public inherited sharing class GenesisApplicationTriggerHandlerV2 {

public static final String COMMITMENT = 'Commitment';

public static final String SETTLEMENT = 'Settlement';

public static final String MAKINSON = 'Makinson';

public static final String MSA = 'MSA';

public static final String PURCELL\_PARTNERS = 'Purcell Partners';

public static final String CREDIT\_APPROVAL\_INTERIM = 'CREDIT APPROVAL - INTERIM';

public static final String CONDITIONAL\_APPROVAL\_INTERIM = 'CONDITIONAL APPROVAL - INTERIM';

public static final String GENERATE\_SCHEDULE = 'generateSchedule';

public static final String POPULATE\_AND\_UPDATE\_RATE\_SCHEDULE = 'populateAndUpdateRateSchedule';

public static final String LOAN = 'LOAN';

public static final String NEW\_PARTIAL = 'NEW-PARTIAL';

public static final String PRINCIPAL\_AND\_INTEREST = 'Principal & Interest';

public static final String INTEREST\_ONLY = 'Interest Only';

public static final String RESIDENTIAL\_ACCOUNT\_MANAGER = 'Residential Account Manager';

public static final String RELATIONSHIP\_MANAGER = 'relationship manager';

public static Boolean recursiveflag = true;

public static Set<String> paymentTypes = new Set<String>{'Principal & Interest','Interest Only'};

private static final String MAX\_RESIDENTIAL = 'Max Residential';

private static final String MAX\_RESIDENTIAL\_RATEMONEY = 'Max Residential RateMoney';

/\*\*

\* @description This method executes various trigger events for the genesis\_\_Applications\_\_c object, calling appropriate handler methods based on the trigger context.

\* @author Swarna Roy | 22-07-2024

\* @param isBefore

\* @param isAfter

\* @param isInsert

\* @param isUpdate

\* @param isDelete

\* @param newApplicationList

\* @param newApplicationMap

\* @param oldApplicationList

\* @param oldApplicationMap

\*\*/

public static void executeTriggerEvents(boolean isBefore, boolean isAfter, boolean isInsert, boolean isUpdate, boolean isDelete,

List<genesis\_\_Applications\_\_c> newApplicationList, Map<Id, genesis\_\_Applications\_\_c> newApplicationMap,

List<genesis\_\_Applications\_\_c> oldApplicationList, Map<Id, genesis\_\_Applications\_\_c> oldApplicationMap) {

if(isBefore) {

if(isInsert) {

beforeInsert(newApplicationList);

} else if(isUpdate) {

beforeUpdate(newApplicationList, oldApplicationMap);

}

} else if(isAfter) {

if(isInsert) {

afterInsert(newApplicationList);

} else if(isUpdate) {

afterUpdate(newApplicationList, oldApplicationMap);

}

}

}

/\*\*

\* @description This method is triggered in the after update event and performs a variety of updates and calculations on loan applications and

related records based on changes in status, ownership, and several other fields to determine if updates are required, then performs actions such as recalculating fees,

updating related records, and triggering additional processes like Simpology messaging or recalculating margins.

It manages specific actions like handling broker contact changes and repayment schedule updates, with a recursive flag to control certain update operations.

\* @author Swarna Roy | 30-07-2024

\* @param newApp

\* @param oldMap

\*\*/

public static void afterUpdate(List<genesis\_\_Applications\_\_c> newApp, Map<Id, genesis\_\_Applications\_\_c> oldMap){

Boolean queryRequired = false;

Date settlementDate = null;

Set<Id> parentIds = new Set<Id>();

Set<Id> recordIds = new Set<Id>();

Set<Id> workflowStatusUpdate = new Set<Id>();

Set<Id> updateFee = new Set<Id>();

Set<Id> updateBrand = new Set<Id>();

Set<Id> updateAge = new Set<Id>();

Set<Id> allParentAppIds = new Set<Id>();

Set<Id> ownerIds = new Set<Id>();

Set<Id> appsToUpdate = new Set<Id>();

Set<Id> appIds = new Set<Id>();

Set<Id> parentInterestIds = new Set<Id>();

Set<Id> pkgIds = new Set<Id>();

Set<Id> ltvUpdateIds = new Set<Id>();

Set<Id> brokerTrailIds = new Set<Id>();

Set<Id> updatePurpose = new Set<Id>();

Set<Id> updateDate = new Set<Id>();

Set<Id> accIds = new Set<Id>();

Set<Id> calculateMarginIds = new Set<Id>();

Set<Id> updateKeyFieldIds = new Set<Id>();

Set<Id> updateExpectedCloseDate = new Set<Id>();

Set<Id> simpologyDocIds = new Set<Id>();

Set<Id> updateAggregator = new Set<Id>();

Set<Id> updateNccpAndSlaTier = new Set<Id>();

Set<Id> recalculateFeeForSmsfResi = new Set<Id>();

Set<Id> marginChangedIds = new Set<Id>();

Set<Id> updateSchedule = new Set<Id>();

Set<Id> loanIds = new Set<Id>();

List<genesis\_\_Applications\_\_c> simpologyBCMApps = new List <genesis\_\_Applications\_\_c>();

List<genesis\_\_Applications\_\_c> allChildAppsRead = new List <genesis\_\_Applications\_\_c>();

List<genesis\_\_Applications\_\_c> allChildAppsUpdate = new List <genesis\_\_Applications\_\_c>();

List<genesis\_\_Applications\_\_c> allParentAppsRead = new List <genesis\_\_Applications\_\_c>();

List<genesis\_\_Applications\_\_c> allParentAppsUpdate = new List <genesis\_\_Applications\_\_c>();

List<genesis\_\_Applications\_\_c> allAppsToUpdate = new List<genesis\_\_Applications\_\_c>();

Map<Id, String> parentOldOwnerMap = new Map<Id, String>();

Map<Id, Decimal> pkgBrokerTrailMap = new Map<Id, Decimal>();

Map<Id, Date> appDateMap = new Map<Id, Date>();

Map<Id, Date> appDisbDateMap = new Map<Id, Date>();

Map<Id, Id> brokerContactAppIds = new Map<Id, Id>();

Map<Id, genesis\_\_Applications\_\_c> allChildAppsMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> allChildAppsMapRead = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> allParentAppsMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> allParentAppsMapRead = new Map<Id, genesis\_\_Applications\_\_c>();

// For NGN-40, NextGen BCM framework

CLO\_BCM\_Publisher.publishStatusPlatformEventHandler(newApp, oldMap);

for(genesis\_\_Applications\_\_c app: newApp) {

genesis\_\_Applications\_\_c oldParentApp = oldMap.get(app.Id);

if(app.Record\_Type\_Name\_\_c == TTConstants.APP\_PACKAGE){

allParentAppIds.add(app.Id);

if((app.genesis\_\_Status\_\_c != oldParentApp.genesis\_\_Status\_\_c

&& app.genesis\_\_Status\_\_c != CONDITIONAL\_APPROVAL\_INTERIM

&& app.genesis\_\_Status\_\_c != CREDIT\_APPROVAL\_INTERIM)

|| app.Workflow\_Status\_\_c != oldParentApp.Workflow\_Status\_\_c){

queryRequired = true;

parentIds.add(app.Id);

if(app.genesis\_\_Status\_\_c != TTConstants.NEW\_PARTIAL && !TTConstants.REHASH\_STATUSES.contains(app.genesis\_\_Status\_\_c)){

RehashUtility.triggerRehash(app.Id);

}

if(app.genesis\_\_Status\_\_c == TTConstants.PRELIM\_REVIEW) {

updateNccpAndSlaTier.add(app.Id);

}

if(app.Workflow\_Status\_\_c != oldParentApp.Workflow\_Status\_\_c) {

workflowStatusUpdate.add(app.Id);

}

}

if(app.OwnerId != oldParentApp.OwnerId){

queryRequired = true;

recordIds.add(app.Id);

ownerIds.add(app.OwnerId);

if(app.genesis\_\_Status\_\_c != TTConstants.NEW\_PARTIAL){

parentOldOwnerMap.put(app.Id,oldParentApp.OwnerId);

}

}

//Simpology Back Channel messaging

if(app.Is\_Simpology\_Application\_\_c == true && app.genesis\_\_Status\_\_c != oldParentApp.genesis\_\_Status\_\_c){

if(app.genesis\_\_Status\_\_c != TTConstants.NEW\_PARTIAL){

simpologyBCMApps.add(app);

}

}

// Brand to be updated when Aggregator changes.

// When Brand is null, the updateBranding method will always populate Brand\_\_c

if(app.Aggregator\_\_c != null && ((app.Aggregator\_\_c != oldParentApp.Aggregator\_\_c)

|| (app.Brand\_\_c == null && app.genesis\_\_Asset\_Class\_\_c != null))) {

queryRequired = true;

updateBrand.add(app.Id);

}

if(app.Rate\_Index\_\_c != oldParentApp.Rate\_Index\_\_c && oldParentApp.Rate\_Index\_\_c != null){

queryRequired = true;

pkgIds.add(app.Id);

}

if(app.Rate\_Index\_\_c != oldParentApp.Rate\_Index\_\_c){

queryRequired = true;

parentInterestIds.add(app.Id);

}

if(app.Is\_Further\_Advance\_App\_\_c == true){

if(app.genesis\_\_Requested\_Loan\_Amount\_\_c != oldParentApp.genesis\_\_Requested\_Loan\_Amount\_\_c && app.genesis\_\_Status\_\_c != TTConstants.NEW\_PARTIAL){

queryRequired = true;

ltvUpdateIds.add(app.Id);

}

} else {

if(app.genesis\_\_Total\_Facility\_Amount\_\_c != oldParentApp.genesis\_\_Total\_Facility\_Amount\_\_c && app.genesis\_\_Status\_\_c != TTConstants.NEW\_PARTIAL){

queryRequired = true;

ltvUpdateIds.add(app.Id);

} else if(app.Is\_Simpology\_Application\_\_c == true && app.genesis\_\_Status\_\_c != oldParentApp.genesis\_\_Status\_\_c && app.genesis\_\_Status\_\_c == TTConstants.NEW\_ENTERED){

queryRequired = true;

ltvUpdateIds.add(app.Id);

}

}

if(app.Broker\_Trail\_\_c != oldParentApp.Broker\_Trail\_\_c){

queryRequired = true;

brokerTrailIds.add(app.Id);

pkgBrokerTrailMap.put(app.Id, app.Broker\_Trail\_\_c);

}

if(app.Penciled\_In\_Status\_\_c != oldParentApp.Penciled\_In\_Status\_\_c || app.genesis\_\_Expected\_Close\_Date\_\_c != oldParentApp.genesis\_\_Expected\_Close\_Date\_\_c

|| app.genesis\_\_Disbursement\_Date\_\_c != oldParentApp.genesis\_\_Disbursement\_Date\_\_c

|| app.NCCP\_\_c != oldParentApp.NCCP\_\_c || app.Funding\_Date\_\_c != oldParentApp.Funding\_Date\_\_c){

queryRequired = true;

updateKeyFieldIds.add(app.Id);

if(app.genesis\_\_Expected\_Close\_Date\_\_c != oldParentApp.genesis\_\_Expected\_Close\_Date\_\_c && app.genesis\_\_Expected\_Close\_Date\_\_c != null) {

updateExpectedCloseDate.add(app.Id);

}

}

if(app.genesis\_\_Asset\_Class\_\_c != TTConstants.RESIDENTIAL\_ASSET\_CLASS && app.genesis\_\_Asset\_Class\_\_c != TTConstants.SMSF\_RESIDENTIAL\_ASSET\_CLASS){

if(app.genesis\_\_CLTV\_\_c != oldParentApp.genesis\_\_CLTV\_\_c){

RecalculateFeeQueueable recalculateFee = new RecalculateFeeQueueable (app.Id);

System.enqueueJob(recalculateFee);

} else if(app.genesis\_\_Loan\_Amount\_\_c != oldParentApp.genesis\_\_Loan\_Amount\_\_c && app.Is\_Further\_Advance\_App\_\_c == true){

CalculateFeeForCommercialAndSMSF.calculateFee(app.Id);

}

} else if(app.genesis\_\_Asset\_Class\_\_c == TTConstants.SMSF\_RESIDENTIAL\_ASSET\_CLASS) {

if(app.genesis\_\_CLTV\_\_c != oldParentApp.genesis\_\_CLTV\_\_c){

recalculateFeeForSmsfResi.add(app.Id);

}

}

if(app.genesis\_\_Asset\_Class\_\_c == TTConstants.RESIDENTIAL\_ASSET\_CLASS && app.genesis\_\_Status\_\_c != TTConstants.NEW\_PARTIAL){

if(app.genesis\_\_CLTV\_\_c != oldParentApp.genesis\_\_CLTV\_\_c){

queryRequired = true;

calculateMarginIds.add(app.Id);

} else if(app.Brand\_\_c != oldParentApp.Brand\_\_c && app.Brand\_\_c != null){

queryRequired = true;

calculateMarginIds.add(app.Id);

}

}

if(app.Is\_Simpology\_Application\_\_c == true){

if(app.genesis\_\_Status\_\_c != oldParentApp.genesis\_\_Status\_\_c && app.genesis\_\_Status\_\_c == TTConstants.NEW\_ENTERED){

simpologyDocIds.add(app.Id);

}

}

if(app.genesis\_\_Status\_\_c != oldParentApp.genesis\_\_Status\_\_c

&& app.genesis\_\_Status\_\_c != TTConstants.NEW\_ENTERED

&& app.genesis\_\_Status\_\_c != TTConstants.NEW\_PARTIAL

&& app.genesis\_\_Status\_\_c != CONDITIONAL\_APPROVAL\_INTERIM

&& app.genesis\_\_Status\_\_c != CREDIT\_APPROVAL\_INTERIM){

updateAggregator.add(app.Id);

}

if(app.Is\_Simpology\_Application\_\_c && app.Broker\_Contact\_\_c != null && app.Broker\_Contact\_\_c != oldParentApp.Broker\_Contact\_\_c){

queryRequired = true;

brokerContactAppIds.put(app.Broker\_Contact\_\_c, app.Id);

}

} else {

allParentAppIds.add(app.genesis\_\_Parent\_Application\_\_c);

if(app.genesis\_\_Status\_\_c != NEW\_PARTIAL && app.Payment\_Type\_\_c != null && paymentTypes.contains(app.Payment\_Type\_\_c)

&& (app.genesis\_\_Payment\_Amount\_\_c != oldParentApp.genesis\_\_Payment\_Amount\_\_c

|| (app.P\_I\_Repayment\_Amount\_\_c == null && app.IO\_Repayment\_Amount\_\_c == null && app.Construction\_Repayment\_Amount\_\_c == null)

|| (app.Payment\_Type\_\_c == PRINCIPAL\_AND\_INTEREST && app.genesis\_\_Payment\_Amount\_\_c != app.P\_I\_Repayment\_Amount\_\_c)

|| (app.Payment\_Type\_\_c == INTEREST\_ONLY && app.Construction\_Term\_\_c == null && app.genesis\_\_Payment\_Amount\_\_c != app.IO\_Repayment\_Amount\_\_c)

|| (app.Payment\_Type\_\_c == INTEREST\_ONLY && app.Construction\_Term\_\_c != null && app.genesis\_\_Payment\_Amount\_\_c != app.Construction\_Repayment\_Amount\_\_c))) {

queryRequired = true;

loanIds.add(app.Id);

}

if(app.genesis\_\_Status\_\_c == TTConstants.CHANGE\_MEMO\_APPROVAL){

if(app.Thinktank\_Margin\_\_c != oldParentApp.Thinktank\_Margin\_\_c || app.Pricing\_Basis\_\_c != oldParentApp.Pricing\_Basis\_\_c || app.Payment\_Type\_\_c != oldParentApp.Payment\_Type\_\_c){

queryRequired = true;

appIds.add(app.Id);

if(app.Thinktank\_Margin\_\_c != oldParentApp.Thinktank\_Margin\_\_c && app.Thinktank\_Margin\_\_c != null) {

marginChangedIds.add(app.Id);

}

}

} else {

if(app.Pricing\_Basis\_\_c != oldParentApp.Pricing\_Basis\_\_c && app.Payment\_Type\_\_c != oldParentApp.Payment\_Type\_\_c){

queryRequired = true;

appIds.add(app.Id);

}

else if(app.Pricing\_Basis\_\_c != oldParentApp.Pricing\_Basis\_\_c && app.Payment\_Type\_\_c == oldParentApp.Payment\_Type\_\_c){

queryRequired = true;

appIds.add(app.Id);

}

else if(app.Pricing\_Basis\_\_c == oldParentApp.Pricing\_Basis\_\_c && app.Payment\_Type\_\_c != oldParentApp.Payment\_Type\_\_c){

queryRequired = true;

appIds.add(app.Id);

}

}

if(app.genesis\_\_Term\_\_c != oldParentApp.genesis\_\_Term\_\_c && app.genesis\_\_Term\_\_c != null){

queryRequired = true;

updateAge.add(app.genesis\_\_Parent\_Application\_\_c);

}

if((app.genesis\_\_Term\_\_c != oldParentApp.genesis\_\_Term\_\_c && app.genesis\_\_Term\_\_c != null)

|| (app.genesis\_\_Loan\_Amount\_\_c != oldParentApp.genesis\_\_Loan\_Amount\_\_c && app.genesis\_\_Loan\_Amount\_\_c != null)){

updateSchedule.add(app.Id);

}

if (app.genesis\_\_CL\_Purpose\_\_c != oldParentApp.genesis\_\_CL\_Purpose\_\_c){

queryRequired = true;

updatePurpose.add(app.genesis\_\_Parent\_Application\_\_c);

}

if(app.genesis\_\_Disbursement\_Date\_\_c != oldParentApp.genesis\_\_Disbursement\_Date\_\_c && app.genesis\_\_Disbursement\_Date\_\_c != null){

queryRequired = true;

updateDate.add(app.Id);

appDisbDateMap.put(app.Id, app.genesis\_\_Disbursement\_Date\_\_c);

if(app.NCCP\_\_c == false && app.genesis\_\_Disbursement\_Date\_\_c.day() > 28){

settlementDate = app.genesis\_\_Disbursement\_Date\_\_c.adddays(-(app.genesis\_\_Disbursement\_Date\_\_c.day()-28));

}else{

settlementDate = app.genesis\_\_Disbursement\_Date\_\_c;

}

appDateMap.put(app.Id, settlementDate);

} else if(app.genesis\_\_Expected\_Close\_Date\_\_c != oldParentApp.genesis\_\_Expected\_Close\_Date\_\_c && app.genesis\_\_Expected\_Close\_Date\_\_c != null){

queryRequired = true;

updateDate.add(app.Id);

appDisbDateMap.put(app.Id, app.genesis\_\_Expected\_Close\_Date\_\_c);

if(app.NCCP\_\_c == false && app.genesis\_\_Expected\_Close\_Date\_\_c.day() > 28){

settlementDate = app.genesis\_\_Expected\_Close\_Date\_\_c.adddays(-(app.genesis\_\_Expected\_Close\_Date\_\_c.day()-28));

} else {

settlementDate = app.genesis\_\_Expected\_Close\_Date\_\_c;

}

appDateMap.put(app.Id,settlementDate);

}

}

if(app.Account\_Name\_\_c != oldParentApp.Account\_Name\_\_c && app.Borrower\_Name\_\_c != app.Account\_Name\_\_c){

accIds.add(app.genesis\_\_Account\_\_c);

} else if(app.Borrower\_Name\_\_c != app.Account\_Name\_\_c){

accIds.add(app.genesis\_\_Account\_\_c);

}

if(app.genesis\_\_Total\_Fee\_Amount\_\_c != oldParentApp.genesis\_\_Total\_Fee\_Amount\_\_c){

updateFee.add(app.Id);

}

}

if(!allParentAppIds.isEmpty() && queryRequired){

allChildAppsRead = [SELECT Id,

Name,

OwnerId,

genesis\_\_Status\_\_c,

Workflow\_Status\_\_c,

Record\_Type\_Name\_\_c,

genesis\_\_Parent\_Application\_\_c,

Thinktank\_Margin\_\_c,

Parent\_Rate\_Index\_\_c,

Pricing\_Basis\_\_c,

Payment\_Type\_\_c,

Broker\_Trail\_\_c,

genesis\_\_CL\_Purpose\_\_r.Name,

genesis\_\_Asset\_Class\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

Funding\_Date\_\_c,

NCCP\_\_c,

genesis\_\_Term\_\_c,

Solicitor\_Firm\_\_c,

Date\_Solicitor\_Instructed\_\_c,

Conditional\_Credit\_Underwriting\_Date\_\_c,

Ready\_To\_Instruct\_Date\_\_c,

genesis\_\_CL\_Product\_\_r.Product\_Type\_\_c,

genesis\_\_Parent\_Application\_\_r.Is\_Simpology\_Application\_\_c,

genesis\_\_Parent\_Application\_\_r.genesis\_\_Status\_\_c,

genesis\_\_Parent\_Application\_\_r.LTVP\_\_c,

genesis\_\_Parent\_Application\_\_r.Brand\_\_c,

genesis\_\_Parent\_Application\_\_r.Broker\_Trail\_\_c,

genesis\_\_Parent\_Application\_\_r.genesis\_\_CL\_Product\_\_r.Document\_Type\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = :TTConstants.APP\_LOAN

AND genesis\_\_Parent\_Application\_\_c in :allParentAppIds];

if(!allChildAppsRead.isEmpty()) {

allChildAppsUpdate = allChildAppsRead.deepClone(true, false, true);

for(genesis\_\_Applications\_\_c splitLoan: allChildAppsRead) {

if(parentInterestIds.contains(splitLoan.genesis\_\_Parent\_Application\_\_c)) {

appIds.add(splitLoan.Id);

}

allChildAppsMapRead.put(splitLoan.Id, splitLoan);

}

for(genesis\_\_Applications\_\_c splitLoan: allChildAppsUpdate) {

allChildAppsMap.put(splitLoan.Id, splitLoan);

}

}

allParentAppsRead = [SELECT Id,

Name,

OwnerId,

genesis\_\_Status\_\_c,

Workflow\_Status\_\_c,

Record\_Type\_Name\_\_c,

Brand\_\_c,

Fee\_Set\_\_c,

Fee\_Set\_\_r.Name,

Aggregator\_\_r.Name,

Brand\_Type\_\_c,

Rate\_Index\_\_c,

Funding\_Date\_\_c,

NCCP\_\_c,

Solicitor\_Firm\_\_c,

Date\_Solicitor\_Instructed\_\_c,

Conditional\_Credit\_Underwriting\_Date\_\_c,

Ready\_To\_Instruct\_Date\_\_c,

genesis\_\_Asset\_Class\_\_c,

Invalid\_Queue\_User\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Borrower\_Age\_\_c,

Borrower\_Age\_at\_term\_end\_\_c,

Consolidated\_Purpose\_\_c,

genesis\_\_CL\_Product\_\_r.Document\_Type\_\_c,

LTVP\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Is\_Simpology\_Application\_\_c,

genesis\_\_CL\_Product\_\_r.Product\_Type\_\_c,

Total\_Collateral\_Amount\_\_c,

genesis\_\_CLTV\_\_c,

Is\_Further\_Advance\_App\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

Broker\_Relationship\_\_r.Brand\_Name\_\_c,

Broker\_Relationship\_\_r.Dual\_Accreditation\_\_c,

Broker\_Relationship\_\_r.Direct\_Accreditation\_\_c,

Resi\_COMMSMSF\_Combination\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = :TTConstants.APP\_PACKAGE

AND Id in :allParentAppIds];

if(!allParentAppsRead.isEmpty()) {

allParentAppsUpdate = allParentAppsRead.deepClone(true, false, true);

for(genesis\_\_Applications\_\_c app: allParentAppsRead) {

allParentAppsMapRead.put(app.Id, app);

}

for(genesis\_\_Applications\_\_c app: allParentAppsUpdate) {

allParentAppsMap.put(app.Id, app);

}

}

}

if(!simpologyBCMApps.isEmpty()) {

CLXUtil.createSimpologyBCM(simpologyBCMApps);

}

if(!parentIds.isEmpty() || !recordIds.isEmpty()){

for(genesis\_\_Applications\_\_c each: updateSplitStatusAndOwner(parentIds, recordIds, workflowStatusUpdate, allChildAppsUpdate, allParentAppsUpdate, allParentAppsMapRead)) {

if(!appsToUpdate.contains(each.Id)) {

allAppsToUpdate.add(each);

}

appsToUpdate.add(each.Id);

}

}

if(!recordIds.isEmpty()){

for(genesis\_\_Applications\_\_c each: updateAndValidateTaskOwner(recordIds, ownerIds, allChildAppsRead, allParentAppsMap, allParentAppsMapRead, parentOldOwnerMap)) {

if(!appsToUpdate.contains(each.Id)) {

allAppsToUpdate.add(each);

}

appsToUpdate.add(each.Id);

}

}

if(!updateNccpAndSlaTier.isEmpty()){

for(genesis\_\_Applications\_\_c each: updateNccpFlagAndSlaTier(updateNccpAndSlaTier, allParentAppsUpdate)) {

if(!appsToUpdate.contains(each.Id)) {

allAppsToUpdate.add(each);

}

appsToUpdate.add(each.Id);

}

}

if(!updateAge.isEmpty()) {

for(genesis\_\_Applications\_\_c each: updateBorrowerAge(updateAge, allChildAppsRead, allParentAppsRead, allParentAppsMap)) {

if(!appsToUpdate.contains(each.Id)) {

allAppsToUpdate.add(each);

}

appsToUpdate.add(each.Id);

}

}

if(!updatePurpose.isEmpty()) {

for(genesis\_\_Applications\_\_c each: updateConsolidatedPurpose(updatePurpose, allChildAppsRead, allParentAppsMap)) {

if(!appsToUpdate.contains(each.Id)) {

allAppsToUpdate.add(each);

}

appsToUpdate.add(each.Id);

}

}

if(!ltvUpdateIds.isEmpty()) {

for(genesis\_\_Applications\_\_c each: calculatePackageLtv(ltvUpdateIds, allParentAppsRead, allParentAppsMap)) {

if(!appsToUpdate.contains(each.Id)) {

allAppsToUpdate.add(each);

}

appsToUpdate.add(each.Id);

}

}

if(!updateDate.isEmpty()) {

for(genesis\_\_Applications\_\_c each: updateLoanDates(updateDate, settlementDate, appDateMap, appDisbDateMap, allChildAppsUpdate)) {

if(!appsToUpdate.contains(each.Id)) {

allAppsToUpdate.add(each);

}

appsToUpdate.add(each.Id);

}

}

if(!brokerContactAppIds.isEmpty()) {

for(genesis\_\_Applications\_\_c each: handleBrokerChange(brokerContactAppIds, allParentAppsMap)) {

if(!appsToUpdate.contains(each.Id)) {

allAppsToUpdate.add(each);

}

appsToUpdate.add(each.Id);

}

}

if(!loanIds.isEmpty() && recursiveflag) {

for(genesis\_\_Applications\_\_c each: updateRepaymentAmount(loanIds, allChildAppsMap)) {

if(!appsToUpdate.contains(each.Id)) {

allAppsToUpdate.add(each);

}

appsToUpdate.add(each.Id);

}

}

if(!allAppsToUpdate.isEmpty()) {

update allAppsToUpdate;

}

if(!updateSchedule.isEmpty()) {

rescheduleLoan(updateSchedule);

}

if(!updateBrand.isEmpty()) {

updateBranding(updateBrand, allParentAppsRead, allParentAppsMap);

}

if(!appIds.isEmpty() || !pkgIds.isEmpty()) {

updateInterestInformation(appIds, pkgIds, marginChangedIds, allParentAppsRead, allChildAppsRead, allParentAppsMap);

}

if(!updateFee.isEmpty()) {

updateFeeAmountIncludingGst(updateFee);

}

if(!recalculateFeeForSmsfResi.isEmpty()) {

recalculateFee(recalculateFeeForSmsfResi, allChildAppsRead);

}

if(!brokerTrailIds.isEmpty()) {

updateBrokerTrail(brokerTrailIds, pkgBrokerTrailMap, allChildAppsRead, allChildAppsMap);

}

if(!accIds.isEmpty()) {

updateBorrowerName(accIds);

}

if(!updateKeyFieldIds.isEmpty()) {

updateKeyFields(updateKeyFieldIds, updateExpectedCloseDate, allChildAppsUpdate, allParentAppsMapRead);

}

if(!calculateMarginIds.isEmpty()) {

calculateMargin(calculateMarginIds, allChildAppsRead, allParentAppsRead);

}

if(!simpologyDocIds.isEmpty()) {

simpologyDocumentInsert(simpologyDocIds);

}

if(!updateAggregator.isEmpty()) {

updateAggregatorOnAccounts(updateAggregator);

}

}

/\*\*

\* @description This method is triggered in the before update event and handles various updates on loan applications, including recalculating LTV ratios, updating workflow statuses, and reassigning ownership based on changes in application fields such as status, loan amounts, and execution flows.

\* @author Swarna Roy | 30-07-2024

\* @param newApp

\* @param oldMap

\*\*/

public static void beforeUpdate(List<genesis\_\_Applications\_\_c> newApp, Map<Id, genesis\_\_Applications\_\_c> oldMap){

Id workflowUserId;

Set <Id> updateLTVIds = new Set <Id>();

Set <Id> updateLTVParentIds = new Set <Id>();

Set <String> statusList = new Set <String>();

Map <String, String> statusFlowMapping = new Map<String, String>();

List<User> workflowUser = new List<User>();

List<genesis\_\_Applications\_\_c> updateOwner = new List<genesis\_\_Applications\_\_c>();

List<genesis\_\_Applications\_\_c> updateWorkflowStatus = new List<genesis\_\_Applications\_\_c>();

for(genesis\_\_Applications\_\_c app: newApp) {

genesis\_\_Applications\_\_c oldParentApp = oldMap.get(app.Id);

if(app.Record\_Type\_Name\_\_c == TTConstants.APP\_PACKAGE){

if(app.genesis\_\_Total\_Facility\_Amount\_\_c != oldParentApp.genesis\_\_Total\_Facility\_Amount\_\_c && app.genesis\_\_Status\_\_c == TTConstants.CHANGE\_MEMO\_APPROVAL){

app.genesis\_\_Requested\_Loan\_Amount\_\_c = app.genesis\_\_Total\_Facility\_Amount\_\_c;

app.genesis\_\_Loan\_Amount\_\_c = app.genesis\_\_Total\_Facility\_Amount\_\_c;

}

if(app.genesis\_\_Status\_\_c != oldParentApp.genesis\_\_Status\_\_c

&& app.genesis\_\_Status\_\_c != TTConstants.NEW\_PARTIAL

&& app.genesis\_\_Status\_\_c != TTConstants.FUNDED

&& app.genesis\_\_Status\_\_c != TTConstants.APPROVED\_CONVERTED\_TO\_CONTRACT){

if(app.genesis\_\_Status\_\_c == TTConstants.TREASURY\_SKIP){

updateOwner.add(app);

} else {

updateWorkflowStatus.add(app);

statusList.add(app.genesis\_\_Status\_\_c);

}

}

} else {

if(app.genesis\_\_Requested\_Loan\_Amount\_\_c != oldParentApp.genesis\_\_Requested\_Loan\_Amount\_\_c){

updateLTVIds.add(app.Id);

updateLTVParentIds.add(app.genesis\_\_Parent\_Application\_\_c);

}

}

}

if(!updateLTVIds.isEmpty()) {

calculateSplitLtv(updateLTVIds, updateLTVParentIds, newApp);

}

if(!updateOwner.isEmpty()) {

workflowUser = [SELECT Id

FROM User

WHERE Name = :TTConstants.JWF\_WORKFLOW

AND IsActive = true LIMIT 1];

if(!workflowUser.isEmpty()) {

workflowUserId = workflowUser[0].Id;

}

for(genesis\_\_Applications\_\_c app: updateOwner) {

app.OwnerId = workflowUserId;

}

}

if(!updateWorkflowStatus.isEmpty()) {

List<Execution\_Flow\_Action\_Junction\_\_c> flow = [SELECT Workflow\_Status\_\_c,

Execution\_Flow\_\_c

FROM Execution\_Flow\_Action\_Junction\_\_c

WHERE Next\_Status\_\_c IN :statusList];

for(Execution\_Flow\_Action\_Junction\_\_c each: flow) {

statusFlowMapping.put(each.Execution\_Flow\_\_c, each.Workflow\_Status\_\_c);

}

for(genesis\_\_Applications\_\_c app: updateWorkflowStatus) {

String status = statusFlowMapping.get(app.Execution\_Flow\_\_c);

if(status != null){

app.Workflow\_Status\_\_c = status;

}

}

}

}

/\*\*

\* @description This method is triggered before inserting new genesis\_\_Applications\_\_c records, performing initial setup tasks such as calculating LTV ratios, setting default values for various fields, and checking sales queue assignments for non-Simpology applications.

\* @author Swarna Roy | 30-07-2024

\* @param newApp

\*\*/

public static void beforeInsert(List<genesis\_\_Applications\_\_c> newApp){

Set <Id> updateLTVIds = new Set <Id>();

Map<Id, String> parentOwnerMap = new Map<Id, String>();

for(genesis\_\_Applications\_\_c app: newApp) {

if(app.Record\_Type\_Name\_\_c == TTConstants.APP\_PACKAGE){

if(app.Is\_Simpology\_Application\_\_c == false){

parentOwnerMap.put(app.Id, app.OwnerId);

}

} else {

updateLTVIds.add(app.genesis\_\_Parent\_Application\_\_c);

}

app.Borrower\_Name\_\_c = app.Account\_Name\_\_c;

app.Document\_Asset\_Class\_\_c = app.genesis\_\_Asset\_Class\_\_c;

app.Start\_DateTime\_at\_Current\_Status\_\_c = Datetime.now();

app.Start\_Date\_at\_Current\_Status\_\_c = Date.today();

app.Start\_Date\_at\_Current\_Workflow\_Status\_\_c = Date.today();

}

if(!updateLTVIds.isEmpty()) {

calculateSplitLtvBeforeInsert(updateLTVIds, newApp);

}

if(!parentOwnerMap.isEmpty()) {

salesQueueCheck(parentOwnerMap, newApp);

}

}

/\*\*

\* @description This method is triggered in the after insert event for genesis\_\_Applications\_\_c records by updating related child and parent records, managing interest information, and consolidating purposes based on the new data.

It performs queries to gather relevant records, updates interest and purpose information, and ensures all necessary records are updated accordingly.

\* @author Swarna Roy | 30-07-2024

\* @param newApp

\*\*/

public static void afterInsert(List<genesis\_\_Applications\_\_c> newApp){

Set<Id> parentIds = new Set<Id>();

Set<Id> updatePurposeIds = new Set<Id>();

Set<Id> allParentAppIds = new Set<Id>();

Set<Id> appsToUpdate = new Set<Id>();

Map<Id,Id> brokerContactAppIds = new Map<Id,Id>();

List<genesis\_\_Applications\_\_c> allChildAppsUpdate = new List <genesis\_\_Applications\_\_c>();

List<genesis\_\_Applications\_\_c> allParentAppsUpdate = new List <genesis\_\_Applications\_\_c>();

List<genesis\_\_Applications\_\_c> packageToUpdate = new List <genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> allChildAppsMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> allParentAppsMap = new Map<Id, genesis\_\_Applications\_\_c>();

for(genesis\_\_Applications\_\_c app: newApp) {

if(app.Record\_Type\_Name\_\_c == TTConstants.APP\_LOAN){

allParentAppIds.add(app.genesis\_\_Parent\_Application\_\_c);

parentIds.add(app.genesis\_\_Parent\_Application\_\_c);

if(app.genesis\_\_CL\_Purpose\_\_c != null){

updatePurposeIds.add(app.genesis\_\_Parent\_Application\_\_c);

}

} else {

allParentAppIds.add(app.Id);

if(app.Is\_Simpology\_Application\_\_c && app.Broker\_Contact\_\_c != null){

brokerContactAppIds.put(app.Broker\_Contact\_\_c, app.Id);

}

}

}

if(!allParentAppIds.isEmpty()){

allChildAppsUpdate = [SELECT Id,

Name,

NCCP\_\_c,

genesis\_\_Term\_\_c,

genesis\_\_CL\_Purpose\_\_r.Name,

genesis\_\_Parent\_Application\_\_r.NCCP\_\_c,

genesis\_\_Parent\_Application\_\_r.genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Parent\_Application\_\_r.genesis\_\_Expected\_Close\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c, genesis\_\_Expected\_Close\_Date\_\_c,

genesis\_\_Asset\_Class\_\_c,

Payment\_Type\_\_c,

genesis\_\_Parent\_Application\_\_r.genesis\_\_CL\_Product\_\_r.Document\_Type\_\_c,

genesis\_\_Parent\_Application\_\_c,

genesis\_\_Parent\_Application\_\_r.LTVP\_\_c,

genesis\_\_Parent\_Application\_\_r.Brand\_\_c,

genesis\_\_Parent\_Application\_\_r.Broker\_Trail\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = :TTConstants.APP\_LOAN

AND genesis\_\_Parent\_Application\_\_c in :allParentAppIds];

if(!allChildAppsUpdate.isEmpty()) {

for(genesis\_\_Applications\_\_c splitLoan: allChildAppsUpdate) {

allChildAppsMap.put(splitLoan.Id, splitLoan);

}

}

allParentAppsUpdate = [SELECT Id,

Name,

Broker\_Trail\_\_c,

Consolidated\_Purpose\_\_c,

genesis\_\_CL\_Product\_Name\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = :TTConstants.APP\_PACKAGE

AND Id in :allParentAppIds];

if(!allParentAppsUpdate.isEmpty()) {

for(genesis\_\_Applications\_\_c app: allParentAppsUpdate) {

allParentAppsMap.put(app.Id, app);

}

}

}

if(!parentIds.isEmpty()){

insertInterestInformationAndUpdateAge(parentIds, newApp, allChildAppsMap, allParentAppsMap);

}

if(!brokerContactAppIds.isEmpty()) {

for(genesis\_\_Applications\_\_c each: handleBrokerChange(brokerContactAppIds, allParentAppsMap)) {

if(!appsToUpdate.contains(each.Id)) {

packageToUpdate.add(each);

}

appsToUpdate.add(each.Id);

}

}

if(!updatePurposeIds.isEmpty()) {

for(genesis\_\_Applications\_\_c each: updateConsolidatedPurpose(updatePurposeIds, allChildAppsUpdate, allParentAppsMap)) {

if(!appsToUpdate.contains(each.Id)) {

packageToUpdate.add(each);

}

appsToUpdate.add(each.Id);

}

}

if(!packageToUpdate.isEmpty()) {

update packageToUpdate;

}

}

/\*\*

\* @description This method updates the status, workflow, and ownership details of both child and parent Application records based on provided sets of IDs and status updates.

It ensures that relevant fields are updated and new values are assigned for each record that meets the criteria.

\* @author Swarna Roy | 31-07-2024

\* @param parentIds

\* @param recordIds

\* @param workflowStatusUpdate

\* @param allChildApps

\* @param allParentsAppUpdate

\* @param allParentAppsMap

\* @return List<genesis\_\_Applications\_\_c>

\*\*/

public static List<genesis\_\_Applications\_\_c> updateSplitStatusAndOwner(Set <Id> parentIds, Set <Id> recordIds, Set <Id> workflowStatusUpdate, List<genesis\_\_Applications\_\_c> allChildApps, List<genesis\_\_Applications\_\_c> allParentsAppUpdate, Map <Id, genesis\_\_Applications\_\_c> allParentAppsMap) {

Boolean isUpdate;

List<genesis\_\_Applications\_\_c> appToUpdate = new List<genesis\_\_Applications\_\_c>();

for(genesis\_\_Applications\_\_c splitLoan: allChildApps) {

isUpdate = false;

String parentId = splitLoan.genesis\_\_Parent\_Application\_\_c;

if(parentIds.contains(parentId)) {

splitLoan.genesis\_\_Status\_\_c = allParentAppsMap.get(parentId).genesis\_\_Status\_\_c;

splitLoan.Workflow\_Status\_\_c = allParentAppsMap.get(parentId).Workflow\_Status\_\_c;

splitLoan.Start\_DateTime\_at\_Current\_Status\_\_c = Datetime.now();

splitLoan.Start\_Date\_at\_Current\_Status\_\_c = Date.today();

if(splitLoan.genesis\_\_Status\_\_c == TTConstants.CONDITIONAL\_APPROVAL) {

splitLoan.Credit\_Approval\_Date\_\_c = Date.today();

if(splitLoan.genesis\_\_CL\_Product\_\_r.Product\_Type\_\_c == TTConstants.COMM\_PROD\_TYPE || splitLoan.genesis\_\_Asset\_Class\_\_c.contains(TTConstants.SMSF\_BRAND\_TYPE)) {

splitLoan.Invoice\_Type\_\_c = COMMITMENT;

}

}

if(splitLoan.genesis\_\_Status\_\_c == TTConstants.READY\_TO\_INSTRUCT) {

splitLoan.Invoice\_Type\_\_c = SETTLEMENT;

}

isUpdate = true;

}

if(workflowStatusUpdate.contains(parentId)) {

splitLoan.Start\_Date\_at\_Current\_Workflow\_Status\_\_c = Date.today();

if(splitLoan.Workflow\_Status\_\_c == TTConstants.SOLICITORS\_INSTRUCTED && splitLoan.Date\_Solicitor\_Instructed\_\_c == null){

splitLoan.Date\_Solicitor\_Instructed\_\_c = Date.today();

}

if((splitLoan.Workflow\_Status\_\_c == TTConstants.CONDITIONAL\_CREDIT\_UNDERWRITING || splitLoan.Workflow\_Status\_\_c == TTConstants.CREDIT\_UNDERWRITING)

&& splitLoan.Conditional\_Credit\_Underwriting\_Date\_\_c == null){

splitLoan.Conditional\_Credit\_Underwriting\_Date\_\_c = Date.today();

}

if(splitLoan.Workflow\_Status\_\_c == TTConstants.READY\_TO\_INSTRUCT && splitLoan.Ready\_To\_Instruct\_Date\_\_c == null){

splitLoan.Ready\_To\_Instruct\_Date\_\_c = Datetime.now();

}

if(String.isBlank(splitLoan.Solicitor\_Firm\_\_c)){

if(splitLoan.genesis\_\_CL\_Product\_\_r.Product\_Type\_\_c == TTConstants.COMM\_PROD\_TYPE || splitLoan.genesis\_\_Asset\_Class\_\_c.contains(TTConstants.SMSF\_BRAND\_TYPE)) {

splitLoan.Solicitor\_Firm\_\_c = MAKINSON;

} else if(splitLoan.genesis\_\_CL\_Product\_\_r.Product\_Type\_\_c == TTConstants.RESI\_PROD\_TYPE && !splitLoan.genesis\_\_Asset\_Class\_\_c.contains(TTConstants.SMSF\_BRAND\_TYPE)) {

splitLoan.Solicitor\_Firm\_\_c = MSA;

}

}

isUpdate = true;

}

if(recordIds.contains(parentId)) {

splitLoan.OwnerId = allParentAppsMap.get(parentId).OwnerId;

isUpdate = true;

}

if(isUpdate) {

appToUpdate.add(splitLoan);

}

}

for(genesis\_\_Applications\_\_c parentApp: allParentsAppUpdate) {

if(parentIds.contains(parentApp.Id)) {

parentApp.Start\_DateTime\_at\_Current\_Status\_\_c = Datetime.now();

parentApp.Start\_Date\_at\_Current\_Status\_\_c = Date.today();

if(parentApp.genesis\_\_Status\_\_c == TTConstants.CONDITIONAL\_APPROVAL) {

parentApp.Credit\_Approval\_Date\_\_c = Date.today();

if(parentApp.genesis\_\_CL\_Product\_\_r.Product\_Type\_\_c == TTConstants.COMM\_PROD\_TYPE || parentApp.genesis\_\_Asset\_Class\_\_c.contains(TTConstants.SMSF\_BRAND\_TYPE)) {

parentApp.Invoice\_Type\_\_c = COMMITMENT;

}

}

if(parentApp.genesis\_\_Status\_\_c == TTConstants.READY\_TO\_INSTRUCT) {

parentApp.Invoice\_Type\_\_c = SETTLEMENT;

}

appToUpdate.add(parentApp);

}

if(workflowStatusUpdate.contains(parentApp.Id)) {

parentApp.Start\_Date\_at\_Current\_Workflow\_Status\_\_c = Date.today();

if(parentApp.Workflow\_Status\_\_c == TTConstants.SOLICITORS\_INSTRUCTED && parentApp.Date\_Solicitor\_Instructed\_\_c == null){

parentApp.Date\_Solicitor\_Instructed\_\_c = Date.today();

}

if((parentApp.Workflow\_Status\_\_c == TTConstants.CONDITIONAL\_CREDIT\_UNDERWRITING || parentApp.Workflow\_Status\_\_c == TTConstants.CREDIT\_UNDERWRITING)

&& parentApp.Conditional\_Credit\_Underwriting\_Date\_\_c == null){

parentApp.Conditional\_Credit\_Underwriting\_Date\_\_c = Date.today();

}

if(parentApp.Workflow\_Status\_\_c == TTConstants.READY\_TO\_INSTRUCT && parentApp.Ready\_To\_Instruct\_Date\_\_c == null){

parentApp.Ready\_To\_Instruct\_Date\_\_c = Datetime.now();

}

if(String.isBlank(parentApp.Solicitor\_Firm\_\_c)){

if(parentApp.genesis\_\_CL\_Product\_\_r.Product\_Type\_\_c == TTConstants.COMM\_PROD\_TYPE || parentApp.genesis\_\_Asset\_Class\_\_c.contains(TTConstants.SMSF\_BRAND\_TYPE)) {

parentApp.Solicitor\_Firm\_\_c = MAKINSON;

} else if(parentApp.genesis\_\_CL\_Product\_\_r.Product\_Type\_\_c == TTConstants.RESI\_PROD\_TYPE && !parentApp.genesis\_\_Asset\_Class\_\_c.contains(TTConstants.SMSF\_BRAND\_TYPE)) {

parentApp.Solicitor\_Firm\_\_c = MSA;

}

}

appToUpdate.add(parentApp);

}

}

return appToUpdate;

}

/\*\*

\* @description This method updates and validates the ownership of Application records and associated tasks based on changes in owner IDs and queue memberships.

It checks if the new owners are valid for the respective queues and updates the Invalid\_Queue\_User\_\_c field for applications if necessary.

Additionally, it ensures that tasks associated with these applications have their ownership updated to match the new owner.

\* @author Swarna Roy | 31-07-2024

\* @param recordIds

\* @param ownerIds

\* @param allChildApps

\* @param allParentAppsMap

\* @param allParentAppsMapRead

\* @param parentOldOwnerMap

\* @return List<genesis\_\_Applications\_\_c>

\*\*/

public static List<genesis\_\_Applications\_\_c> updateAndValidateTaskOwner(Set <Id> recordIds,

Set <Id> ownerIds,

List <genesis\_\_Applications\_\_c> allChildApps,

Map <Id, genesis\_\_Applications\_\_c> allParentAppsMap,

Map <Id, genesis\_\_Applications\_\_c> allParentAppsMapRead,

Map<Id, String> parentOldOwnerMap) {

Set<Id> pkgIds = new Set<Id>();

Set<Id> packageIds = new Set<Id>();

Set<Id> applIds = new Set<Id>();

Set<Id> packageAppToUpdateId = new Set<Id>();

Id queueId;

Id workflowUserId;

String dept;

Boolean userOfQueue = false;

List<Task> taskToUpdate = new List<Task>();

List<User> workflowUser = new List<User>();

List <GroupMember> allGroupMembers = new List <GroupMember>();

List<genesis\_\_Applications\_\_c> packageToUpdate = new List<genesis\_\_Applications\_\_c>();

Map<Id, String> queueMap = new Map<Id, String>();

Map<String, Id> queueNameMap = new Map<String, Id>();

Map<Id, List<String>> parentGroupMemeberMap = new Map<Id, List<String>>();

genesis\_\_Applications\_\_c packageApp = new genesis\_\_Applications\_\_c();

Map<Id, genesis\_\_Application\_Department\_\_c> appDeptMap = new Map<Id, genesis\_\_Application\_Department\_\_c>();

Map<Id, Task> appTaskMap = new Map<Id, Task>();

Map<Id, genesis\_\_Departments\_Members\_\_c> depQueueMap = new Map<Id, genesis\_\_Departments\_Members\_\_c>();

List<Group> queues = [SELECT Id,

Name

FROM Group

WHERE Type = :TTConstants.QUEUE];

workflowUser = [SELECT Id

FROM User

WHERE Name = :TTConstants.JWF\_WORKFLOW

AND IsActive = true LIMIT 1];

if(!workflowUser.isEmpty()) {

workflowUserId = workflowUser[0].Id;

}

for(Group queue : queues){

queueMap.put(queue.Id, queue.Name);

queueNameMap.put(queue.Name, queue.Id);

}

if(!ownerIds.isEmpty()) {

allGroupMembers = [SELECT Id,

GroupId,

UserOrGroupId

FROM GroupMember

WHERE UserOrGroupId IN: ownerIds];

for(GroupMember each: allGroupMembers) {

if(parentGroupMemeberMap.get(each.UserOrGroupId) != null) {

parentGroupMemeberMap.get(each.UserOrGroupId).add(each.GroupId);

} else {

parentGroupMemeberMap.put(each.UserOrGroupId, new List<String>{each.GroupId});

}

}

}

for(genesis\_\_Applications\_\_c splitLoan: allChildApps) {

if(recordIds.contains(splitLoan.genesis\_\_Parent\_Application\_\_c)) {

String recordId = splitLoan.genesis\_\_Parent\_Application\_\_c;

if(queueMap.containsKey(parentOldOwnerMap.get(recordId))){

if(!queueMap.containsKey(allParentAppsMapRead.get(recordId).OwnerId)){

pkgIds.add(recordId);

}

} else {

if(queueMap.containsKey(allParentAppsMapRead.get(recordId).OwnerId)){

applIds.add(recordId);

} else {

if(allParentAppsMapRead.get(recordId).OwnerId != workflowUserId){

packageIds.add(recordId);

}

}

}

}

}

if(!packageIds.isEmpty() || !pkgIds.isEmpty()){

List<Task> appTasks = [SELECT Id,

OwnerId,

Status,

genesis\_\_Application\_\_c

FROM Task

WHERE genesis\_\_Application\_\_c IN :recordIds

AND Status = :TTConstants.OPEN];

for(Task appTaskL : appTasks){

appTaskMap.put(appTaskL.Id, appTaskL);

}

}

if(!packageIds.isEmpty() || !applIds.isEmpty()) {

List<genesis\_\_Application\_Department\_\_c> appDepts = [SELECT Id,

genesis\_\_Application\_\_c,

genesis\_\_Department\_\_c,

genesis\_\_Department\_\_r.Name,

genesis\_\_Application\_\_r.genesis\_\_Asset\_Class\_\_c,

genesis\_\_Application\_\_r.COMMSMSF\_Credit\_queue\_\_c,

genesis\_\_Application\_\_r.Brand\_\_c

FROM genesis\_\_Application\_Department\_\_c

WHERE genesis\_\_Application\_\_c IN :recordIds

AND genesis\_\_Status\_\_c = :TTConstants.ACTIVE];

for(genesis\_\_Application\_Department\_\_c appDept : appDepts){

appDeptMap.put(appDept.genesis\_\_Application\_\_c, appDept);

}

}

if(!packageIds.isEmpty()){

List<genesis\_\_Departments\_Members\_\_c> deptQueues = [SELECT Id,

genesis\_\_Department\_\_c,

genesis\_\_Queue\_Id\_\_c

FROM genesis\_\_Departments\_Members\_\_c];

for(genesis\_\_Departments\_Members\_\_c deptQueue : deptQueues){

depQueueMap.put(deptQueue.Id, deptQueue);

}

for(Id packageId : packageIds){

String ownerId = allParentAppsMapRead.get(packageId).OwnerId;

if(appDeptMap.containsKey(packageId)){

dept = appDeptMap.get(packageId).genesis\_\_Department\_\_c;

for(Id depQueueId : depQueueMap.keySet()){

genesis\_\_Departments\_Members\_\_c depQueue = depQueueMap.get(depQueueId);

if(dept == depQueue.genesis\_\_Department\_\_c){

queueId = depQueue.genesis\_\_Queue\_Id\_\_c;

List <String> validUser = parentGroupMemeberMap.get(ownerId);

if(validUser != null && validUser.contains(queueId)) {

userOfQueue = true;

}

}

}

if(userOfQueue == false){

if(!packageAppToUpdateId.contains(packageId)){

packageApp = allParentAppsMap.get(packageId);

packageApp.Invalid\_Queue\_User\_\_c = true;

packageToUpdate.add(packageApp);

packageAppToUpdateId.add(packageId);

}

}

}

//Task ownership change

for(Id taskId : appTaskMap.keySet()){

Task appTask = appTaskMap.get(taskId);

if(appTask.genesis\_\_Application\_\_c == packageId && appTask.OwnerId != ownerId){

appTask.OwnerId = ownerId;

taskToUpdate.add(appTask);

}

}

}

}

if(!pkgIds.isEmpty()){

for(Id pkgId : pkgIds){

String ownerId = allParentAppsMapRead.get(pkgId).OwnerId;

queueId = parentOldOwnerMap.get(pkgId);

List <String> validUser = parentGroupMemeberMap.get(ownerId);

if(validUser == null || !validUser.contains(queueId)){

if(!packageAppToUpdateId.contains(pkgId)){

packageApp = allParentAppsMap.get(pkgId);

packageApp.Invalid\_Queue\_User\_\_c = true;

packageToUpdate.add(packageApp);

packageAppToUpdateId.add(pkgId);

}

}

//Task ownership change

for(Id taskIdN : appTaskMap.keySet()){

Task appTaskN = appTaskMap.get(taskIdN);

if(appTaskN.genesis\_\_Application\_\_c == pkgId && appTaskN.OwnerId != ownerId){

appTaskN.OwnerId = ownerId;

taskToUpdate.add(appTaskN);

}

}

}

}

if(!applIds.isEmpty()){

for(Id applId : applIds){

String ownerId = allParentAppsMapRead.get(applId).OwnerId;

if(appDeptMap.containsKey(applId)){

genesis\_\_Application\_Department\_\_c appDeptData = appDeptMap.get(applId);

dept = appDeptData.genesis\_\_Department\_\_r.Name;

if(dept == TTConstants.CREDIT\_DEPT || dept == TTConstants.CREDIT\_UNDERWRITING\_DEPT){

if(appDeptData.genesis\_\_Application\_\_r.genesis\_\_Asset\_Class\_\_c == TTConstants.RESIDENTIAL\_ASSET\_CLASS){

if(queueNameMap.get(TTConstants.RESI\_CREDIT\_RATEMONEY\_QUEUE) != ownerId && appDeptData.genesis\_\_Application\_\_r.Brand\_\_c == TTConstants.RATEMONEY){

if(!packageAppToUpdateId.contains(applId)){

packageApp = allParentAppsMap.get(applId);

packageApp.ownerId = queueNameMap.get(TTConstants.RESI\_CREDIT\_RATEMONEY\_QUEUE);

packageToUpdate.add(packageApp);

packageAppToUpdateId.add(applId);

}

} else if(queueNameMap.get(TTConstants.RESI\_CREDIT\_RATEMONEY\_QUEUE) == ownerId && appDeptData.genesis\_\_Application\_\_r.Brand\_\_c != TTConstants.RATEMONEY){

if(!packageAppToUpdateId.contains(applId)){

packageApp = allParentAppsMap.get(applId);

packageApp.ownerId = queueNameMap.get(TTConstants.RESI\_CREDIT\_TT\_QUEUE);

packageToUpdate.add(packageApp);

packageAppToUpdateId.add(applId);

}

}

} else {

if(appDeptData.genesis\_\_Application\_\_r.COMMSMSF\_Credit\_queue\_\_c != null){

if(queueNameMap.get(appDeptData.genesis\_\_Application\_\_r.COMMSMSF\_Credit\_queue\_\_c) != ownerId){

if(!packageAppToUpdateId.contains(applId)){

packageApp = allParentAppsMap.get(applId);

packageApp.ownerId = queueNameMap.get(appDeptData.genesis\_\_Application\_\_r.COMMSMSF\_Credit\_queue\_\_c);

packageToUpdate.add(packageApp);

packageAppToUpdateId.add(applId);

}

}

}

}

}

}

}

}

if (!taskToUpdate.isEmpty()) {

update taskToUpdate;

}

return packageToUpdate;

}

/\*\*

\* @description This method updates the total fee amounts, including GST, for Applications records based on aggregated fee data from related Fee records, and subsequently adjusts the fee amounts for parent application records.

\* @author Swarna Roy | 31-07-2024

\* @param parentIds

\*\*/

public static void updateFeeAmountIncludingGst(Set <Id> parentIds) {

String appId;

String pkgId;

Decimal totalFee = 0.00;

Decimal sumFee = 0.00;

Decimal sumGstFee = 0.00;

Set<Id> packageIds = new Set<Id>();

Map<Id, Decimal> feeSumMap = new Map<Id, Decimal>();

Map<Id, Decimal> splitFeeSumMap = new Map<Id, Decimal>();

Map<Id, Decimal> splitFeeGstSumMap = new Map<Id, Decimal>();

List<genesis\_\_Applications\_\_c> appToUpdate = new List<genesis\_\_Applications\_\_c>();

List<genesis\_\_Applications\_\_c> packageAppToUpdate = new List<genesis\_\_Applications\_\_c>();

AggregateResult[] feeSummary = [SELECT sum(Fee\_Amount\_inc\_GST\_\_c) totalFees,

genesis\_\_Application\_\_c applicationId

FROM clcommon\_\_Fee\_\_c where genesis\_\_Application\_\_c IN :parentIds

GROUP BY genesis\_\_Application\_\_c];

for(AggregateResult appFees : feeSummary){

totalFee = (Decimal)appFees.get(TTConstants.TOTAL\_FEES);

appId = (String)appFees.get(TTConstants.APPLICATION\_ID);

feeSumMap.put(appId,totalFee);

}

for(genesis\_\_Applications\_\_c apps: [SELECT Id,

Total\_Fee\_Amount\_inc\_GST\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Id IN :parentIds]) {

if(feeSumMap.containsKey(apps.Id)) {

apps.Total\_Fee\_Amount\_inc\_GST\_\_c = feeSumMap.get(apps.Id);

appToUpdate.add(apps);

}

}

if(!appToUpdate.isEmpty()){

update appToUpdate;

}

AggregateResult[] splitFeeSums = [SELECT sum(genesis\_\_Total\_Fee\_Amount\_\_c) feeSum,

sum(Total\_Fee\_Amount\_inc\_GST\_\_c) feeGstSum,

genesis\_\_Parent\_Application\_\_c parentAppId

FROM genesis\_\_Applications\_\_c where Id IN :parentIds

AND genesis\_\_Parent\_Application\_\_c != null

GROUP BY genesis\_\_Parent\_Application\_\_c];

for(AggregateResult splitFeeSum : splitFeeSums){

sumFee = (Decimal)splitFeeSum.get(TTConstants.FEE\_SUM);

sumGstFee = (Decimal)splitFeeSum.get(TTConstants.FEE\_GST\_SUM);

pkgId = (String)splitFeeSum.get(TTConstants.PARENT\_APP\_ID);

splitFeeSumMap.put(pkgId,sumFee);

splitFeeGstSumMap.put(pkgId,sumGstFee);

packageIds.add(pkgId);

}

if(!packageIds.isEmpty()){

for(genesis\_\_Applications\_\_c packageApp: [SELECT Id,

genesis\_\_Total\_Fee\_Amount\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Id IN :packageIds]) {

if(splitFeeSumMap.containsKey(packageApp.Id)) {

packageApp.genesis\_\_Total\_Fee\_Amount\_\_c = splitFeeSumMap.get(packageApp.Id);

packageApp.Total\_Fee\_Amount\_inc\_GST\_\_c = splitFeeGstSumMap.get(packageApp.Id);

packageAppToUpdate.add(packageApp);

}

}

}

if(!packageAppToUpdate.isEmpty()){

update packageAppToUpdate;

}

}

/\*\*

\* @description This method updates the branding and rate index of Application records based on their asset class, aggregator details, and broker relationships, applying specific fee set changes as needed.

It also updates parent application records, adjusts fee sets for certain conditions, and recalculates fees for updated packages.

\* @author Swarna Roy | 31-07-2024

\* @param parentIds

\* @param allParentAppsRead

\* @param allParentAppsMap

\*\*/

public static void updateBranding(Set <Id> parentIds, List<genesis\_\_Applications\_\_c> allParentAppsRead, Map <Id, genesis\_\_Applications\_\_c> allParentAppsMap) {

Set<Id> pkgIds = new Set<Id>();

Boolean isRateMoney = false;

String type;

String aggName;

genesis\_\_Applications\_\_c parentApp;

Map<String,Id> feeSetIdMap = new Map<String,Id>();

List<Aggregator\_Brand\_Mapping\_\_mdt> brands;

List<clcommon\_\_Fee\_Set\_\_c> feeSetList = new List<clcommon\_\_Fee\_Set\_\_c>();

List<genesis\_\_Applications\_\_c> packageAppToBrand = new List<genesis\_\_Applications\_\_c>();

if(!parentIds.isEmpty()){

feeSetList = [SELECT Id, Name FROM clcommon\_\_Fee\_Set\_\_c];

for(clcommon\_\_Fee\_Set\_\_c feeSet:feeSetList) {

feeSetIdMap.put(feeSet.Name, feeSet.Id);

}

//Venky added broker branding filter as part of TB-2712

brands = [SELECT Id,

Aggregator\_\_c,

Brand\_\_c,

Residential\_Commercial\_\_c

FROM Aggregator\_Brand\_Mapping\_\_mdt

WHERE Broker\_Branding\_\_c =false];

for(genesis\_\_Applications\_\_c packageApp: allParentAppsRead) {

if(parentIds.contains(packageApp.Id)) {

parentApp = allParentAppsMap.get(packageApp.Id);

if(packageApp.genesis\_\_Asset\_Class\_\_c == TTConstants.RESIDENTIAL\_ASSET\_CLASS){

if(parentApp.Rate\_Index\_\_c != ConstructionLoanUtil.CONSTRUCTION\_LOAN\_REFERENCE\_RATE) {

parentApp.Rate\_Index\_\_c = TTConstants.RESIDENTIAL\_LOAN\_RATE;

}

parentApp.Brand\_Type\_\_c = TTConstants.RESIDENTIAL\_BRAND\_TYPE;

} else if(packageApp.genesis\_\_Asset\_Class\_\_c == TTConstants.COMMERCIAL\_ASSET\_CLASS){

if(parentApp.Rate\_Index\_\_c != TTConstants.MID\_TICKET\_COMMERCIAL\_LOAN\_RATE && parentApp.Rate\_Index\_\_c != TTConstants.BUSINESS\_LOAN\_REFERENCE\_RATE) {

parentApp.Rate\_Index\_\_c = TTConstants.COMMERCIAL\_LOAN\_RATE;

}

parentApp.Brand\_Type\_\_c = TTConstants.COMMERCIAL\_BRAND\_TYPE;

} else if(packageApp.genesis\_\_Asset\_Class\_\_c == TTConstants.SMSF\_COMMERCIAL\_ASSET\_CLASS){

if(parentApp.Rate\_Index\_\_c != TTConstants.MID\_TICKET\_SMSF\_COMMERCIAL\_LOAN\_RATE) {

parentApp.Rate\_Index\_\_c = TTConstants.COMMERCIAL\_LOAN\_RATE;

}

parentApp.Brand\_Type\_\_c = TTConstants.COMMERCIAL\_BRAND\_TYPE;

} else if(packageApp.genesis\_\_Asset\_Class\_\_c == TTConstants.SMSF\_RESIDENTIAL\_ASSET\_CLASS){

parentApp.Rate\_Index\_\_c = TTConstants.SMSF\_LOAN\_RATE;

parentApp.Brand\_Type\_\_c = TTConstants.SMSF\_BRAND\_TYPE;

}

if(packageApp.Aggregator\_\_c != null && packageApp.genesis\_\_Asset\_Class\_\_c != null) {

parentApp.Brand\_\_c = TTConstants.THINKTANK;

aggName = packageApp.Aggregator\_\_r.Name;

if(packageApp.genesis\_\_Asset\_Class\_\_c == TTConstants.RESIDENTIAL\_ASSET\_CLASS

|| (packageApp.genesis\_\_Asset\_Class\_\_c == TTConstants.SMSF\_RESIDENTIAL\_ASSET\_CLASS

&& aggName != TTConstants.AFG && aggName != TTConstants.AFG\_BUSINESS)) {

type = TTConstants.RESI\_PROD\_TYPE;

} else {

type = TTConstants.COMM\_PROD\_TYPE;

}

Boolean brokerAccIsDualAccredited = false;

if (packageApp.Broker\_Relationship\_\_r.Dual\_Accreditation\_\_c &&

packageApp.Broker\_Relationship\_\_r.Direct\_Accreditation\_\_c)

{

brokerAccIsDualAccredited = true;

}

//Venky updated brand logic to cater for broker company branding jira TB-1617

if (packageApp.Broker\_Relationship\_\_r.Brand\_Name\_\_c != null

&& (packageApp.genesis\_\_Asset\_Class\_\_c == TTConstants.COMMERCIAL\_ASSET\_CLASS

|| packageApp.genesis\_\_Asset\_Class\_\_c == TTConstants.SMSF\_COMMERCIAL\_ASSET\_CLASS

|| packageApp.genesis\_\_Asset\_Class\_\_c == TTConstants.SMSF\_RESIDENTIAL\_ASSET\_CLASS

|| packageApp.genesis\_\_Asset\_Class\_\_c == TTConstants.RESIDENTIAL\_ASSET\_CLASS

|| packageApp.Resi\_COMMSMSF\_Combination\_\_c)) {

parentApp.Brand\_\_c = packageApp.Broker\_Relationship\_\_r.Brand\_Name\_\_c;

}

// Mapping for Dual Accreditated Broker + Product combination. COMP-100 - Owen

else if (brokerAccIsDualAccredited && (packageApp.genesis\_\_Asset\_Class\_\_c == TTConstants.COMMERCIAL\_ASSET\_CLASS

|| packageApp.genesis\_\_Asset\_Class\_\_c == TTConstants.SMSF\_COMMERCIAL\_ASSET\_CLASS)) {

parentApp.Brand\_\_c = TTConstants.THINKTANK;

}

// Default brand mapping

else {

for (Aggregator\_Brand\_Mapping\_\_mdt brand : brands) {

if (aggName == brand.Aggregator\_\_c &&

type == brand.Residential\_Commercial\_\_c

) {

parentApp.Brand\_\_c = brand.Brand\_\_c;

if (aggName.startsWith(TTConstants.RATE\_MONEY)){

isRateMoney = true;

}

}

}

}

if(packageApp.genesis\_\_Asset\_Class\_\_c == TTConstants.RESIDENTIAL\_ASSET\_CLASS && packageApp.Fee\_Set\_\_c != feeSetIdMap.get(TTConstants.RATEMONEY)

&& packageApp.Fee\_Set\_\_r.Name != CalculateCustomFeeController.MID\_DOC\_CONSTRUCTION && packageApp.Fee\_Set\_\_r.Name != CalculateCustomFeeController.FULL\_DOC\_CONSTRUTION && isRateMoney == true){

if(packageApp.Fee\_Set\_\_r.Name == MAX\_RESIDENTIAL) {

parentApp.Fee\_Set\_\_c = feeSetIdMap.get(MAX\_RESIDENTIAL\_RATEMONEY);

} else {

parentApp.Fee\_Set\_\_c = feeSetIdMap.get(TTConstants.RATEMONEY);

}

pkgIds.add(packageApp.Id);

}

}

packageAppToBrand.add(parentApp);

}

}

}

if(!packageAppToBrand.isEmpty()) {

update packageAppToBrand;

}

if(!pkgIds.isEmpty()) {

delete [SELECT Id FROM clcommon\_\_Fee\_\_c WHERE genesis\_\_Application\_\_c IN :pkgIds];

for(Id pkgId : pkgIds){

CalculateCustomFeeController.calculateFeeAmount(pkgId);

}

}

}

/\*\*

\* @description This method calculates and updates the age of borrowers at the end of loan terms for parent Application records based on the maximum term of associated child applications and the borrower's current age.

\* @author Swarna Roy | 31-07-2024

\* @param parentIds

\* @param allChildAppsRead

\* @param allParentAppsRead

\* @param allParentAppsMap

\* @return List<genesis\_\_Applications\_\_c>

\*\*/

public static List<genesis\_\_Applications\_\_c> updateBorrowerAge(Set<Id> parentIds, List<genesis\_\_Applications\_\_c> allChildAppsRead, List<genesis\_\_Applications\_\_c> allParentAppsRead, Map <Id, genesis\_\_Applications\_\_c> allParentAppsMap){

List<genesis\_\_Applications\_\_c> packageAppToUpdate = new List<genesis\_\_Applications\_\_c>();

Map<Id, Decimal> appTermMap = new Map<Id, Decimal>();

Map<Id, Decimal> ageMap = new Map<Id, Decimal>();

Decimal borrowerAge = 0;

Decimal totTerm = 0;

genesis\_\_Applications\_\_c parentApp;

if(!parentIds.isEmpty()){

List<clcommon\_\_Party\_\_c> appBorrowerParties = [SELECT Id,

clcommon\_\_Contact\_\_r.Age\_\_c,

genesis\_\_Application\_\_c

FROM clcommon\_\_Party\_\_c

WHERE genesis\_\_Application\_\_c IN :parentIds

AND Party\_Type\_Name\_\_c = :TTConstants.BORROWER];

for(clcommon\_\_Party\_\_c appBorrowerParty : appBorrowerParties){

if(appBorrowerParty.clcommon\_\_Contact\_\_r.Age\_\_c != null){

ageMap.put(appBorrowerParty.genesis\_\_Application\_\_c, appBorrowerParty.clcommon\_\_Contact\_\_r.Age\_\_c);

} else {

ageMap.put(appBorrowerParty.genesis\_\_Application\_\_c, 0);

}

}

for(genesis\_\_Applications\_\_c app : allChildAppsRead){

Id parentAppId = app.genesis\_\_Parent\_Application\_\_c;

if(appTermMap.containsKey(parentAppId)) {

if(app.genesis\_\_Term\_\_c > appTermMap.get(parentAppId)) {

appTermMap.put(parentAppId, app.genesis\_\_Term\_\_c);

}

} else {

appTermMap.put(parentAppId, app.genesis\_\_Term\_\_c);

}

}

for (genesis\_\_Applications\_\_c packageApp: allParentAppsRead) {

if(parentIds.contains(packageApp.Id)) {

parentApp = allParentAppsMap.get(packageApp.Id);

if(appTermMap.containsKey(packageApp.Id)) {

totTerm = appTermMap.get(packageApp.Id);

}

if(ageMap.containsKey(packageApp.Id)) {

borrowerAge = ageMap.get(packageApp.Id);

}

if(borrowerAge > 0){

parentApp.Borrower\_Age\_at\_term\_end\_\_c = (borrowerAge + (totTerm/12)).setScale(0);

parentApp.Borrower\_Age\_\_c = borrowerAge;

if(packageApp.Borrower\_Age\_at\_term\_end\_\_c > 999){

parentApp.Borrower\_Age\_at\_term\_end\_\_c = 999;

}

if(packageApp.Borrower\_Age\_\_c > 999){

parentApp.Borrower\_Age\_\_c = 999;

}

packageAppToUpdate.add(parentApp);

}

}

}

}

return packageAppToUpdate;

}

/\*\*

\* @description This method updates the interest rate and related details for a set of Applications and their associated Rate Schedule Setup records based on margin changes, rate indexes, and pricing basis.

It creates new rate schedules for applications lacking them, updates existing schedules with the latest margin rates, and adjusts the brand type of parent applications based on their rate index.

\* @author Swarna Roy | 31-07-2024

\* @param appIds

\* @param pkgIds

\* @param marginChangedIds

\* @param allParentAppsRead

\* @param allChildAppsRead

\* @param allParentAppsMap

\*\*/

public static void updateInterestInformation(Set<Id> appIds, Set<Id> pkgIds, Set <Id> marginChangedIds,

List<genesis\_\_Applications\_\_c> allParentAppsRead,

List<genesis\_\_Applications\_\_c> allChildAppsRead,

Map <Id, genesis\_\_Applications\_\_c> allParentAppsMap) {

String rateIndex;

Id indexId;

String rateType;

Decimal margin;

genesis\_\_Applications\_\_c parentApp;

List<genesis\_\_Rate\_Schedule\_Setup\_\_c> rateSchdlToInsert = new List<genesis\_\_Rate\_Schedule\_Setup\_\_c>();

List<genesis\_\_Rate\_Schedule\_Setup\_\_c> rateScheduleSetupUpdate = new List<genesis\_\_Rate\_Schedule\_Setup\_\_c>();

List<genesis\_\_Applications\_\_c> pkgAppToUpdate = new List<genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Rate\_Schedule\_Setup\_\_c> schdlMap = new Map<Id, genesis\_\_Rate\_Schedule\_Setup\_\_c>();

Map<String, Id> indexMap = new Map<String, Id>();

Map<Id, Decimal> marginCalculationResults = new Map<Id, Decimal>();

Map<Id, String> rateTypeMap = new Map<Id, String>();

if(!pkgIds.isEmpty()){

for (genesis\_\_Applications\_\_c packageApp: allParentAppsRead) {

if(!pkgIds.contains(packageApp.Id)) {

continue;

}

parentApp = allParentAppsMap.get(packageApp.Id);

if(packageApp.Rate\_Index\_\_c == TTConstants.RESIDENTIAL\_LOAN\_RATE){

parentApp.Brand\_Type\_\_c = TTConstants.RESIDENTIAL\_BRAND\_TYPE;

} else if(packageApp.Rate\_Index\_\_c == TTConstants.COMMERCIAL\_LOAN\_RATE){

parentApp.Brand\_Type\_\_c = TTConstants.COMMERCIAL\_BRAND\_TYPE;

} else if(packageApp.Rate\_Index\_\_c == TTConstants.SMSF\_LOAN\_RATE){

parentApp.Brand\_Type\_\_c = TTConstants.SMSF\_BRAND\_TYPE;

}

pkgAppToUpdate.add(parentApp);

}

}

if(!pkgAppToUpdate.isEmpty()) {

update pkgAppToUpdate;

}

if(!appIds.isEmpty()){

List<genesis\_\_Rate\_Schedule\_Setup\_\_c> rateSchdls = [SELECT Id,

genesis\_\_Margin\_Rate\_\_c,

genesis\_\_Spread\_\_c,

genesis\_\_Application\_\_c,

genesis\_\_Application\_\_r.Parent\_Rate\_Index\_\_c,

genesis\_\_Application\_\_r.Thinktank\_Margin\_\_c

FROM genesis\_\_Rate\_Schedule\_Setup\_\_c

WHERE genesis\_\_Application\_\_c IN :appIds];

for(genesis\_\_Rate\_Schedule\_Setup\_\_c rateSchdl : rateSchdls){

schdlMap.put(rateSchdl.genesis\_\_Application\_\_c,rateSchdl);

}

List<clcommon\_\_Floating\_Rate\_Index\_\_c> rateIndexes = [SELECT Id,

Name

FROM clcommon\_\_Floating\_Rate\_Index\_\_c

WHERE clcommon\_\_Active\_\_c = true];

for(clcommon\_\_Floating\_Rate\_Index\_\_c rate : rateIndexes){

indexMap.put(rate.Name, rate.Id);

}

for(genesis\_\_Applications\_\_c splitApp : allChildAppsRead){

if(!appIds.contains(splitApp.Id)) {

continue;

}

if(splitApp.Pricing\_Basis\_\_c != null){

if(splitApp.Pricing\_Basis\_\_c == TTConstants.VARIABLE){

rateType = TTConstants.FLOATING;

} else{

rateType = splitApp.Pricing\_Basis\_\_c;

}

} else{

rateType = TTConstants.FIXED;

}

rateTypeMap.put(splitApp.Id, rateType);

if(!schdlMap.containsKey(splitApp.Id) && splitApp.Parent\_Rate\_Index\_\_c != null && indexMap.containsKey(splitApp.Parent\_Rate\_Index\_\_c)) {

rateIndex = splitApp.Parent\_Rate\_Index\_\_c;

indexId = indexMap.get(rateIndex);

genesis\_\_Rate\_Schedule\_Setup\_\_c rateSchdl = new genesis\_\_Rate\_Schedule\_Setup\_\_c();

rateSchdl.genesis\_\_Index\_\_c = indexId;

rateSchdl.genesis\_\_Application\_\_c = splitApp.Id;

rateSchdl.genesis\_\_Rate\_Type\_\_c = rateType;

if(splitApp.genesis\_\_Parent\_Application\_\_r.Broker\_Trail\_\_c != null) {

rateSchdl.genesis\_\_Spread\_\_c = splitApp.genesis\_\_Parent\_Application\_\_r.Broker\_Trail\_\_c;

}

rateSchdl.genesis\_\_Start\_Date\_\_c = Date.today();

rateSchdlToInsert.add(rateSchdl);

}

}

}

if(!rateSchdlToInsert.isEmpty()){

INSERT rateSchdlToInsert;

}

if(!appIds.isEmpty()) {

marginCalculationResults = CLXUtil.marginCalculation(allChildAppsRead, appIds);

List<genesis\_\_Rate\_Schedule\_Setup\_\_c> rateSchUpds = [SELECT Id,

genesis\_\_Margin\_Rate\_\_c,

genesis\_\_Base\_Rate\_\_c,

genesis\_\_Spread\_\_c,

Schedule\_Action\_Method\_\_c,

genesis\_\_Application\_\_c,

genesis\_\_Application\_\_r.Parent\_Rate\_Index\_\_c,

genesis\_\_Application\_\_r.genesis\_\_CL\_Product\_Name\_\_c,

genesis\_\_Application\_\_r.Thinktank\_Margin\_\_c

FROM genesis\_\_Rate\_Schedule\_Setup\_\_c

WHERE genesis\_\_Application\_\_c IN :appIds];

for(genesis\_\_Rate\_Schedule\_Setup\_\_c rate : rateSchUpds){

if(!appIds.contains(rate.genesis\_\_Application\_\_c)) {

continue;

}

rateIndex = rate.genesis\_\_Application\_\_r.Parent\_Rate\_Index\_\_c;

rate.genesis\_\_Rate\_Type\_\_c = rateTypeMap.get(rate.genesis\_\_Application\_\_c);

if(marginChangedIds.contains(rate.genesis\_\_Application\_\_c)) {

rate.genesis\_\_Margin\_Rate\_\_c = rate.genesis\_\_Application\_\_r.Thinktank\_Margin\_\_c;

} else {

margin = marginCalculationResults.get(rate.genesis\_\_Application\_\_c);

if(margin != 1000){

rate.genesis\_\_Margin\_Rate\_\_c = margin;

}

}

if(rateIndex != null && rate.genesis\_\_Application\_\_r.genesis\_\_CL\_Product\_Name\_\_c != ConstructionLoanUtil.CONSTRUCTION\_LOAN && indexMap.containsKey(rateIndex)){

indexId = indexMap.get(rateIndex);

rate.genesis\_\_Index\_\_c = indexId;

}

if(rate.Schedule\_Action\_Method\_\_c != POPULATE\_AND\_UPDATE\_RATE\_SCHEDULE) {

rate.Schedule\_Action\_Method\_\_c = POPULATE\_AND\_UPDATE\_RATE\_SCHEDULE;

rateScheduleSetupUpdate.add(rate);

}

}

if(!rateScheduleSetupUpdate.isEmpty()) {

update rateScheduleSetupUpdate;

}

}

}

/\*\*

\* @description This method computes LTV ratio for a set of parent Applications based on their associated collateral values.

It updates the CLTV field and Total Collateral Amount for each application while handling various conditions related to application status, collateral valuation, and asset class.

\* @author Swarna Roy | 31-07-2024

\* @param parentIds

\* @param allParentAppsRead

\* @param allParentAppsMap

\* @return List<genesis\_\_Applications\_\_c>

\*\*/

public static List<genesis\_\_Applications\_\_c> calculatePackageLtv(Set<Id> parentIds, List<genesis\_\_Applications\_\_c> allParentAppsRead, Map <Id, genesis\_\_Applications\_\_c> allParentAppsMap){

Decimal totalSum = 0.00;

Decimal minValue;

String colCmprId;

Decimal pkgAmt;

Boolean isAppLocked = false;

Boolean valuationTypeEmpty = false;

genesis\_\_Applications\_\_c packageApp;

Map<Id, Boolean> appLockMap = new Map<Id, Boolean>();

List<genesis\_\_Applications\_\_c> cltvUpdate = new List<genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Application\_Collateral\_\_c> collateralMap = new Map<Id, genesis\_\_Application\_Collateral\_\_c>();

if(!parentIds.isEmpty()){

appLockMap = Approval.isLocked(new List<Id>(parentIds));

List<genesis\_\_Application\_Collateral\_\_c> allApplCol = [SELECT Id,

Name,

genesis\_\_Collateral\_\_c,

genesis\_\_Application\_\_c,

genesis\_\_Collateral\_\_r.Current\_Value\_\_c,

genesis\_\_Collateral\_\_r.clcommon\_\_Estimated\_Value\_\_c,

genesis\_\_Collateral\_\_r.contract\_Amount\_\_c,

genesis\_\_Collateral\_\_r.Valuation\_Type\_\_c,

genesis\_\_Application\_\_r.genesis\_\_CL\_Product\_\_r.Product\_Type\_\_c

FROM genesis\_\_Application\_Collateral\_\_c

WHERE genesis\_\_Application\_\_c IN :parentIds

AND is\_Security\_\_c = true];

for(genesis\_\_Application\_Collateral\_\_c appCol : allApplCol){

collateralMap.put(appCol.Id, appCol);

}

for(genesis\_\_Applications\_\_c updateApp : allParentAppsRead){

if(parentIds.contains(updateApp.Id)) {

packageApp = allParentAppsMap.get(updateApp.Id);

if(appLockMap.size() > 0){

isAppLocked = appLockMap.get(updateApp.Id);

}

if(isAppLocked == false || updateApp.genesis\_\_Status\_\_c == TTConstants.CHANGE\_MEMO\_APPROVAL){

totalSum = 0.00;

minValue = 0.00;

colCmprId = updateApp.Id;

if(updateApp.Is\_Further\_Advance\_App\_\_c == true){

pkgAmt = updateApp.genesis\_\_Requested\_Loan\_Amount\_\_c;

} else {

pkgAmt = updateApp.genesis\_\_Total\_Facility\_Amount\_\_c;

}

if(collateralMap.size() > 0){

for(Id appColId : collateralMap.keySet()){

genesis\_\_Application\_Collateral\_\_c appColt = collateralMap.get(appColId);

if(appColt.genesis\_\_Application\_\_c == colCmprId){

if(appColt.genesis\_\_Collateral\_\_r.Valuation\_Type\_\_c == null) {

valuationTypeEmpty = true;

}

if(appColt.genesis\_\_Collateral\_\_r.Current\_Value\_\_c != null){

if(appColt.genesis\_\_Collateral\_\_r.contract\_Amount\_\_c == null){

minValue = appColt.genesis\_\_Collateral\_\_r.Current\_Value\_\_c;

} else {

if(appColt.genesis\_\_Collateral\_\_r.Current\_Value\_\_c < appColt.genesis\_\_Collateral\_\_r.contract\_Amount\_\_c){

minValue = appColt.genesis\_\_Collateral\_\_r.Current\_Value\_\_c;

} else {

minValue = appColt.genesis\_\_Collateral\_\_r.contract\_Amount\_\_c;

}

}

} else {

if(appColt.genesis\_\_Collateral\_\_r.contract\_Amount\_\_c == null){

minValue = appColt.genesis\_\_Collateral\_\_r.clcommon\_\_Estimated\_Value\_\_c;

} else {

minValue = appColt.genesis\_\_Collateral\_\_r.contract\_Amount\_\_c;

}

}

totalSum = totalSum + minValue;

}

}

if(pkgAmt != null && totalSum > 0.00 && (valuationTypeEmpty || updateApp.Rate\_Index\_\_c == ConstructionLoanUtil.CONSTRUCTION\_LOAN\_REFERENCE\_RATE

|| (updateApp.genesis\_\_Asset\_Class\_\_c != TTConstants.RESIDENTIAL\_ASSET\_CLASS && updateApp.genesis\_\_Asset\_Class\_\_c != TTConstants.SMSF\_RESIDENTIAL\_ASSET\_CLASS))){

packageApp.genesis\_\_CLTV\_\_c =(pkgAmt/totalSum).setScale(4, System.RoundingMode.UP);

packageApp.Total\_Collateral\_Amount\_\_c = totalSum;

cltvUpdate.add(packageApp);

}

} else {

if((valuationTypeEmpty || updateApp.Rate\_Index\_\_c == ConstructionLoanUtil.CONSTRUCTION\_LOAN\_REFERENCE\_RATE

|| (updateApp.genesis\_\_Asset\_Class\_\_c != TTConstants.RESIDENTIAL\_ASSET\_CLASS && updateApp.genesis\_\_Asset\_Class\_\_c != TTConstants.SMSF\_RESIDENTIAL\_ASSET\_CLASS))) {

packageApp.genesis\_\_CLTV\_\_c = 0.00;

packageApp.Total\_Collateral\_Amount\_\_c = 0.00;

cltvUpdate.add(packageApp);

}

}

if(totalSum > 0 && pkgAmt > 0){

if(updateApp.genesis\_\_Asset\_Class\_\_c == TTConstants.RESIDENTIAL\_ASSET\_CLASS){

CalculateCustomFeeController.calculateFeeAmount(colCmprId);

}

}

if(updateApp.Is\_Simpology\_Application\_\_c == false){

RelationshipExposureCalculation.calculateExposuresParty(colCmprId);

} else {

if(updateApp.genesis\_\_Status\_\_c != TTConstants.NEW\_PARTIAL && updateApp.genesis\_\_Status\_\_c != TTConstants.NEW\_ENTERED){

RelationshipExposureCalculation.calculateExposuresParty(colCmprId);

}

}

}

}

}

}

return cltvUpdate;

}

/\*\*

\* @description This method updates the Broker Trail field for child applications based on the broker trail value from their parent application, recalculates margins, and updates associated rate schedule records with the new broker trail and margin rates.

\* @author Swarna Roy | 31-07-2024

\* @param packageIds

\* @param pkgBrokerTrailMap

\* @param allChildAppsRead

\* @param allChildAppsMap

\*\*/

public static void updateBrokerTrail(Set<Id> packageIds, Map<Id, Decimal> pkgBrokerTrailMap, List<genesis\_\_Applications\_\_c> allChildAppsRead, Map <Id, genesis\_\_Applications\_\_c> allChildAppsMap){

Set<Id> appIds = new Set<Id>();

Decimal brokerTrail = 0;

Decimal margin;

genesis\_\_Applications\_\_c childApp;

Map<Id, Decimal> marginCalculationResults = new Map<Id, Decimal>();

List<genesis\_\_Applications\_\_c> splitAppToUpdate = new List<genesis\_\_Applications\_\_c>();

List<genesis\_\_Rate\_Schedule\_Setup\_\_c> splitRateSchdlToUpdate = new List<genesis\_\_Rate\_Schedule\_Setup\_\_c>();

Map<Id,Decimal> appBrokerTrailMap = new Map<Id,Decimal>();

for(genesis\_\_Applications\_\_c splitApp: allChildAppsRead){

if(packageIds.contains(splitApp.genesis\_\_Parent\_Application\_\_c)) {

brokerTrail = pkgBrokerTrailMap.get(splitApp.genesis\_\_Parent\_Application\_\_c);

childApp = allChildAppsMap.get(splitApp.Id);

childApp.Broker\_Trail\_\_c = brokerTrail;

splitAppToUpdate.add(childApp);

appIds.add(splitApp.Id);

appBrokerTrailMap.put(splitApp.Id, brokerTrail);

}

}

if(!splitAppToUpdate.isEmpty()){

update splitAppToUpdate;

}

if(!appIds.isEmpty()){

marginCalculationResults = CLXUtil.marginCalculation(allChildAppsRead, appIds);

for(genesis\_\_Rate\_Schedule\_Setup\_\_c splitRateSchdl: [SELECT Id,

genesis\_\_Spread\_\_c,

genesis\_\_Index\_\_c,

genesis\_\_Margin\_Rate\_\_c,

genesis\_\_Base\_Rate\_\_c,

Schedule\_Action\_Method\_\_c,

genesis\_\_Application\_\_c

FROM genesis\_\_Rate\_Schedule\_Setup\_\_c

WHERE genesis\_\_Application\_\_c IN :appIds]){

splitRateSchdl.genesis\_\_Spread\_\_c = appBrokerTrailMap.get(splitRateSchdl.genesis\_\_Application\_\_c);

margin = marginCalculationResults.get(splitRateSchdl.genesis\_\_Application\_\_c);

if(margin != 1000){

splitRateSchdl.genesis\_\_Margin\_Rate\_\_c = margin;

}

if(splitRateSchdl.Schedule\_Action\_Method\_\_c != POPULATE\_AND\_UPDATE\_RATE\_SCHEDULE) {

splitRateSchdl.Schedule\_Action\_Method\_\_c = POPULATE\_AND\_UPDATE\_RATE\_SCHEDULE;

splitRateSchdlToUpdate.add(splitRateSchdl);

}

}

}

if(!splitRateSchdlToUpdate.isEmpty()){

update splitRateSchdlToUpdate;

}

}

/\*\*

\* @description This method updates the Consolidated Purpose field for parent applications based on the purposes of their associated child applications.

\* @author Swarna Roy | 31-07-2024

\* @param parentIds

\* @param allChildAppsRead

\* @param allParentAppsMap

\* @return List<genesis\_\_Applications\_\_c>

\*\*/

public static List<genesis\_\_Applications\_\_c> updateConsolidatedPurpose(Set <Id> parentIds, List<genesis\_\_Applications\_\_c> allChildAppsRead, Map <Id, genesis\_\_Applications\_\_c> allParentAppsMap){

String appId;

String purpose;

String consolPurpose;

String pkgPurpose;

Map<Id, String> appPurposeMap = new Map<Id, String>();

List<genesis\_\_Applications\_\_c> packageAppToUpdate = new List<genesis\_\_Applications\_\_c>();

if(!parentIds.isEmpty()){

for(Id parentId : parentIds){

consolPurpose = null;

for(genesis\_\_Applications\_\_c appPurpose : allChildAppsRead){

purpose = appPurpose.genesis\_\_CL\_Purpose\_\_r.Name;

appId = appPurpose.genesis\_\_Parent\_Application\_\_c;

if(parentId == appId && purpose != null){

if(consolPurpose != null){

consolPurpose = consolPurpose + ',' + purpose;

} else {

consolPurpose = purpose;

}

}

}

appPurposeMap.put(parentId, consolPurpose);

}

}

for(genesis\_\_Applications\_\_c packageApp: allParentAppsMap.values()) {

if(appPurposeMap.containsKey(packageApp.Id)) {

pkgPurpose = appPurposeMap.get(packageApp.Id);

packageApp.Consolidated\_Purpose\_\_c = pkgPurpose;

packageAppToUpdate.add(packageApp);

}

}

return packageAppToUpdate;

}

/\*\*

\* @description This method updates loan-related date fields for a list of child applications based on provided date maps and a settlement date.

\* @author Swarna Roy | 31-07-2024

\* @param appIds

\* @param settlementDate

\* @param appDateMap

\* @param appDisbDateMap

\* @param allChildAppsUpdate

\* @return List<genesis\_\_Applications\_\_c>

\*\*/

public static List<genesis\_\_Applications\_\_c> updateLoanDates(Set<Id> appIds, Date settlementDate, Map<Id, Date> appDateMap, Map<Id, Date> appDisbDateMap, List<genesis\_\_Applications\_\_c> allChildAppsUpdate){

List<genesis\_\_Applications\_\_c> splitAppToUpdate = new List<genesis\_\_Applications\_\_c>();

if(!appIds.isEmpty()){

for(genesis\_\_Applications\_\_c splitApp: allChildAppsUpdate){

if(appIds.contains(splitApp.Id)) {

splitApp.genesis\_\_Expected\_First\_Payment\_Date\_\_c = appDateMap.get(splitApp.Id).addMonths(1);

splitApp.genesis\_\_Expected\_Close\_Date\_\_c = appDisbDateMap.get(splitApp.Id);

splitApp.genesis\_\_Expected\_Start\_Date\_\_c= appDisbDateMap.get(splitApp.Id);

splitApp.genesis\_\_Due\_Day\_\_c = appDateMap.get(splitApp.Id).day();

splitAppToUpdate.add(splitApp);

}

}

}

return splitAppToUpdate;

}

/\*\*

\* @description This method quries the Borrower Name field with the Account Name field for applications associated with the specified account IDs, updating the records if they differ.

\* @author Swarna Roy | 31-07-2024

\* @param accIds

\*\*/

public static void updateBorrowerName(Set <Id> accIds){

List<genesis\_\_Applications\_\_c> appToUpdate = new List<genesis\_\_Applications\_\_c>();

if(!accIds.isEmpty()){

for(genesis\_\_Applications\_\_c allApp : [SELECT Id, Account\_Name\_\_c, Borrower\_Name\_\_c FROM genesis\_\_Applications\_\_c WHERE genesis\_\_Account\_\_c IN :accIds]){

if(allApp.Borrower\_Name\_\_c != allApp.Account\_Name\_\_c){

allApp.Borrower\_Name\_\_c = allApp.Account\_Name\_\_c;

appToUpdate.add(allApp);

}

}

}

if(!appToUpdate.isEmpty()){

update appToUpdate;

}

}

/\*\*

\* @description This method queries key fields from parent applications to child applications, updates the status of related rate schedules, and adjusts the schedule generation flag based on specified conditions.

\* @author Swarna Roy | 31-07-2024

\* @param pkgIds

\* @param updateExpectedCloseDate

\* @param allChildAppsUpdate

\* @param allParentAppsMapRead

\*\*/

public static void updateKeyFields(Set<Id> pkgIds, Set<Id> updateExpectedCloseDate, List<genesis\_\_Applications\_\_c> allChildAppsUpdate, Map<Id, genesis\_\_Applications\_\_c> allParentAppsMapRead){

Set<Id> splitIds = new Set<Id>();

List<genesis\_\_Applications\_\_c> appToUpdate = new List<genesis\_\_Applications\_\_c>();

List<genesis\_\_Rate\_Schedule\_Setup\_\_c> rateSchedulesToUpdate = new List<genesis\_\_Rate\_Schedule\_Setup\_\_c>();

if(!pkgIds.isEmpty()){

for(genesis\_\_Applications\_\_c allSplit : allChildAppsUpdate){

if(pkgIds.contains(allSplit.genesis\_\_Parent\_Application\_\_c)) {

genesis\_\_Applications\_\_c pkgData = allParentAppsMapRead.get(allSplit.genesis\_\_Parent\_Application\_\_c);

allSplit.Penciled\_In\_Status\_\_c = pkgData.Penciled\_In\_Status\_\_c;

allSplit.genesis\_\_Expected\_Close\_Date\_\_c = pkgData.genesis\_\_Expected\_Close\_Date\_\_c;

allSplit.genesis\_\_Disbursement\_Date\_\_c = pkgData.genesis\_\_Disbursement\_Date\_\_c;

allSplit.Funding\_Date\_\_c = pkgData.Funding\_Date\_\_c;

allSplit.NCCP\_\_c = pkgData.NCCP\_\_c;

appToUpdate.add(allSplit);

if(updateExpectedCloseDate.contains(allSplit.genesis\_\_Parent\_Application\_\_c)) {

if(allSplit.genesis\_\_Parent\_Application\_\_r.Is\_Simpology\_Application\_\_c == false){

splitIds.add(allSplit.Id);

} else {

if(allSplit.genesis\_\_Parent\_Application\_\_r.genesis\_\_Status\_\_c != TTConstants.NEW\_PARTIAL && allSplit.genesis\_\_Parent\_Application\_\_r.genesis\_\_Status\_\_c != TTConstants.NEW\_ENTERED){

splitIds.add(allSplit.Id);

}

}

}

}

}

}

if(!appToUpdate.isEmpty()){

UPDATE appToUpdate;

}

List<genesis\_\_Rate\_Schedule\_Setup\_\_c> rateSchedules = [SELECT Id,

genesis\_\_Margin\_Rate\_\_c,

genesis\_\_Base\_Rate\_\_c,

genesis\_\_Spread\_\_c,

Schedule\_Action\_Method\_\_c,

genesis\_\_Application\_\_c

FROM genesis\_\_Rate\_Schedule\_Setup\_\_c

WHERE genesis\_\_Application\_\_c IN :splitIds];

// Generate Schedule

for(genesis\_\_Rate\_Schedule\_Setup\_\_c rate : rateSchedules){

if(rate.Schedule\_Action\_Method\_\_c == null) {

rate.Schedule\_Action\_Method\_\_c = GENERATE\_SCHEDULE;

rateSchedulesToUpdate.add(rate);

}

}

if(!rateSchedulesToUpdate.isEmpty()) {

update rateSchedulesToUpdate;

}

}

/\*\*

\* @description This method computes and updates the margin rates for a set of child applications based on interest rate indexes and application details, ensuring that rates are recalculated and applied correctly according to property type, document type, and payment type.

It also checks for locked records or pending change memos before performing calculations and updates the rate schedule setup records accordingly.

\* @author Swarna Roy | 31-07-2024

\* @param parentIds

\* @param allChildAppsRead

\* @param allParentAppsRead

\*\*/

public static void calculateMargin(Set<Id> parentIds, List<genesis\_\_Applications\_\_c> allChildAppsRead, List<genesis\_\_Applications\_\_c> allParentAppsRead){

List<Interest\_Rate\_Index\_\_mdt> rateindexlist;

List<Interest\_Rate\_Index\_\_mdt> rateMoneyIndexes;

List<Interest\_Rate\_Index\_\_mdt> nonRateMoneyIndexes;

Set<Id> childIds = new Set<Id>();

Set<Id> isAppLocked = new Set<Id>();

Map<Id, String> parentDataMap = new Map<Id, String>();

Map<Id, Decimal> marginResults = new Map<Id, Decimal>();

Map<String, String> chgMemoMap = new Map<String, String>();

Map<Id, String> propertyTypeMap = new Map<Id, String>();

Map<Id, genesis\_\_Applications\_\_c> childAppMap = new Map<Id, genesis\_\_Applications\_\_c>();

List<genesis\_\_Rate\_Schedule\_Setup\_\_c> rateSchedules = new List<genesis\_\_Rate\_Schedule\_Setup\_\_c>();

List<genesis\_\_Rate\_Schedule\_Setup\_\_c> rateSchToUpdate = new List<genesis\_\_Rate\_Schedule\_Setup\_\_c>();

Map<Id, genesis\_\_Rate\_Schedule\_Setup\_\_c> rateSchdlMap = new Map<Id, genesis\_\_Rate\_Schedule\_Setup\_\_c>();

if(!parentIds.isEmpty()){

for(genesis\_\_Applications\_\_c parentData : allParentAppsRead){

if(parentIds.contains(parentData.Id)) {

parentDataMap.put(parentData.Id, parentData.genesis\_\_CL\_Product\_\_r.Document\_Type\_\_c);

}

}

List<genesis\_\_Change\_Memo\_\_c> chngMemosList = [SELECT Id,

genesis\_\_Application\_\_c

FROM genesis\_\_Change\_Memo\_\_c

WHERE genesis\_\_Application\_\_c IN :parentIds

AND genesis\_\_Status\_\_c = :TTConstants.PENDING

AND Thinktank\_Margin\_\_c != null];

for(genesis\_\_Change\_Memo\_\_c chngMemos : chngMemosList){

chgMemoMap.put(chngMemos.genesis\_\_Application\_\_c, chngMemos.Id);

}

List<genesis\_\_Application\_Collateral\_\_c> collateralList = [ SELECT Id,

isPrimary\_\_c,

genesis\_\_Application\_\_c,

genesis\_\_Collateral\_\_r.clcommon\_\_Property\_Type\_\_c

FROM genesis\_\_Application\_Collateral\_\_c

WHERE genesis\_\_Application\_\_c IN :parentIds

AND isPrimary\_\_c = true];

for(genesis\_\_Application\_Collateral\_\_c collaterals : collateralList){

propertyTypeMap.put(collaterals.genesis\_\_Application\_\_c, collaterals.genesis\_\_Collateral\_\_r.clcommon\_\_Property\_Type\_\_c);

}

for(genesis\_\_Applications\_\_c split : allChildAppsRead){

if(parentIds.contains(split.genesis\_\_Parent\_Application\_\_c) && split.genesis\_\_Asset\_Class\_\_c == TTConstants.RESIDENTIAL\_ASSET\_CLASS) {

childAppMap.put(split.Id, split);

childIds.add(split.Id);

}

}

if(!childIds.isEmpty()){

rateSchedules = [SELECT Id,

genesis\_\_Application\_\_c,

genesis\_\_Base\_Rate\_\_c,

genesis\_\_Spread\_\_c,

genesis\_\_Margin\_Rate\_\_c,

Schedule\_Action\_Method\_\_c,

genesis\_\_Application\_\_r.genesis\_\_Status\_\_c,

genesis\_\_Application\_\_r.genesis\_\_Parent\_Application\_\_r.genesis\_\_Status\_\_c,

genesis\_\_Application\_\_r.genesis\_\_Parent\_Application\_\_r.Is\_Simpology\_Application\_\_c

FROM genesis\_\_Rate\_Schedule\_Setup\_\_c

WHERE genesis\_\_Application\_\_c IN :childIds];

for(genesis\_\_Rate\_Schedule\_Setup\_\_c rate : rateSchedules){

rateSchdlMap.put(rate.genesis\_\_Application\_\_c,rate);

}

rateMoneyIndexes = [SELECT Id,

Type\_\_c,

Lower\_LVR\_Range\_\_c,

Higher\_LVR\_Range\_\_c,

Interest\_Rate\_Full\_Doc\_\_c,

Interest\_Rate\_Mid\_Doc\_\_c,

Investment\_Full\_Doc\_\_c,

Investment\_Mid\_Doc\_\_c,

Interest\_Only\_Full\_Doc\_\_c,

Interest\_Only\_Mid\_Doc\_\_c

FROM Interest\_Rate\_Index\_\_mdt

WHERE Type\_\_c = :TTConstants.RATEMONEY];

nonRateMoneyIndexes = [SELECT Id,

Type\_\_c,

Lower\_LVR\_Range\_\_c,

Higher\_LVR\_Range\_\_c,

Interest\_Rate\_Full\_Doc\_\_c,

Interest\_Rate\_Mid\_Doc\_\_c,

Investment\_Full\_Doc\_\_c,

Investment\_Mid\_Doc\_\_c,

Interest\_Only\_Full\_Doc\_\_c,

Interest\_Only\_Mid\_Doc\_\_c

FROM Interest\_Rate\_Index\_\_mdt

WHERE Type\_\_c != :TTConstants.RATEMONEY];

}

List<clcommon\_\_Locked\_Record\_Configuration\_Snapshot\_\_c> lockedRecordSnapshots = [SELECT clcommon\_\_Primary\_Object\_ID\_\_c

FROM clcommon\_\_Locked\_Record\_Configuration\_Snapshot\_\_c

WHERE clcommon\_\_Primary\_Object\_ID\_\_c IN :parentIds];

for(clcommon\_\_Locked\_Record\_Configuration\_Snapshot\_\_c isLocked: lockedRecordSnapshots) {

isAppLocked.add(isLocked.clcommon\_\_Primary\_Object\_ID\_\_c);

}

}

for(Id indSplitId : childIds){

genesis\_\_Applications\_\_c child = childAppMap.get(indSplitId);

String paymentType = child.Payment\_Type\_\_c;

Decimal interestRate = 0;

Decimal ltvp = child.genesis\_\_Parent\_Application\_\_r.LTVP\_\_c;

String documentType = parentDataMap.get(child.genesis\_\_Parent\_Application\_\_c);

String propertyType = propertyTypeMap.get(child.genesis\_\_Parent\_Application\_\_c);

genesis\_\_Rate\_Schedule\_Setup\_\_c childRate = rateSchdlMap.get(indSplitId);

// Check to see if App is locked, if so break out of recalculation

if(isAppLocked.contains(child.genesis\_\_Parent\_Application\_\_c) || chgMemoMap.containsKey(child.genesis\_\_Parent\_Application\_\_c)

|| propertyType == null || documentType == null || childRate == null || ltvp == null) {

continue;

}

if(child.genesis\_\_Parent\_Application\_\_r.Brand\_\_c == TTConstants.RATEMONEY){

rateindexlist = rateMoneyIndexes;

} else {

rateindexlist = nonRateMoneyIndexes;

}

for(Interest\_Rate\_Index\_\_mdt rateindex : rateindexlist){

if(ltvp > rateindex.Lower\_LVR\_Range\_\_c && ltvp <= rateindex.Higher\_LVR\_Range\_\_c){

if(propertyType == TTConstants.OWNER\_OCCUPIED) {

if(documentType == TTConstants.FULL\_DOC){

interestRate = rateindex.Interest\_Rate\_Full\_Doc\_\_c;

} else if(documentType == TTConstants.MID\_DOC){

interestRate = rateindex.Interest\_Rate\_Mid\_Doc\_\_c;

}

} else if(propertyType == TTConstants.INVESTMENT) {

if(paymentType == TTConstants.PRINCIPAL\_INTEREST){

if(documentType == TTConstants.FULL\_DOC ){

interestRate = rateindex.Interest\_Rate\_Full\_Doc\_\_c + rateindex.Investment\_Full\_Doc\_\_c;

} else if(documentType == TTConstants.MID\_DOC){

interestRate = rateindex.Interest\_Rate\_Mid\_Doc\_\_c + rateindex.Investment\_Mid\_Doc\_\_c;

}

} else if(paymentType == TTConstants.INTEREST\_ONLY){

if(documentType == TTConstants.FULL\_DOC){

interestRate = rateindex.Interest\_Rate\_Full\_Doc\_\_c + rateindex.Investment\_Full\_Doc\_\_c + rateindex.Interest\_Only\_Full\_Doc\_\_c;

} else if(documentType == TTConstants.MID\_DOC){

interestRate = rateindex.Interest\_Rate\_Mid\_Doc\_\_c + rateindex.Investment\_Mid\_Doc\_\_c + rateindex.Interest\_Only\_Mid\_Doc\_\_c;

}

}

}

}

}

//Venky added the below fix to avoid argument cannot be null error when one of the field involved in the operation is null

if(interestRate > 0){

marginResults.put(indSplitId, interestRate);

} else {

marginResults.put(indSplitId, 0);

}

}

for(genesis\_\_Rate\_Schedule\_Setup\_\_c childRate : rateSchedules){

Decimal interestRate = marginResults.get(childRate.genesis\_\_Application\_\_c);

if(interestRate == null || interestRate == 0) {

continue;

}

if(childRate.genesis\_\_Spread\_\_c != null && childRate.genesis\_\_Base\_Rate\_\_c != null) {

childRate.genesis\_\_Margin\_Rate\_\_c = interestRate - childRate.genesis\_\_Spread\_\_c - childRate.genesis\_\_Base\_Rate\_\_c;

} else if(childRate.genesis\_\_Base\_Rate\_\_c != null) {

childRate.genesis\_\_Margin\_Rate\_\_c = interestRate - childRate.genesis\_\_Base\_Rate\_\_c;

} else if(childRate.genesis\_\_Spread\_\_c != null) {

childRate.genesis\_\_Margin\_Rate\_\_c = interestRate - childRate.genesis\_\_Spread\_\_c;

}

if(childRate.genesis\_\_Application\_\_r.genesis\_\_Status\_\_c != TTConstants.CHANGE\_MEMO\_APPROVAL && childRate.Schedule\_Action\_Method\_\_c != POPULATE\_AND\_UPDATE\_RATE\_SCHEDULE){

if(childRate.genesis\_\_Application\_\_r.genesis\_\_Parent\_Application\_\_r.Is\_Simpology\_Application\_\_c == false){

childRate.Schedule\_Action\_Method\_\_c = POPULATE\_AND\_UPDATE\_RATE\_SCHEDULE;

} else {

if(childRate.genesis\_\_Application\_\_r.genesis\_\_Parent\_Application\_\_r.genesis\_\_Status\_\_c != TTConstants.NEW\_ENTERED){

childRate.Schedule\_Action\_Method\_\_c = POPULATE\_AND\_UPDATE\_RATE\_SCHEDULE;

}

}

}

rateSchToUpdate.add(childRate);

}

if(!rateSchToUpdate.isEmpty()) {

update rateSchToUpdate;

}

}

/\*\*

\* @description This method creates and inserts new Document\_Checklist\_\_c records for applications, based on required document categories and conditions, where no existing checklist records are present.

\* @author Swarna Roy | 31-07-2024

\* @param packageIds

\*\*/

public static void simpologyDocumentInsert(Set <Id> packageIds){

String docType;

String assetClassType;

Boolean settleCheck;

List<Document\_Checklist\_\_c> docChecklistToInsert = new List<Document\_Checklist\_\_c>();

Map<Id, Document\_Checklist\_\_c> docChkListDataMap = new Map<Id, Document\_Checklist\_\_c>();

Map<Id, clcommon\_\_Document\_Category\_\_c> masterDocDataMap = new Map<Id, clcommon\_\_Document\_Category\_\_c>();

if(!packageIds.isEmpty()){

List<clcommon\_\_Document\_Category\_\_c> docCats = [SELECT Id,

genesis\_\_Application\_\_c,

clcommon\_\_Category\_Name\_\_c,

clcommon\_\_Document\_Definition\_\_r.clcommon\_\_Description\_\_c,

clcommon\_\_Parent\_Document\_Category\_\_c,

clcommon\_\_Document\_Definition\_\_c,

clcommon\_\_Priority\_\_c,

clcommon\_\_Status\_\_c,

clcommon\_\_Parent\_Document\_Category\_\_r.clcommon\_\_Category\_Name\_\_c,

genesis\_\_Application\_\_r.genesis\_\_Asset\_Class\_\_c,

genesis\_\_Application\_\_r.Settlement\_Doc\_Checklist\_\_c

FROM clcommon\_\_Document\_Category\_\_c

WHERE genesis\_\_Application\_\_c IN :packageIds

AND clcommon\_\_Required\_\_c = true];

for(clcommon\_\_Document\_Category\_\_c docCat : docCats){

masterDocDataMap.put(docCat.Id, docCat);

}

List<Document\_Checklist\_\_c> exisChks = [SELECT Id,

Category\_Name\_\_c,

Application\_\_c

FROM Document\_Checklist\_\_c

WHERE Application\_\_c IN :packageIds];

for(Document\_Checklist\_\_c exisChk : exisChks){

docChkListDataMap.put(exisChk.Id, exisChk);

}

for(Id docId : masterDocDataMap.keySet()){

Map<String, Id> docCategoryNameMap = new Map<String, Id>();

clcommon\_\_Document\_Category\_\_c docs = masterDocDataMap.get(docId);

for(Id docChkId : docChkListDataMap.keySet()){

Document\_Checklist\_\_c docChkList = docChkListDataMap.get(docChkId);

if(docChkList.Application\_\_c == docs.genesis\_\_Application\_\_c){

docCategoryNameMap.put(docChkList.Category\_Name\_\_c, docChkList.Application\_\_c);

}

}

if(!docCategoryNameMap.containsKey(docs.clcommon\_\_Category\_Name\_\_c)){

docType = docs.clcommon\_\_Parent\_Document\_Category\_\_r.clcommon\_\_Category\_Name\_\_c;

assetClassType = docs.genesis\_\_Application\_\_r.genesis\_\_Asset\_Class\_\_c;

settleCheck = docs.genesis\_\_Application\_\_r.Settlement\_Doc\_Checklist\_\_c;

if(!(docType == TTConstants.RESI\_DOCS && assetClassType != TTConstants.RESIDENTIAL\_ASSET\_CLASS)

&& !(docType == TTConstants.COMM\_DOCS && assetClassType != TTConstants.COMMERCIAL\_ASSET\_CLASS)

&& !(docType == TTConstants.SMSF\_DOCS && assetClassType != TTConstants.SMSF\_RESIDENTIAL\_ASSET\_CLASS && assetClassType != TTConstants.SMSF\_COMMERCIAL\_ASSET\_CLASS)

&& !(docType == TTConstants.SETTLEMENT\_DOCS && settleCheck == false)) {

Document\_Checklist\_\_c docChk = new Document\_Checklist\_\_c();

docChk.Application\_\_c = docs.genesis\_\_Application\_\_c;

docChk.Category\_Name\_\_c = docs.clcommon\_\_Category\_Name\_\_c;

docChk.Description\_\_c = docs.clcommon\_\_Document\_Definition\_\_r.clcommon\_\_Description\_\_c;

docChk.Document\_Definition\_\_c = docs.clcommon\_\_Document\_Definition\_\_c;

docChk.Parent\_Document\_Category\_\_c = docs.clcommon\_\_Parent\_Document\_Category\_\_c;

docChk.Priority\_\_c = docs.clcommon\_\_Priority\_\_c;

docChk.Status\_\_c = docs.clcommon\_\_Status\_\_c;

docChecklistToInsert.add(docChk);

}

}

}

}

if(!docChecklistToInsert.isEmpty()){

insert docChecklistToInsert;

}

}

/\*\*

\* @description This method updates the Aggregator\_based\_Access\_\_c field on Account records based on aggregated values from related parties associated with specified applications.

\* @author Swarna Roy | 31-07-2024

\* @param updateAggregatorIds

\*\*/

public static void updateAggregatorOnAccounts(Set <Id> updateAggregatorIds){

List<Account> accountsList = new List<Account>();

Map<Id, String> aggregatorMap = new Map<Id, String>();

Map<Id, String> oldAggregatorMap = new Map<Id, String>();

if(updateAggregatorIds.size() > 0) {

//Get the Application parties and their unique account Id values

for(clcommon\_\_Party\_\_c party : [SELECT Id,

Party\_Type\_Name\_\_c,

clcommon\_\_Account\_\_r.Aggregator\_based\_Access\_\_c,

Aggregator\_Name\_portal\_Access\_\_c

FROM clcommon\_\_Party\_\_c

WHERE Party\_Type\_Name\_\_c IN (:TTConstants.BORROWER, :TTConstants.COBORROWER, :TTConstants.GUARANTOR, :TTConstants.BARE\_TRUST)

AND genesis\_\_Application\_\_c IN :updateAggregatorIds AND clcommon\_\_Account\_\_c != null]) {

if(aggregatorMap.containsKey(party.clcommon\_\_Account\_\_c) && !aggregatorMap.get(party.clcommon\_\_Account\_\_c).toLowerCase().contains(party.Aggregator\_Name\_portal\_Access\_\_c.toLowerCase())) {

String currentValue = aggregatorMap.get(party.clcommon\_\_Account\_\_c);

String newValue = currentValue + ',' + party.Aggregator\_Name\_portal\_Access\_\_c;

aggregatorMap.put(party.clcommon\_\_Account\_\_c, newValue);

} else if(!aggregatorMap.containsKey(party.clcommon\_\_Account\_\_c)) {

aggregatorMap.put(party.clcommon\_\_Account\_\_c, party.Aggregator\_Name\_portal\_Access\_\_c);

}

oldAggregatorMap.put(party.clcommon\_\_Account\_\_c, party.clcommon\_\_Account\_\_r.Aggregator\_based\_Access\_\_c);

}

}

//map account Id and aggregator based access values

for(Id accId : aggregatorMap.keySet()){

if(aggregatorMap.get(accId) != oldAggregatorMap.get(accId)) {

Account acc = new Account();

acc.Id = accId;

acc.Aggregator\_based\_Access\_\_c = aggregatorMap.get(accId);

accountsList.add(acc);

}

}

if(!accountsList.isEmpty()) {

update accountsList;

}

}

/\*@Future

public static void recalculateFeeFuture(String appId){

//calculating fee for commercial & smsf

List<genesis\_\_Applications\_\_c> childapps = [SELECT Id FROM genesis\_\_Applications\_\_c WHERE genesis\_\_Parent\_Application\_\_c =: appId];

if(!childapps.isEmpty()){

for(genesis\_\_Applications\_\_c split : childapps){

CalculateFeeForCommercialAndSMSF.calculateFee(split.Id);

}

}

}\*/

/\*\*

\* @description This method recalculates fees for specified child applications if their parent application's ID is in the provided set.

\* @author Swarna Roy | 31-07-2024

\* @param parentIds

\* @param childApps

\*\*/

public static void recalculateFee(Set <Id> parentIds, List<genesis\_\_Applications\_\_c> childApps){

for(genesis\_\_Applications\_\_c split : childApps){

if(parentIds.contains(split.genesis\_\_Parent\_Application\_\_c)) {

CalculateFeeForCommercialAndSMSF.calculateFee(split.Id);

}

}

}

/\*\*

\* @description This method calculates and updates the CLTV and total collateral amount for loan applications, considering collateral details, loan amounts, and application lock status.

\* @author Swarna Roy | 31-07-2024

\* @param appIds

\* @param parentIds

\* @param newApp

\*\*/

public static void calculateSplitLtv(Set <Id> appIds, Set <Id> parentIds, List<genesis\_\_Applications\_\_c> newApp){

Decimal totalSum = 0.00;

Decimal minValue;

String colCmprId;

Map<Id,Boolean> appLockMap = new Map<Id,Boolean>();

Boolean isAppLocked = false;

Boolean valuationTypeEmpty = false;

Map<Id, genesis\_\_Application\_Collateral\_\_c> collateralMap = new Map<Id, genesis\_\_Application\_Collateral\_\_c>();

if(!parentIds.isEmpty()){

List<genesis\_\_Application\_Collateral\_\_c> allApplCol = [SELECT Id,

Name,

genesis\_\_Collateral\_\_c,

genesis\_\_Application\_\_c,

genesis\_\_Collateral\_\_r.Current\_Value\_\_c,

genesis\_\_Collateral\_\_r.clcommon\_\_Estimated\_Value\_\_c,

genesis\_\_Collateral\_\_r.contract\_Amount\_\_c,

genesis\_\_Collateral\_\_r.Valuation\_Type\_\_c,

genesis\_\_Application\_\_r.genesis\_\_CL\_Product\_\_r.Product\_Type\_\_c

FROM genesis\_\_Application\_Collateral\_\_c

WHERE genesis\_\_Application\_\_c IN :parentIds

AND is\_Security\_\_c = true];

for(genesis\_\_Application\_Collateral\_\_c appCol : allApplCol){

collateralMap.put(appCol.Id,appCol);

}

}

if(!appIds.isEmpty()) {

appLockMap = Approval.isLocked(new List<Id>(appIds));

}

for(genesis\_\_Applications\_\_c app : newApp){

if(appLockMap.size() > 0){

isAppLocked = appLockMap.get(app.Id);

}

if(isAppLocked == false || app.genesis\_\_Status\_\_c == TTConstants.CHANGE\_MEMO\_APPROVAL){

totalSum = 0.00;

minValue = 0.00;

if(appIds.contains(app.Id)){

colCmprId = app.genesis\_\_Parent\_Application\_\_c;

if(collateralMap.size() > 0){

for(Id appColId : collateralMap.keySet()){

genesis\_\_Application\_Collateral\_\_c appColt = collateralMap.get(appColId);

if(appColt.genesis\_\_Application\_\_c == colCmprId){

if(appColt.genesis\_\_Collateral\_\_r.Valuation\_Type\_\_c == null) {

valuationTypeEmpty = true;

}

if(appColt.genesis\_\_Collateral\_\_r.Current\_Value\_\_c != null){

if(appColt.genesis\_\_Collateral\_\_r.contract\_Amount\_\_c == null){

minValue = appColt.genesis\_\_Collateral\_\_r.Current\_Value\_\_c;

} else {

if(appColt.genesis\_\_Collateral\_\_r.Current\_Value\_\_c < appColt.genesis\_\_Collateral\_\_r.contract\_Amount\_\_c){

minValue = appColt.genesis\_\_Collateral\_\_r.Current\_Value\_\_c;

} else {

minValue = appColt.genesis\_\_Collateral\_\_r.contract\_Amount\_\_c;

}

}

}

else{

if(appColt.genesis\_\_Collateral\_\_r.contract\_Amount\_\_c == null){

minValue = appColt.genesis\_\_Collateral\_\_r.clcommon\_\_Estimated\_Value\_\_c;

} else {

minValue = appColt.genesis\_\_Collateral\_\_r.contract\_Amount\_\_c;

}

}

totalSum = totalSum+minValue;

}

}

if(app.genesis\_\_Requested\_Loan\_Amount\_\_c != null && totalSum > 0.00 && (valuationTypeEmpty || (app.genesis\_\_Asset\_Class\_\_c != TTConstants.RESIDENTIAL\_ASSET\_CLASS

&& app.genesis\_\_Asset\_Class\_\_c != TTConstants.SMSF\_RESIDENTIAL\_ASSET\_CLASS))){

app.genesis\_\_CLTV\_\_c = (app.genesis\_\_Requested\_Loan\_Amount\_\_c/totalSum).setScale(4, System.RoundingMode.UP);

app.Total\_Collateral\_Amount\_\_c = totalSum;

}

} else {

if(valuationTypeEmpty || (app.genesis\_\_Asset\_Class\_\_c != TTConstants.RESIDENTIAL\_ASSET\_CLASS

&& app.genesis\_\_Asset\_Class\_\_c != TTConstants.SMSF\_RESIDENTIAL\_ASSET\_CLASS)) {

app.genesis\_\_CLTV\_\_c = 0.00;

app.Total\_Collateral\_Amount\_\_c = 0.00;

}

}

}

}

}

}

/\*\*

\* @description This method computes and sets the CLTV and total collateral amount for new loan applications based on the collateral details and requested loan amounts, while also handling cases with missing valuation types or asset classes.

\* @author Swarna Roy | 31-07-2024

\* @param parentIds

\* @param newApp

\*\*/

public static void calculateSplitLtvBeforeInsert(Set <Id> parentIds, List<genesis\_\_Applications\_\_c> newApp){

Decimal totalSum = 0.00;

Decimal minValue;

String colCmprId;

Boolean valuationTypeEmpty = false;

Map<Id, genesis\_\_Application\_Collateral\_\_c> collateralMap = new Map<Id, genesis\_\_Application\_Collateral\_\_c>();

if(!parentIds.isEmpty()){

List<genesis\_\_Application\_Collateral\_\_c> allApplCol = [SELECT Id,

Name,

genesis\_\_Collateral\_\_c,

genesis\_\_Application\_\_c,

genesis\_\_Collateral\_\_r.Current\_Value\_\_c,

genesis\_\_Collateral\_\_r.clcommon\_\_Estimated\_Value\_\_c,

genesis\_\_Collateral\_\_r.contract\_Amount\_\_c,

genesis\_\_Collateral\_\_r.Valuation\_Type\_\_c,

genesis\_\_Application\_\_r.genesis\_\_CL\_Product\_\_r.Product\_Type\_\_c

FROM genesis\_\_Application\_Collateral\_\_c

WHERE genesis\_\_Application\_\_c IN :parentIds

AND is\_Security\_\_c = true];

for(genesis\_\_Application\_Collateral\_\_c appCol : allApplCol){

collateralMap.put(appCol.Id,appCol);

}

}

for(genesis\_\_Applications\_\_c app : newApp){

totalSum = 0.00;

minValue = 0.00;

if(app.Record\_Type\_Name\_\_c == TTConstants.APP\_LOAN){

app.genesis\_\_Index\_Linked\_\_c = true;

colCmprId = app.genesis\_\_Parent\_Application\_\_c;

if(collateralMap.size() > 0){

for(Id appColId : collateralMap.keySet()){

genesis\_\_Application\_Collateral\_\_c appColt = collateralMap.get(appColId);

if(appColt.genesis\_\_Application\_\_c == colCmprId){

if(appColt.genesis\_\_Collateral\_\_r.Valuation\_Type\_\_c == null) {

valuationTypeEmpty = true;

}

if(appColt.genesis\_\_Collateral\_\_r.Current\_Value\_\_c != null){

if(appColt.genesis\_\_Collateral\_\_r.contract\_Amount\_\_c == null){

minValue = appColt.genesis\_\_Collateral\_\_r.Current\_Value\_\_c;

} else {

if(appColt.genesis\_\_Collateral\_\_r.Current\_Value\_\_c < appColt.genesis\_\_Collateral\_\_r.contract\_Amount\_\_c){

minValue = appColt.genesis\_\_Collateral\_\_r.Current\_Value\_\_c;

} else {

minValue = appColt.genesis\_\_Collateral\_\_r.contract\_Amount\_\_c;

}

}

} else {

if(appColt.genesis\_\_Collateral\_\_r.contract\_Amount\_\_c == null){

minValue = appColt.genesis\_\_Collateral\_\_r.clcommon\_\_Estimated\_Value\_\_c;

} else {

minValue = appColt.genesis\_\_Collateral\_\_r.contract\_Amount\_\_c;

}

}

totalSum = totalSum + minValue;

}

}

if(app.genesis\_\_Requested\_Loan\_Amount\_\_c != null && totalSum > 0.00 && (valuationTypeEmpty || (app.genesis\_\_Asset\_Class\_\_c != TTConstants.RESIDENTIAL\_ASSET\_CLASS

&& app.genesis\_\_Asset\_Class\_\_c != TTConstants.SMSF\_RESIDENTIAL\_ASSET\_CLASS))){

app.genesis\_\_CLTV\_\_c =(app.genesis\_\_Requested\_Loan\_Amount\_\_c/totalSum).setScale(4,System.RoundingMode.UP);

app.Total\_Collateral\_Amount\_\_c = totalSum;

}

} else {

if(valuationTypeEmpty || (app.genesis\_\_Asset\_Class\_\_c != TTConstants.RESIDENTIAL\_ASSET\_CLASS

&& app.genesis\_\_Asset\_Class\_\_c != TTConstants.SMSF\_RESIDENTIAL\_ASSET\_CLASS)) {

app.genesis\_\_CLTV\_\_c = 0.00;

app.Total\_Collateral\_Amount\_\_c = 0.00;

}

}

}

}

}

/\*\*

\* @description This method verifies if the owner of each application is a valid user within the specified sales queue and flags applications with owners not in the queue as having an invalid queue user.

\* @author Swarna Roy | 31-07-2024

\* @param parentOwnerMap

\* @param newApp

\*\*/

public static void salesQueueCheck(Map<Id, String> parentOwnerMap, List<genesis\_\_Applications\_\_c> newApp){

Id packageId;

Id ownerId;

Set<Id> queueUser = new Set<Id>();

Id salesQueueId = [SELECT Id FROM Group WHERE Name = :TTConstants.SALES\_DEPT AND Type = :TTConstants.QUEUE].Id;

if(parentOwnerMap.size() > 0){

List<GroupMember> validUsers = [SELECT Id, UserOrGroupId FROM GroupMember WHERE GroupId = :salesQueueId];

for(GroupMember validUser : validUsers){

queueUser.add(validUser.UserOrGroupId);

}

}

for(genesis\_\_Applications\_\_c app : newApp){

if(app.Is\_Simpology\_Application\_\_c == false && app.Record\_Type\_Name\_\_c == TTConstants.APP\_PACKAGE){

packageId = app.Id;

if(parentOwnerMap.containsKey(packageId)){

ownerId = parentOwnerMap.get(packageId);

if(!queueUser.contains(ownerId)){

app.Invalid\_Queue\_User\_\_c = true;

}

}

}

}

}

/\*\*

\* @description This method inserts new rate schedule records for applications, updates existing application records with the latest interest and borrower age information, and adjusts related rate schedules based on specific criteria.

It calculates the borrower’s age at the end of the term, updates related loan fields, and manages interest rate information based on the loan product and index.

The method also handles updates for related child applications and rate schedules, ensuring that the data remains consistent and accurate.

\* @author Swarna Roy | 31-07-2024

\* @param parentIds

\* @param newApp

\* @param allChildAppsMap

\* @param allParentAppsMap

\*\*/

public static void insertInterestInformationAndUpdateAge(Set <Id> parentIds, List<genesis\_\_Applications\_\_c> newApp, Map <Id, genesis\_\_Applications\_\_c> allChildAppsMap, Map <Id, genesis\_\_Applications\_\_c> allParentAppsMap){

Set<Id> appIds = new Set<Id>();

Set<Id> splitIds = new Set<Id>();

Set<Id> recordIds = new Set<Id>();

List<genesis\_\_Rate\_Schedule\_Setup\_\_c> rateSchdlToInsert = new List<genesis\_\_Rate\_Schedule\_Setup\_\_c>();

List<genesis\_\_Applications\_\_c> pkgAppToUpdate = new List<genesis\_\_Applications\_\_c>();

List<genesis\_\_Applications\_\_c> splitAppToUpdate = new List<genesis\_\_Applications\_\_c>();

Map<String, Id> indexMap = new Map<String, Id>();

Map<Id, Decimal> appTermMap = new Map<Id, Decimal>();

Map<Id, Decimal> borrowerAgeMap = new Map<Id, Decimal>();

Map<Id, Decimal> brokerTrailMap = new Map<Id, Decimal>();

Decimal borrowerAge = 0;

Decimal totTerm = 0;

String colCmprId;

String rateIndex;

Id indexId;

Decimal margin;

Boolean isUpdate = false;

Boolean updateSolicitorFirm = false;

genesis\_\_Applications\_\_c packageApp;

genesis\_\_Applications\_\_c splitApp;

Set <Id> packageAppIdsToUpdate = new Set<Id>();

Map<Id, Boolean> includeMarginMap = new Map<Id, Boolean>();

Map<Id, Decimal> marginCalculationResults = new Map<Id, Decimal>();

Map<Id, genesis\_\_Rate\_Schedule\_Setup\_\_c> schdlMap = new Map<Id, genesis\_\_Rate\_Schedule\_Setup\_\_c>();

List<genesis\_\_Rate\_Schedule\_Setup\_\_c> rateScheduleSetupUpdate = new List<genesis\_\_Rate\_Schedule\_Setup\_\_c>();

Map<Id, String> pkgUniqueCheckMap = new Map<Id, String>();

if(!parentIds.isEmpty()){

List<clcommon\_\_Party\_\_c> appParties = [SELECT Id,

clcommon\_\_Contact\_\_r.Age\_\_c,

genesis\_\_Application\_\_c

FROM clcommon\_\_Party\_\_c

WHERE genesis\_\_Application\_\_c in :parentIds

AND Party\_Type\_Name\_\_c = :TTConstants.BORROWER];

for(clcommon\_\_Party\_\_c appParty : appParties){

if(appParty.clcommon\_\_Contact\_\_r.Age\_\_c != null){

borrowerAgeMap.put(appParty.genesis\_\_Application\_\_c, appParty.clcommon\_\_Contact\_\_r.Age\_\_c);

} else {

borrowerAgeMap.put(appParty.genesis\_\_Application\_\_c, 0);

}

}

for(genesis\_\_Applications\_\_c app : allChildAppsMap.values()){

Id parentAppId = app.genesis\_\_Parent\_Application\_\_c;

if(appTermMap.containsKey(parentAppId)) {

if(app.genesis\_\_Term\_\_c > appTermMap.get(parentAppId)) {

appTermMap.put(parentAppId, app.genesis\_\_Term\_\_c);

}

} else {

appTermMap.put(parentAppId, app.genesis\_\_Term\_\_c);

}

}

for(Id each : parentIds){

packageApp = allParentAppsMap.get(each);

if(packageApp.Broker\_Trail\_\_c != null){

brokerTrailMap.put(packageApp.Id, packageApp.Broker\_Trail\_\_c);

}

}

}

List<clcommon\_\_Floating\_Rate\_Index\_\_c> rateIndexes = [SELECT Id, Name

FROM clcommon\_\_Floating\_Rate\_Index\_\_c

WHERE clcommon\_\_Active\_\_c = true];

for(clcommon\_\_Floating\_Rate\_Index\_\_c rate : rateIndexes){

indexMap.put(rate.Name, rate.Id);

}

for(genesis\_\_Applications\_\_c app : newApp){

if(app.Record\_Type\_Name\_\_c == TTConstants.APP\_LOAN){

colCmprId = app.genesis\_\_Parent\_Application\_\_c;

if(app.genesis\_\_CL\_Product\_Name\_\_c == TTConstants.MID\_TICKET\_COMMERCIAL) {

if(allParentAppsMap.get(app.genesis\_\_Parent\_Application\_\_c).genesis\_\_CL\_Product\_Name\_\_c == TTConstants.SMSF\_COMMERCIAL\_MASTER) {

rateIndex = TTConstants.MID\_TICKET\_SMSF\_COMMERCIAL\_LOAN\_RATE;

} else {

rateIndex = TTConstants.MID\_TICKET\_COMMERCIAL\_LOAN\_RATE;

}

} else if(app.genesis\_\_CL\_Product\_Name\_\_c == TTConstants.BUSINESS\_LOAN || app.genesis\_\_CL\_Product\_Name\_\_c == TTConstants.RESIDUAL\_STOCK\_LOAN) {

rateIndex = TTConstants.BUSINESS\_LOAN\_REFERENCE\_RATE;

} else if(app.genesis\_\_CL\_Product\_Name\_\_c == ConstructionLoanUtil.CONSTRUCTION\_LOAN) {

rateIndex = ConstructionLoanUtil.CONSTRUCTION\_LOAN\_REFERENCE\_RATE;

updateSolicitorFirm = true;

} else if(app.Parent\_Rate\_Index\_\_c != null) {

rateIndex = app.Parent\_Rate\_Index\_\_c;

}

splitIds.add(app.Id);

if(appTermMap.containsKey(colCmprId)) {

totTerm = appTermMap.get(colCmprId);

}

if(borrowerAgeMap.containsKey(colCmprId)) {

borrowerAge = borrowerAgeMap.get(colCmprId);

}

if(app.genesis\_\_Term\_\_c != null && app.genesis\_\_Term\_\_c > 0){

if(app.genesis\_\_Term\_\_c > totTerm){

totTerm = app.genesis\_\_Term\_\_c;

}

}

if(rateIndex != null){

if(rateIndex != app.Parent\_Rate\_Index\_\_c) {

packageApp = allParentAppsMap.get(app.genesis\_\_Parent\_Application\_\_c);

packageApp.Rate\_Index\_\_c = rateIndex;

if(updateSolicitorFirm) {

packageApp.Solicitor\_Firm\_\_c = PURCELL\_PARTNERS;

}

pkgAppToUpdate.add(packageApp);

packageAppIdsToUpdate.add(app.genesis\_\_Parent\_Application\_\_c);

} else {

if(indexMap.containsKey(rateIndex)) {

indexId = indexMap.get(rateIndex);

genesis\_\_Rate\_Schedule\_Setup\_\_c rateSchdl = new genesis\_\_Rate\_Schedule\_Setup\_\_c();

rateSchdl.genesis\_\_Index\_\_c = indexId;

rateSchdl.genesis\_\_Application\_\_c = app.Id;

if(app.Pricing\_Basis\_\_c != null){

if(app.Pricing\_Basis\_\_c == TTConstants.VARIABLE){

rateSchdl.genesis\_\_Rate\_Type\_\_c = TTConstants.FLOATING;

} else {

rateSchdl.genesis\_\_Rate\_Type\_\_c = app.Pricing\_Basis\_\_c;

}

} else {

rateSchdl.genesis\_\_Rate\_Type\_\_c = TTConstants.FIXED;

}

if(brokerTrailMap.containsKey(colCmprId)){

rateSchdl.genesis\_\_Spread\_\_c = brokerTrailMap.get(colCmprId);

}

rateSchdl.genesis\_\_Start\_Date\_\_c = Date.today();

rateSchdlToInsert.add(rateSchdl);

appIds.add(app.Id);

}

}

}

if(borrowerAge > 0){

if(!pkgUniqueCheckMap.containsKey(colCmprId)){

packageApp = allParentAppsMap.get(colCmprId);

packageApp.Borrower\_Age\_at\_term\_end\_\_c = (borrowerAge + (totTerm/12)).setScale(0);

packageApp.Borrower\_Age\_\_c = borrowerAge;

if(packageApp.Borrower\_Age\_at\_term\_end\_\_c > 999){

packageApp.Borrower\_Age\_at\_term\_end\_\_c = 999;

}

if(packageApp.Borrower\_Age\_\_c > 999){

packageApp.Borrower\_Age\_\_c = 999;

}

if(!packageAppIdsToUpdate.contains(colCmprId)) {

pkgAppToUpdate.add(packageApp);

}

}

pkgUniqueCheckMap.put(colCmprId, TTConstants.CHECK);

}

}

}

if(!splitIds.isEmpty()){

for(Id each: splitIds) {

splitApp = allChildAppsMap.get(each);

if(splitApp.genesis\_\_Parent\_Application\_\_r.NCCP\_\_c == true){

splitApp.NCCP\_\_c = splitApp.genesis\_\_Parent\_Application\_\_r.NCCP\_\_c;

isUpdate = true;

}

if(splitApp.genesis\_\_Parent\_Application\_\_r.genesis\_\_Disbursement\_Date\_\_c != null){

splitApp.genesis\_\_Disbursement\_Date\_\_c = splitApp.genesis\_\_Parent\_Application\_\_r.genesis\_\_Disbursement\_Date\_\_c;

recordIds.add(splitApp.Id);

isUpdate = true;

} else {

if(splitApp.genesis\_\_Parent\_Application\_\_r.genesis\_\_Expected\_Close\_Date\_\_c != null){

splitApp.genesis\_\_Expected\_Close\_Date\_\_c = splitApp.genesis\_\_Parent\_Application\_\_r.genesis\_\_Expected\_Close\_Date\_\_c;

recordIds.add(splitApp.Id);

isUpdate = true;

}

}

if(isUpdate == true){

splitAppToUpdate.add(splitApp);

}

}

}

if(!splitAppToUpdate.isEmpty()){

pkgAppToUpdate.addAll(splitAppToUpdate);

}

if(!pkgAppToUpdate.isEmpty()){

UPDATE pkgAppToUpdate;

}

if(!rateSchdlToInsert.isEmpty()){

INSERT rateSchdlToInsert;

}

if(!appIds.isEmpty()){

List<genesis\_\_Rate\_Schedule\_Setup\_\_c> rateSchUpds = [SELECT Id,

genesis\_\_Margin\_Rate\_\_c,

genesis\_\_Base\_Rate\_\_c,

genesis\_\_Spread\_\_c,

Schedule\_Action\_Method\_\_c,

genesis\_\_Application\_\_c

FROM genesis\_\_Rate\_Schedule\_Setup\_\_c

WHERE genesis\_\_Application\_\_c IN :appIds];

for(genesis\_\_Rate\_Schedule\_Setup\_\_c rateSchdlUpd : rateSchUpds){

schdlMap.put(rateSchdlUpd.genesis\_\_Application\_\_c, rateSchdlUpd);

}

marginCalculationResults = CLXUtil.marginCalculation(allChildAppsMap.values(), appIds);

for(Id appId: appIds) {

genesis\_\_Rate\_Schedule\_Setup\_\_c rateUpd = schdlMap.get(appId);

margin = marginCalculationResults.get(appId);

if(margin != 1000){

rateUpd.genesis\_\_Margin\_Rate\_\_c = margin;

}

if(rateUpd.Schedule\_Action\_Method\_\_c != POPULATE\_AND\_UPDATE\_RATE\_SCHEDULE) {

rateUpd.Schedule\_Action\_Method\_\_c = POPULATE\_AND\_UPDATE\_RATE\_SCHEDULE;

rateScheduleSetupUpdate.add(rateUpd);

}

includeMarginMap.put(appId, true);

}

if(!rateScheduleSetupUpdate.isEmpty()) {

update rateScheduleSetupUpdate;

}

}

}

/\*\*

\* @description This method updates the NCCP flag and SLA Tier field for a list of applications based on the asset class and borrower types associated with the specified application IDs.

\* @author Swarna Roy | 31-07-2024

\* @param appIds

\* @param appList

\* @return List<genesis\_\_Applications\_\_c>

\*\*/

public static List<genesis\_\_Applications\_\_c> updateNccpFlagAndSlaTier(Set<Id> appIds, List<genesis\_\_Applications\_\_c> appList) {

Set <String> borrowerTypes = new Set <String>();

List<genesis\_\_Applications\_\_c> appToUpdate = new List<genesis\_\_Applications\_\_c>();

if(appIds.isEmpty()) {

throw new CustomException(TTConstants.NO\_APP\_ID);

}

// Getting all borrower and coborrower parties

List<clcommon\_\_Party\_\_c> parties = [SELECT clcommon\_\_Account\_\_r.clcommon\_\_Legal\_Entity\_Type\_\_r.Name

FROM clcommon\_\_Party\_\_c

WHERE genesis\_\_Application\_\_c IN :appIds

AND clcommon\_\_Type\_\_r.Name IN (:TTConstants.BORROWER, :TTConstants.COBORROWER)];

for(clcommon\_\_Party\_\_c each: parties) {

borrowerTypes.add(each.clcommon\_\_Account\_\_r.clcommon\_\_Legal\_Entity\_Type\_\_r.Name);

}

for(genesis\_\_Applications\_\_c app: appList) {

if(appIds.contains(app.Id)) {

if(app.genesis\_\_Asset\_Class\_\_c == TTConstants.RESIDENTIAL\_ASSET\_CLASS) {

if(borrowerTypes.size() == 1 && borrowerTypes.contains(TTConstants.INDIVIDUAL)) {

app.NCCP\_\_c = true;

app.SLA\_Tier\_\_c = TTConstants.SLA\_TIER1\_STATUS;

} else {

app.NCCP\_\_c = false;

}

if(borrowerTypes.contains(TTConstants.COMPANY)) {

app.SLA\_Tier\_\_c = TTConstants.SLA\_TIER2\_STATUS;

}

} else {

app.NCCP\_\_c = false;

}

if(app.genesis\_\_CL\_Product\_\_r.Product\_Type\_\_c == TTConstants.RESI\_PROD\_TYPE) {

if(borrowerTypes.contains(TTConstants.TRUST)) {

app.SLA\_Tier\_\_c = TTConstants.SLA\_TIER3\_STATUS;

}

} else {

app.SLA\_Tier\_\_c = TTConstants.SLA\_TIER4\_STATUS;

}

appToUpdate.add(app);

}

}

return appToUpdate;

}

/\*\*

\* @description This method sets the Schedule\_Action\_Method\_\_c field to GENERATE\_SCHEDULE for rate schedule records associated with the specified application IDs that currently have a null action method.

\* @author Swarna Roy | 31-07-2024

\* @param updateSchedule

\*\*/

public static void rescheduleLoan(Set<Id> updateSchedule) {

List<genesis\_\_Rate\_Schedule\_Setup\_\_c> rateSchedules = [SELECT Id

FROM genesis\_\_Rate\_Schedule\_Setup\_\_c

WHERE genesis\_\_Application\_\_c IN :updateSchedule

AND Schedule\_Action\_Method\_\_c = null];

// Generate Schedule

for(genesis\_\_Rate\_Schedule\_Setup\_\_c rate : rateSchedules){

rate.Schedule\_Action\_Method\_\_c = GENERATE\_SCHEDULE;

}

update rateSchedules;

}

/\*\*

\* @description This method updates repayment amounts for genesis\_\_Applications\_\_c records based on their amortization schedules, construction terms, and interest-only periods, and then performs a bulk update on those records.

\* @author Swarna Roy | 30-07-2024

\* @param loanIds

\*\*/

public static List<genesis\_\_Applications\_\_c> updateRepaymentAmount(Set<Id> loanIds, Map <Id, genesis\_\_Applications\_\_c> allChildAppsMap){

recursiveflag = false;

genesis\_\_Applications\_\_c splitAppToUpdate;

List<genesis\_\_Applications\_\_c> appsToUpdate = new List<genesis\_\_Applications\_\_c>();

List <genesis\_\_Applications\_\_c> allApps = [SELECT Id,

Construction\_Term\_\_c,

genesis\_\_Term\_\_c,

genesis\_\_Interest\_Only\_Period\_\_c,

(SELECT genesis\_\_Total\_Due\_Amount\_\_c

FROM genesis\_\_Amortization\_Schedule\_\_r

ORDER BY genesis\_\_Payment\_Number\_\_c ASC)

FROM genesis\_\_Applications\_\_c

WHERE Id IN :loanIds];

for(genesis\_\_Applications\_\_c app: allApps) {

splitAppToUpdate = allChildAppsMap.get(app.Id);

if(app.genesis\_\_Amortization\_Schedule\_\_r.size() != app.genesis\_\_Term\_\_c

|| app.genesis\_\_Interest\_Only\_Period\_\_c >= app.genesis\_\_Term\_\_c

|| app.Construction\_Term\_\_c >= app.genesis\_\_Term\_\_c){

continue;

}

if(app.Construction\_Term\_\_c != null && app.genesis\_\_Interest\_Only\_Period\_\_c != null) {

Integer constructionTerm = Integer.valueOf(app.Construction\_Term\_\_c);

Integer interestOnlyPeriod = Integer.valueOf(app.genesis\_\_Interest\_Only\_Period\_\_c);

splitAppToUpdate.Construction\_Repayment\_Amount\_\_c = app.genesis\_\_Amortization\_Schedule\_\_r[0].genesis\_\_Total\_Due\_Amount\_\_c;

if(constructionTerm == interestOnlyPeriod) {

splitAppToUpdate.IO\_Repayment\_Amount\_\_c = 0;

} else {

splitAppToUpdate.IO\_Repayment\_Amount\_\_c = app.genesis\_\_Amortization\_Schedule\_\_r[constructionTerm].genesis\_\_Total\_Due\_Amount\_\_c;

}

splitAppToUpdate.P\_I\_Repayment\_Amount\_\_c = app.genesis\_\_Amortization\_Schedule\_\_r[interestOnlyPeriod].genesis\_\_Total\_Due\_Amount\_\_c;

} else if(app.genesis\_\_Interest\_Only\_Period\_\_c != null) {

Integer interestOnlyPeriod = Integer.valueOf(app.genesis\_\_Interest\_Only\_Period\_\_c);

splitAppToUpdate.Construction\_Repayment\_Amount\_\_c = 0;

splitAppToUpdate.IO\_Repayment\_Amount\_\_c = app.genesis\_\_Amortization\_Schedule\_\_r[0].genesis\_\_Total\_Due\_Amount\_\_c;

splitAppToUpdate.P\_I\_Repayment\_Amount\_\_c = app.genesis\_\_Amortization\_Schedule\_\_r[interestOnlyPeriod].genesis\_\_Total\_Due\_Amount\_\_c;

} else {

splitAppToUpdate.Construction\_Repayment\_Amount\_\_c = 0;

splitAppToUpdate.IO\_Repayment\_Amount\_\_c = 0;

splitAppToUpdate.P\_I\_Repayment\_Amount\_\_c = app.genesis\_\_Amortization\_Schedule\_\_r[0].genesis\_\_Total\_Due\_Amount\_\_c;

}

appsToUpdate.add(splitAppToUpdate);

}

return appsToUpdate;

}

/\*\*

\* @description This method asynchronously updates clcommon\_\_Party\_\_c records with new Relationship\_Manager details based on changes to Contact records and updates the corresponding genesis\_\_Applications\_\_c records with these manager IDs.

\* @author Venky added this for jira TB-19.

\* @param brokerContactAppIds

\*\*/

public static List<genesis\_\_Applications\_\_c> handleBrokerChange(Map<Id,Id> brokerContactAppIds, Map <Id, genesis\_\_Applications\_\_c> allParentAppsMap){

genesis\_\_Applications\_\_c packageApp;

List<clcommon\_\_Party\_\_c> rams = new List<clcommon\_\_Party\_\_c>();

List<genesis\_\_Applications\_\_c> packages = new List<genesis\_\_Applications\_\_c>();

for(Contact contact : [SELECT Id, Residential\_Account\_Manager\_\_c, OwnerId, Owner.title

FROM Contact

WHERE Id IN:brokerContactAppIds.keySet()

LIMIT :brokerContactAppIds.size()]){

packageApp = allParentAppsMap.get(brokerContactAppIds.get(contact.Id));

if(contact.Residential\_Account\_Manager\_\_c != null ){

clcommon\_\_Party\_\_c party = new clcommon\_\_Party\_\_c();

party.clcommon\_\_User\_\_c = contact.Residential\_Account\_Manager\_\_c;

party.Role\_\_c = RESIDENTIAL\_ACCOUNT\_MANAGER;

party.genesis\_\_Application\_\_c = brokerContactAppIds.get(contact.Id);

party.genesis\_\_Is\_Internal\_User\_\_c = true;

rams.add(party);

packageApp.Residential\_Account\_Manager\_ID\_\_c = contact.Residential\_Account\_Manager\_\_c;

packages.add(packageApp);

} else if(contact.OwnerId != null && contact.Owner.title != null && contact.Owner.title.containsIgnoreCase(RELATIONSHIP\_MANAGER)){

clcommon\_\_Party\_\_c party = new clcommon\_\_Party\_\_c();

party.clcommon\_\_User\_\_c = contact.OwnerId;

party.Role\_\_c = RELATIONSHIP\_MANAGER;

party.genesis\_\_Application\_\_c = brokerContactAppIds.get(contact.Id);

party.genesis\_\_Is\_Internal\_User\_\_c = true;

rams.add(party);

packageApp.Relationship\_Manager\_ID\_\_c = contact.OwnerId;

packages.add(packageApp);

}

}

if(rams.size() > 0){

insert rams;

}

return packages;

}

}

//////////////

Test class : -

@isTest

public class GenesisApplicationTriggerHandlerV2Test{

@TestSetup

static void setup() {

clcommon\_\_Legal\_Entity\_\_c legal = new clcommon\_\_Legal\_Entity\_\_c();

legal.Name= 'Individual' ;

insert legal;

List <clcommon\_\_Legal\_Entity\_\_c> individualLegalEntity = [ SELECT Id FROM clcommon\_\_Legal\_Entity\_\_c WHERE Name = 'Individual'];

System.assertEquals(true,!individualLegalEntity.isEmpty(), 'Individual legal entity not found');

Id accRecordType = Schema.SObjectType.Account.getRecordTypeInfosByName().get('Aggregator').getRecordTypeId();

Id accBrokerRecordType = Schema.SObjectType.Account.getRecordTypeInfosByName().get('Broker').getRecordTypeId();

Account brokerAggregatorAccount = new Account();

brokerAggregatorAccount.Name = 'Raghav Broker B';

brokerAggregatorAccount.Type = 'Aggregator';

brokerAggregatorAccount.Salutation\_\_c = 'Mr.';

brokerAggregatorAccount.First\_Name\_\_c = 'Raghav';

brokerAggregatorAccount.Last\_Name\_\_c = 'Broker B';

brokerAggregatorAccount.clcommon\_\_Email\_\_c = 'BrokerB@mailinator.com';

brokerAggregatorAccount.loan\_\_Broker\_\_c = true;

brokerAggregatorAccount.recordTypeId = accRecordType;

brokerAggregatorAccount.clcommon\_\_Legal\_Entity\_Type\_\_c = legal.Id;

insert brokerAggregatorAccount;

Account acc = new Account ();

acc.Name = 'Raghav Mahajan';

acc.RecordTypeId = accRecordType;

acc.Type = 'Aggregator';

acc.Salutation\_\_c = 'Mr.';

acc.First\_Name\_\_c = 'Raghav';

acc.Last\_Name\_\_c = 'Mahajan';

acc.clcommon\_\_Legal\_Entity\_Type\_\_c = legal.Id;

insert acc;

Account account = new Account ();

account.Name = 'Robin Mahajan';

account.RecordTypeId = accRecordType;

account.Type = 'Individual';

account.Salutation\_\_c = 'Mr.';

account.First\_Name\_\_c = 'Robin';

account.Last\_Name\_\_c = 'Mahajan';

account.clcommon\_\_Legal\_Entity\_Type\_\_c = legal.Id;

insert account;

Account accounts = new Account ();

accounts.Name = 'Aghav Mahajan';

accounts.RecordTypeId = accRecordType;

accounts.Type = 'Individual';

accounts.Salutation\_\_c = 'Mr.';

accounts.First\_Name\_\_c = 'Aghav';

accounts.Last\_Name\_\_c = 'Mahajan';

accounts.clcommon\_\_Legal\_Entity\_Type\_\_c = legal.Id;

insert accounts;

List<Account> assertAccount = [SELECT Id

FROM Account

WHERE RecordTypeId != null ];

System.assertEquals(true,assertAccount.size() > 0,'Account Not Inserted');

Id contRecordType = Schema.SObjectType.Contact.getRecordTypeInfosByName().get('Individual Contact').getRecordTypeId();

Id brokerContRecordType = Schema.SObjectType.Contact.getRecordTypeInfosByName().get('Broker Contact').getRecordTypeId();

Contact contact = new Contact();

contact.RecordTypeId = contRecordType;

contact.Salutation = 'Mr.';

contact.FirstName = 'Raghav';

contact.MiddleName = 'Kumar';

contact.LastName = 'Mahajan';

contact.BirthDate = Date.newInstance(1998,12,9);

contact.Email = 'raghav.mahajan@mailinator.com';

contact.clcommon\_\_Marital\_Status\_\_c = 'Married';

contact.Number\_of\_Dependants\_\_c = 2;

contact.clcommon\_\_Legal\_Entity\_Type\_\_c = legal.Id;

insert contact;

List<Contact> assertContact= [SELECT Id

FROM Contact

WHERE RecordTypeId != null ];

System.assertEquals(true,assertContact.size() > 0,'Contact Not Inserted');

// Rate Money Accounts and Contact are for COMP-100 test

Account mateMoneyAggregatorAccount = new Account();

mateMoneyAggregatorAccount.Name = 'Rate Money Pty Ltd';

mateMoneyAggregatorAccount.Type = 'Aggregator';

mateMoneyAggregatorAccount.loan\_\_Broker\_\_c = true;

mateMoneyAggregatorAccount.recordTypeId = accRecordType;

insert mateMoneyAggregatorAccount;

Account mateMoneyBrokerAccount = new Account();

mateMoneyBrokerAccount.Name = 'Rate Money Pty Ltd';

mateMoneyBrokerAccount.Type = 'Broker';

mateMoneyBrokerAccount.loan\_\_Broker\_\_c = true;

mateMoneyBrokerAccount.recordTypeId = accBrokerRecordType;

mateMoneyBrokerAccount.clcommon\_\_Legal\_Entity\_Type\_\_c = legal.Id;

mateMoneyBrokerAccount.Direct\_Accreditation\_\_c = true;

mateMoneyBrokerAccount.Dual\_Accreditation\_\_c = true;

insert mateMoneyBrokerAccount;

Contact mateMoneyBrokerContact = new Contact();

mateMoneyBrokerContact.RecordTypeId = brokerContRecordType;

mateMoneyBrokerContact.FirstName = 'Owen';

mateMoneyBrokerContact.LastName = 'Rees';

mateMoneyBrokerContact.BirthDate = Date.newInstance(1985,10,10);

mateMoneyBrokerContact.Email = 'owen.mule@mailinator.com';

mateMoneyBrokerContact.clcommon\_\_Marital\_Status\_\_c = 'Married';

mateMoneyBrokerContact.Number\_of\_Dependants\_\_c = 2;

mateMoneyBrokerContact.clcommon\_\_Legal\_Entity\_Type\_\_c = legal.Id;

insert mateMoneyBrokerContact;

clcommon\_\_Floating\_Rate\_Index\_\_c floatingRate = new clcommon\_\_Floating\_Rate\_Index\_\_c();

floatingRate.clcommon\_\_Active\_\_c = true;

floatingRate.Name = 'Test';

insert floatingRate;

clcommon\_\_Floating\_Rate\_Index\_\_c floatingRateIndex = new clcommon\_\_Floating\_Rate\_Index\_\_c();

floatingRateIndex.clcommon\_\_Active\_\_c = true;

floatingRateIndex.Name = 'Residential Loan Rate';

insert floatingRateIndex;

List<clcommon\_\_Floating\_Rate\_Index\_\_c> assertFloatRate = [SELECT Id, Name

FROM clcommon\_\_Floating\_Rate\_Index\_\_c ];

System.assertEquals(true, assertFloatRate.size() > 0,'Floating Rate Not Inserted');

clcommon\_\_CL\_Purpose\_\_c purpose = new clcommon\_\_CL\_Purpose\_\_c();

purpose.Name = 'Test Purpose';

purpose.clcommon\_\_Purpose\_Code\_\_c = '1234';

insert purpose;

clcommon\_\_CL\_Purpose\_\_c purposeCL = new clcommon\_\_CL\_Purpose\_\_c();

purposeCL.Name = 'Refinance';

purposeCL.clcommon\_\_Purpose\_Code\_\_c = '12345';

insert purposeCL;

clcommon\_\_CL\_Purpose\_\_c purposeSplit = new clcommon\_\_CL\_Purpose\_\_c();

purposeSplit.Name = 'Purchase';

purposeSplit.clcommon\_\_Purpose\_Code\_\_c = '123456';

insert purposeSplit;

clcommon\_\_CL\_Purpose\_\_c purposeSplitCommon = new clcommon\_\_CL\_Purpose\_\_c();

purposeSplitCommon.Name = 'Equity Out';

purposeSplitCommon.clcommon\_\_Purpose\_Code\_\_c = '1234567';

insert purposeSplitCommon;

clcommon\_\_CL\_Purpose\_\_c purposeSplitsCommon = new clcommon\_\_CL\_Purpose\_\_c();

purposeSplitsCommon.Name = 'Refinance/Equity Out';

purposeSplitsCommon.clcommon\_\_Purpose\_Code\_\_c = '3453';

insert purposeSplitsCommon;

List<clcommon\_\_CL\_Purpose\_\_c> clpur = [SELECT Id, Name

FROM clcommon\_\_CL\_Purpose\_\_c ];

System.assertEquals(true, clpur.size() > 0,'CL purpose Not Inserted');

clcommon\_\_Asset\_Class\_\_c assetClass = new clcommon\_\_Asset\_Class\_\_c(Name = 'Residential Loan');

insert assetClass;

List<clcommon\_\_Asset\_Class\_\_c> assertAssetFlow = [SELECT Id

FROM clcommon\_\_Asset\_Class\_\_c];

System.assertEquals(true,assertAssetFlow.size() > 0,'Asset Class Not Inserted');

clcommon\_\_Asset\_Class\_\_c assetClassCommercial = new clcommon\_\_Asset\_Class\_\_c(Name = 'Commercial Loan');

insert assetClassCommercial;

List<clcommon\_\_Asset\_Class\_\_c> assertassetClassCom = [SELECT Id

FROM clcommon\_\_Asset\_Class\_\_c];

System.assertEquals(true,assertassetClassCom.size() > 0,'Asset Class Not Inserted');

clcommon\_\_Asset\_Class\_\_c assetClassCommercialSMSF = new clcommon\_\_Asset\_Class\_\_c(Name = 'SMSF Commercial Loan');

insert assetClassCommercialSMSF;

List<clcommon\_\_Asset\_Class\_\_c> assertClassCommercialSmsf = [SELECT Id

FROM clcommon\_\_Asset\_Class\_\_c];

System.assertEquals(true,assertClassCommercialSmsf.size() > 0,'Asset Class Not Inserted');

clcommon\_\_Asset\_Class\_\_c assetClassResidentialSMSF = new clcommon\_\_Asset\_Class\_\_c(Name = 'SMSF Residential Loan');

insert assetClassResidentialSMSF;

List<clcommon\_\_Asset\_Class\_\_c> assertassetResidentialSMSF = [SELECT Id

FROM clcommon\_\_Asset\_Class\_\_c];

System.assertEquals(true,assertassetResidentialSMSF.size() > 0,'Asset Class Not Inserted');

Execution\_Flow\_\_c execflow = new Execution\_Flow\_\_c();

execflow.name = 'Residential Loan Flow - Manual';

execflow.Active\_\_c = True;

insert execflow;

Execution\_Flow\_\_c execFlowCommercial = new Execution\_Flow\_\_c();

execFlowCommercial.name = 'Commercial Loan Flow - Manual';

execFlowCommercial.Active\_\_c = True;

insert execFlowCommercial;

Execution\_Flow\_\_c execflows = new Execution\_Flow\_\_c();

execflows.name = 'Residential Loan Flow - Manual';

execflows.Active\_\_c = True;

insert execflows;

Execution\_Flow\_\_c execFlowSimpology = new Execution\_Flow\_\_c();

execFlowSimpology.name = 'Residential Loan Flow - Simpology';

execFlowSimpology.Active\_\_c = True;

insert execFlowSimpology;

clcommon\_\_Fee\_Set\_\_c feeSet = new clcommon\_\_Fee\_Set\_\_c();

feeSet.Name='RateMoney';

insert feeSet;

clcommon\_\_CL\_Product\_\_c product = new clcommon\_\_CL\_Product\_\_c();

product.clcommon\_\_Product\_Name\_\_c= 'Full Doc Residential Master';

product.JWF\_Execution\_Flow\_\_c = execflow.Id;

product.clcommon\_\_Asset\_Class\_\_c = assetClass.Id;

product.Product\_Type\_\_c = 'Residential';

insert product;

List<clcommon\_\_CL\_Product\_\_c> assertProduct = [ SELECT Id

FROM clcommon\_\_CL\_Product\_\_c];

System.assertEquals(true,assertProduct.size() > 0,'Product Not Inserted');

clcommon\_\_CL\_Product\_\_c productCommercial = new clcommon\_\_CL\_Product\_\_c();

productCommercial.clcommon\_\_Product\_Name\_\_c= 'Full Doc Commercial Master';

productCommercial.JWF\_Execution\_Flow\_\_c = execFlowCommercial.Id;

productCommercial.clcommon\_\_Asset\_Class\_\_c = assetClassCommercial.Id;

insert productCommercial;

List<clcommon\_\_CL\_Product\_\_c> assertproductCommercial = [ SELECT Id

FROM clcommon\_\_CL\_Product\_\_c];

System.assertEquals(true,assertproductCommercial.size() > 0,'Product Not Inserted');

clcommon\_\_CL\_Product\_\_c productResidential = new clcommon\_\_CL\_Product\_\_c();

productResidential.clcommon\_\_Product\_Name\_\_c= 'SMSF Residential Master';

productResidential.JWF\_Execution\_Flow\_\_c = execflow.Id;

productResidential.clcommon\_\_Asset\_Class\_\_c = assetClassResidentialSMSF.Id;

productResidential.Product\_Type\_\_c = 'Residential';

insert productResidential;

List<clcommon\_\_CL\_Product\_\_c> assertProductResidential = [ SELECT Id

FROM clcommon\_\_CL\_Product\_\_c];

System.assertEquals(true,assertProductResidential.size() > 0,'Product Not Inserted');

clcommon\_\_CL\_Product\_\_c productMaster = new clcommon\_\_CL\_Product\_\_c();

productMaster.clcommon\_\_Product\_Name\_\_c= 'SMSF Commercial Master';

productMaster.JWF\_Execution\_Flow\_\_c = execflow.Id;

productMaster.clcommon\_\_Asset\_Class\_\_c = assetClassCommercialSMSF.Id;

insert productMaster;

List<clcommon\_\_CL\_Product\_\_c> assertproductMaster = [ SELECT Id

FROM clcommon\_\_CL\_Product\_\_c];

System.assertEquals(true,assertproductMaster.size() > 0,'Product Not Inserted');

clcommon\_\_CL\_Product\_\_c midTicketCommercial = new clcommon\_\_CL\_Product\_\_c();

midTicketCommercial.clcommon\_\_Product\_Name\_\_c= 'Mid Ticket Commercial';

midTicketCommercial.JWF\_Execution\_Flow\_\_c = execFlowCommercial.Id;

midTicketCommercial.clcommon\_\_Asset\_Class\_\_c = assetClassCommercial.Id;

insert midTicketCommercial;

List<clcommon\_\_CL\_Product\_\_c> assertMidTicketCommercial = [ SELECT Id

FROM clcommon\_\_CL\_Product\_\_c

WHERE clcommon\_\_Product\_Name\_\_c = 'Mid Ticket Commercial'];

System.assertEquals(true,assertMidTicketCommercial.size() > 0,'Product Not Inserted');

Id recordTypeIdPackage = Schema.SObjectType.genesis\_\_Applications\_\_c.getRecordTypeInfosByName().get('Package').getRecordTypeId();

genesis\_\_Applications\_\_c packageApp = new genesis\_\_Applications\_\_c();

packageApp.genesis\_\_Status\_\_c = 'NEW-PARTIAL';

packageApp.Thinktank\_Margin\_\_c = 0.25;

packageApp.genesis\_\_Interest\_Index\_\_c = floatingRateIndex.Id;

packageApp.recordtypeid = recordTypeIdPackage;

packageApp.genesis\_\_CL\_Product\_\_c = product.Id;

packageApp.genesis\_\_Loan\_Amount\_\_c = 15000;

packageApp.genesis\_\_CL\_Purpose\_\_c = purpose.Id;

packageApp.Broker\_Trail\_\_c = 0.25;

packageApp.genesis\_\_Requested\_Loan\_Amount\_\_c = 40000;

packageApp.Rate\_Index\_\_c = 'Residential Loan Rate';

packageApp.genesis\_\_CLTV\_\_c = 0.50;

packageApp.Aggregator\_\_c = brokerAggregatorAccount.Id;

packageApp.genesis\_\_Payment\_Amount\_\_c = 100000;

packageApp.P\_I\_Repayment\_Amount\_\_c = 50000;

packageApp.Total\_Fee\_Amount\_inc\_GST\_\_c = 100;

packageApp.Funding\_Date\_\_c = Date.newInstance(2021,11,20);

packageApp.genesis\_\_Disbursement\_Date\_\_c = Date.newInstance(2020,7,15);

packageApp.genesis\_\_Account\_\_c = acc.Id;

packageApp.Borrower\_Name\_\_c = 'Tony';

packageApp.genesis\_\_Total\_Fee\_Amount\_\_c = 1000;

packageApp.Workflow\_Status\_\_c = 'WORKSHOP';

packageApp.Pricing\_Basis\_\_c = 'Variable';

packageApp.genesis\_\_Product\_Type\_\_c = genesis.LendingConstants.LOAN;

packageApp.genesis\_\_Interest\_Rate\_\_c =8.5;

packageApp.genesis\_\_Term\_\_c = 12;

packageApp.IO\_Repayment\_Amount\_\_c = 80000;

packageApp.Execution\_Flow\_\_c = execflows.Id;

packageApp.Fee\_Set\_\_c = feeSet.id;

packageApp.genesis\_\_Total\_Facility\_Amount\_\_c = 134.50;

packageApp.NCCP\_\_c = true;

insert packageApp;

genesis\_\_Applications\_\_c packageApplication = new genesis\_\_Applications\_\_c();

packageApplication.genesis\_\_Status\_\_c = 'NEW-PARTIAL';

packageApplication.Thinktank\_Margin\_\_c = 0.25;

packageApplication.genesis\_\_Interest\_Index\_\_c = floatingRateIndex.Id;

packageApplication.recordtypeid = recordTypeIdPackage;

packageApplication.genesis\_\_CL\_Product\_\_c = product.Id;

packageApplication.genesis\_\_Loan\_Amount\_\_c = 15000;

packageApplication.genesis\_\_CL\_Purpose\_\_c = purpose.Id;

packageApplication.Broker\_Trail\_\_c = 0.25;

packageApplication.genesis\_\_Requested\_Loan\_Amount\_\_c = 40000;

packageApplication.Rate\_Index\_\_c = 'Residential Loan Rate';

packageApplication.genesis\_\_CLTV\_\_c = 0.50;

packageApplication.genesis\_\_Payment\_Amount\_\_c = 100000;

packageApplication.P\_I\_Repayment\_Amount\_\_c = 50000;

packageApplication.Aggregator\_\_c = brokerAggregatorAccount.Id;

packageApplication.Total\_Fee\_Amount\_inc\_GST\_\_c = 100;

packageApplication.Funding\_Date\_\_c = Date.newInstance(2021,11,20);

packageApplication.genesis\_\_Disbursement\_Date\_\_c = Date.newInstance(2020,7,15);

packageApplication.genesis\_\_Account\_\_c = acc.Id;

packageApplication.Borrower\_Name\_\_c = 'Tony';

packageApplication.genesis\_\_Total\_Fee\_Amount\_\_c = 1000;

packageApplication.Workflow\_Status\_\_c = 'WORKSHOP';

packageApplication.Pricing\_Basis\_\_c = 'Variable';

packageApplication.genesis\_\_Product\_Type\_\_c = genesis.LendingConstants.LOAN;

packageApplication.genesis\_\_Interest\_Rate\_\_c =8.5;

packageApplication.genesis\_\_Term\_\_c = 12;

packageApplication.Execution\_Flow\_\_c = execFlowSimpology.Id;

packageApplication.Fee\_Set\_\_c = feeSet.id;

packageApplication.genesis\_\_Total\_Facility\_Amount\_\_c = 134.50;

packageApplication.NCCP\_\_c = true;

packageApplication.IO\_Repayment\_Amount\_\_c = 80000;

packageApplication.Settlement\_Notes\_\_c = 'psuedo unique identifier 1001';

insert packageApplication;

genesis\_\_Rate\_Schedule\_Setup\_\_c rateSchedule = new genesis\_\_Rate\_Schedule\_Setup\_\_c();

rateSchedule.genesis\_\_Margin\_Rate\_\_c = 100;

rateSchedule.genesis\_\_Spread\_\_c = 50;

rateSchedule.genesis\_\_Application\_\_c = packageApp.Id;

insert rateSchedule;

clcommon\_\_Fee\_\_c existingFee = new clcommon\_\_Fee\_\_c();

existingFee.clcommon\_\_Original\_Amount\_\_c = 0.00;

existingFee.genesis\_\_Application\_\_c = packageApp.Id;

existingFee.clcommon\_\_Date\_\_c = System.TODAY();

insert existingFee;

List<clcommon\_\_Fee\_\_c> assertfee = [SELECT Id FROM clcommon\_\_Fee\_\_c];

System.assertEquals(true,assertfee.size() > 0,'Fee Not Inserted');

clcommon\_\_Party\_Type\_\_c partyType = new clcommon\_\_Party\_Type\_\_c();

partyType.Name = 'BORROWER';

insert partyType;

clcommon\_\_Party\_\_c party = new clcommon\_\_Party\_\_c();

party.clcommon\_\_Account\_\_c = acc.Id;

party.clcommon\_\_Active\_\_c = true;

party.genesis\_\_Application\_\_c = packageApp.Id;

party.clcommon\_\_Type\_\_c = partyType.Id;

party.clcommon\_\_Contact\_\_c = contact.Id;

insert party;

clcommon\_\_Party\_\_c partyClcommon = new clcommon\_\_Party\_\_c();

partyClcommon.clcommon\_\_Account\_\_c = acc.Id;

partyClcommon.clcommon\_\_Active\_\_c = true;

partyClcommon.genesis\_\_Application\_\_c = packageApp.Id;

partyClcommon.clcommon\_\_Type\_\_c = partyType.Id;

partyClcommon.clcommon\_\_Contact\_\_c = contact.Id;

insert partyClcommon;

Id recordTypeIdLoan = Schema.SObjectType.genesis\_\_Applications\_\_c.getRecordTypeInfosByName().get('Loan').getRecordTypeId();

genesis\_\_Applications\_\_c splitLoanApp = new genesis\_\_Applications\_\_c();

splitLoanApp.genesis\_\_Status\_\_c = 'NEW - ENTERED';

splitLoanApp.genesis\_\_Account\_\_c = account.Id;

splitLoanApp.Thinktank\_Margin\_\_c = 0.15;

splitLoanApp.recordtypeid = recordTypeIdLoan;

splitLoanApp.genesis\_\_Parent\_Application\_\_c = packageApp.id;

splitLoanApp.Pricing\_Basis\_\_c = 'Fixed';

splitLoanApp.genesis\_\_CL\_Product\_\_c = product.Id;

splitLoanApp.genesis\_\_Loan\_Amount\_\_c = 20000;

splitLoanApp.Rate\_Index\_\_c = 'Residential Loan Rate';

splitLoanApp.genesis\_\_CLTV\_\_c = 0.65;

splitLoanApp.genesis\_\_Interest\_Index\_\_c = floatingRateIndex.Id;

splitLoanApp.Workflow\_Status\_\_c = 'PRELIMINARY REVIEW';

splitLoanApp.genesis\_\_Product\_Type\_\_c = genesis.LendingConstants.LOAN;

splitLoanApp.genesis\_\_CL\_Product\_\_c = product.Id;

splitLoanApp.genesis\_\_CL\_Purpose\_\_c = purposeSplit.Id;

splitLoanApp.genesis\_\_Payment\_Amount\_\_c = 100000;

splitLoanApp.P\_I\_Repayment\_Amount\_\_c = 50000;

splitLoanApp.Broker\_Trail\_\_c = 0.35;

splitLoanApp.genesis\_\_Requested\_Loan\_Amount\_\_c = 30000;

splitLoanApp.Total\_Fee\_Amount\_inc\_GST\_\_c = 120;

splitLoanApp.Funding\_Date\_\_c = Date.newInstance(2021,11,10);

splitLoanApp.genesis\_\_Disbursement\_Date\_\_c = Date.newInstance(2019,11,20);

splitLoanApp.genesis\_\_Total\_Fee\_Amount\_\_c = 1200;

splitLoanApp.genesis\_\_Product\_Type\_\_c = genesis.LendingConstants.LOAN;

splitLoanApp.genesis\_\_Interest\_Rate\_\_c =8.5;

splitLoanApp.genesis\_\_Term\_\_c = 48;

splitLoanApp.Execution\_Flow\_\_c = execflow.Id;

splitLoanApp.Fee\_Set\_\_c = feeSet.id;

splitLoanApp.genesis\_\_Total\_Facility\_Amount\_\_c = 130.50;

splitLoanApp.IO\_Repayment\_Amount\_\_c = 80000;

splitLoanApp.Payment\_Type\_\_c = 'Interest Only';

splitLoanApp.Construction\_Term\_\_c = null;

insert splitLoanApp;

genesis\_\_Rate\_Schedule\_Setup\_\_c rateScheduleSplit = new genesis\_\_Rate\_Schedule\_Setup\_\_c();

rateScheduleSplit.genesis\_\_Margin\_Rate\_\_c = 120;

rateScheduleSplit.genesis\_\_Spread\_\_c = 70;

rateScheduleSplit.genesis\_\_Application\_\_c = splitLoanApp.Id;

insert rateScheduleSplit;

clcommon\_\_Collateral\_\_c col = new clcommon\_\_Collateral\_\_c();

col.clcommon\_\_Collateral\_Name\_\_c = 'Test collateral';

col.Street\_Address\_\_c = 'tetststtk yududi';

col.clcommon\_\_City\_\_c = 'Melborne';

col.clcommon\_\_State\_\_c = 'TAS';

col.clcommon\_\_Postal\_Code\_\_c ='748839';

col.clcommon\_\_Purchased\_Price\_\_c = 15000;

col.Current\_Value\_\_c = 24000;

col.Valuation\_Type\_\_c = 'AVM';

insert col;

genesis\_\_Application\_Collateral\_\_c appCol = new genesis\_\_Application\_Collateral\_\_c();

appCol.genesis\_\_Application\_\_c = packageApplication.id;

appCol.genesis\_\_Collateral\_\_c = col.id;

appCol.isPrimary\_\_c = true;

appCol.is\_Security\_\_c=true;

insert appCol;

genesis\_\_Application\_Collateral\_\_c appCollateral = new genesis\_\_Application\_Collateral\_\_c();

appCollateral.genesis\_\_Application\_\_c = packageApp.id;

appCollateral.genesis\_\_Collateral\_\_c = col.id;

appCollateral.isPrimary\_\_c = true;

appCollateral.is\_Security\_\_c=true;

insert appCollateral;

clcommon\_\_Document\_Definition\_\_c docDef = new clcommon\_\_Document\_Definition\_\_c();

docDef.clcommon\_\_Description\_\_c = 'ABC';

insert docDef;

clcommon\_\_Document\_Category\_\_c docCategory = new clcommon\_\_Document\_Category\_\_c();

docCategory.clcommon\_\_Category\_Name\_\_c = 'Test Category 1';

docCategory.genesis\_\_Application\_\_c = packageApp.Id;

docCategory.clcommon\_\_Required\_\_c = true;

docCategory.clcommon\_\_Document\_Definition\_\_c = docDef.id;

insert docCategory;

clcommon\_\_Document\_Category\_\_c docCategoryClcommon = new clcommon\_\_Document\_Category\_\_c();

docCategoryClcommon.clcommon\_\_Category\_Name\_\_c = 'Test Category 1';

docCategoryClcommon.genesis\_\_Application\_\_c = packageApplication.Id;

docCategoryClcommon.clcommon\_\_Required\_\_c = true;

docCategoryClcommon.clcommon\_\_Document\_Definition\_\_c = docDef.id;

insert docCategoryClcommon;

Document\_Checklist\_\_c document = new Document\_Checklist\_\_c();

document.Status\_\_c = 'OPEN';

document.Active\_\_c = true;

document.Application\_\_c = packageApp.Id;

insert document;

Document\_Checklist\_\_c documentChecklist = new Document\_Checklist\_\_c();

documentChecklist.Status\_\_c = 'OPEN';

documentChecklist.Active\_\_c = true;

documentChecklist.Application\_\_c = packageApplication.Id;

insert documentChecklist;

List<Document\_Checklist\_\_c> assertdocument = [ SELECT Id FROM Document\_Checklist\_\_c];

System.assertEquals(true, !assertdocument.isEmpty(), 'Document Checklist not inserted');

}

@isTest

static void updateSplitStatusOnPackageChangeTest() {

List<genesis\_\_Applications\_\_c> oldApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

Consolidated\_Purpose\_\_c,

genesis\_\_Parent\_Application\_\_c,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

List<Contact> assertContact= [SELECT Id,Email

FROM Contact

WHERE RecordTypeId != null ];

System.assertEquals(true,assertContact.size() > 0,'Contact Not Inserted');

List<genesis\_\_Application\_Collateral\_\_c> appCol = [SELECT Id, genesis\_\_Application\_\_c FROM genesis\_\_Application\_Collateral\_\_c];

appCol[0].genesis\_\_Application\_\_c = oldApplist[0].Id;

update appCol;

Group queue = [SELECT Id, Name FROM Group WHERE Type = 'Queue' AND Name = 'Sales'];

oldApplist[0].OwnerId = queue.Id;

System.assertEquals(true, acc.size() > 0,'Account Not Inserted');

Group queueType = [SELECT Id, Name FROM Group WHERE Type = 'Queue' AND Name = 'Credit'];

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(oldApplist[0].Id, oldApplist[0]);

List<genesis\_\_Applications\_\_c> newApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

createddate,

Consolidated\_Purpose\_\_c,

genesis\_\_Parent\_Application\_\_c,

Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

newApplist[0].genesis\_\_Status\_\_c = 'NEW - ENTERED';

newApplist[0].genesis\_\_Interest\_Rate\_\_c = 9.8;

newApplist[0].genesis\_\_Term\_\_c = 24;

newApplist[0].genesis\_\_Loan\_Amount\_\_c = 18000;

newApplist[0].genesis\_\_Total\_Fee\_Amount\_\_c = 2000;

newApplist[0].Workflow\_Status\_\_c = 'CONDITIONAL APPROVAL';

newApplist[0].Thinktank\_Margin\_\_c = 0.41;

newApplist[0].genesis\_\_CLTV\_\_c = 0.70;

newApplist[0].Funding\_Date\_\_c = Date.newInstance(2021,11,24);

newApplist[0].Rate\_Index\_\_c = 'Residential SMSF Rate';

newApplist[0].Broker\_Trail\_\_c = 0.67;

newApplist[0].genesis\_\_Requested\_Loan\_Amount\_\_c = 10000;

newApplist[0].genesis\_\_Disbursement\_Date\_\_c = Date.newInstance(2018,05,13);

newApplist[0].genesis\_\_Total\_Facility\_Amount\_\_c = 2002;

newApplist[0].Aggregator\_\_c = acc[0].Id;

newApplist[0].OwnerId = queueType.Id;

update newApplist;

genesis\_\_Department\_\_c departmentGenesis = new genesis\_\_Department\_\_c();

departmentGenesis.Name = TTConstants.CREDIT\_UNDERWRITING\_DEPT;

insert departmentGenesis;

List<genesis\_\_Department\_\_c> assertdeparts = [ SELECT Id FROM genesis\_\_Department\_\_c];

System.assertEquals(true, !assertdeparts.isEmpty(), 'Department2 not inserted');

genesis\_\_Application\_Department\_\_c applicationDepartment = new genesis\_\_Application\_Department\_\_c();

applicationDepartment.genesis\_\_Department\_\_c = departmentGenesis.Id;

applicationDepartment.genesis\_\_Application\_\_c = newApplist[0].Id;

applicationDepartment.genesis\_\_Status\_\_c = TTConstants.ACTIVE;

insert applicationDepartment;

List< genesis\_\_Application\_Department\_\_c> assertDept = [ SELECT Id ,genesis\_\_Department\_\_c FROM genesis\_\_Application\_Department\_\_c];

System.assertEquals(true, !assertDept.isEmpty(), 'Application Department not inserted');

List<Task> taskList = new List<Task>();

Task tasks = new Task();

tasks.genesis\_\_Application\_\_c = newApplist[0].Id;

tasks.genesis\_\_Department\_\_c = assertDept[0].genesis\_\_Department\_\_c;

tasks.Status = 'Open';

tasks.Subject = 'Urgent Task-Increase Loan amount for Further Advance';

taskList.add(tasks);

insert taskList;

newMap.put(newApplist[0].Id, newApplist[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

newApplist,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApplist, oldMap, 'AFTER\_UPDATE');

// obj.afterUpdate(newApplist, oldMap);

}

Test.stopTest();

//System.assertEquals(true,!Application.isEmpty(),'App Not Found');

}

@isTest

static void updateSplitOnPackageChangeTest() {

List<genesis\_\_Applications\_\_c> oldApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Parent\_Rate\_Index\_\_c,

genesis\_\_Parent\_Application\_\_c,

Payment\_Type\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

P\_I\_Repayment\_Amount\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

IO\_Repayment\_Amount\_\_c,

Construction\_Term\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

genesis\_\_Payment\_Amount\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Loan'];

List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

System.assertEquals(true, acc.size() > 0,'Account Not Inserted');

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(oldApplist[0].Id, oldApplist[0]);

List<genesis\_\_Applications\_\_c> newApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Parent\_Rate\_Index\_\_c,

genesis\_\_Parent\_Application\_\_c,

Payment\_Type\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

P\_I\_Repayment\_Amount\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Payment\_Amount\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

IO\_Repayment\_Amount\_\_c,

Construction\_Term\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

createddate ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Loan'];

List<clcommon\_\_CL\_Purpose\_\_c> purposeSplit = [SELECT Id FROM clcommon\_\_CL\_Purpose\_\_c WHERE Name = 'Equity Out'];

newApplist[0].genesis\_\_Status\_\_c = 'CHANGE MEMO APPROVAL';

newApplist[0].genesis\_\_Interest\_Rate\_\_c = 9.8;

newApplist[0].genesis\_\_Term\_\_c = 24;

newApplist[0].genesis\_\_Loan\_Amount\_\_c = 18000;

newApplist[0].genesis\_\_Total\_Fee\_Amount\_\_c = 2000;

newApplist[0].Workflow\_Status\_\_c = 'CONDITIONAL APPROVAL';

newApplist[0].Thinktank\_Margin\_\_c = 0.40;

newApplist[0].genesis\_\_CLTV\_\_c = 0.70;

newApplist[0].Funding\_Date\_\_c = Date.newInstance(2021,11,24);

newApplist[0].Rate\_Index\_\_c = 'Residential Loan Rate';

newApplist[0].Broker\_Trail\_\_c = 0.67;

newApplist[0].genesis\_\_Requested\_Loan\_Amount\_\_c = 10000;

newApplist[0].genesis\_\_Disbursement\_Date\_\_c = Date.newInstance(2018,05,13);

newApplist[0].genesis\_\_Total\_Facility\_Amount\_\_c = 2002;

newApplist[0].Aggregator\_\_c = acc[0].Id;

newApplist[0].genesis\_\_CL\_Purpose\_\_c = purposeSplit[0].Id;

update newApplist;

newMap.put(newApplist[0].Id, newApplist[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

newApplist,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApplist, oldMap, 'AFTER\_UPDATE');

// obj.afterUpdate(newApplist, oldMap);

}

Test.stopTest();

}

@isTest

static void updateStatusOnPackageChangeTest() {

List<genesis\_\_Applications\_\_c> oldApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Parent\_Rate\_Index\_\_c,

genesis\_\_Parent\_Application\_\_c,

Payment\_Type\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

P\_I\_Repayment\_Amount\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

genesis\_\_Payment\_Amount\_\_c,

Rate\_Index\_\_c,

IO\_Repayment\_Amount\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Loan'];

System.assertEquals(true, oldAppList.size() > 0,'Application Not Inserted');

List<genesis\_\_Application\_Collateral\_\_c> appCol = [SELECT Id, genesis\_\_Application\_\_c FROM genesis\_\_Application\_Collateral\_\_c];

appCol[0].genesis\_\_Application\_\_c = oldApplist[0].genesis\_\_Parent\_Application\_\_c;

update appCol;

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(oldApplist[0].Id, oldApplist[0]);

List<genesis\_\_Applications\_\_c> newApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Parent\_Rate\_Index\_\_c,

genesis\_\_Parent\_Application\_\_c,

Payment\_Type\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

P\_I\_Repayment\_Amount\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

IO\_Repayment\_Amount\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Payment\_Amount\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

createddate ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Loan'];

newApplist[0].genesis\_\_Status\_\_c = 'CHANGE MEMO APPROVAL';

update newApplist;

newMap.put(newApplist[0].Id, newApplist[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

newApplist,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApplist, oldMap, 'AFTER\_UPDATE');

// obj.afterUpdate(newApplist, oldMap);

}

Test.stopTest();

}

@isTest

static void genesisAppTriggerHandlerTest(){

List<genesis\_\_Applications\_\_c> app = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

genesis\_\_Payment\_Amount\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

Execution\_Flow\_\_r.Name,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

System.assertEquals(true, app.size() > 0,'App Not Inserted');

Id accRecordType = Schema.SObjectType.Account.getRecordTypeInfosByName().get('Aggregator').getRecordTypeId();

Account newAcc = new Account();

newAcc.Name = 'Sanjeev';

newAcc.Type = 'Aggregator';

newAcc.Salutation\_\_c = 'Mr.';

newAcc.First\_Name\_\_c = 'Sanjeev ';

newAcc.Last\_Name\_\_c = 'Kumar';

newAcc.clcommon\_\_Email\_\_c = 'sk@mailinator.com';

newAcc.recordTypeId = accRecordType;

insert newAcc;

List<Account> acc = [SELECT Id, Name FROM Account

WHERE Name = 'Sanjeev'];

System.assertEquals(true, acc.size() > 0,'Account Not Inserted');

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(app[0].Id, app[0]);

app[0].Aggregator\_\_c = acc[0].Id;

app[0].Funding\_Date\_\_c = Date.newInstance(2019,11,24);

app[0].Borrower\_Name\_\_c = 'Bishal';

app[0].genesis\_\_CLTV\_\_c = 0.01;

update app;

newMap.put(app[0].Id,app[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

app,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(app, oldMap, 'AFTER\_UPDATE');

// obj.afterUpdate(app, oldMap);

}

Test.stopTest();

}

@isTest

static void genesisAppliTriggerHandlerTest(){

List<genesis\_\_Applications\_\_c> app = [SELECT Id,

OwnerId,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Execution\_Flow\_\_r.Name,

LTVP\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

Consolidated\_Purpose\_\_c,

genesis\_\_Parent\_Application\_\_c,

Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

System.assertEquals(true, app.size() > 0,'App Not Inserted');

List<clcommon\_\_CL\_Product\_\_c> prod = [SELECT Id FROM clcommon\_\_CL\_Product\_\_c WHERE clcommon\_\_Product\_Name\_\_c= 'Full Doc Commercial Master'];

System.assertEquals(true, prod.size() > 0,'Prod Not Fethced');

Id accRecordType = Schema.SObjectType.Account.getRecordTypeInfosByName().get('Aggregator').getRecordTypeId();

Account newAcc = new Account();

newAcc.Name = 'Sanjeev';

newAcc.Type = 'Aggregator';

newAcc.Salutation\_\_c = 'Mr.';

newAcc.First\_Name\_\_c = 'Sanjeev ';

newAcc.Last\_Name\_\_c = 'Kumar';

newAcc.clcommon\_\_Email\_\_c = 'sk@mailinator.com';

newAcc.recordTypeId = accRecordType;

insert newAcc;

List<Account> acc = [SELECT Id, Name FROM Account

WHERE Name = 'Sanjeev'];

System.assertEquals(true, acc.size() > 0,'Account Not Inserted');

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(app[0].Id, app[0]);

app[0].Aggregator\_\_c = acc[0].Id;

app[0].genesis\_\_Status\_\_c = 'TREASURY SKIP';

app[0].genesis\_\_CL\_Product\_\_c = prod[0].Id;

update app;

List<genesis\_\_Applications\_\_c> newApp = [SELECT Id,

OwnerId,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Execution\_Flow\_\_r.Name,

LTVP\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

createddate,

Consolidated\_Purpose\_\_c,

genesis\_\_Parent\_Application\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

newMap.put(newApp[0].Id,newApp[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

app,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(app, oldMap, 'AFTER\_UPDATE');

// obj.afterUpdate(app, oldMap);

}

Test.stopTest();

}

@isTest

static void genesisApplicTriggerHandlerTest(){

List<genesis\_\_Applications\_\_c> app = [SELECT Id,

OwnerId,

Execution\_Flow\_\_r.Name,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

LTVP\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

createddate,

Consolidated\_Purpose\_\_c,

genesis\_\_Parent\_Application\_\_c,

Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

System.assertEquals(true, app.size() > 0,'App Not Inserted');

List<clcommon\_\_CL\_Product\_\_c> prod = [SELECT Id FROM clcommon\_\_CL\_Product\_\_c WHERE clcommon\_\_Product\_Name\_\_c= 'SMSF Residential Master'];

System.assertEquals(true, prod.size() > 0,'Prod Not Fethced');

Id accRecordType = Schema.SObjectType.Account.getRecordTypeInfosByName().get('Aggregator').getRecordTypeId();

Account newAcc = new Account();

newAcc.Name = 'Sanjeev';

newAcc.Type = 'Aggregator';

newAcc.Salutation\_\_c = 'Mr.';

newAcc.First\_Name\_\_c = 'Sanjeev ';

newAcc.Last\_Name\_\_c = 'Kumar';

newAcc.clcommon\_\_Email\_\_c = 'sk@mailinator.com';

newAcc.recordTypeId = accRecordType;

insert newAcc;

List<Account> acc = [SELECT Id, Name FROM Account

WHERE Name = 'Sanjeev'];

System.assertEquals(true, acc.size() > 0,'Account Not Inserted');

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(app[0].Id, app[0]);

app[0].Aggregator\_\_c = acc[0].Id;

app[0].genesis\_\_Status\_\_c = 'TREASURY SKIP';

app[0].genesis\_\_CL\_Product\_\_c = prod[0].Id;

update app;

newMap.put(app[0].Id,app[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

app,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(app, oldMap, 'AFTER\_UPDATE');

// obj.afterUpdate(app, oldMap);

}

Test.stopTest();

}

@isTest

static void runAppAsQueueOwner(){

List<genesis\_\_Applications\_\_c> app = [SELECT Id,

Record\_Type\_Name\_\_c,

genesis\_\_Status\_\_c,

Workflow\_Status\_\_c,

OwnerId,

Is\_Simpology\_Application\_\_c,

genesis\_\_Parent\_Application\_\_c,

genesis\_\_CL\_Product\_\_c,

Aggregator\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

Set<Id> appIds = new Set<Id>();

appIds.add(app[0].Id);

Group queue = [SELECT Id, Name FROM Group WHERE Type = 'Queue' AND Name = 'Credit Underwriter - Thinktank'];

System.assertEquals(true,queue.Id != null, 'Queue not found');

app[0].genesis\_\_Status\_\_c = 'NEW - ENTERED';

update app;

oldMap.put(app[0].Id, app[0]);

List<clcommon\_\_CL\_Product\_\_c> clProduct = [SELECT Id FROM clcommon\_\_CL\_Product\_\_c WHERE clcommon\_\_Product\_Name\_\_c = 'SMSF Residential Master'];

List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

List<genesis\_\_Applications\_\_c> newApp = [SELECT Id,Record\_Type\_Name\_\_c,genesis\_\_Status\_\_c,Workflow\_Status\_\_c,OwnerId,Is\_Simpology\_Application\_\_c,genesis\_\_Parent\_Application\_\_c,genesis\_\_CL\_Product\_\_c,Aggregator\_\_c,genesis\_\_Total\_Facility\_Amount\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

newApp[0].OwnerId = queue.Id;

newApp[0].genesis\_\_Status\_\_c = 'NEW - ENTERED';

newApp[0].genesis\_\_CL\_Product\_\_c = clProduct[0].Id;

newApp[0].Aggregator\_\_c = acc[0].Id;

update newApp;

newMap.put(newApp[0].Id, newApp[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

true, // isBefore

false, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

newApp,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApp, oldMap, 'BEFORE\_UPDATE');

// obj.beforeUpdate(newApp, oldMap);

}

Test.stopTest();

}

@isTest

static void runAppAsQueueOwnerTest(){

List<genesis\_\_Applications\_\_c> app = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Parent\_Rate\_Index\_\_c,

genesis\_\_Parent\_Application\_\_c,

Payment\_Type\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

Set<Id> appIds = new Set<Id>();

appIds.add(app[0].Id);

Group queue = [SELECT Id, Name FROM Group WHERE Type = 'Queue' AND Name = 'Credit Underwriter - Thinktank'];

System.assertEquals(true,queue.Id != null, 'Queue not found');

app[0].genesis\_\_Status\_\_c = 'NEW - ENTERED';

update app;

oldMap.put(app[0].Id, app[0]);

List<clcommon\_\_CL\_Product\_\_c> clProduct = [SELECT Id FROM clcommon\_\_CL\_Product\_\_c WHERE clcommon\_\_Product\_Name\_\_c = 'SMSF Residential Master'];

List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

List<genesis\_\_Applications\_\_c> newApp = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Parent\_Rate\_Index\_\_c,

genesis\_\_Parent\_Application\_\_c,

Payment\_Type\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

newApp[0].OwnerId = queue.Id;

newApp[0].genesis\_\_Status\_\_c = 'NEW - ENTERED';

newApp[0].genesis\_\_CL\_Product\_\_c = clProduct[0].Id;

newApp[0].Aggregator\_\_c = acc[0].Id;

update newApp;

newMap.put(newApp[0].Id, newApp[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

true, // isBefore

false, // isAfter

true, // isInsert

false, // isUpdate

false, // isDelete

newApp,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApp, oldMap, 'BEFORE\_INSERT');

// obj.beforeInsert(newApp);

}

Test.stopTest();

}

@isTest

static void runBrandMapping(){

List<genesis\_\_Applications\_\_c> app = [SELECT Id,

Record\_Type\_Name\_\_c,

genesis\_\_Status\_\_c,

Workflow\_Status\_\_c,

OwnerId,

Is\_Simpology\_Application\_\_c,

genesis\_\_Parent\_Application\_\_c,

genesis\_\_CL\_Product\_\_c,

Aggregator\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

System.assertEquals(true, app.size() > 0,'Application Not Inserted');

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(app[0].Id, app[0]);

List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

List<clcommon\_\_CL\_Product\_\_c> clProduct = [SELECT Id FROM clcommon\_\_CL\_Product\_\_c WHERE clcommon\_\_Product\_Name\_\_c = 'Full Doc Commercial Master'];

List<genesis\_\_Applications\_\_c> newApp = [SELECT Id,

Record\_Type\_Name\_\_c,

genesis\_\_Status\_\_c,

Workflow\_Status\_\_c,

OwnerId,

Is\_Simpology\_Application\_\_c,

genesis\_\_Parent\_Application\_\_c,

genesis\_\_CL\_Product\_\_c,

Aggregator\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

newApp[0].genesis\_\_CL\_Product\_\_c = clProduct[0].Id;

newApp[0].Aggregator\_\_c = acc[0].Id;

update newApp;

newMap.put(newApp[0].Id, newApp[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

true, // isBefore

false, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

newApp,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApp, oldMap, 'BEFORE\_UPDATE');

// obj.beforeUpdate(newApp, oldMap);

}

Test.stopTest();

}

@isTest

static void updateSplitStatusOnPackageChangeTestUpd() {

List<genesis\_\_Applications\_\_c> oldApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Parent\_Rate\_Index\_\_c,

genesis\_\_Parent\_Application\_\_c,

Payment\_Type\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Loan'];

List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

System.assertEquals(true, acc.size() > 0,'Account Not Inserted');

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(oldApplist[0].Id, oldApplist[0]);

List<genesis\_\_Applications\_\_c> newApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Parent\_Rate\_Index\_\_c,

genesis\_\_Parent\_Application\_\_c,

Payment\_Type\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

createddate ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Loan'];

List<clcommon\_\_CL\_Purpose\_\_c> purposeSplit = [SELECT Id FROM clcommon\_\_CL\_Purpose\_\_c WHERE Name = 'Equity Out'];

newApplist[0].genesis\_\_Status\_\_c = 'CHANGE MEMO APPROVAL';

newApplist[0].genesis\_\_Interest\_Rate\_\_c = 9.8;

newApplist[0].genesis\_\_Term\_\_c = 24;

newApplist[0].genesis\_\_Loan\_Amount\_\_c = 18000;

newApplist[0].genesis\_\_Total\_Fee\_Amount\_\_c = 2000;

newApplist[0].Workflow\_Status\_\_c = 'CONDITIONAL APPROVAL';

newApplist[0].Thinktank\_Margin\_\_c = 0.40;

newApplist[0].genesis\_\_CLTV\_\_c = 0.70;

newApplist[0].Funding\_Date\_\_c = Date.newInstance(2021,11,24);

newApplist[0].Rate\_Index\_\_c = 'Residential Loan Rate';

newApplist[0].Broker\_Trail\_\_c = 0.67;

newApplist[0].genesis\_\_Requested\_Loan\_Amount\_\_c = 10000;

newApplist[0].genesis\_\_Disbursement\_Date\_\_c = Date.newInstance(2018,05,13);

newApplist[0].genesis\_\_Total\_Facility\_Amount\_\_c = 3002;

newApplist[0].Aggregator\_\_c = acc[0].Id;

newApplist[0].genesis\_\_CL\_Purpose\_\_c = purposeSplit[0].Id;

update newApplist;

newMap.put(newApplist[0].Id, newApplist[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

true, // isBefore

false, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

newApplist,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApplist, oldMap, 'BEFORE\_UPDATE');

// obj.beforeUpdate(newApplist, oldMap);

}

Test.stopTest();

}

@isTest

static void updateSplitStatusOnPackageChangeTestUpdate() {

List<genesis\_\_Applications\_\_c> oldApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Parent\_Rate\_Index\_\_c,

genesis\_\_Parent\_Application\_\_c,

Payment\_Type\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Loan'];

List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

System.assertEquals(true, acc.size() > 0,'Account Not Inserted');

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(oldApplist[0].Id, oldApplist[0]);

List<genesis\_\_Applications\_\_c> newApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Parent\_Rate\_Index\_\_c,

genesis\_\_Parent\_Application\_\_c,

Payment\_Type\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

createddate ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Loan'];

List<clcommon\_\_CL\_Purpose\_\_c> purposeSplit = [SELECT Id FROM clcommon\_\_CL\_Purpose\_\_c WHERE Name = 'Equity Out'];

newApplist[0].genesis\_\_Status\_\_c = 'CHANGE MEMO APPROVAL';

newApplist[0].genesis\_\_Interest\_Rate\_\_c = 9.8;

newApplist[0].genesis\_\_Term\_\_c = 24;

newApplist[0].genesis\_\_Loan\_Amount\_\_c = 18000;

newApplist[0].genesis\_\_Total\_Fee\_Amount\_\_c = 2000;

newApplist[0].Workflow\_Status\_\_c = 'CONDITIONAL APPROVAL';

newApplist[0].Thinktank\_Margin\_\_c = 0.40;

newApplist[0].genesis\_\_CLTV\_\_c = 0.70;

newApplist[0].Funding\_Date\_\_c = Date.newInstance(2021,11,24);

newApplist[0].Rate\_Index\_\_c = 'Residential Loan Rate';

newApplist[0].Broker\_Trail\_\_c = 0.67;

newApplist[0].genesis\_\_Requested\_Loan\_Amount\_\_c = 10000;

newApplist[0].genesis\_\_Disbursement\_Date\_\_c = Date.newInstance(2018,05,13);

newApplist[0].genesis\_\_Total\_Facility\_Amount\_\_c = 3002;

newApplist[0].Aggregator\_\_c = acc[0].Id;

newApplist[0].genesis\_\_CL\_Purpose\_\_c = purposeSplit[0].Id;

update newApplist;

newMap.put(newApplist[0].Id, newApplist[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

true, // isBefore

false, // isAfter

true, // isInsert

false, // isUpdate

false, // isDelete

newApplist,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApplist, oldMap, 'BEFORE\_INSERT');

// obj.beforeInsert(newApplist);

}

Test.stopTest();

}

@isTest

static void updateSpStatusOnPackageChangeTestUpdate() {

List<genesis\_\_Applications\_\_c> oldApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Parent\_Rate\_Index\_\_c,

genesis\_\_Parent\_Application\_\_c,

Payment\_Type\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Loan'];

List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

System.assertEquals(true, acc.size() > 0,'Account Not Inserted');

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(oldApplist[0].Id, oldApplist[0]);

List<genesis\_\_Applications\_\_c> newApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Parent\_Rate\_Index\_\_c,

genesis\_\_Parent\_Application\_\_c,

Payment\_Type\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

genesis\_\_CL\_Product\_Name\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

createddate ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Loan'];

List<clcommon\_\_CL\_Purpose\_\_c> purposeSplit = [SELECT Id FROM clcommon\_\_CL\_Purpose\_\_c WHERE Name = 'Equity Out'];

newApplist[0].genesis\_\_Status\_\_c = 'CHANGE MEMO APPROVAL';

newApplist[0].genesis\_\_Interest\_Rate\_\_c = 9.8;

newApplist[0].genesis\_\_Term\_\_c = 24;

newApplist[0].genesis\_\_Loan\_Amount\_\_c = 18000;

newApplist[0].genesis\_\_Total\_Fee\_Amount\_\_c = 2000;

newApplist[0].Workflow\_Status\_\_c = 'CONDITIONAL APPROVAL';

newApplist[0].Thinktank\_Margin\_\_c = 0.40;

newApplist[0].genesis\_\_CLTV\_\_c = 0.70;

newApplist[0].Funding\_Date\_\_c = Date.newInstance(2021,11,24);

newApplist[0].Rate\_Index\_\_c = 'Residential Loan Rate';

newApplist[0].Broker\_Trail\_\_c = 0.67;

newApplist[0].genesis\_\_Requested\_Loan\_Amount\_\_c = 10000;

newApplist[0].genesis\_\_Disbursement\_Date\_\_c = Date.newInstance(2018,05,13);

newApplist[0].genesis\_\_Total\_Facility\_Amount\_\_c = 3002;

newApplist[0].Aggregator\_\_c = acc[0].Id;

newApplist[0].genesis\_\_CL\_Purpose\_\_c = purposeSplit[0].Id;

update newApplist;

newMap.put(newApplist[0].Id, newApplist[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

true, // isInsert

false, // isUpdate

false, // isDelete

newApplist,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApplist, oldMap, 'AFTER\_INSERT');

// obj.afterInsert(newApplist);

}

Test.stopTest();

}

@isTest

static void midTicketApplicationTest() {

List<genesis\_\_Applications\_\_c> oldApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Parent\_Rate\_Index\_\_c,

genesis\_\_Parent\_Application\_\_c,

Payment\_Type\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Loan'];

List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

System.assertEquals(true, acc.size() > 0,'Account Not Inserted');

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(oldApplist[0].Id, oldApplist[0]);

List<genesis\_\_Applications\_\_c> newApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Parent\_Rate\_Index\_\_c,

genesis\_\_Parent\_Application\_\_c,

Payment\_Type\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

genesis\_\_CL\_Product\_Name\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

createddate ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Loan'];

List<clcommon\_\_CL\_Purpose\_\_c> purposeSplit = [SELECT Id FROM clcommon\_\_CL\_Purpose\_\_c WHERE Name = 'Equity Out'];

List<clcommon\_\_CL\_Product\_\_c> assertMidTicketCommercial = [ SELECT Id

FROM clcommon\_\_CL\_Product\_\_c

WHERE clcommon\_\_Product\_Name\_\_c = 'Mid Ticket Commercial'];

System.assertEquals(true,assertMidTicketCommercial.size() > 0,'Product Not Inserted');

newApplist[0].genesis\_\_Status\_\_c = 'CHANGE MEMO APPROVAL';

newApplist[0].genesis\_\_Interest\_Rate\_\_c = 9.8;

newApplist[0].genesis\_\_Term\_\_c = 24;

newApplist[0].genesis\_\_Loan\_Amount\_\_c = 18000;

newApplist[0].genesis\_\_Total\_Fee\_Amount\_\_c = 2000;

newApplist[0].Workflow\_Status\_\_c = 'CONDITIONAL APPROVAL';

newApplist[0].Thinktank\_Margin\_\_c = 0.40;

newApplist[0].genesis\_\_CLTV\_\_c = 0.70;

newApplist[0].Funding\_Date\_\_c = Date.newInstance(2021,11,24);

newApplist[0].Rate\_Index\_\_c = 'Residential Loan Rate';

newApplist[0].Broker\_Trail\_\_c = 0.67;

newApplist[0].genesis\_\_Requested\_Loan\_Amount\_\_c = 10000;

newApplist[0].genesis\_\_Disbursement\_Date\_\_c = Date.newInstance(2018,05,13);

newApplist[0].genesis\_\_Total\_Facility\_Amount\_\_c = 3002;

newApplist[0].Aggregator\_\_c = acc[0].Id;

newApplist[0].genesis\_\_CL\_Purpose\_\_c = purposeSplit[0].Id;

newApplist[0].genesis\_\_CL\_Product\_\_c = assertMidTicketCommercial[0].Id;

update newApplist;

newMap.put(newApplist[0].Id, newApplist[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

true, // isInsert

false, // isUpdate

false, // isDelete

newApplist,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApplist, oldMap, 'AFTER\_INSERT');

// obj.afterInsert(newApplist);

}

Test.stopTest();

}

@isTest

static void genesisApplicationSplitTriggerHandlerTest() {

List<genesis\_\_Applications\_\_c> oldApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Execution\_Flow\_\_r.Name,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

genesis\_\_Payment\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name = 'Residential Loan Flow - Simpology'];

List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

List<genesis\_\_Application\_Collateral\_\_c> appCol = [SELECT Id, genesis\_\_Application\_\_c FROM genesis\_\_Application\_Collateral\_\_c];

appCol[0].genesis\_\_Application\_\_c = oldApplist[0].Id;

update appCol;

System.assertEquals(true, acc.size() > 0,'Account Not Inserted');

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(oldApplist[0].Id, oldApplist[0]);

Contact dualAccredBrokerCon = [

SELECT Id

FROM Contact

WHERE Name = 'Owen Rees'

LIMIT 1

];

List<genesis\_\_Applications\_\_c> newApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Execution\_Flow\_\_r.Name,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

genesis\_\_Payment\_Amount\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

Broker\_Contact\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name = 'Residential Loan Flow - Simpology'];

newApplist[0].genesis\_\_Status\_\_c = 'NEW - ENTERED';

newApplist[0].Broker\_Contact\_\_c = dualAccredBrokerCon.Id;

update newApplist;

newMap.put(newApplist[0].Id, newApplist[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

newApplist,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApplist, oldMap, 'AFTER\_UPDATE');

// obj.afterUpdate(newApplist, oldMap);

}

Test.stopTest();

}

@isTest

static void updateSplitStatusOnPackageChangeTestCheck() {

List<genesis\_\_Applications\_\_c> oldApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

Consolidated\_Purpose\_\_c,

genesis\_\_Parent\_Application\_\_c,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

List<genesis\_\_Application\_Collateral\_\_c> appCol = [SELECT Id, genesis\_\_Application\_\_c FROM genesis\_\_Application\_Collateral\_\_c];

appCol[0].genesis\_\_Application\_\_c = oldApplist[0].Id;

update appCol;

Group queue = [SELECT Id, Name FROM Group WHERE Type = 'Queue' AND Name = 'Sales'];

oldApplist[0].OwnerId = queue.Id;

System.assertEquals(true, acc.size() > 0,'Account Not Inserted');

Group queueType = [SELECT Id, Name FROM Group WHERE Type = 'Queue' AND Name = 'Credit'];

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(oldApplist[0].Id, oldApplist[0]);

List<genesis\_\_Applications\_\_c> newApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

createddate,

Consolidated\_Purpose\_\_c,

genesis\_\_Parent\_Application\_\_c,

Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

newApplist[0].genesis\_\_Status\_\_c = 'PRELIM REVIEW';

newApplist[0].genesis\_\_Interest\_Rate\_\_c = 9.8;

newApplist[0].genesis\_\_Term\_\_c = 24;

newApplist[0].genesis\_\_Loan\_Amount\_\_c = 18000;

newApplist[0].genesis\_\_Total\_Fee\_Amount\_\_c = 2000;

newApplist[0].Workflow\_Status\_\_c = 'CONDITIONAL APPROVAL';

newApplist[0].Thinktank\_Margin\_\_c = 0.41;

newApplist[0].genesis\_\_CLTV\_\_c = 0.70;

newApplist[0].Funding\_Date\_\_c = Date.newInstance(2021,11,24);

newApplist[0].Rate\_Index\_\_c = 'Residential SMSF Rate';

newApplist[0].Broker\_Trail\_\_c = 0.67;

newApplist[0].genesis\_\_Requested\_Loan\_Amount\_\_c = 10000;

newApplist[0].genesis\_\_Disbursement\_Date\_\_c = Date.newInstance(2018,05,13);

newApplist[0].genesis\_\_Total\_Facility\_Amount\_\_c = 2002;

newApplist[0].Aggregator\_\_c = acc[0].Id;

newApplist[0].OwnerId = queueType.Id;

update newApplist;

newMap.put(newApplist[0].Id, newApplist[0]);

genesis\_\_Department\_\_c departmentCheck = new genesis\_\_Department\_\_c();

departmentCheck.Name = TTConstants.CREDIT\_UNDERWRITING\_DEPT;

insert departmentCheck;

List<genesis\_\_Department\_\_c> assertdeparts = [ SELECT Id FROM genesis\_\_Department\_\_c];

System.assertEquals(true, !assertdeparts.isEmpty(), 'Department2 not inserted');

genesis\_\_Application\_Department\_\_c deptApplicationCheck = new genesis\_\_Application\_Department\_\_c();

deptApplicationCheck.genesis\_\_Department\_\_c = departmentCheck.Id;

deptApplicationCheck.genesis\_\_Application\_\_c = newApplist[0].Id;

deptApplicationCheck.genesis\_\_Status\_\_c = TTConstants.ACTIVE;

insert deptApplicationCheck;

List< genesis\_\_Application\_Department\_\_c> assertDept = [ SELECT Id ,genesis\_\_Department\_\_c FROM genesis\_\_Application\_Department\_\_c];

System.assertEquals(true, !assertDept.isEmpty(), 'Application Department not inserted');

List<Task> taskList = new List<Task>();

Task tasks = new Task();

tasks.genesis\_\_Application\_\_c = newApplist[0].Id;

tasks.genesis\_\_Department\_\_c = assertDept[0].genesis\_\_Department\_\_c;

tasks.Status = 'Open';

tasks.Subject = 'Urgent Task-Increase Loan amount for Further Advance';

taskList.add(tasks);

insert taskList;

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

newApplist,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApplist, oldMap, 'AFTER\_UPDATE');

// obj.afterUpdate(newApplist, oldMap);

}

Test.stopTest();

}

@isTest

static void updateSplitStatusOnPackageChangeTestApp() {

List<genesis\_\_Applications\_\_c> oldApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Parent\_Rate\_Index\_\_c,

genesis\_\_Parent\_Application\_\_c,

Payment\_Type\_\_c,

genesis\_\_Asset\_Class\_\_c,

P\_I\_Repayment\_Amount\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Payment\_Amount\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Loan'];

List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

System.assertEquals(true, acc.size() > 0,'Account Not Inserted');

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(oldApplist[0].Id, oldApplist[0]);

List<genesis\_\_Applications\_\_c> newApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Parent\_Rate\_Index\_\_c,

genesis\_\_Parent\_Application\_\_c,

Payment\_Type\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

P\_I\_Repayment\_Amount\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_Payment\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

createddate ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Loan'];

List<clcommon\_\_CL\_Purpose\_\_c> purposeSplit = [SELECT Id FROM clcommon\_\_CL\_Purpose\_\_c WHERE Name = 'Equity Out'];

newApplist[0].genesis\_\_Status\_\_c = 'NEW - ENTERED';

newApplist[0].genesis\_\_Interest\_Rate\_\_c = 9.8;

newApplist[0].genesis\_\_Term\_\_c = 24;

newApplist[0].genesis\_\_Loan\_Amount\_\_c = 18000;

newApplist[0].genesis\_\_Total\_Fee\_Amount\_\_c = 2000;

newApplist[0].Workflow\_Status\_\_c = 'CONDITIONAL APPROVAL';

newApplist[0].Thinktank\_Margin\_\_c = 0.40;

newApplist[0].genesis\_\_CLTV\_\_c = 0.70;

newApplist[0].Funding\_Date\_\_c = Date.newInstance(2021,11,24);

newApplist[0].Rate\_Index\_\_c = 'Residential Loan Rate';

newApplist[0].Broker\_Trail\_\_c = 0.67;

newApplist[0].genesis\_\_Requested\_Loan\_Amount\_\_c = 10000;

newApplist[0].genesis\_\_Disbursement\_Date\_\_c = Date.newInstance(2018,05,13);

newApplist[0].genesis\_\_Total\_Facility\_Amount\_\_c = 2002;

newApplist[0].Aggregator\_\_c = acc[0].Id;

newApplist[0].genesis\_\_CL\_Purpose\_\_c = purposeSplit[0].Id;

newApplist[0].Payment\_Type\_\_c = 'Principal & Interest';

update newApplist;

newMap.put(newApplist[0].Id, newApplist[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

newApplist,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApplist, oldMap, 'AFTER\_UPDATE');

// obj.afterUpdate(newApplist, oldMap);

}

Test.stopTest();

}

@isTest

static void updateSplitStatusOnPackageChangeTestApplication() {

List<genesis\_\_Applications\_\_c> oldApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Parent\_Rate\_Index\_\_c,

genesis\_\_Parent\_Application\_\_c,

Payment\_Type\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

P\_I\_Repayment\_Amount\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Payment\_Amount\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Loan'];

List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

System.assertEquals(true, acc.size() > 0,'Account Not Inserted');

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(oldApplist[0].Id, oldApplist[0]);

List<genesis\_\_Applications\_\_c> newApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Parent\_Rate\_Index\_\_c,

genesis\_\_Parent\_Application\_\_c,

Payment\_Type\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Payment\_Amount\_\_c,

genesis\_\_Status\_\_c,

P\_I\_Repayment\_Amount\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

createddate ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Loan'];

List<clcommon\_\_CL\_Purpose\_\_c> purposeSplit = [SELECT Id FROM clcommon\_\_CL\_Purpose\_\_c WHERE Name = 'Equity Out'];

newApplist[0].genesis\_\_Status\_\_c = 'NEW - ENTERED';

newApplist[0].genesis\_\_Interest\_Rate\_\_c = 9.8;

newApplist[0].genesis\_\_Term\_\_c = 24;

newApplist[0].genesis\_\_Loan\_Amount\_\_c = 18000;

newApplist[0].genesis\_\_Total\_Fee\_Amount\_\_c = 2000;

newApplist[0].Workflow\_Status\_\_c = 'CONDITIONAL APPROVAL';

newApplist[0].Thinktank\_Margin\_\_c = 0.40;

newApplist[0].genesis\_\_CLTV\_\_c = 0.70;

newApplist[0].Funding\_Date\_\_c = Date.newInstance(2021,11,24);

newApplist[0].Rate\_Index\_\_c = 'Residential Loan Rate';

newApplist[0].Broker\_Trail\_\_c = 0.67;

newApplist[0].genesis\_\_Requested\_Loan\_Amount\_\_c = 10000;

newApplist[0].genesis\_\_Disbursement\_Date\_\_c = Date.newInstance(2018,05,13);

newApplist[0].genesis\_\_Total\_Facility\_Amount\_\_c = 2002;

newApplist[0].Aggregator\_\_c = acc[0].Id;

newApplist[0].genesis\_\_CL\_Purpose\_\_c = purposeSplit[0].Id;

newApplist[0].Pricing\_Basis\_\_c = 'Variable';

newApplist[0].Payment\_Type\_\_c = 'Principal & Interest';

update newApplist;

newMap.put(newApplist[0].Id, newApplist[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

newApplist,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApplist, oldMap, 'AFTER\_UPDATE');

// obj.afterUpdate(newApplist, oldMap);

}

Test.stopTest();

}

@isTest

static void genesisApplicationTriggerHandlerTestUpdates(){

List<genesis\_\_Applications\_\_c> app = [SELECT Id,

OwnerId,

Execution\_Flow\_\_r.Name,

Is\_Further\_Advance\_App\_\_c,

LTVP\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

createddate,

Consolidated\_Purpose\_\_c,

genesis\_\_Parent\_Application\_\_c,

Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

System.assertEquals(true, app.size() > 0,'App Not Inserted');

List<clcommon\_\_CL\_Product\_\_c> prod = [SELECT Id FROM clcommon\_\_CL\_Product\_\_c WHERE clcommon\_\_Product\_Name\_\_c= 'SMSF Commercial Master'];

System.assertEquals(true, prod.size() > 0,'Prod Not Fethced');

Id accRecordType = Schema.SObjectType.Account.getRecordTypeInfosByName().get('Aggregator').getRecordTypeId();

Account newAcc = new Account();

newAcc.Name = 'Sanjeev';

newAcc.Type = 'Aggregator';

newAcc.Salutation\_\_c = 'Mr.';

newAcc.First\_Name\_\_c = 'Sanjeev ';

newAcc.Last\_Name\_\_c = 'Kumar';

newAcc.clcommon\_\_Email\_\_c = 'sk@mailinator.com';

newAcc.recordTypeId = accRecordType;

insert newAcc;

List<Account> acc = [SELECT Id, Name FROM Account

WHERE Name = 'Sanjeev'];

System.assertEquals(true, acc.size() > 0,'Account Not Inserted');

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(app[0].Id, app[0]);

app[0].genesis\_\_Disbursement\_Date\_\_c = Date.newInstance(2015, 11, 11);

app[0].Aggregator\_\_c = acc[0].Id;

app[0].genesis\_\_CL\_Product\_\_c = prod[0].Id;

app[0].Broker\_Trail\_\_c = 3.4;

update app;

newMap.put(app[0].Id,app[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

app,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(app, oldMap, 'AFTER\_UPDATE');

// obj.afterUpdate(app, oldMap);

}

Test.stopTest();

}

@isTest

static void genesisApplicationTriggerHandlerChecksTest(){

List<genesis\_\_Applications\_\_c> app = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

Parent\_Rate\_Index\_\_c,

genesis\_\_Parent\_Application\_\_c,

Payment\_Type\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

P\_I\_Repayment\_Amount\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

IO\_Repayment\_Amount\_\_c,

Construction\_Term\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

genesis\_\_Payment\_Amount\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

createddate,

Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Loan'];

System.assertEquals(true, app.size() > 0,'App Not Inserted');

List<clcommon\_\_CL\_Product\_\_c> prod = [SELECT Id FROM clcommon\_\_CL\_Product\_\_c WHERE clcommon\_\_Product\_Name\_\_c= 'SMSF Commercial Master'];

System.assertEquals(true, prod.size() > 0,'Prod Not Fethced');

Id accRecordType = Schema.SObjectType.Account.getRecordTypeInfosByName().get('Aggregator').getRecordTypeId();

Account newAcc = new Account();

newAcc.Name = 'Sanjeev';

newAcc.Type = 'Aggregator';

newAcc.Salutation\_\_c = 'Mr.';

newAcc.First\_Name\_\_c = 'Sanjeev ';

newAcc.Last\_Name\_\_c = 'Kumar';

newAcc.clcommon\_\_Email\_\_c = 'sk@mailinator.com';

newAcc.recordTypeId = accRecordType;

insert newAcc;

List<Account> acc = [SELECT Id, Name FROM Account

WHERE Name = 'Sanjeev'];

System.assertEquals(true, acc.size() > 0,'Account Not Inserted');

List<clcommon\_\_CL\_Purpose\_\_c> clpur = [SELECT Id, Name

FROM clcommon\_\_CL\_Purpose\_\_c WHERE clcommon\_\_Purpose\_Code\_\_c = '3453'];

System.assertEquals(true, clpur.size() > 0,'Purpose Not Fetched');

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(app[0].Id, app[0]);

app[0].genesis\_\_Disbursement\_Date\_\_c = Date.newInstance(2015, 11, 11);

app[0].Thinktank\_Margin\_\_c = 4.53;

app[0].genesis\_\_Status\_\_c = 'CHANGE MEMO APPROVAL';

app[0].Aggregator\_\_c = acc[0].Id;

app[0].genesis\_\_CL\_Purpose\_\_c = clpur[0].Id;

app[0].genesis\_\_CL\_Product\_\_c = prod[0].Id;

app[0].genesis\_\_Term\_\_c = 55;

update app;

newMap.put(app[0].Id,app[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

app,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(app, oldMap, 'AFTER\_UPDATE');

// obj.afterUpdate(app, oldMap);

}

Test.stopTest();

}

/\*@isTest

static void genesisApplicantsTriggerHandlerTest() {

List<genesis\_\_Applications\_\_c> oldApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

Consolidated\_Purpose\_\_c,

genesis\_\_Parent\_Application\_\_c,

Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

List<Group> testGroup = [SELECT Id FROM Group WHERE Type = 'Queue' AND Name ='Sales' LIMIT 1];

oldApplist[0].OwnerId = testGroup[0].Id;

update oldApplist;

List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

List<genesis\_\_Application\_Collateral\_\_c> appCol = [SELECT Id, genesis\_\_Application\_\_c FROM genesis\_\_Application\_Collateral\_\_c];

appCol[0].genesis\_\_Application\_\_c = oldApplist[0].Id;

update appCol;

System.assertEquals(true, oldApplist.size() > 0,'App Not Inserted');

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(oldApplist[0].Id, oldApplist[0]);

List<genesis\_\_Applications\_\_c> newApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

createddate,

Consolidated\_Purpose\_\_c,

genesis\_\_Parent\_Application\_\_c,

Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

List<user> testuser = [SELECT Id FROM user WHERE profile.Name = 'System Administrator' ];

newApplist[0].Aggregator\_\_c = acc[0].Id;

newApplist[0].OwnerId = testuser[0].Id;

newApplist[0].genesis\_\_Status\_\_c = 'INTERIM STATUS FORWARD';

update newApplist;

newMap.put(newApplist[0].Id, newApplist[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApplist, oldMap, 'AFTER\_UPDATE');

obj.afterUpdate(newApplist, oldMap);

}

Test.stopTest();

}\*/

@isTest

static void genesisApplicationCheckTriggerHandlerTest() {

List<genesis\_\_Applications\_\_c> oldApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

Consolidated\_Purpose\_\_c,

genesis\_\_Parent\_Application\_\_c,

Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

List<user> testuserNew = [SELECT Id FROM user WHERE Id= :userinfo.getuserid() LIMIT 1];

oldApplist[0].OwnerId = testuserNew[0].Id;

update oldApplist;

Group queueType = [SELECT Id, Name FROM Group WHERE Type = 'Queue' AND Name = 'Credit'];

List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

List<genesis\_\_Application\_Collateral\_\_c> appCol = [SELECT Id, genesis\_\_Application\_\_c FROM genesis\_\_Application\_Collateral\_\_c];

appCol[0].genesis\_\_Application\_\_c = oldApplist[0].Id;

update appCol;

System.assertEquals(true, acc.size() > 0,'Account Not Inserted');

genesis\_\_Department\_\_c departmentCheck = new genesis\_\_Department\_\_c();

departmentCheck.Name = TTConstants.CREDIT\_UNDERWRITING\_DEPT;

insert departmentCheck;

List<genesis\_\_Department\_\_c> assertdeparts = [ SELECT Id FROM genesis\_\_Department\_\_c];

System.assertEquals(true, !assertdeparts.isEmpty(), 'Department2 not inserted');

genesis\_\_Application\_Department\_\_c deptApplicationCheck = new genesis\_\_Application\_Department\_\_c();

deptApplicationCheck.genesis\_\_Department\_\_c = departmentCheck.Id;

deptApplicationCheck.genesis\_\_Application\_\_c = oldApplist[0].Id;

deptApplicationCheck.genesis\_\_Status\_\_c = TTConstants.ACTIVE;

insert deptApplicationCheck;

List< genesis\_\_Application\_Department\_\_c> assertDept = [ SELECT Id ,genesis\_\_Department\_\_c FROM genesis\_\_Application\_Department\_\_c];

System.assertEquals(true, !assertDept.isEmpty(), 'Application Department not inserted');

List<Task> taskList = new List<Task>();

Task tasks = new Task();

tasks.genesis\_\_Application\_\_c = oldApplist[0].Id;

tasks.genesis\_\_Department\_\_c = assertDept[0].genesis\_\_Department\_\_c;

tasks.Status = 'Open';

tasks.Subject = 'Urgent Task-Increase Loan amount for Further Advance';

taskList.add(tasks);

insert taskList;

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(oldApplist[0].Id, oldApplist[0]);

List<genesis\_\_Applications\_\_c> newApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

createddate,

Consolidated\_Purpose\_\_c,

genesis\_\_Parent\_Application\_\_c,

Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

newApplist[0].Aggregator\_\_c = acc[0].Id;

newApplist[0].OwnerId = queueType.Id;

newApplist[0].genesis\_\_Status\_\_c = 'CHANGE MEMO';

update newApplist;

newMap.put(newApplist[0].Id, newApplist[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

newApplist,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApplist, oldMap, 'AFTER\_UPDATE');

// obj.afterUpdate(newApplist, oldMap);

}

Test.stopTest();

}

@isTest

static void genesisApplicationUpdateTriggerHandlerTest() {

List<genesis\_\_Applications\_\_c> oldApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c, createddate ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

List<user> testuser = [SELECT Id FROM user WHERE profile.Name = 'System Administrator' AND Id != :userinfo.getuserid() LIMIT 1];

oldApplist[0].ownerid = testuser[0].Id;

update oldApplist;

System.assertEquals(true, oldApplist.size() > 0,'App Not Inserted');

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(oldApplist[0].Id, oldApplist[0]);

List<genesis\_\_Applications\_\_c> newApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c, createddate ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

Group queue = [SELECT Id, Name FROM Group WHERE Type = 'Queue' AND Name = 'Credit Underwriter - Thinktank'];

newApplist[0].ownerid = queue.Id;

newApplist[0].genesis\_\_Status\_\_c = 'FORMAL CREDIT UNDERWRITING';

newAppList[0].Brand\_\_c = 'RateMoney';

update newApplist;

genesis\_\_Department\_\_c dept = new genesis\_\_Department\_\_c();

dept.Name = 'Credit Underwriting';

INSERT dept;

genesis\_\_Application\_Department\_\_c dep = new genesis\_\_Application\_Department\_\_c();

dep.genesis\_\_Application\_\_c = newApplist[0].Id;

dep.genesis\_\_Department\_\_c = dept.Id;

dep.genesis\_\_Status\_\_c = 'Active';

INSERT dep;

newMap.put(newApplist[0].Id, newApplist[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

newApplist,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApplist, oldMap, 'AFTER\_UPDATE');

// obj.afterUpdate(newApplist, oldMap);

}

Test.stopTest();

}

@isTest

static void genesisCheckApplicationTriggerHandlerTest() {

List<genesis\_\_Applications\_\_c> oldApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c, createddate ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

List<user> testuser = [SELECT Id FROM user WHERE profile.Name = 'System Administrator' AND Id != :userinfo.getuserid() LIMIT 1];

oldApplist[0].ownerid = testuser[0].Id;

update oldApplist;

System.assertEquals(true, oldApplist.size() > 0,'App Not Inserted');

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(oldApplist[0].Id, oldApplist[0]);

List<genesis\_\_Applications\_\_c> newApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c, createddate ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

Group queue = [SELECT Id, Name FROM Group WHERE Type = 'Queue' AND Name = 'Credit Underwriter - Thinktank'];

newApplist[0].ownerid = queue.Id;

newApplist[0].genesis\_\_Status\_\_c = 'FORMAL CREDIT UNDERWRITING';

newAppList[0].Brand\_\_c = 'Thinktank';

update newApplist;

genesis\_\_Department\_\_c dept = new genesis\_\_Department\_\_c();

dept.Name = 'Credit Underwriting';

INSERT dept;

genesis\_\_Application\_Department\_\_c dep = new genesis\_\_Application\_Department\_\_c();

dep.genesis\_\_Application\_\_c = newApplist[0].Id;

dep.genesis\_\_Department\_\_c = dept.Id;

dep.genesis\_\_Status\_\_c = 'Active';

INSERT dep;

newMap.put(newApplist[0].Id, newApplist[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

newApplist,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApplist, oldMap, 'AFTER\_UPDATE');

// obj.afterUpdate(newApplist, oldMap);

}

Test.stopTest();

}

@isTest

static void genesisApplicationTriggerSplitHandlerTest() {

List<genesis\_\_Applications\_\_c> oldApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c, createddate ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(oldApplist[0].Id, oldApplist[0]);

System.assertEquals(true, oldApplist.size() > 0,'App Not Inserted');

List<genesis\_\_Applications\_\_c> newApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c, createddate ,Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

newApplist[0].genesis\_\_Status\_\_c = 'PRELIM REVIEW - COMPLETION';

update newApplist;

newMap.put(newApplist[0].Id, newApplist[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

newApplist,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApplist, oldMap, 'AFTER\_UPDATE');

// obj.afterUpdate(newApplist, oldMap);

}

Test.stopTest();

}

@isTest

static void runBrandMappingTestt(){

List<genesis\_\_Applications\_\_c> app = [SELECT Id,

Record\_Type\_Name\_\_c,

genesis\_\_Status\_\_c,

Workflow\_Status\_\_c,

OwnerId,

Is\_Simpology\_Application\_\_c,

genesis\_\_Parent\_Application\_\_c,

genesis\_\_CL\_Product\_\_c,

Aggregator\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

Execution\_Flow\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

System.assertEquals(true, app.size() > 0,'Application Not Inserted');

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(app[0].Id, app[0]);

List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

List<clcommon\_\_CL\_Product\_\_c> clProduct = [SELECT Id FROM clcommon\_\_CL\_Product\_\_c WHERE clcommon\_\_Product\_Name\_\_c = 'Full Doc Commercial Master'];

List<genesis\_\_Applications\_\_c> newApp = [SELECT Id,

Record\_Type\_Name\_\_c,

genesis\_\_Status\_\_c,

Workflow\_Status\_\_c,

OwnerId,

Is\_Simpology\_Application\_\_c,

genesis\_\_Parent\_Application\_\_c,

genesis\_\_CL\_Product\_\_c,

Aggregator\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c ,

Execution\_Flow\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

newApp[0].genesis\_\_CL\_Product\_\_c = clProduct[0].Id;

newApp[0].Aggregator\_\_c = acc[0].Id;

newApp[0].genesis\_\_Total\_Facility\_Amount\_\_c = 100.23;

newApp[0].genesis\_\_Status\_\_c = 'CHANGE MEMO APPROVAL';

update newApp;

newMap.put(newApp[0].Id, newApp[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

true, // isBefore

false, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

newApp,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApp, oldMap, 'BEFORE\_UPDATE');

// obj.beforeUpdate(newApp, oldMap);

}

Test.stopTest();

}

@isTest

static void runBrandMapTestt(){

List<genesis\_\_Applications\_\_c> app = [SELECT Id,

Record\_Type\_Name\_\_c,

genesis\_\_Status\_\_c,

Workflow\_Status\_\_c,

OwnerId,

Is\_Simpology\_Application\_\_c,

genesis\_\_Parent\_Application\_\_c,

genesis\_\_CL\_Product\_\_c,

Aggregator\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

Execution\_Flow\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

System.assertEquals(true, app.size() > 0,'Application Not Inserted');

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(app[0].Id, app[0]);

List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

List<clcommon\_\_CL\_Product\_\_c> clProduct = [SELECT Id FROM clcommon\_\_CL\_Product\_\_c WHERE clcommon\_\_Product\_Name\_\_c = 'Full Doc Commercial Master'];

List<genesis\_\_Applications\_\_c> newApp = [SELECT Id,

Record\_Type\_Name\_\_c,

genesis\_\_Status\_\_c,

Workflow\_Status\_\_c,

OwnerId,

Is\_Simpology\_Application\_\_c,

genesis\_\_Parent\_Application\_\_c,

genesis\_\_CL\_Product\_\_c,

Aggregator\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

Execution\_Flow\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

newApp[0].genesis\_\_CL\_Product\_\_c = clProduct[0].Id;

newApp[0].Aggregator\_\_c = acc[0].Id;

newApp[0].genesis\_\_Total\_Facility\_Amount\_\_c = 100.23;

newApp[0].genesis\_\_Status\_\_c = 'TREASURY SKIP';

update newApp;

newMap.put(newApp[0].Id, newApp[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

true, // isBefore

false, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

newApp,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApp, oldMap, 'BEFORE\_UPDATE');

// obj.beforeUpdate(newApp, oldMap);

}

Test.stopTest();

}

@isTest

static void genesisApplicationCheckTriggerHandlerrTest() {

List<clcommon\_\_CL\_Product\_\_c> assertproductCommercial = [ SELECT Id

FROM clcommon\_\_CL\_Product\_\_c

WHERE clcommon\_\_Product\_Name\_\_c= 'Full Doc Commercial Master'];

List<genesis\_\_Applications\_\_c> oldApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

Consolidated\_Purpose\_\_c,

genesis\_\_Parent\_Application\_\_c,

Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

List<user> testuserNew = [SELECT Id FROM user WHERE Id= :userinfo.getuserid() LIMIT 1];

oldApplist[0].OwnerId = testuserNew[0].Id;

update oldApplist;

Group queueType = [SELECT Id, Name FROM Group WHERE Type = 'Queue' AND Name = 'Credit'];

List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

List<genesis\_\_Application\_Collateral\_\_c> appCol = [SELECT Id, genesis\_\_Application\_\_c FROM genesis\_\_Application\_Collateral\_\_c];

appCol[0].genesis\_\_Application\_\_c = oldApplist[0].Id;

update appCol;

System.assertEquals(true, acc.size() > 0,'Account Not Inserted');

genesis\_\_Department\_\_c departmentCheck = new genesis\_\_Department\_\_c();

departmentCheck.Name = TTConstants.CREDIT\_UNDERWRITING\_DEPT;

insert departmentCheck;

List<genesis\_\_Department\_\_c> assertdeparts = [ SELECT Id FROM genesis\_\_Department\_\_c];

System.assertEquals(true, !assertdeparts.isEmpty(), 'Department2 not inserted');

genesis\_\_Application\_Department\_\_c deptApplicationCheck = new genesis\_\_Application\_Department\_\_c();

deptApplicationCheck.genesis\_\_Department\_\_c = departmentCheck.Id;

deptApplicationCheck.genesis\_\_Application\_\_c = oldApplist[0].Id;

deptApplicationCheck.genesis\_\_Status\_\_c = TTConstants.ACTIVE;

insert deptApplicationCheck;

List< genesis\_\_Application\_Department\_\_c> assertDept = [ SELECT Id ,genesis\_\_Department\_\_c FROM genesis\_\_Application\_Department\_\_c];

System.assertEquals(true, !assertDept.isEmpty(), 'Application Department not inserted');

List<Task> taskList = new List<Task>();

Task tasks = new Task();

tasks.genesis\_\_Application\_\_c = oldApplist[0].Id;

tasks.genesis\_\_Department\_\_c = assertDept[0].genesis\_\_Department\_\_c;

tasks.Status = 'Open';

tasks.Subject = 'Urgent Task-Increase Loan amount for Further Advance';

taskList.add(tasks);

insert taskList;

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(oldApplist[0].Id, oldApplist[0]);

List<genesis\_\_Applications\_\_c> newApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

createddate,

Consolidated\_Purpose\_\_c,

genesis\_\_Parent\_Application\_\_c,

Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

newApplist[0].Aggregator\_\_c = acc[0].Id;

newApplist[0].OwnerId = queueType.Id;

newApplist[0].genesis\_\_Status\_\_c = 'CHANGE MEMO';

newApplist[0].genesis\_\_CL\_Product\_\_c = assertproductCommercial[0].Id;

update newApplist;

newMap.put(newApplist[0].Id, newApplist[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

newApplist,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApplist, oldMap, 'AFTER\_UPDATE');

// obj.afterUpdate(newApplist, oldMap);

}

Test.stopTest();

}

@isTest

static void genesisApplicationCheckTriggerHandlerrTestOne() {

List<clcommon\_\_CL\_Product\_\_c> assertproductCommercial = [ SELECT Id

FROM clcommon\_\_CL\_Product\_\_c

WHERE clcommon\_\_Product\_Name\_\_c= 'Full Doc Commercial Master'];

List<genesis\_\_Applications\_\_c> oldApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

Consolidated\_Purpose\_\_c,

genesis\_\_Parent\_Application\_\_c,

Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

List<user> testuserNew = [SELECT Id FROM user WHERE Id= :userinfo.getuserid() LIMIT 1];

oldApplist[0].OwnerId = testuserNew[0].Id;

update oldApplist;

Group queueType = [SELECT Id, Name FROM Group WHERE Type = 'Queue' AND Name = 'Credit'];

List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

List<genesis\_\_Application\_Collateral\_\_c> appCol = [SELECT Id, genesis\_\_Application\_\_c FROM genesis\_\_Application\_Collateral\_\_c];

appCol[0].genesis\_\_Application\_\_c = oldApplist[0].Id;

update appCol;

System.assertEquals(true, acc.size() > 0,'Account Not Inserted');

genesis\_\_Department\_\_c departmentCheck = new genesis\_\_Department\_\_c();

departmentCheck.Name = TTConstants.CREDIT\_UNDERWRITING\_DEPT;

insert departmentCheck;

List<genesis\_\_Department\_\_c> assertdeparts = [ SELECT Id FROM genesis\_\_Department\_\_c];

System.assertEquals(true, !assertdeparts.isEmpty(), 'Department2 not inserted');

genesis\_\_Application\_Department\_\_c deptApplicationCheck = new genesis\_\_Application\_Department\_\_c();

deptApplicationCheck.genesis\_\_Department\_\_c = departmentCheck.Id;

deptApplicationCheck.genesis\_\_Application\_\_c = oldApplist[0].Id;

deptApplicationCheck.genesis\_\_Status\_\_c = TTConstants.ACTIVE;

insert deptApplicationCheck;

List< genesis\_\_Application\_Department\_\_c> assertDept = [ SELECT Id ,genesis\_\_Department\_\_c FROM genesis\_\_Application\_Department\_\_c];

System.assertEquals(true, !assertDept.isEmpty(), 'Application Department not inserted');

List<Task> taskList = new List<Task>();

Task tasks = new Task();

tasks.genesis\_\_Application\_\_c = oldApplist[0].Id;

tasks.genesis\_\_Department\_\_c = assertDept[0].genesis\_\_Department\_\_c;

tasks.Status = 'Open';

tasks.Subject = 'Urgent Task-Increase Loan amount for Further Advance';

taskList.add(tasks);

insert taskList;

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(oldApplist[0].Id, oldApplist[0]);

List<genesis\_\_Applications\_\_c> newApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

createddate,

Consolidated\_Purpose\_\_c,

genesis\_\_Parent\_Application\_\_c,

Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

newApplist[0].Aggregator\_\_c = acc[0].Id;

newApplist[0].OwnerId = queueType.Id;

newApplist[0].genesis\_\_Status\_\_c = 'CHANGE MEMO';

newApplist[0].genesis\_\_CL\_Product\_\_c = assertproductCommercial[0].Id;

newApplist[0].genesis\_\_CLTV\_\_c = 0.70;

update newApplist;

newMap.put(newApplist[0].Id, newApplist[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

newApplist,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApplist, oldMap, 'AFTER\_UPDATE');

// obj.afterUpdate(newApplist, oldMap);

}

Test.stopTest();

}

@isTest

static void genesisAppplicationnCheckTriggerHandlerrTest() {

List<clcommon\_\_CL\_Product\_\_c> assertproductCommercial = [ SELECT Id

FROM clcommon\_\_CL\_Product\_\_c

WHERE clcommon\_\_Product\_Name\_\_c= 'SMSF Residential Master'];

List<genesis\_\_Applications\_\_c> oldApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

Consolidated\_Purpose\_\_c,

genesis\_\_Parent\_Application\_\_c,

Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

List<user> testuserNew = [SELECT Id FROM user WHERE Id= :userinfo.getuserid() LIMIT 1];

oldApplist[0].OwnerId = testuserNew[0].Id;

update oldApplist;

Group queueType = [SELECT Id, Name FROM Group WHERE Type = 'Queue' AND Name = 'Credit'];

List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

List<genesis\_\_Application\_Collateral\_\_c> appCol = [SELECT Id, genesis\_\_Application\_\_c FROM genesis\_\_Application\_Collateral\_\_c];

appCol[0].genesis\_\_Application\_\_c = oldApplist[0].Id;

update appCol;

System.assertEquals(true, acc.size() > 0,'Account Not Inserted');

genesis\_\_Department\_\_c departmentCheck = new genesis\_\_Department\_\_c();

departmentCheck.Name = TTConstants.CREDIT\_UNDERWRITING\_DEPT;

insert departmentCheck;

List<genesis\_\_Department\_\_c> assertdeparts = [ SELECT Id FROM genesis\_\_Department\_\_c];

System.assertEquals(true, !assertdeparts.isEmpty(), 'Department2 not inserted');

genesis\_\_Application\_Department\_\_c deptApplicationCheck = new genesis\_\_Application\_Department\_\_c();

deptApplicationCheck.genesis\_\_Department\_\_c = departmentCheck.Id;

deptApplicationCheck.genesis\_\_Application\_\_c = oldApplist[0].Id;

deptApplicationCheck.genesis\_\_Status\_\_c = TTConstants.ACTIVE;

insert deptApplicationCheck;

List< genesis\_\_Application\_Department\_\_c> assertDept = [ SELECT Id ,genesis\_\_Department\_\_c FROM genesis\_\_Application\_Department\_\_c];

System.assertEquals(true, !assertDept.isEmpty(), 'Application Department not inserted');

List<Task> taskList = new List<Task>();

Task tasks = new Task();

tasks.genesis\_\_Application\_\_c = oldApplist[0].Id;

tasks.genesis\_\_Department\_\_c = assertDept[0].genesis\_\_Department\_\_c;

tasks.Status = 'Open';

tasks.Subject = 'Urgent Task-Increase Loan amount for Further Advance';

taskList.add(tasks);

insert taskList;

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(oldApplist[0].Id, oldApplist[0]);

List<genesis\_\_Applications\_\_c> newApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

createddate,

Consolidated\_Purpose\_\_c,

genesis\_\_Parent\_Application\_\_c,

Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

newApplist[0].Aggregator\_\_c = acc[0].Id;

newApplist[0].OwnerId = queueType.Id;

newApplist[0].genesis\_\_Status\_\_c = 'CHANGE MEMO';

newApplist[0].genesis\_\_CL\_Product\_\_c = assertproductCommercial[0].Id;

update newApplist;

newMap.put(newApplist[0].Id, newApplist[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

newApplist,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApplist, oldMap, 'AFTER\_UPDATE');

// obj.afterUpdate(newApplist, oldMap);

}

Test.stopTest();

}

@isTest

static void genesissApplicationnCheckkTriggerHandlerrTest() {

List<clcommon\_\_CL\_Product\_\_c> assertproductCommercial = [ SELECT Id

FROM clcommon\_\_CL\_Product\_\_c

WHERE clcommon\_\_Product\_Name\_\_c= 'SMSF Commercial Master'];

List<genesis\_\_Applications\_\_c> oldApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

Consolidated\_Purpose\_\_c,

genesis\_\_Parent\_Application\_\_c,

Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

List<user> testuserNew = [SELECT Id FROM user WHERE Id= :userinfo.getuserid() LIMIT 1];

oldApplist[0].OwnerId = testuserNew[0].Id;

update oldApplist;

Group queueType = [SELECT Id, Name FROM Group WHERE Type = 'Queue' AND Name = 'Credit'];

List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

List<genesis\_\_Application\_Collateral\_\_c> appCol = [SELECT Id, genesis\_\_Application\_\_c FROM genesis\_\_Application\_Collateral\_\_c];

appCol[0].genesis\_\_Application\_\_c = oldApplist[0].Id;

update appCol;

System.assertEquals(true, acc.size() > 0,'Account Not Inserted');

genesis\_\_Department\_\_c departmentCheck = new genesis\_\_Department\_\_c();

departmentCheck.Name = TTConstants.CREDIT\_UNDERWRITING\_DEPT;

insert departmentCheck;

List<genesis\_\_Department\_\_c> assertdeparts = [ SELECT Id FROM genesis\_\_Department\_\_c];

System.assertEquals(true, !assertdeparts.isEmpty(), 'Department2 not inserted');

genesis\_\_Application\_Department\_\_c deptApplicationCheck = new genesis\_\_Application\_Department\_\_c();

deptApplicationCheck.genesis\_\_Department\_\_c = departmentCheck.Id;

deptApplicationCheck.genesis\_\_Application\_\_c = oldApplist[0].Id;

deptApplicationCheck.genesis\_\_Status\_\_c = TTConstants.ACTIVE;

insert deptApplicationCheck;

List< genesis\_\_Application\_Department\_\_c> assertDept = [ SELECT Id ,genesis\_\_Department\_\_c FROM genesis\_\_Application\_Department\_\_c];

System.assertEquals(true, !assertDept.isEmpty(), 'Application Department not inserted');

List<Task> taskList = new List<Task>();

Task tasks = new Task();

tasks.genesis\_\_Application\_\_c = oldApplist[0].Id;

tasks.genesis\_\_Department\_\_c = assertDept[0].genesis\_\_Department\_\_c;

tasks.Status = 'Open';

tasks.Subject = 'Urgent Task-Increase Loan amount for Further Advance';

taskList.add(tasks);

insert taskList;

Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

oldMap.put(oldApplist[0].Id, oldApplist[0]);

List<genesis\_\_Applications\_\_c> newApplist = [SELECT Id,

OwnerId,

LTVP\_\_c,

Is\_Further\_Advance\_App\_\_c,

Brand\_\_c,

genesis\_\_Asset\_Class\_\_c,

genesis\_\_Expected\_Close\_Date\_\_c,

Record\_Type\_Name\_\_c,

Is\_Simpology\_Application\_\_c,

Account\_Name\_\_c,

Penciled\_In\_Status\_\_c,

genesis\_\_Status\_\_c,

Thinktank\_Margin\_\_c,

genesis\_\_Interest\_Index\_\_c,

recordtypeid, genesis\_\_CL\_Product\_\_c,

genesis\_\_Loan\_Amount\_\_c,

genesis\_\_CL\_Purpose\_\_c,

Broker\_Trail\_\_c,

genesis\_\_Requested\_Loan\_Amount\_\_c,

Rate\_Index\_\_c,

genesis\_\_CLTV\_\_c,

Aggregator\_\_c,

Total\_Fee\_Amount\_inc\_GST\_\_c,

Funding\_Date\_\_c,

genesis\_\_Disbursement\_Date\_\_c,

genesis\_\_Account\_\_c,

Borrower\_Name\_\_c,

genesis\_\_Total\_Fee\_Amount\_\_c,

Workflow\_Status\_\_c,

Pricing\_Basis\_\_c,

genesis\_\_Interest\_Rate\_\_c,

genesis\_\_Term\_\_c,

Execution\_Flow\_\_c,

genesis\_\_Total\_Facility\_Amount\_\_c,

NCCP\_\_c,

createddate,

Consolidated\_Purpose\_\_c,

genesis\_\_Parent\_Application\_\_c,

Integration\_Aggregator\_\_c

FROM genesis\_\_Applications\_\_c

WHERE Record\_Type\_Name\_\_c = 'Package'

AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

newApplist[0].Aggregator\_\_c = acc[0].Id;

newApplist[0].OwnerId = queueType.Id;

newApplist[0].genesis\_\_Status\_\_c = 'CHANGE MEMO';

newApplist[0].genesis\_\_CL\_Product\_\_c = assertproductCommercial[0].Id;

update newApplist;

newMap.put(newApplist[0].Id, newApplist[0]);

Profile profileList = [SELECT Id,

Name

FROM Profile

WHERE name = 'System Administrator'];

User adminUser = new User();

adminUser.Alias = 'tt-test';

adminUser.Email = 'tt-test-admin-user@123.com';

adminUser.EmailEncodingKey = 'UTF-8';

adminUser.FirstName = 'TTTEST';

adminUser.LastName = 'ADMIN';

adminUser.LanguageLocaleKey = 'en\_US';

adminUser.LocaleSidKey = 'en\_US';

adminUser.ProfileId = profileList.Id;

adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

adminUser.UserName = 'tt-test-admin-user@123.com';

adminUser.IsActive = true;

insert adminUser;

Test.startTest();

System.runAs(adminUser){

GenesisApplicationTriggerHandlerV2.executeTriggerEvents(

false, // isBefore

true, // isAfter

false, // isInsert

true, // isUpdate

false, // isDelete

newApplist,

newMap,

null,

oldMap

);

// GenesisApplicationTriggerHandlerV2 obj = new GenesisApplicationTriggerHandlerV2(newApplist, oldMap, 'AFTER\_UPDATE');

// obj.afterUpdate(newApplist, oldMap);

}

Test.stopTest();

}

// @isTest

// static void genesissApplicationnCheckkTriggerHandlerrTestTwo() {

// List<clcommon\_\_CL\_Product\_\_c> assertproductCommercial = [ SELECT Id

// FROM clcommon\_\_CL\_Product\_\_c

// WHERE clcommon\_\_Product\_Name\_\_c= 'SMSF Commercial Master'];

// List<genesis\_\_Applications\_\_c> oldApplist = [SELECT Id,

// OwnerId,

// LTVP\_\_c,

// Is\_Further\_Advance\_App\_\_c,

// Brand\_\_c,

// genesis\_\_Asset\_Class\_\_c,

// genesis\_\_Expected\_Close\_Date\_\_c,

// Record\_Type\_Name\_\_c,

// Is\_Simpology\_Application\_\_c,

// Account\_Name\_\_c,

// Penciled\_In\_Status\_\_c,

// genesis\_\_Status\_\_c,

// Thinktank\_Margin\_\_c,

// genesis\_\_Interest\_Index\_\_c,

// recordtypeid, genesis\_\_CL\_Product\_\_c,

// genesis\_\_Loan\_Amount\_\_c,

// genesis\_\_CL\_Purpose\_\_c,

// Broker\_Trail\_\_c,

// genesis\_\_Requested\_Loan\_Amount\_\_c,

// Rate\_Index\_\_c,

// genesis\_\_CLTV\_\_c,

// Aggregator\_\_c,

// Total\_Fee\_Amount\_inc\_GST\_\_c,

// Funding\_Date\_\_c,

// genesis\_\_Disbursement\_Date\_\_c,

// genesis\_\_Account\_\_c,

// Borrower\_Name\_\_c,

// genesis\_\_Total\_Fee\_Amount\_\_c,

// Workflow\_Status\_\_c,

// Pricing\_Basis\_\_c,

// genesis\_\_Interest\_Rate\_\_c,

// genesis\_\_Term\_\_c,

// Execution\_Flow\_\_c,

// genesis\_\_Total\_Facility\_Amount\_\_c,

// NCCP\_\_c,

// Consolidated\_Purpose\_\_c,

// genesis\_\_Parent\_Application\_\_c,

// Integration\_Aggregator\_\_c

// FROM genesis\_\_Applications\_\_c

// WHERE Record\_Type\_Name\_\_c = 'Package'

// AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

// List<user> testuserNew = [SELECT Id FROM user WHERE Id= :userinfo.getuserid() LIMIT 1];

// oldApplist[0].OwnerId = testuserNew[0].Id;

// update oldApplist;

// Group queueType = [SELECT Id, Name FROM Group WHERE Type = 'Queue' AND Name = 'Credit'];

// List<Account> acc = [SELECT Id, Name, Type FROM Account WHERE Name = 'Raghav Mahajan'];

// List<genesis\_\_Application\_Collateral\_\_c> appCol = [SELECT Id, genesis\_\_Application\_\_c FROM genesis\_\_Application\_Collateral\_\_c];

// appCol[0].genesis\_\_Application\_\_c = oldApplist[0].Id;

// update appCol;

// System.assertEquals(true, acc.size() > 0,'Account Not Inserted');

// genesis\_\_Department\_\_c departmentCheck = new genesis\_\_Department\_\_c();

// departmentCheck.Name = TTConstants.CREDIT\_UNDERWRITING\_DEPT;

// insert departmentCheck;

// List<genesis\_\_Department\_\_c> assertdeparts = [ SELECT Id FROM genesis\_\_Department\_\_c];

// System.assertEquals(true, !assertdeparts.isEmpty(), 'Department2 not inserted');

// genesis\_\_Application\_Department\_\_c deptApplicationCheck = new genesis\_\_Application\_Department\_\_c();

// deptApplicationCheck.genesis\_\_Department\_\_c = departmentCheck.Id;

// deptApplicationCheck.genesis\_\_Application\_\_c = oldApplist[0].Id;

// deptApplicationCheck.genesis\_\_Status\_\_c = TTConstants.ACTIVE;

// insert deptApplicationCheck;

// List< genesis\_\_Application\_Department\_\_c> assertDept = [ SELECT Id ,genesis\_\_Department\_\_c FROM genesis\_\_Application\_Department\_\_c];

// System.assertEquals(true, !assertDept.isEmpty(), 'Application Department not inserted');

// List<Task> taskList = new List<Task>();

// Task tasks = new Task();

// tasks.genesis\_\_Application\_\_c = oldApplist[0].Id;

// tasks.genesis\_\_Department\_\_c = assertDept[0].genesis\_\_Department\_\_c;

// tasks.Status = 'Open';

// tasks.Subject = 'Urgent Task-Increase Loan amount for Further Advance';

// taskList.add(tasks);

// insert taskList;

// Map<Id, genesis\_\_Applications\_\_c> oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

// Map<Id, genesis\_\_Applications\_\_c> newMap = new Map<Id, genesis\_\_Applications\_\_c>();

// oldMap.put(oldApplist[0].Id, oldApplist[0]);

// List<genesis\_\_Applications\_\_c> newApplist = [SELECT Id,

// OwnerId,

// LTVP\_\_c,

// Is\_Further\_Advance\_App\_\_c,

// Brand\_\_c,

// genesis\_\_Asset\_Class\_\_c,

// genesis\_\_Expected\_Close\_Date\_\_c,

// Record\_Type\_Name\_\_c,

// Is\_Simpology\_Application\_\_c,

// Account\_Name\_\_c,

// Penciled\_In\_Status\_\_c,

// genesis\_\_Status\_\_c,

// Thinktank\_Margin\_\_c,

// genesis\_\_Interest\_Index\_\_c,

// recordtypeid, genesis\_\_CL\_Product\_\_c,

// genesis\_\_Loan\_Amount\_\_c,

// genesis\_\_CL\_Purpose\_\_c,

// Broker\_Trail\_\_c,

// genesis\_\_Requested\_Loan\_Amount\_\_c,

// Rate\_Index\_\_c,

// genesis\_\_CLTV\_\_c,

// Aggregator\_\_c,

// Total\_Fee\_Amount\_inc\_GST\_\_c,

// Funding\_Date\_\_c,

// genesis\_\_Disbursement\_Date\_\_c,

// genesis\_\_Account\_\_c,

// Borrower\_Name\_\_c,

// genesis\_\_Total\_Fee\_Amount\_\_c,

// Workflow\_Status\_\_c,

// Pricing\_Basis\_\_c,

// genesis\_\_Interest\_Rate\_\_c,

// genesis\_\_Term\_\_c,

// Execution\_Flow\_\_c,

// genesis\_\_Total\_Facility\_Amount\_\_c,

// NCCP\_\_c,

// createddate,

// Consolidated\_Purpose\_\_c,

// genesis\_\_Parent\_Application\_\_c,

// Integration\_Aggregator\_\_c

// FROM genesis\_\_Applications\_\_c

// WHERE Record\_Type\_Name\_\_c = 'Package'

// AND Execution\_Flow\_\_r.Name != 'Residential Loan Flow - Simpology'];

// newApplist[0].Aggregator\_\_c = acc[0].Id;

// newApplist[0].OwnerId = queueType.Id;

// newApplist[0].genesis\_\_Status\_\_c = 'CHANGE MEMO';

// newApplist[0].genesis\_\_CL\_Product\_\_c = assertproductCommercial[0].Id;

// update newApplist;

// newMap.put(newApplist[0].Id, newApplist[0]);

// Profile profileList = [SELECT Id,

// Name

// FROM Profile

// WHERE name = 'System Administrator'];

// User adminUser = new User();

// adminUser.Alias = 'tt-test';

// adminUser.Email = 'tt-test-admin-user@123.com';

// adminUser.EmailEncodingKey = 'UTF-8';

// adminUser.FirstName = 'TTTEST';

// adminUser.LastName = 'ADMIN';

// adminUser.LanguageLocaleKey = 'en\_US';

// adminUser.LocaleSidKey = 'en\_US';

// adminUser.ProfileId = profileList.Id;

// adminUser.TimeZoneSidKey = 'America/Los\_Angeles';

// adminUser.UserName = 'tt-test-admin-user@123.com';

// adminUser.IsActive = true;

// insert adminUser;

// Test.startTest();

// System.runAs(adminUser){

// GenesisApplicationTriggerHandlerV2.recalculateFeeFuture(newApplist[0].Id);

// }

// Test.stopTest();

// }

/\*\*

\* @description Tests various aggregator to brand mapping scenarios for package applications

\* @author Owen Rees | 01-07-2024

\*\*/

// @isTest

// private static void updateBrandingTest() {

// genesis\_\_Applications\_\_c packageApp = [

// SELECT Id

// FROM genesis\_\_Applications\_\_c

// WHERE Record\_Type\_Name\_\_c = :TTConstants.APP\_PACKAGE

// AND Settlement\_Notes\_\_c = 'psuedo unique identifier 1001'

// LIMIT 1

// ];

// Account dualAccredAggregator = [

// SELECT Id

// FROM Account

// WHERE Name = 'Rate Money Pty Ltd'

// AND Type = 'Aggregator'

// LIMIT 1

// ];

// Account dualAccredBrokerAcc = [

// SELECT Id

// FROM Account

// WHERE Name = 'Rate Money Pty Ltd'

// AND Type = 'Broker'

// LIMIT 1

// ];

// Contact dualAccredBrokerCon = [

// SELECT Id

// FROM Contact

// WHERE Name = 'Owen Rees'

// LIMIT 1

// ];

// List<clcommon\_\_CL\_Product\_\_c> assertproductCommercial = [

// SELECT Id

// FROM clcommon\_\_CL\_Product\_\_c

// WHERE clcommon\_\_Product\_Name\_\_c = 'SMSF Commercial Master'

// ];

// // Below set to invoke the COMP-100 brand mapping

// packageApp.Aggregator\_\_c = dualAccredAggregator.Id;

// packageApp.Broker\_Relationship\_\_c = dualAccredBrokerAcc.Id;

// packageApp.Broker\_Contact\_\_c = dualAccredBrokerCon.Id;

// packageApp.genesis\_\_CL\_Product\_\_c = assertproductCommercial[0].Id;

// update packageApp;

// // Variables named to match the brandMapping method call in the handler class

// Set<Id> updateBrand = new Set<Id>();

// List<genesis\_\_Applications\_\_c> allParentAppsRead = new List <genesis\_\_Applications\_\_c>();

// Map<Id, genesis\_\_Applications\_\_c> allParentAppsMap = new Map<Id, genesis\_\_Applications\_\_c>();

// allParentAppsRead = [

// SELECT Id,

// Name,

// OwnerId,

// genesis\_\_Status\_\_c,

// Workflow\_Status\_\_c,

// Record\_Type\_Name\_\_c,

// Brand\_\_c,

// Fee\_Set\_\_c,

// Fee\_Set\_\_r.Name,

// Aggregator\_\_r.Name,

// Brand\_Type\_\_c,

// Rate\_Index\_\_c,

// Funding\_Date\_\_c,

// NCCP\_\_c,

// Solicitor\_Firm\_\_c,

// Date\_Solicitor\_Instructed\_\_c,

// Conditional\_Credit\_Underwriting\_Date\_\_c,

// Ready\_To\_Instruct\_Date\_\_c,

// genesis\_\_Asset\_Class\_\_c,

// Invalid\_Queue\_User\_\_c,

// Total\_Fee\_Amount\_inc\_GST\_\_c,

// genesis\_\_Total\_Fee\_Amount\_\_c,

// Borrower\_Age\_\_c,

// Borrower\_Age\_at\_term\_end\_\_c,

// Consolidated\_Purpose\_\_c,

// genesis\_\_CL\_Product\_\_r.Document\_Type\_\_c,

// LTVP\_\_c,

// Broker\_Trail\_\_c,

// genesis\_\_Total\_Facility\_Amount\_\_c,

// genesis\_\_Requested\_Loan\_Amount\_\_c,

// Is\_Simpology\_Application\_\_c,

// genesis\_\_CL\_Product\_\_r.Product\_Type\_\_c,

// Total\_Collateral\_Amount\_\_c,

// genesis\_\_CLTV\_\_c,

// Is\_Further\_Advance\_App\_\_c,

// Penciled\_In\_Status\_\_c,

// genesis\_\_Expected\_Close\_Date\_\_c,

// genesis\_\_Disbursement\_Date\_\_c,

// Broker\_Relationship\_\_r.Brand\_Name\_\_c,

// Broker\_Relationship\_\_r.Dual\_Accreditation\_\_c,

// Broker\_Relationship\_\_r.Direct\_Accreditation\_\_c,

// Resi\_COMMSMSF\_Combination\_\_c

// FROM genesis\_\_Applications\_\_c

// WHERE Record\_Type\_Name\_\_c = :TTConstants.APP\_PACKAGE

// AND Settlement\_Notes\_\_c = 'psuedo unique identifier 1001'

// LIMIT 1

// ];

// for(genesis\_\_Applications\_\_c app : allParentAppsRead) {

// updateBrand.add(app.Id);

// allParentAppsMap.put(app.Id, app);

// }

// // Dummy variables so we can construct handler object

// List<genesis\_\_Applications\_\_c> dummy\_newApplist = new List<genesis\_\_Applications\_\_c>();

// Map<Id, genesis\_\_Applications\_\_c> dummy\_oldMap = new Map<Id, genesis\_\_Applications\_\_c>();

// GenesisApplicationTriggerHandlerV2 handlerObj = new GenesisApplicationTriggerHandlerV2(allParentAppsRead, allParentAppsMap, 'AFTER\_UPDATE');

// handlerObj.updateBranding(updateBrand, allParentAppsRead, allParentAppsMap);

// genesis\_\_Applications\_\_c updatedPackageApp = [

// SELECT Id, Brand\_\_c

// FROM genesis\_\_Applications\_\_c

// WHERE Record\_Type\_Name\_\_c = :TTConstants.APP\_PACKAGE

// AND Settlement\_Notes\_\_c = 'psuedo unique identifier 1001'

// LIMIT 1

// ];

// Assert.areEqual(

// updatedPackageApp.Brand\_\_c, TTConstants.THINKTANK,

// 'Package app Brand is not set to: ' + TTConstants.THINKTANK

// );

// // For default brand mapping

// dualAccredBrokerAcc.Direct\_Accreditation\_\_c = false;

// dualAccredBrokerAcc.Dual\_Accreditation\_\_c = false;

// update dualAccredBrokerAcc;

// allParentAppsRead = [

// SELECT Id,

// Name,

// OwnerId,

// genesis\_\_Status\_\_c,

// Workflow\_Status\_\_c,

// Record\_Type\_Name\_\_c,

// Brand\_\_c,

// Fee\_Set\_\_c,

// Fee\_Set\_\_r.Name,

// Aggregator\_\_r.Name,

// Brand\_Type\_\_c,

// Rate\_Index\_\_c,

// Funding\_Date\_\_c,

// NCCP\_\_c,

// Solicitor\_Firm\_\_c,

// Date\_Solicitor\_Instructed\_\_c,

// Conditional\_Credit\_Underwriting\_Date\_\_c,

// Ready\_To\_Instruct\_Date\_\_c,

// genesis\_\_Asset\_Class\_\_c,

// Invalid\_Queue\_User\_\_c,

// Total\_Fee\_Amount\_inc\_GST\_\_c,

// genesis\_\_Total\_Fee\_Amount\_\_c,

// Borrower\_Age\_\_c,

// Borrower\_Age\_at\_term\_end\_\_c,

// Consolidated\_Purpose\_\_c,

// genesis\_\_CL\_Product\_\_r.Document\_Type\_\_c,

// LTVP\_\_c,

// Broker\_Trail\_\_c,

// genesis\_\_Total\_Facility\_Amount\_\_c,

// genesis\_\_Requested\_Loan\_Amount\_\_c,

// Is\_Simpology\_Application\_\_c,

// genesis\_\_CL\_Product\_\_r.Product\_Type\_\_c,

// Total\_Collateral\_Amount\_\_c,

// genesis\_\_CLTV\_\_c,

// Is\_Further\_Advance\_App\_\_c,

// Penciled\_In\_Status\_\_c,

// genesis\_\_Expected\_Close\_Date\_\_c,

// genesis\_\_Disbursement\_Date\_\_c,

// Broker\_Relationship\_\_r.Brand\_Name\_\_c,

// Broker\_Relationship\_\_r.Dual\_Accreditation\_\_c,

// Broker\_Relationship\_\_r.Direct\_Accreditation\_\_c,

// Resi\_COMMSMSF\_Combination\_\_c

// FROM genesis\_\_Applications\_\_c

// WHERE Record\_Type\_Name\_\_c = :TTConstants.APP\_PACKAGE

// AND Settlement\_Notes\_\_c = 'psuedo unique identifier 1001'

// LIMIT 1

// ];

// allParentAppsMap.clear();

// for(genesis\_\_Applications\_\_c app : allParentAppsRead) {

// allParentAppsMap.put(app.Id, app);

// }

// handlerObj.updateBranding(updateBrand, allParentAppsRead, allParentAppsMap);

// updatedPackageApp = [

// SELECT Id, Brand\_\_c, Aggregator\_\_r.Name

// FROM genesis\_\_Applications\_\_c

// WHERE Record\_Type\_Name\_\_c = :TTConstants.APP\_PACKAGE

// AND Settlement\_Notes\_\_c = 'psuedo unique identifier 1001'

// LIMIT 1

// ];

// List<Aggregator\_Brand\_Mapping\_\_mdt> brandMappings = [

// SELECT Id,

// Aggregator\_\_c,

// Brand\_\_c,

// Residential\_Commercial\_\_c

// FROM Aggregator\_Brand\_Mapping\_\_mdt

// WHERE Broker\_Branding\_\_c = false];

// String appResiCommType = TTConstants.RESI\_PROD\_TYPE;

// String aggregatorName = updatedPackageApp.Aggregator\_\_r.Name;

// String brandAssert = '';

// for (Aggregator\_Brand\_Mapping\_\_mdt brandMapping : brandMappings) {

// if (aggregatorName == brandMapping.Aggregator\_\_c &&

// appResiCommType == brandMapping.Residential\_Commercial\_\_c

// ) {

// brandAssert = brandMapping.Brand\_\_c;

// }

// }

// Assert.areNotEqual(

// '', brandAssert,

// 'Aggregator\_Brand\_Mapping\_\_mdt match not found'

// );

// Assert.areEqual(

// updatedPackageApp.Brand\_\_c, brandAssert,

// 'Package app Brand is not set to: ' + brandAssert

// );

// // For TB-1617 broker company branding (e.g. LJ Hooker)

// dualAccredBrokerAcc.Brand\_Name\_\_c = 'LJ Hooker';

// update dualAccredBrokerAcc;

// allParentAppsRead = [

// SELECT Id,

// Name,

// OwnerId,

// genesis\_\_Status\_\_c,

// Workflow\_Status\_\_c,

// Record\_Type\_Name\_\_c,

// Brand\_\_c,

// Fee\_Set\_\_c,

// Fee\_Set\_\_r.Name,

// Aggregator\_\_r.Name,

// Brand\_Type\_\_c,

// Rate\_Index\_\_c,

// Funding\_Date\_\_c,

// NCCP\_\_c,

// Solicitor\_Firm\_\_c,

// Date\_Solicitor\_Instructed\_\_c,

// Conditional\_Credit\_Underwriting\_Date\_\_c,

// Ready\_To\_Instruct\_Date\_\_c,

// genesis\_\_Asset\_Class\_\_c,

// Invalid\_Queue\_User\_\_c,

// Total\_Fee\_Amount\_inc\_GST\_\_c,

// genesis\_\_Total\_Fee\_Amount\_\_c,

// Borrower\_Age\_\_c,

// Borrower\_Age\_at\_term\_end\_\_c,

// Consolidated\_Purpose\_\_c,

// genesis\_\_CL\_Product\_\_r.Document\_Type\_\_c,

// LTVP\_\_c,

// Broker\_Trail\_\_c,

// genesis\_\_Total\_Facility\_Amount\_\_c,

// genesis\_\_Requested\_Loan\_Amount\_\_c,

// Is\_Simpology\_Application\_\_c,

// genesis\_\_CL\_Product\_\_r.Product\_Type\_\_c,

// Total\_Collateral\_Amount\_\_c,

// genesis\_\_CLTV\_\_c,

// Is\_Further\_Advance\_App\_\_c,

// Penciled\_In\_Status\_\_c,

// genesis\_\_Expected\_Close\_Date\_\_c,

// genesis\_\_Disbursement\_Date\_\_c,

// Broker\_Relationship\_\_r.Brand\_Name\_\_c,

// Broker\_Relationship\_\_r.Dual\_Accreditation\_\_c,

// Broker\_Relationship\_\_r.Direct\_Accreditation\_\_c,

// Resi\_COMMSMSF\_Combination\_\_c

// FROM genesis\_\_Applications\_\_c

// WHERE Record\_Type\_Name\_\_c = :TTConstants.APP\_PACKAGE

// AND Settlement\_Notes\_\_c = 'psuedo unique identifier 1001'

// LIMIT 1

// ];

// allParentAppsMap.clear();

// for(genesis\_\_Applications\_\_c app : allParentAppsRead) {

// allParentAppsMap.put(app.Id, app);

// }

// handlerObj.updateBranding(updateBrand, allParentAppsRead, allParentAppsMap);

// updatedPackageApp = [

// SELECT Id, Brand\_\_c

// FROM genesis\_\_Applications\_\_c

// WHERE Record\_Type\_Name\_\_c = :TTConstants.APP\_PACKAGE

// AND Settlement\_Notes\_\_c = 'psuedo unique identifier 1001'

// LIMIT 1

// ];

// Assert.areEqual(

// updatedPackageApp.Brand\_\_c, 'LJ Hooker',

// 'Package app Brand is not set to: LJ Hooker'

// );

// }

}