UpdateContext(

 {

 MELoading: true,

 CurrentBuildScheduleType: CurrentEntryActivityDefinition.BuildSchedule,

 CurrentCustomQuestionID: CurrentEntryActivityDefinition.QuestionGroupID,

 QuestionMEIID: CurrentEntryMonthlyEntryInstance.MonthlyEntryInstanceID,

 QuestionMESN: CurrentEntryMonthlyEntryInstance.MonthlyEntryInstanceSequenceNumber,

 MaxSessionID: Max(

 '[dbo].[SessionInstance]',

 SessionInstanceID

 )

 }

);

If(

 !IsBlank(CurrentCustomQuestionID) && CurrentCustomQuestionID <> 0 && !MEDelete,

 'PC-PopulateCustomQuestions'.Run()

);

If(

 MonthlyEntryType = "New" || (Coalesce(

 CurrentEntryActivityDefinition.LinkedFRActivityDefinition1,

 0

 ) = 0 && Coalesce(

 CurrentEntryActivityDefinition.LinkedFRActivityDefinition2,

 0

 ) = 0 && CurrentEntryMonthlyEntry.FRManageSessionsFieldAccess = "Locked"),

 Collect(

 '[dbo].[SessionInstance]',

 {

 SessionInstanceID: MaxSessionID + CountRows(

 Filter(

 MECurrentSessionInstances,

 SessionInstanceID <> 0

 )

 ) + 1,

 SessionInstanceSequenceNumber: 0,

 SessionStandardID: 0,

 SessionStandardSequenceNumber: 0,

 SessionInstanceActive: false,

 ActivityType: "temp"

 }

 );

 Collect(

 '[dbo].[SessionInstance]',

 DropColumns(

 UpdateIf(

 Filter(

 MECurrentSessionInstances,

 SessionInstanceID <> 0

 ),

 true,

 {SessionInstanceID: SessionInstanceID + MaxSessionID}

 ),

 "Updated"

 )

 );

 RemoveIf(

 '[dbo].[SessionInstance]',

 SessionInstanceID = MaxSessionID + CountRows(

 Filter(

 MECurrentSessionInstances,

 SessionInstanceID <> 0

 )

 ) + 1

 ),

 Collect(

 '[dbo].[SessionInstance]',

 DropColumns(

 UpdateIf(

 Filter(

 MECurrentSessionInstances,

 SessionInstanceID <> 0

 ),

 true,

 {SessionInstanceSequenceNumber: SessionInstanceSequenceNumber + 1}

 ),

 "Updated"

 )

 )

);

ForAll(

 Filter(

 MECurrentSessionInstances,

 SessionInstanceID <> 0

 ),

 Collect(

 '[dbo].[FlexRegProgramSessions]',

 {

 FlexRegProgramSessionsActive: true,

 MonthlyEntryInstanceID: QuestionMEIID,

 MonthlyEntryInstanceSequenceNumber: QuestionMESN,

 SessionInstanceID: SessionInstanceID,

 SessionInstanceSequenceNumber: SessionInstanceSequenceNumber

 }

 )

);

ForAll(

 SessionInstancesToInactivate,

 Patch(

 '[dbo].[FlexRegProgramSessions]',

 LookUp(

 '[dbo].[FlexRegProgramSessions]',

 SessionInstanceID = TEMPSessionInstanceID && SessionInstanceSequenceNumber = TEMPSessionInstanceSequenceNumber

 ),

 {FlexRegProgramSessionsActive: false}

 );

 Patch(

 '[dbo].[SessionInstance]',

 LookUp(

 '[dbo].[SessionInstance]',

 SessionInstanceID = TEMPSessionInstanceID && SessionInstanceSequenceNumber = TEMPSessionInstanceSequenceNumber

 ),

 {SessionInstanceActive: false}

 )

);

RemoveIf(

 MECurrentSessionInstances,

 SessionInstanceID <> 0

);/\*This section handles sessions that were created manually, which is identified by SessionInstanceID = 0\*/

If(

 CountRows(

 Filter(

 MECurrentSessionInstances,

 SessionInstanceID = 0

 )

 ) > 0,

 Refresh('[dbo].[SessionInstance]');

 UpdateContext(

 {

 MaxSessionID: Max(

 '[dbo].[SessionInstance]',

 SessionInstanceID

 )

 }

 );

 Collect(

 '[dbo].[SessionInstance]',

 {

 SessionInstanceID: MaxSessionID + CountRows(

 Filter(

 MECurrentSessionInstances,

 SessionInstanceID = 0

 )

 ) + 1,

 SessionInstanceSequenceNumber: 0,

 SessionStandardID: 0,

 SessionStandardSequenceNumber: 0,

 SessionInstanceActive: false,

 ActivityType: "temp"

 }

 );

 Clear(MENewSessionInstances);

 ForAll(

 Filter(

 MECurrentSessionInstances,

 SessionInstanceID = 0

 ),

 Collect(

 MENewSessionInstances,

 {

 SessionInstanceID: MaxSessionID + CountRows(MENewSessionInstances) + 1,

 SessionInstanceSequenceNumber: SessionInstanceSequenceNumber,

 SessionStandardID: SessionStandardID,

 SessionStandardSequenceNumber: SessionStandardSequenceNumber,

 SessionInstanceActive: SessionInstanceActive,

 SessionName: SessionName,

 SessionDescription: SessionDescription,

 ActivityType: CurrentEntryActivityDefinition.ActivityType,

 StartDate: StartDate,

 EndDate: EndDate,

 StartTime: StartTime,

 EndTime: EndTime,

 CreatedBy: CreatedBy,

 CreatedDate: CreatedDate,

 MinAge: MinAge,

 MaxAge: MaxAge,

 MinParticipants: MinParticipants,

 MaxParticipants: MaxParticipants,

 FacilityMemberFee: FacilityMemberFee,

 CommunityMemberFee: CommunityMemberFee,

 WrapAroundFacility1: WrapAroundFacility1,

 WrapAroundFacility2: WrapAroundFacility2,

 WrapAroundFacility3: WrapAroundFacility3,

 WrapAroundFacility4: WrapAroundFacility4,

 WrapAroundFacility5: WrapAroundFacility5

 }

 )

 );

 Collect(

 '[dbo].[SessionInstance]',

 MENewSessionInstances

 );

 RemoveIf(

 '[dbo].[SessionInstance]',

 SessionInstanceID = MaxSessionID + CountRows(

 Filter(

 MECurrentSessionInstances,

 SessionInstanceID = 0

 )

 ) + 1

 );

 ForAll(

 MENewSessionInstances,

 Collect(

 '[dbo].[FlexRegProgramSessions]',

 {

 FlexRegProgramSessionsActive: true,

 MonthlyEntryInstanceID: QuestionMEIID,

 MonthlyEntryInstanceSequenceNumber: QuestionMESN,

 SessionInstanceID: SessionInstanceID,

 SessionInstanceSequenceNumber: SessionInstanceSequenceNumber

 }

 )

 )

);

ClearCollect(

 TEMPCurrentBuildScheduleInstance,

 Sort(

 Filter(

 '[dbo].[BuildScheduleInstance]',

 BuildScheduleType = CurrentBuildScheduleType

 ),

 BuildStartDate,

 Descending

 )

);

ClearCollect(

 CurrentBuildScheduleInstance,

 First(

 Sort(

 Filter(

 TEMPCurrentBuildScheduleInstance,

 DateDiff(

 BuildStartDate,

 CurrentStartDate,

 Days

 ) >= 0

 ),

 BuildStartDate,

 Descending

 )

 )

);

If(

 CountRows(CurrentBuildScheduleInstance) = 0 || DateDiff(

 Today(),

 First(CurrentBuildScheduleInstance).EntryDeadline,

 Days

 ) < 0 && !CurrentEntryMonthlyEntryInstance.PendingApproval,

 'ProgramCatalog-Sendanemailwhenbuildrequestisafterthedeadline'.Run(

 CurrentUser.displayName,

 CurrentStartDate,

 CurrentSite,

 CurrentEntryActivityDefinition.ActivityDefinitionName

 )

);

If(

 MonthlyEntryType = "Edit" && !CurrentEntryMonthlyEntryInstance.PendingApproval,

 SubmitForm(MEPreviousSN)

);

'ProgramCatalog-CallPopulateScheduleView'.Run();

If(

 CurrentEntryMonthlyEntryInstance.PendingApproval,

 ClearCollect(

 ApproverList,

 {ApproverEmails: "first"}

 );

 ForAll(

 Filter(

 '[dbo].[Users]',

 UserBranch = CurrentSite && BranchApprover && UserActive

 ),

 If(

 First(ApproverList).ApproverEmails <> "first",

 Patch(

 ApproverList,

 First(ApproverList),

 {

 ApproverEmails: Concatenate(

 First(ApproverList).ApproverEmails,

 "; "

 )

 }

 ),

 Patch(

 ApproverList,

 First(ApproverList),

 {ApproverEmails: ""}

 )

 );

 Patch(

 ApproverList,

 First(ApproverList),

 {

 ApproverEmails: Concatenate(

 First(ApproverList).ApproverEmails,

 UserEmail

 )

 }

 )

 );

 'PC-Emailforlatefeebasedrequest'.Run(

 "not being used",

 If(

 First(ApproverList).ApproverEmails = "first",

 "info@seattleymca.org",

 First(ApproverList).ApproverEmails

 ),

 CurrentUser.mail,

 BuildName,

 StartDate,

 CurrentEntryMonthlyEntryInstance.MonthlyEntryInstanceID,

 LateReason,

 First(CurrentBuildScheduleInstance).EntryDeadline,

 CurrentEntryMonthlyEntryInstance.MonthlyEntryInstanceSequenceNumber,

 If(

 MEDelete,

 "delete an",

 If(

 CurrentEntryMonthlyEntryInstance.MonthlyEntryInstanceSequenceNumber = 1,

 "build a new",

 "edit an existing"

 )

 )

 )

);

If(

 !IsBlank(CurrentCustomQuestionID) && CurrentCustomQuestionID <> 0 && !MEDelete,

 UpdateContext({MEGoToCustomQuestionPopup: true}),

 UpdateContext({MEDelete: false});

 Navigate(

 HomeScreen,

 ScreenTransition.None

 )

);

UpdateIf(

 CurrentNonStandardCharges,

 true,

 {

 InstanceID: QuestionMEIID,

 InstanceSequenceNumber: QuestionMESN

 }

);

Collect(

 '[dbo].[InstanceCharges]',

 DropColumns(

 CurrentNonStandardCharges,

 "InstanceChargeID"

 )

);

If(

 MonthlyEntryType = "Edit",

 ForAll(

 RenameColumns(

 CurrentNonStandardCharges,

 "InstanceChargeID",

 "TEMPInstanceChargeID"

 ),

 If(

 !IsBlank(TEMPInstanceChargeID),

 Patch(

 '[dbo].[InstanceCharges]',

 LookUp(

 '[dbo].[InstanceCharges]',

 InstanceChargeID = TEMPInstanceChargeID

 ),

 {InstanceChargeActive: false}

 )

 )

 )

);

UpdateContext({MELoading: false})