using System;

using System.Collections.Generic;

using DataCollectorScriptLibrary;

using Reporter.Model;

using User = DataCollectorScriptLibrary.User;

public class TRN\_ABC\_report : InnerCode

{

public TRN\_ABC\_report(CollectionContainer container)

: base(container)

{

}

#region Code

#region Configuration

#region Load

//#LOADSNIPPET "TRN\_ABC\_snippet"

/\*

#LOAD "Utilities\Utilities.ReportLogProvider.cs"

#LOAD "Utilities\Utilities.IntervalTimerProvider.cs"

#LOAD "Utilities\Utilities.HierarchyCacheProvider.cs"

#LOAD "Utilities\Utilities.ProductivityProvider.cs"

#LOAD "Utilities\Utilities.WorkItemExtConcatenatorProvider.cs"

#LOAD "Utilities\Utilities.WorkItemExtConcatenatorProviderLinear.cs"

#LOAD "Utilities\Utilities.ReportControlProvider.cs"

#LOAD "Utilities\Utilities.TasknameDecoratorProvider.cs"

#LOAD "BaseReports\BaseReports.Custom.cs"

#LOAD "BaseReports\BaseReports.SimpleAggregator.cs"

#LOAD "Utilities\Utilities.ConfigurationControlProvider.cs"

#LOAD "Utilities\Utilities.Utilities.cs"

#LOAD "Utilities\Utilities.WorktimeSettingsProvider.cs"

\*/

//#LOAD "Utilities\Utilities.Default\_KPI\_Snippet.cs"

//#LOAD "Utilities\Utilities.HTML\_Design\_Snippet.cs"

#endregion

/\*\*\*\*\*\* HierarchyCacheProviderConfig \*\*\*\*\*\*/

private const string HierarchyCacheProviderItemSeparator = " >> ";

private const string HierarchyCacheProviderEncloserFormat = "{1} ({0})";

/\*\*\*\*\*\* WorkitemExtConcatenatorConfig \*\*\*\*\*\*/

Dictionary<ItemType, int> ItemTypeImportance = new Dictionary<ItemType, int>

{

{ItemType.Holiday, 1},

{ItemType.SickLeave, 2},

{ItemType.Manual, 3},

{ItemType.CalendarMeeting, 4},

{ItemType.AdhocMeeting, 5},

{ItemType.Mobile, 6},

{ItemType.Pc, 7},

};

public HashSet<int> ExcludedTaskIds = new HashSet<int>();

public HashSet<ItemType> ExcludedWorkItemTypes = new HashSet<ItemType>();

public double TaskIdsWeight = 1;

public Dictionary<int, int> PositiveTaskIds = new Dictionary<int, int>();

public Dictionary<int, int> NegativeTaskIds = new Dictionary<int, int>();

public double ItemTypeImportanceWeight = 100;

public bool EnableNetting = true;

public bool EnableIgnoreHolidays = true;

#endregion

public TRN\_ABC\_snippet snippetVariable;

#region OutputTypeSwitches

public bool OutputHtml = false;

public bool OutputKpi = true;

public bool OutputXls = true;

public bool OutputEV = true;

public bool OutputNorma = true;

public bool WriteDebug = false;

public bool bigData = false;

public List<int> GroupLevels = new List<int>() { 2, 3 };

#endregion

public void Execute()

{

snippetVariable = new TRN\_ABC\_snippet(this.collectionContainer);

helper = this;

snippetVariable.helper = this;

snippetVariable.GroupLevels = GroupLevels;

snippetVariable.ReportStartDate = helper.GetReportContext().LocalStartDate;

snippetVariable.ReportEndDate = helper.GetReportContext().LocalEndDate;

snippetVariable.OutputHtml = this.OutputHtml;

snippetVariable.OutputKpi = this.OutputKpi;

snippetVariable.OutputXls = this.OutputXls;

snippetVariable.OutputEV = this.OutputEV;

snippetVariable.OutputNorma = this.OutputNorma;

snippetVariable.WriteDebug = this.WriteDebug;

snippetVariable.bigData = this.bigData;

//snippetVariable.Tasks = Tasks;

//snippetVariable.Productivity = this.Productivity;

if (!Initialize()) return;

if (!snippetVariable.Initialize()) return;

IntervalTimer.StartTimer("Main.Execute");

// Write the function if you need pre process before fetching the tups

snippetVariable.PreProcess();

var tempUserId = -1;

var tempDateSplit = DateTime.MinValue.Date;

IntervalTimer.StartTimer("Main.Execute.ProcessTups");

foreach (var tup in GetNextTup())

{

var Xtup = new WorkItemExt(tup);

// Change user or day

if (EnableNetting)

{

if (tup.UserId != tempUserId || UtcToLocalDate(tup.StartDate.Date) != tempDateSplit)

{

var WorkItemList = WorkItemExtConcatenator.Flush();

if (WorkItemList != null)

{

DistributeXtups(WorkItemList);

}

tempUserId = tup.UserId;

tempDateSplit = UtcToLocalDate(tup.StartDate.Date);

}

}

WorkItemExtConcatenator.SetXtupAttributes(Xtup);

// Write the function if you need to pre process the tup before sending to be processed

// TupPreprocess();

IntervalTimer.StartTimer("Main.Execute.ProcessTup");

if (EnableNetting)

{

var WorkItemList = WorkItemExtConcatenator.Push(Xtup);

if (WorkItemList != null)

{

DistributeXtups(WorkItemList);

}

// DistributeXtups( ?? new List<WorkItemExt>());

}

else

{

DistributeXtups(new List<WorkItemExt> { Xtup });

}

IntervalTimer.StopTimer("Main.Execute.ProcessTup");

// If you want to process the processed Xtups you can use this, note that at this point you have WorkItemExt types

/\*

\* var TupsWithNetValues = DistibuteXtups(WorkItemExtConcatenator.Push(new WorkItemExt(tup)));

\* XtupPostProcess(TupsWithNetValues);

\*/

// Note that you have to implement this after the Flush also (so use a function)

}

IntervalTimer.StopTimer("Main.Execute.ProcessTups");

if (EnableNetting)

{

var WorkItemList = WorkItemExtConcatenator.Flush();

if (WorkItemList != null)

{

DistributeXtups(WorkItemList);

}

}

// Write the function if you need post process after fetching the tups

snippetVariable.PostProcess();

var ShowReport = snippetVariable.WriteOutput() > 0;

IntervalTimer.StopTimer("Main.Execute");

if (ShowReport)

{

ReportLogger.OutputReportLog();

IntervalTimer.OutputIntervalTimes();

SetExcelFileName();

var LogSheet = helper.dataSet.Tables["Logs"];

if (LogSheet != null)

{

LogSheet.Hide = true;

}

}

else

{

helper.SetShowReport(false);

}

}

public List<WorkItemExt> DistributeXtups(List<WorkItemExt> XtupList)

{

// Distribute the processed tups as you wish or process with business logic

// var CollectedKeySet = new HashSet<string>();

foreach (var Xtup in XtupList)

{

snippetVariable.doProcess(Xtup);

}

return XtupList;

}

public InnerCode helper;

public WorkItemExtConcatenatorConfig ConcatenatorConfig = new WorkItemExtConcatenatorConfig();

public WorkItemExtConcatenatorProviderLinear WorkItemExtConcatenator;

public HierarchyCacheProviderConfig HierarchyCacheProviderDefaultConfig = new HierarchyCacheProviderConfig();

public HierarchyCacheProvider<UserExt, User> Users;

public HierarchyCacheProvider<TaskExt, Task> Tasks;

public IntervalTimerProvider IntervalTimer;

public ReportLogProvider ReportLogger;

public ProductivityProvider Productivity;

public WorktimeSettingsProvider WorktimeSettings;

public RuntimeContextProvider RuntimeContext;

#region Initialization

public bool Configure()

{

ConcatenatorConfig.ItemTypeImportance = ItemTypeImportance;

ConcatenatorConfig.ExcludedTaskIds = ExcludedTaskIds;

ConcatenatorConfig.ExcludedWorkItemTypes = ExcludedWorkItemTypes;

ConcatenatorConfig.TaskIdsWeight = TaskIdsWeight;

ConcatenatorConfig.PositiveTaskIds = PositiveTaskIds;

ConcatenatorConfig.NegativeTaskIds = NegativeTaskIds;

ConcatenatorConfig.DisableNetting = !EnableNetting;

ConcatenatorConfig.EnableIgnoreHolidays = EnableIgnoreHolidays;

HierarchyCacheProviderDefaultConfig.Separator = HierarchyCacheProviderItemSeparator;

HierarchyCacheProviderDefaultConfig.EncloserFormat = HierarchyCacheProviderEncloserFormat;

return true;

}

public bool Initialize()

{

if (!Configure()) return false;

try

{

IntervalTimer = new IntervalTimerProvider(helper);

ReportLogger = new ReportLogProvider("Central", helper);

RuntimeContext = new RuntimeContextProvider(helper, ReportLogger, IntervalTimer);

Users = new HierarchyCacheProvider<UserExt, User>("Users Cache", HierarchyCacheProviderDefaultConfig,

RuntimeContext);

Tasks = new HierarchyCacheProvider<TaskExt, Task>("Tasks Cache", HierarchyCacheProviderDefaultConfig,

RuntimeContext);

Productivity = new ProductivityProvider("Productivity", RuntimeContext, Users, Tasks, ReportLogger.Log,

null);

WorktimeSettings = new WorktimeSettingsProvider("Worktime settings", RuntimeContext);

WorkItemExtConcatenator = new WorkItemExtConcatenatorProviderLinear(ConcatenatorConfig, RuntimeContext,

Productivity, WorktimeSettings);

}

catch (Exception e)

{

if (ReportLogger != null && ReportLogger.LogHasContent)

{

ReportLogger.OutputReportLog();

}

helper.Log(string.Format("{0} exception caught during initialization.", e));

return false;

}

return true;

}

public void SetExcelFileName()

{

var repContext = helper.GetReportContext();

helper.SetFileName(string.Format("{0}\_{1}\_{2}\_{3}", repContext.DefinitionName, repContext.CompanyName,

repContext.LocalStartDate.ToString("yyyMMdd"),

repContext.LocalEndDate.ToString("yyyMMdd")));

}

#endregion

#endregion

}