using System;

using System.Threading;

using System.Xml;

using System.Linq;

using System.Collections.Generic;

using System.Reflection;

using System.IO;

using System.Data.SqlClient;

using Microsoft.SqlServer.Dts.Runtime;

using Microsoft.SqlServer.Dts.Tasks.ExecuteSQLTask;

using Microsoft.SqlServer.Dts.Tasks.FileSystemTask;

using Microsoft.SqlServer.Dts.Tasks.ScriptTask;

using Microsoft.SqlServer.Dts.Tasks.ExecuteProcess;

using Microsoft.SqlServer.Dts.Tasks.SendMailTask;

using Microsoft.SqlServer.Dts.Tasks.ExpressionTask;

using Microsoft.SqlServer.Dts.Tasks.FtpTask;

using Microsoft.SqlServer.Dts.Pipeline.Wrapper;

using Microsoft.SqlServer.Dts.Tasks.ExecutePackageTask;

using Microsoft.SqlServer.Dts.Tasks.BulkInsertTask;

using Microsoft.SqlServer.Dts.Tasks.XMLTask;

using ClosedXML.Excel;

namespace SSISAccelerator

{

class Program

{

static void Main(string[] args)

{

string connectionString = "";

string OutputFolder = "";

Console.WriteLine("Enter the Package Folder path:");

string packageFolder = Console.ReadLine();

Console.WriteLine("Enter the Data Save Type (SQL or EXCEL):");

string DataSaveType = Console.ReadLine();

if (DataSaveType == "SQL")

{

Console.WriteLine("Enter the Connection String:");

connectionString = Console.ReadLine();

}

if (DataSaveType == "EXCEL")

{

Console.WriteLine("Enter the Output Folder path:");

OutputFolder = Console.ReadLine();

}

if (DataSaveType != "EXCEL" && DataSaveType != "SQL")

{

Console.WriteLine("Wrong Input");

Thread.Sleep(5000);

return;

}

//string packageFolder = @"C:\SSIS\";

//string OutputFolder = @"C:\SSIS\Output\";

//string DataSaveType = "SQL";//"SQL",EXCEL

//string connectionString = "Data Source = TIGER03189; Initial Catalog = AdventureWorks; Integrated Security = True; ";

string PackageAnalysisFilePath = OutputFolder+@"PackageAnalysisResult.xlsx";

string dataFlowlFilePath = OutputFolder;

string PackageDetailsFilePath = OutputFolder + @"PackageDetails.xlsx";

if (DataSaveType == "EXCEL")

{

DeleteAllFilesInDirectory(dataFlowlFilePath);

}

var analyzer = new SSISPackageAnalyzer(packageFolder, connectionString, PackageAnalysisFilePath, dataFlowlFilePath, PackageDetailsFilePath,DataSaveType);

analyzer.AnalyzeAllPackages();

// Initialize the SSISProjectAnalyzer class

//SSISProjectAnalyzer projanalyzer = new SSISProjectAnalyzer();

// Process the directory and extract connection details from all .dtproj files

//projanalyzer.ProcessProjectDirectory(packageFolder, connectionString);

Console.WriteLine("Running...");

}

public static void DeleteAllFilesInDirectory(string directoryPath)

{

try

{

// Check if the directory exists

if (Directory.Exists(directoryPath))

{

// Get all file paths in the directory

string[] filePaths = Directory.GetFiles(directoryPath);

// Loop through each file and delete it

foreach (var filePath in filePaths)

{

// Delete the file

File.Delete(filePath);

//Console.WriteLine($"Deleted: {filePath}");

}

//Console.WriteLine("All files have been deleted.");

}

else

{

Console.WriteLine("Directory does not exist.");

}

}

catch (Exception ex)

{

Console.WriteLine($"An error occurred: {ex.Message}");

}

}

}

/\*class SSISProjectAnalyzer

{

// Method to get project-level connections from a given .dtproj file

public List<string> GetProjectConnections(string projectPath, string SqlConnection)

{

var connectionDetails = new List<string>();

Application app = new Application();

//Package package = app.LoadPackage(projectPath, null);

XmlDocument doc = new XmlDocument();

string projectName = Path.GetFileNameWithoutExtension(projectPath);

try

{

doc.Load(projectPath);

}

catch (Exception ex)

{

Console.WriteLine($"Error: {ex.Message}");

}

try

{

XmlNamespaceManager nsManager = new XmlNamespaceManager(doc.NameTable);

nsManager.AddNamespace("SSIS", "www.microsoft.com/SqlServer/SSIS");

XmlNodeList connNodes = doc.GetElementsByTagName("SSIS:ConnectionManager");

XmlNodeList connStringNodes = doc.GetElementsByTagName("SSIS:Parameter");

string name = "";

string connectionString = "";

foreach (XmlNode connNode in connNodes)

{

name = connNode.Attributes["SSIS:Name"]?.Value;

name = name.Replace(".conmgr", "");

//Console.WriteLine(name);

foreach (XmlNode connStringNode in connStringNodes)

{

string paramName = connStringNode.Attributes["SSIS:Name"]?.Value;

if (paramName.Contains(name+".ConnectionString") && paramName.Contains (name))

{

connectionString = connStringNode.SelectSingleNode(".//SSIS:Property[@SSIS:Name='Value']", nsManager)?.InnerText;

}

}

connectionDetails.Add($"{name}: {connectionString}" );

string projectDirectory = Path.GetDirectoryName(projectPath);

InsertConnectionDetails(projectName, name, projectDirectory, SqlConnection, connectionString);

}

}

catch (Exception ex)

{

Console.WriteLine($"Error reading {projectPath}: {ex.Message}");

}

return connectionDetails;

}

// Method to scan a directory and process all .dtproj files

public void ProcessProjectDirectory(string packageFolder ,string metadataConnectionString)

{

string \_connectionString = metadataConnectionString;

// Get all .dtproj files in the directory and subdirectories

var projectFiles = Directory.GetFiles(packageFolder, "\*.dtproj", SearchOption.AllDirectories);

// Iterate through each .dtproj file and extract connection details

foreach (var projectFile in projectFiles)

{

if (projectFile.IndexOf(@"\obj\", StringComparison.OrdinalIgnoreCase) >= 0)

{

Console.WriteLine($"Skipping directory: {projectFile} (contains 'obj')");

continue;

}

Console.WriteLine($"Processing project: {projectFile}");

// Get connections for the current project

var connections = GetProjectConnections(projectFile, \_connectionString);

}

}

public void InsertConnectionDetails(string projectName, string connectionName, string projectPath, string \_connectionString, string ProjectConnectionString)

{

if (connectionName != "")

{

try

{

using (SqlConnection conn = new SqlConnection(\_connectionString))

{

conn.Open();

string query = "INSERT INTO ProjectConnectionsDetails (ProjectName, ConnectionName, ConnectionString, projectPath) " +

"VALUES (@ProjectName, @ConnectionName, @ConnectionString, @projectPath)";

using (SqlCommand cmd = new SqlCommand(query, conn))

{

// Add parameters to avoid SQL injection

cmd.Parameters.AddWithValue("@ProjectName", projectName);

cmd.Parameters.AddWithValue("@ConnectionName", connectionName);

cmd.Parameters.AddWithValue("@ConnectionString", ProjectConnectionString);

cmd.Parameters.AddWithValue("@projectPath", projectPath);

cmd.ExecuteNonQuery();

}

}

}

catch (Exception ex)

{

Console.WriteLine($"Error inserting into database: {ex.Message}");

Console.WriteLine($"Error inserting into database: {ex.Message}");

}

}

}

}\*/

public class SSISPackageAnalyzer

{

int containerCount = 0;

int containerTaskCount = 0;

private string \_connectionString;

private string \_packageFolder;

private HashSet<string> processedPackagePaths;

string PackagePath = "";

string PackageName = "";

int ComponentCount = 0;

private string PackageAnalysisFilePath;

private string DataFlowlFilePath;

private string PackageDetailsFilePath;

private string DataSaveType;

List<string> ComponentNameCHeck = new List<string>();

public SSISPackageAnalyzer(string packageFolder, string metadataConnectionString, string packageAnalysisFilePathfilepath, string dataFlowlFilePath, string packageDetailsFilePath, string datasavetype)

{

\_packageFolder = packageFolder;

\_connectionString = metadataConnectionString;

processedPackagePaths = new HashSet<string>();

PackageAnalysisFilePath = packageAnalysisFilePathfilepath;

DataFlowlFilePath = dataFlowlFilePath;

PackageDetailsFilePath = packageDetailsFilePath;

DataSaveType = datasavetype;

}

public void AnalyzeAllPackages()

{

TruncateTable();

string[] directories = Directory.GetDirectories(\_packageFolder, "\*", SearchOption.AllDirectories);

foreach (string directory in directories)

{

if (directory.IndexOf(@"\obj\", StringComparison.OrdinalIgnoreCase) >= 0)

{

// Console.WriteLine($"Skipping directory: {directory} (contains 'obj')");

continue;

}

try

{

string[] packageFiles = Directory.GetFiles(directory, "\*.dtsx");

string[] connectionmanagerfiles = Directory.GetFiles(directory, "\*.conmgr");

string[] paramfiles = Directory.GetFiles(directory, "\*.params");

foreach (string packagePath in packageFiles)

{

if (processedPackagePaths.Contains(packagePath))

{

continue;

}

try

{

processedPackagePaths.Add(packagePath);

AnalyzeSinglePackage(packagePath);

}

catch (Exception ex)

{

LogError(packagePath, ex);

}

}

foreach (string ConnectionManagerPath in connectionmanagerfiles)

{

if (processedPackagePaths.Contains(ConnectionManagerPath))

{

continue;

}

try

{

processedPackagePaths.Add(ConnectionManagerPath);

AnalyzeSingleConnectionManager(ConnectionManagerPath);

}

catch (Exception ex)

{

LogError(ConnectionManagerPath, ex);

}

}

foreach (string paramfile in paramfiles)

{

if (processedPackagePaths.Contains(paramfile))

{

continue;

}

try

{

processedPackagePaths.Add(paramfile);

AnalyzeParamManager(paramfile);

}

catch (Exception ex)

{

LogError(paramfile, ex);

}

}

//Console.WriteLine($"Completed directory: {directory} ");

}

catch (Exception ex)

{

Console.WriteLine($"Error accessing directory {directory}: {ex.Message}");

}

// Console.WriteLine($"Completed directory: {directory} ");

}

SaveUdateConnectionName(PackageDetailsFilePath);

Console.WriteLine("Completed...");

}

private void AnalyzeParamManager(string paramfile)

{

XmlDocument doc = new XmlDocument();

doc.Load(paramfile);

XmlElement root = doc.DocumentElement;

XmlNamespaceManager nsManager = new XmlNamespaceManager(doc.NameTable);

nsManager.AddNamespace("SSIS", "www.microsoft.com/SqlServer/SSIS");

var metadata = new PackageAnalysisResult

{

ProjectParameterDetails = new List<ProjectParameterInfo>(),

PackagePath = Path.GetDirectoryName(paramfile),

PackageName = Path.GetFileName(paramfile),

};

XmlNodeList parameterNodes = doc.SelectNodes("//SSIS:Parameter", nsManager);

foreach (XmlNode parameterNode in parameterNodes)

{

string parameterName = parameterNode.Attributes["SSIS:Name"]?.Value;

XmlNode valueNode = parameterNode.SelectSingleNode("SSIS:Properties/SSIS:Property[@SSIS:Name='Value']", nsManager);

XmlNode DatatypeNode = parameterNode.SelectSingleNode("SSIS:Properties/SSIS:Property[@SSIS:Name='DataType']", nsManager);

string connectionString = valueNode?.InnerText;

string DataType = DatatypeNode?.InnerText;

if(DataType=="3")

{

DataType = "Boolean";

}

else if (DataType == "6")

{

DataType = "Byte";

}

else if (DataType == "16")

{

DataType = "DateTime";

}

else if (DataType == "15")

{

DataType = "Decimal";

}

else if (DataType == "14")

{

DataType = "Double";

}

else if (DataType == "7")

{

DataType = "Int16";

}

else if (DataType == "9")

{

DataType = "Int32";

}

else if (DataType == "11")

{

DataType = "Int64";

}

else if (DataType == "5")

{

DataType = "SByte";

}

else if (DataType == "13")

{

DataType = "Single";

}

else if (DataType == "18")

{

DataType = "String";

}

else if (DataType == "10")

{

DataType = "Unit32";

}

else if (DataType == "12")

{

DataType = "Unit64";

}

metadata.ProjectParameterDetails.Add(new ProjectParameterInfo

{

ParameterName= parameterName,

DataType= DataType,

Value= connectionString

});

}

SaveProjectParametermetadata(metadata, PackageDetailsFilePath);

}

private void AnalyzeSingleConnectionManager(string ConnectionManagerPath)

{

XmlDocument doc = new XmlDocument();

doc.Load(ConnectionManagerPath);

string connectionStringName = "";

string connectionName = "";

string connectionID = "";

string connectionExpression = "";

string connectionType = "";

// Get the root element

XmlElement root = doc.DocumentElement;

XmlNamespaceManager nsManager = new XmlNamespaceManager(doc.NameTable);

nsManager.AddNamespace("DTS", "www.microsoft.com/SqlServer/Dts");

// Access specific elements or attributes

XmlNode connectionStringNode = root.SelectSingleNode("//DTS:ConnectionManager/DTS:ObjectData/DTS:ConnectionManager/@DTS:ConnectionString", nsManager);

XmlNode connectionNameNode = root.SelectSingleNode("//DTS:ConnectionManager/@DTS:ObjectName", nsManager);

XmlNode connectionTypeNode = root.SelectSingleNode("//DTS:ConnectionManager/@DTS:CreationName", nsManager);

XmlNode connectionIDNode = root.SelectSingleNode("//DTS:ConnectionManager/@DTS:DTSID", nsManager);

XmlNodeList propertyExpressionNodes = root.SelectNodes("//DTS:PropertyExpression", nsManager);

if (propertyExpressionNodes != null)

{

foreach (XmlNode propertyExpressionNode in propertyExpressionNodes)

{

// Extract the Name attribute

XmlAttribute nameAttribute = propertyExpressionNode.Attributes["DTS:Name"];

string name = nameAttribute?.Value ?? "Name not found.";

// Extract the value

string value = propertyExpressionNode.InnerText;

connectionExpression += ($"{name} : {value} ");

//Console.WriteLine($"Property Expression Name: {name}");

//Console.WriteLine($"Property Expression Value: {value}");

// Console.WriteLine();

}

}

if (connectionStringNode != null)

{

connectionStringName = connectionStringNode.Value;

}

if (connectionNameNode != null)

{

connectionName = connectionNameNode.Value;

}

if (connectionTypeNode != null)

{

connectionType = connectionTypeNode.Value;

}

if (connectionIDNode != null)

{

connectionID = connectionIDNode.Value;

}

var metadata = new PackageAnalysisResult

{

Connections = new List<ConnectionInfo>(),

PackagePath = Path.GetDirectoryName(ConnectionManagerPath),

PackageName = Path.GetFileName(ConnectionManagerPath),

};

metadata.Connections.Add(new ConnectionInfo

{

ConnectionName = connectionName,

ConnectionString = connectionStringName,

ConnectionExpressions = connectionExpression,

ConnectionType = connectionType,

ConnectionID = connectionID,

IsProjectConnection = "1"

});

SaveConnectionsmetadata(metadata,PackageDetailsFilePath);

}

private void AnalyzeSinglePackage(string packagePath)

{

containerCount = 0;

containerTaskCount = 0;

ComponentCount = 0;

Application app = new Application();

Package package = app.LoadPackage(packagePath, null);

XmlDocument doc = new XmlDocument();

PackageName = Path.GetFileName(packagePath);

PackagePath = Path.GetDirectoryName(packagePath);

ComponentNameCHeck = new List<string>();

try

{

doc.Load(packagePath);

TraverseXml(doc.DocumentElement);

}

catch (Exception ex)

{

Console.WriteLine($"Error: {ex.Message}");

}

var metadata = new PackageAnalysisResult

{

PackageName = Path.GetFileName(packagePath),

CreatedDate = package.CreationDate,

CreatedBy = package.CreatorName,

Tasks = CountPackageTasks(package),

Connections = CountPackageConnections(package),

PackagePath = Path.GetDirectoryName(packagePath),

Containers = CountPackageContainers(package),

DTSXXML = doc.OuterXml,

Seqtasks = new List<TaskInfo>(),

Foreachtasks = new List<TaskInfo>(),

Forlooptasks = new List<TaskInfo>(),

Variables = GetPackageVariables(package),

DataFlowTaskDetails = new List<DataFlowTaskInfo>(),

};

foreach (Executable executable in package.Executables)

{

//Console.WriteLine($"Executable Type: {executable.GetType().Name}");

if (executable is ForEachLoop foreachContainer)

{

metadata.Foreachtasks.AddRange(ProcessForEachLoopContainerDetails(foreachContainer, new List<ContainerInfo>(), package));

}

else if (executable is Sequence sequenceContainer)

{

metadata.Seqtasks.AddRange(ProcessSequenceContainerDetails(sequenceContainer, new List<ContainerInfo>(), package));

}

else if (executable is ForLoop forLoop)

{

metadata.Forlooptasks.AddRange(ProcessForLoopContainerDetails(forLoop, new List<ContainerInfo>(), package));

}

else if (executable is TaskHost taskHost)

{

if (taskHost.InnerObject is MainPipe dataFlowTask)

{

ExtractDataFlowTask(taskHost, "0");

}

}

}

metadata.SequenceContainerTaskCount = CountSequenceContainerTasks(package);

metadata.ForeachContainerTaskCount = CountForeacheContainerTasks(package);

metadata.ForLoopContainerTaskCount = CountForloopContainerTasks(package);

metadata.ExecutionTime = MeasurePackagePerformance(package);

SavePackageMetadata(metadata, PackageAnalysisFilePath, PackageDetailsFilePath);

ExtractPrecedenceConstraintsForTask(package);

ExtractEventHandlersForPackage(package);

}

static void TraverseXml(XmlNode node)

{

if (node != null)

{

foreach (XmlNode childNode in node.ChildNodes)

{

TraverseXml(childNode);

}

}

}

private List<VariableInfo> GetPackageVariables(Package package)

{

var variables = new List<VariableInfo>();

foreach (Variable variable in package.Variables)

{

if (!variable.SystemVariable)

{

variables.Add(new VariableInfo

{

Name = variable.Name,

Value = variable.Value?.ToString(),

DataType = variable.DataType.ToString(),

Namespace = variable.Namespace,

IsParameter = 0

});

}

}

foreach (Parameter Parameter in package.Parameters)

{

variables.Add(new VariableInfo

{

Name = Parameter.Name,

Value = Parameter.Value?.ToString(),

DataType = Parameter.DataType.ToString(),

IsParameter = 1

});

}

/\*foreach (var variable in variables)

{

Console.WriteLine($"Variable Name: {variable.Name}, Value: {variable.Value}, DataType: {variable.DataType} ,Namespace: {variable.Namespace}");

}\*/

return variables;

}

private List<TaskInfo> CountSequenceContainerTasks(Package package)

{

var tasksInSequence = new List<TaskInfo>();

foreach (Executable executable in package.Executables)

{

// Console.WriteLine($"foreach containers name: {executable.GetType().Name}");

if (executable is Sequence sequence)

{

// Recursively handle containers (like Sequence, ForEachLoop)

ProcessContainerSequenceLoop(sequence, tasksInSequence, package);

}

if (executable is ForEachLoop container)

{

// Recursively handle containers (like Sequence, ForEachLoop)

ProcessContainerForEachLoop(container, tasksInSequence, package);

}

if (executable is ForLoop forloop)

{

ProcessContainerforLoop(forloop, tasksInSequence, package);

}

}

containerTaskCount = containerTaskCount + tasksInSequence.Count;

//Console.WriteLine($"Total tasks in sequence containers (including nested containers): {tasksInSequence.Count}");

return tasksInSequence; // Return total task count for all sequence containers

}

private List<TaskInfo> CountForeacheContainerTasks(Package package)

{

var tasksInForEach = new List<TaskInfo>();

foreach (Executable executable in package.Executables)

{

if (executable is ForEachLoop container)

{

// Recursively handle containers (like Sequence, ForEachLoop)

ProcessContainerForEachLoop(container, tasksInForEach, package);

}

if (executable is Sequence sequence)

{

// Recursively handle containers (like Sequence, ForEachLoop)

ProcessContainerSequenceLoop(sequence, tasksInForEach, package);

}

if (executable is ForLoop forloop)

{

ProcessContainerforLoop(forloop, tasksInForEach, package);

}

}

containerTaskCount = containerTaskCount + tasksInForEach.Count;

//Console.WriteLine($"Total tasks in ForEach containers (including nested containers): : {tasksInForEach.Count}");

return tasksInForEach; // Return total task count for all sequence containers

}

private List<TaskInfo> CountForloopContainerTasks(Package package)

{

var tasksInForLoop = new List<TaskInfo>();

foreach (Executable executable in package.Executables)

{

if (executable is ForEachLoop container)

{

// Recursively handle containers (like Sequence, ForEachLoop)

ProcessContainerForEachLoop(container, tasksInForLoop, package);

}

if (executable is Sequence sequence)

{

// Recursively handle containers (like Sequence, ForEachLoop)

ProcessContainerSequenceLoop(sequence, tasksInForLoop, package);

}

if (executable is ForLoop forloop)

{

// Recursively handle containers (like Sequence, ForEachLoop)

ProcessContainerforLoop(forloop, tasksInForLoop, package);

}

}

containerTaskCount = containerTaskCount + tasksInForLoop.Count;

//Console.WriteLine($"Total tasks in ForEach containers (including nested containers): : {tasksInForEach.Count}");

return tasksInForLoop; // Return total task count for all sequence containers

}

private void ProcessContainerForEachLoop(ForEachLoop container, List<TaskInfo> tasksInForEach, Package package)

{

// Check if the container is a ForEachLoop

if (container is ForEachLoop foreachLoop)

{

}

// Check for nested containers and recursively process them

foreach (Executable nestedExecutable in container.Executables)

{

if (nestedExecutable is ForEachLoop nestedContainer)

{

//ProcessContainerForEachLoop(nestedContainer, tasksInForEach, package);

var tasksInLoop = ProcessForEachLoopContainerDetails(nestedContainer, new List<ContainerInfo>(), package);

tasksInForEach.AddRange(tasksInLoop);

}

if (nestedExecutable is Sequence sequence)

{

//ProcessContainerSequenceLoop(sequence, tasksInForEach, package);

var tasksInLoop = ProcessSequenceContainerDetails(sequence, new List<ContainerInfo>(), package);

tasksInForEach.AddRange(tasksInLoop);

}

if (nestedExecutable is ForLoop forLoop)

{

//ProcessContainerSequenceLoop(seq, tasksInForEach, package);

var tasksInLoop = ProcessForLoopContainerDetails(forLoop, new List<ContainerInfo>(), package);

tasksInForEach.AddRange(tasksInLoop);

}

}

}

private void ProcessContainerSequenceLoop(Sequence container, List<TaskInfo> tasksInForEach, Package package)

{

// Check if the container is a ForEachLoop

if (container is Sequence sequence)

{

}

// Check for nested containers and recursively process them

foreach (Executable nestedExecutable in container.Executables)

{

if (nestedExecutable is ForEachLoop nestedContainer)

{

//ProcessContainerForEachLoop(nestedContainer, tasksInForEach, package);

var tasksInLoop = ProcessForEachLoopContainerDetails(nestedContainer, new List<ContainerInfo>(), package);

tasksInForEach.AddRange(tasksInLoop);

}

if (nestedExecutable is Sequence seq)

{

//ProcessContainerSequenceLoop(seq, tasksInForEach, package);

var tasksInLoop = ProcessSequenceContainerDetails(seq, new List<ContainerInfo>(), package);

tasksInForEach.AddRange(tasksInLoop);

}

if (nestedExecutable is ForLoop forLoop)

{

//ProcessContainerSequenceLoop(seq, tasksInForEach, package);

var tasksInLoop = ProcessForLoopContainerDetails(forLoop, new List<ContainerInfo>(), package);

tasksInForEach.AddRange(tasksInLoop);

}

}

}

private void ProcessContainerforLoop(ForLoop container, List<TaskInfo> tasksInForEach, Package package)

{

// Check if the container is a ForEachLoop

if (container is ForLoop forloop)

{

}

// Check for nested containers and recursively process them

foreach (Executable nestedExecutable in container.Executables)

{

if (nestedExecutable is ForEachLoop nestedContainer)

{

//ProcessContainerForEachLoop(nestedContainer, tasksInForEach, package);

var tasksInLoop = ProcessForEachLoopContainerDetails(nestedContainer, new List<ContainerInfo>(), package);

tasksInForEach.AddRange(tasksInLoop);

}

if (nestedExecutable is Sequence seq)

{

//ProcessContainerSequenceLoop(seq, tasksInForEach, package);

var tasksInLoop = ProcessSequenceContainerDetails(seq, new List<ContainerInfo>(), package);

tasksInForEach.AddRange(tasksInLoop);

}

if (nestedExecutable is ForLoop forLoop)

{

//ProcessContainerSequenceLoop(seq, tasksInForEach, package);

var tasksInLoop = ProcessForLoopContainerDetails(forLoop, new List<ContainerInfo>(), package);

tasksInForEach.AddRange(tasksInLoop);

}

}

}

private List<TaskInfo> CountPackageTasks(Package package)

{

var tasks = new List<TaskInfo>();

var Paperty = new List<PropertyInfo>();

foreach (Executable executable in package.Executables)

{

if (executable is TaskHost taskHost)

{

ExtractTaskDetails(taskHost, "", "", "", "0", "", "", "", "");

tasks.Add(new TaskInfo

{

TaskName = taskHost.Name,

});

}

}

/\* foreach (var task in tasks)

{

Console.WriteLine($"Task Name: {task.TaskName}, Task Type: {task.TaskType}, Task Query: {task.TaskSqlQuery}, Var: {task.Variables} EPD: {task.ExecuteProcessDetails}");

}\*/

return tasks;

}

private void ExtractEventHandlersForPackage(Package package)

{

if (package.EventHandlers.Count > 0)

{

string EventhandlerName = package.Name;

string EventhandlerType = "Package";

String EventName = "";

foreach (DtsEventHandler eventHandler in package.EventHandlers)

{

EventName = eventHandler.Name;

foreach (Executable eventexecutable in eventHandler.Executables)

{

if (eventexecutable is TaskHost taskHost)

{

ExtractEventTaskDetails(taskHost, EventhandlerName, EventhandlerType, EventName, "", "", "", "");

}

else if (eventexecutable is Sequence seq)

{

ExtractEventSequenceTaskDetails(seq, EventhandlerName, EventhandlerType, EventName);

}

else if (eventexecutable is ForEachLoop foreachLoop)

{

ExtractEventForeachTaskDetails(foreachLoop, EventhandlerName, EventhandlerType, EventName);

}

else if (eventexecutable is ForLoop forLoop)

{

ExtractEventForLoopTaskDetails(forLoop, EventhandlerName, EventhandlerType, EventName);

}

}

}

}

}

private void ExtractEventHandlersForSequence(Sequence sequence)

{

if (sequence.EventHandlers.Count > 0)

{

string EventhandlerName = sequence.Name;

string EventhandlerType = "Sequence";

String EventName = "";

foreach (DtsEventHandler eventHandler in sequence.EventHandlers)

{

EventName = eventHandler.Name;

foreach (Executable eventexecutable in eventHandler.Executables)

{

if (eventexecutable is TaskHost taskHost)

{

ExtractEventTaskDetails(taskHost, EventhandlerName, EventhandlerType, EventName, "", "", "", "");

}

else if (eventexecutable is Sequence seq)

{

ExtractEventSequenceTaskDetails(seq, EventhandlerName, EventhandlerType, EventName);

}

else if (eventexecutable is ForEachLoop foreachLoop)

{

ExtractEventForeachTaskDetails(foreachLoop, EventhandlerName, EventhandlerType, EventName);

}

else if (eventexecutable is ForLoop forLoop)

{

ExtractEventForLoopTaskDetails(forLoop, EventhandlerName, EventhandlerType, EventName);

}

}

}

}

}

private void ExtractEventHandlersForForEachLoop(ForEachLoop forEachLoop)

{

if (forEachLoop.EventHandlers.Count > 0)

{

string EventhandlerName = forEachLoop.Name;

string EventhandlerType = "ForEachLoop";

String EventName = "";

foreach (DtsEventHandler eventHandler in forEachLoop.EventHandlers)

{

EventName = eventHandler.Name;

foreach (Executable eventexecutable in eventHandler.Executables)

{

if (eventexecutable is TaskHost taskHost)

{

ExtractEventTaskDetails(taskHost, EventhandlerName, EventhandlerType, EventName, "", "", "", "");

}

else if (eventexecutable is Sequence seq)

{

ExtractEventSequenceTaskDetails(seq, EventhandlerName, EventhandlerType, EventName);

}

else if (eventexecutable is ForEachLoop foreachLoop)

{

ExtractEventForeachTaskDetails(foreachLoop, EventhandlerName, EventhandlerType, EventName);

}

else if (eventexecutable is ForLoop forLoop)

{

ExtractEventForLoopTaskDetails(forLoop, EventhandlerName, EventhandlerType, EventName);

}

}

}

}

}

private void ExtractEventHandlersForForLoop(ForLoop forLoop)

{

if (forLoop.EventHandlers.Count > 0)

{

string EventhandlerName = forLoop.Name;

string EventhandlerType = "ForLoop";

String EventName = "";

foreach (DtsEventHandler eventHandler in forLoop.EventHandlers)

{

EventName = eventHandler.Name;

foreach (Executable eventexecutable in eventHandler.Executables)

{

if (eventexecutable is TaskHost taskHost)

{

ExtractEventTaskDetails(taskHost, EventhandlerName, EventhandlerType, EventName, "", "", "", "");

}

else if (eventexecutable is Sequence seq)

{

ExtractEventSequenceTaskDetails(seq, EventhandlerName, EventhandlerType, EventName);

}

else if (eventexecutable is ForEachLoop foreachLoop)

{

ExtractEventForeachTaskDetails(foreachLoop, EventhandlerName, EventhandlerType, EventName);

}

else if (eventexecutable is ForLoop forloop)

{

ExtractEventForLoopTaskDetails(forloop, EventhandlerName, EventhandlerType, EventName);

}

}

}

}

}

private void ExtractEventHandlersForTask(TaskHost taskhost)

{

if (taskhost.EventHandlers.Count > 0)

{

string EventhandlerName = taskhost.Name;

string EventhandlerType = taskhost.InnerObject.GetType().Name;

string EventName = "";

if (taskhost.InnerObject is MainPipe)

{

EventhandlerType = "DataFlowTask";

}

else if (taskhost.InnerObject is ExecutePackageTask)

{

EventhandlerType = "ExecutePackageTask";

}

else

{

EventhandlerType = taskhost.InnerObject.GetType().Name;

}

foreach (DtsEventHandler eventHandler in taskhost.EventHandlers)

{

EventName = eventHandler.Name;

foreach (Executable eventexecutable in eventHandler.Executables)

{

if (eventexecutable is TaskHost taskHost)

{

ExtractEventTaskDetails(taskHost, EventhandlerName, EventhandlerType, EventName, "", "", "", "");

}

else if (eventexecutable is Sequence seq)

{

ExtractEventSequenceTaskDetails(seq, EventhandlerName, EventhandlerType, EventName);

}

else if (eventexecutable is ForEachLoop foreachLoop)

{

ExtractEventForeachTaskDetails(foreachLoop, EventhandlerName, EventhandlerType, EventName);

}

else if (eventexecutable is ForLoop forLoop)

{

ExtractEventForLoopTaskDetails(forLoop, EventhandlerName, EventhandlerType, EventName);

}

}

}

}

}

private List<PrecedenceConstraintInfo> ExtractPrecedenceConstraintsForTask(Package package)

{

// var DataFlowTaskdetails = new List<DataFlowTaskInfo>();

var PrecedenceConstraintDetails = new List<PrecedenceConstraintInfo>();

var Paperty = new List<PropertyInfo>();

var metadata = new PackageAnalysisResult

{

PrecedenceConstraintDetails = new List<PrecedenceConstraintInfo>(),

};

if (package.PrecedenceConstraints.Count == 0)

{

foreach (Executable executable in package.Executables)

{

// Check if the executable is a SequenceContainer or ForEachLoopContainer

if (executable is Sequence sequence)

{

ExtractPrecedenceConstraintsForSequence(sequence);

}

if (executable is ForEachLoop forEachLoop)

{

ExtractPrecedenceConstraintsForForeach(forEachLoop);

}

if (executable is ForLoop forLoop)

{

ExtractPrecedenceConstraintsForForloop(forLoop);

}

}

}

else

{

foreach (PrecedenceConstraint precedenceConstraint in package.PrecedenceConstraints)

{

string precedenceConstraintFrom = "";

string precedenceConstraintTo = "";

string precedenceConstraintValue = precedenceConstraint.Value.ToString();

string PrecedenceConstraintExpression = precedenceConstraint.Expression.ToString();

string PrecedenceConstraintEvalOP = precedenceConstraint.EvalOp.ToString();

string PrecedenceConstraintLogicalAnd = precedenceConstraint.LogicalAnd.ToString();

{

if (precedenceConstraint.PrecedenceExecutable is TaskHost fromTaskHost)

{

precedenceConstraintFrom = fromTaskHost.Name;

}

else if (precedenceConstraint.PrecedenceExecutable is Sequence fromSequence)

{

precedenceConstraintFrom = fromSequence.Name;

ExtractPrecedenceConstraintsForSequence(fromSequence);

}

else if (precedenceConstraint.PrecedenceExecutable is ForEachLoop fromforeach)

{

precedenceConstraintFrom = fromforeach.Name;

ExtractPrecedenceConstraintsForForeach(fromforeach);

}

else if (precedenceConstraint.PrecedenceExecutable is ForLoop fromforloop)

{

precedenceConstraintFrom = fromforloop.Name;

ExtractPrecedenceConstraintsForForloop(fromforloop);

}

if (precedenceConstraint.ConstrainedExecutable is TaskHost totaskHost)

{

precedenceConstraintTo = totaskHost.Name;

}

else if (precedenceConstraint.ConstrainedExecutable is Sequence ToSequence)

{

precedenceConstraintTo = ToSequence.Name;

ExtractPrecedenceConstraintsForSequence(ToSequence);

}

else if (precedenceConstraint.ConstrainedExecutable is ForEachLoop Toforeach)

{

precedenceConstraintTo = Toforeach.Name;

ExtractPrecedenceConstraintsForForeach(Toforeach);

}

else if (precedenceConstraint.ConstrainedExecutable is ForLoop Toforloop)

{

precedenceConstraintTo = Toforloop.Name;

ExtractPrecedenceConstraintsForForloop(Toforloop);

}

metadata.PrecedenceConstraintDetails.Add(new PrecedenceConstraintInfo

{

PackageName = PackageName,

PackagePath = PackagePath,

PrecedenceConstraintFrom = precedenceConstraintFrom,

PrecedenceConstraintTo = precedenceConstraintTo,

PrecedenceConstraintValue = precedenceConstraintValue,

PrecedenceConstraintExpression = PrecedenceConstraintExpression,

PrecedenceConstraintEvalOP = PrecedenceConstraintEvalOP,

PrecedenceConstraintLogicalAnd = PrecedenceConstraintLogicalAnd,

ContainerName = ""

});

}

}

}

SavePrecedenceConstraintMetadata(metadata, PackageDetailsFilePath);

return metadata.PrecedenceConstraintDetails;

}

private List<PrecedenceConstraintInfo> ExtractPrecedenceConstraintsForSequence(Sequence sequence)

{

// var DataFlowTaskdetails = new List<DataFlowTaskInfo>();

var PrecedenceConstraintDetails = new List<PrecedenceConstraintInfo>();

var Paperty = new List<PropertyInfo>();

var metadata = new PackageAnalysisResult

{

PrecedenceConstraintDetails = new List<PrecedenceConstraintInfo>(),

};

if (sequence.PrecedenceConstraints.Count == 0)

{

foreach (Executable executable in sequence.Executables)

{

// Check if the executable is a SequenceContainer or ForEachLoopContainer

if (executable is Sequence seq)

{

ExtractPrecedenceConstraintsForSequence(seq);

}

if (executable is ForEachLoop forEachLoop)

{

ExtractPrecedenceConstraintsForForeach(forEachLoop);

}

if (executable is ForLoop forLoop)

{

ExtractPrecedenceConstraintsForForloop(forLoop);

}

}

}

else

{

foreach (PrecedenceConstraint precedenceConstraint in sequence.PrecedenceConstraints)

{

string precedenceConstraintFrom = "";

string precedenceConstraintTo = "";

string precedenceConstraintValue = precedenceConstraint.Value.ToString();

string PrecedenceConstraintExpression = precedenceConstraint.Expression.ToString();

string PrecedenceConstraintEvalOP = precedenceConstraint.EvalOp.ToString();

string PrecedenceConstraintLogicalAnd = precedenceConstraint.LogicalAnd.ToString();

{

if (precedenceConstraint.PrecedenceExecutable is TaskHost fromTaskHost)

{

precedenceConstraintFrom = fromTaskHost.Name;

}

else if (precedenceConstraint.PrecedenceExecutable is Sequence fromSequence)

{

precedenceConstraintFrom = fromSequence.Name;

ExtractPrecedenceConstraintsForSequence(fromSequence);

}

else if (precedenceConstraint.PrecedenceExecutable is ForEachLoop fromforeach)

{

precedenceConstraintFrom = fromforeach.Name;

ExtractPrecedenceConstraintsForForeach(fromforeach);

}

else if (precedenceConstraint.PrecedenceExecutable is ForLoop fromforloop)

{

precedenceConstraintFrom = fromforloop.Name;

ExtractPrecedenceConstraintsForForloop(fromforloop);

}

if (precedenceConstraint.ConstrainedExecutable is TaskHost totaskHost)

{

precedenceConstraintTo = totaskHost.Name;

}

else if (precedenceConstraint.ConstrainedExecutable is Sequence ToSequence)

{

precedenceConstraintTo = ToSequence.Name;

ExtractPrecedenceConstraintsForSequence(ToSequence);

}

else if (precedenceConstraint.ConstrainedExecutable is ForEachLoop Toforeach)

{

precedenceConstraintTo = Toforeach.Name;

ExtractPrecedenceConstraintsForForeach(Toforeach);

}

else if (precedenceConstraint.ConstrainedExecutable is ForLoop Toforloop)

{

precedenceConstraintTo = Toforloop.Name;

ExtractPrecedenceConstraintsForForloop(Toforloop);

}

metadata.PrecedenceConstraintDetails.Add(new PrecedenceConstraintInfo

{

PackageName = PackageName,

PackagePath = PackagePath,

PrecedenceConstraintFrom = precedenceConstraintFrom,

PrecedenceConstraintTo = precedenceConstraintTo,

PrecedenceConstraintValue = precedenceConstraintValue,

PrecedenceConstraintExpression = PrecedenceConstraintExpression,

PrecedenceConstraintEvalOP = PrecedenceConstraintEvalOP,

PrecedenceConstraintLogicalAnd = PrecedenceConstraintLogicalAnd,

ContainerName = sequence.Name

});

}

}

}

SavePrecedenceConstraintMetadata(metadata, PackageDetailsFilePath);

return metadata.PrecedenceConstraintDetails;

}

private List<PrecedenceConstraintInfo> ExtractPrecedenceConstraintsForForeach(ForEachLoop forEach)

{

// var DataFlowTaskdetails = new List<DataFlowTaskInfo>();

var PrecedenceConstraintDetails = new List<PrecedenceConstraintInfo>();

var Paperty = new List<PropertyInfo>();

var metadata = new PackageAnalysisResult

{

PrecedenceConstraintDetails = new List<PrecedenceConstraintInfo>(),

};

if (forEach.PrecedenceConstraints.Count == 0)

{

foreach (Executable executable in forEach.Executables)

{

// Check if the executable is a SequenceContainer or ForEachLoopContainer

if (executable is Sequence sequence)

{

ExtractPrecedenceConstraintsForSequence(sequence);

}

if (executable is ForEachLoop forEachLoop)

{

ExtractPrecedenceConstraintsForForeach(forEachLoop);

}

if (executable is ForLoop forLoop)

{

ExtractPrecedenceConstraintsForForloop(forLoop);

}

}

}

else

{

foreach (PrecedenceConstraint precedenceConstraint in forEach.PrecedenceConstraints)

{

string precedenceConstraintFrom = "";

string precedenceConstraintTo = "";

string precedenceConstraintValue = precedenceConstraint.Value.ToString();

string PrecedenceConstraintExpression = precedenceConstraint.Expression.ToString();

string PrecedenceConstraintEvalOP = precedenceConstraint.EvalOp.ToString();

string PrecedenceConstraintLogicalAnd = precedenceConstraint.LogicalAnd.ToString();

{

if (precedenceConstraint.PrecedenceExecutable is TaskHost fromTaskHost)

{

precedenceConstraintFrom = fromTaskHost.Name;

}

else if (precedenceConstraint.PrecedenceExecutable is Sequence fromSequence)

{

precedenceConstraintFrom = fromSequence.Name;

ExtractPrecedenceConstraintsForSequence(fromSequence);

}

else if (precedenceConstraint.PrecedenceExecutable is ForEachLoop fromforeach)

{

precedenceConstraintFrom = fromforeach.Name;

ExtractPrecedenceConstraintsForForeach(fromforeach);

}

else if (precedenceConstraint.PrecedenceExecutable is ForLoop fromforloop)

{

precedenceConstraintFrom = fromforloop.Name;

ExtractPrecedenceConstraintsForForloop(fromforloop);

}

if (precedenceConstraint.ConstrainedExecutable is TaskHost totaskHost)

{

precedenceConstraintTo = totaskHost.Name;

}

else if (precedenceConstraint.ConstrainedExecutable is Sequence ToSequence)

{

precedenceConstraintTo = ToSequence.Name;

ExtractPrecedenceConstraintsForSequence(ToSequence);

}

else if (precedenceConstraint.ConstrainedExecutable is ForEachLoop Toforeach)

{

precedenceConstraintTo = Toforeach.Name;

ExtractPrecedenceConstraintsForForeach(Toforeach);

}

else if (precedenceConstraint.ConstrainedExecutable is ForLoop Toforloop)

{

precedenceConstraintTo = Toforloop.Name;

ExtractPrecedenceConstraintsForForloop(Toforloop);

}

metadata.PrecedenceConstraintDetails.Add(new PrecedenceConstraintInfo

{

PackageName = PackageName,

PackagePath = PackagePath,

PrecedenceConstraintFrom = precedenceConstraintFrom,

PrecedenceConstraintTo = precedenceConstraintTo,

PrecedenceConstraintValue = precedenceConstraintValue,

PrecedenceConstraintExpression = PrecedenceConstraintExpression,

PrecedenceConstraintEvalOP = PrecedenceConstraintEvalOP,

PrecedenceConstraintLogicalAnd = PrecedenceConstraintLogicalAnd,

ContainerName = forEach.Name

});

}

}

}

SavePrecedenceConstraintMetadata(metadata, PackageDetailsFilePath);

return metadata.PrecedenceConstraintDetails;

}

private List<PrecedenceConstraintInfo> ExtractPrecedenceConstraintsForForloop(ForLoop forLoop)

{

// var DataFlowTaskdetails = new List<DataFlowTaskInfo>();

var PrecedenceConstraintDetails = new List<PrecedenceConstraintInfo>();

var Paperty = new List<PropertyInfo>();

var metadata = new PackageAnalysisResult

{

PrecedenceConstraintDetails = new List<PrecedenceConstraintInfo>(),

};

if (forLoop.PrecedenceConstraints.Count == 0)

{

foreach (Executable executable in forLoop.Executables)

{

// Check if the executable is a SequenceContainer or ForEachLoopContainer

if (executable is Sequence sequence)

{

ExtractPrecedenceConstraintsForSequence(sequence);

}

if (executable is ForEachLoop forEachLoop)

{

ExtractPrecedenceConstraintsForForeach(forEachLoop);

}

if (executable is ForLoop forloop)

{

ExtractPrecedenceConstraintsForForloop(forloop);

}

}

}

else

{

foreach (PrecedenceConstraint precedenceConstraint in forLoop.PrecedenceConstraints)

{

string precedenceConstraintFrom = "";

string precedenceConstraintTo = "";

string precedenceConstraintValue = precedenceConstraint.Value.ToString();

string PrecedenceConstraintExpression = precedenceConstraint.Expression.ToString();

string PrecedenceConstraintEvalOP = precedenceConstraint.EvalOp.ToString();

string PrecedenceConstraintLogicalAnd = precedenceConstraint.LogicalAnd.ToString();

{

if (precedenceConstraint.PrecedenceExecutable is TaskHost fromTaskHost)

{

precedenceConstraintFrom = fromTaskHost.Name;

}

else if (precedenceConstraint.PrecedenceExecutable is Sequence fromSequence)

{

precedenceConstraintFrom = fromSequence.Name;

ExtractPrecedenceConstraintsForSequence(fromSequence);

}

else if (precedenceConstraint.PrecedenceExecutable is ForEachLoop fromforeach)

{

precedenceConstraintFrom = fromforeach.Name;

ExtractPrecedenceConstraintsForForeach(fromforeach);

}

else if (precedenceConstraint.PrecedenceExecutable is ForLoop fromforloop)

{

precedenceConstraintFrom = fromforloop.Name;

ExtractPrecedenceConstraintsForForloop(fromforloop);

}

if (precedenceConstraint.ConstrainedExecutable is TaskHost totaskHost)

{

precedenceConstraintTo = totaskHost.Name;

}

else if (precedenceConstraint.ConstrainedExecutable is Sequence ToSequence)

{

precedenceConstraintTo = ToSequence.Name;

ExtractPrecedenceConstraintsForSequence(ToSequence);

}

else if (precedenceConstraint.ConstrainedExecutable is ForEachLoop Toforeach)

{

precedenceConstraintTo = Toforeach.Name;

ExtractPrecedenceConstraintsForForeach(Toforeach);

}

else if (precedenceConstraint.ConstrainedExecutable is ForLoop Toforloop)

{

precedenceConstraintTo = Toforloop.Name;

ExtractPrecedenceConstraintsForForloop(Toforloop);

}

metadata.PrecedenceConstraintDetails.Add(new PrecedenceConstraintInfo

{

PackageName = PackageName,

PackagePath = PackagePath,

PrecedenceConstraintFrom = precedenceConstraintFrom,

PrecedenceConstraintTo = precedenceConstraintTo,

PrecedenceConstraintValue = precedenceConstraintValue,

PrecedenceConstraintExpression = PrecedenceConstraintExpression,

PrecedenceConstraintEvalOP = PrecedenceConstraintEvalOP,

PrecedenceConstraintLogicalAnd = PrecedenceConstraintLogicalAnd,

ContainerName = forLoop.Name

});

}

}

}

SavePrecedenceConstraintMetadata(metadata, PackageDetailsFilePath);

return metadata.PrecedenceConstraintDetails;

}

private void ExtractEventTaskDetails(TaskHost taskHost, string EventHandlerName, string EventHandlerType, string EventType,

string ContainerName, string ContainerType, string ContainerExpression, string ContainerEnumDetails)

{

ExtractTaskDetails(taskHost, EventHandlerName, EventHandlerType, EventType, "1",

ContainerName, ContainerType, ContainerExpression, ContainerEnumDetails);

//ExtractPrecedenceConstraintsForTask(taskHost,package);

}

private void ExtractEventSequenceTaskDetails(Sequence sequence, String EventhandlerName, string EventhandlerType, string EventName)

{

String ContainerName = sequence.Name;

string ContainerType = "Sequence";

foreach (Executable eventexecutable in sequence.Executables)

{

if (eventexecutable is TaskHost taskHost)

{

ExtractEventTaskDetails(taskHost, EventhandlerName, EventhandlerType, EventName, ContainerName, ContainerType, "", "");

}

else if (eventexecutable is Sequence seq)

{

ExtractEventSequenceTaskDetails(seq, EventhandlerName, EventhandlerType, EventName);

}

else if (eventexecutable is ForEachLoop foreachLoop)

{

ExtractEventForeachTaskDetails(foreachLoop, EventhandlerName, EventhandlerType, EventName);

}

else if (eventexecutable is ForLoop forLoop)

{

ExtractEventForLoopTaskDetails(forLoop, EventhandlerName, EventhandlerType, EventName);

}

}

}

private void ExtractEventForeachTaskDetails(ForEachLoop foreachloop, string EventhandlerName, string EventhandlerType, string EventName)

{

String ContainerName = foreachloop.Name;

string ContainerType = "ForEachLoop";

String ContainerExpression = GetForEachLoopExpressions(foreachloop);

String ContainerEnum = GetForEachLoopEnumerator(foreachloop);

foreach (Executable eventexecutable in foreachloop.Executables)

{

if (eventexecutable is TaskHost taskHost)

{

ExtractEventTaskDetails(taskHost, EventhandlerName, EventhandlerType, EventName, ContainerName, ContainerType, ContainerExpression, ContainerEnum);

}

else if (eventexecutable is Sequence seq)

{

ExtractEventSequenceTaskDetails(seq, EventhandlerName, EventhandlerType, EventName);

}

else if (eventexecutable is ForEachLoop foreachLoop)

{

ExtractEventForeachTaskDetails(foreachLoop, EventhandlerName, EventhandlerType, EventName);

}

else if (eventexecutable is ForLoop forLoop)

{

ExtractEventForLoopTaskDetails(forLoop, EventhandlerName, EventhandlerType, EventName);

}

}

}

private void ExtractEventForLoopTaskDetails(ForLoop forloop, string EventhandlerName, string EventhandlerType, string EventName)

{

String ContainerName = forloop.Name;

string ContainerType = "ForLoop";

String ContainerExpression = GetForLoopExpressions(forloop);

String ContainerEnum = GetForLoopEnumerator(forloop);

foreach (Executable eventexecutable in forloop.Executables)

{

if (eventexecutable is TaskHost taskHost)

{

ExtractEventTaskDetails(taskHost, EventhandlerName, EventhandlerType, EventName, ContainerName, ContainerType, ContainerExpression, ContainerEnum);

}

else if (eventexecutable is Sequence seq)

{

ExtractEventSequenceTaskDetails(seq, EventhandlerName, EventhandlerType, EventName);

}

else if (eventexecutable is ForEachLoop foreachLoop)

{

ExtractEventForeachTaskDetails(foreachLoop, EventhandlerName, EventhandlerType, EventName);

}

else if (eventexecutable is ForLoop forLoop)

{

ExtractEventForLoopTaskDetails(forLoop, EventhandlerName, EventhandlerType, EventName);

}

}

}

private string ExtractVariablesForTask(TaskHost taskHost)

{

var variablesUsed = new List<string>();

var variables = new List<VariableInfo>();

// Check for any task that uses package variables (e.g., ExecuteSQLTask, FileSystemTask, etc.)

var taskType = taskHost.InnerObject.GetType();

if (taskHost.InnerObject is ExecuteSQLTask executeSQLtask)

{

if (!string.IsNullOrEmpty(executeSQLtask.SqlStatementSource))

{

var expressionVariables = ExtractVariablesFromExpression(executeSQLtask.SqlStatementSource);

variablesUsed.AddRange(expressionVariables);

}

// Similarly, check for connection string (if it contains variables)

else if (!string.IsNullOrEmpty(executeSQLtask.Connection))

{

var connectionVariables = ExtractVariablesFromExpression(executeSQLtask.Connection);

variablesUsed.AddRange(connectionVariables);

}

}

else if (taskHost.InnerObject is FileSystemTask fileSystemTask)

{

// Check if the FileSystemTask uses any package variables

if (fileSystemTask.IsSourcePathVariable is true)

{

variablesUsed.Add($"Source Path: {fileSystemTask.Source}");

}

if (fileSystemTask.IsDestinationPathVariable is true)

{

variablesUsed.Add($"Destination Path: {fileSystemTask.Destination}");

}

}

else if (taskHost.InnerObject is ScriptTask scriptTask)

{

// If it's a ScriptTask, you may want to check for variables being passed in/out

string[] readOnlyVariables = scriptTask.ReadOnlyVariables.Split(',');

string[] readWriteVariables = scriptTask.ReadWriteVariables.Split(',');

// Add both read-only and read-write variables to the list

variablesUsed.AddRange(readOnlyVariables);

variablesUsed.AddRange(readWriteVariables);

}

return string.Join(", ", variablesUsed);

}

private List<string> ExtractVariablesFromExpression(string expression)

{

var variables = new List<string>();

// For simplicity, let's assume the variables are represented by something like: @[VariableName]

var regex = new System.Text.RegularExpressions.Regex(@"@\[(.\*?)\]");

var matches = regex.Matches(expression);

foreach (System.Text.RegularExpressions.Match match in matches)

{

if (match.Groups.Count > 1)

{

variables.Add(match.Groups[1].Value); // Add the variable name (e.g., "User::MyVariable")

}

}

return variables;

}

private string ExtractParametersForTask(TaskHost taskHost)

{

var parametersUsed = new List<string>();

var taskParameters = new List<TaskParameterInfo>();

//Console.WriteLine($"taskname: {taskHost.InnerObject.GetType().Name},{taskHost.Name}");

// Handle tasks with parameters, like ExecuteSQLTask

if (taskHost.InnerObject is ExecuteSQLTask sqlTask)

{

PropertyInfo parameterBindingsProperty = sqlTask.GetType().GetProperty("ParameterBindings", BindingFlags.Public | BindingFlags.NonPublic | BindingFlags.Instance);

if (parameterBindingsProperty != null)

{

// Get the ParameterBindings collection

var parameterBindings = (System.Collections.IEnumerable)parameterBindingsProperty.GetValue(sqlTask);

foreach (var binding in parameterBindings)

{

// Use reflection to get details of each parameter binding

var nameProperty = binding.GetType().GetProperty("ParameterName");

var directionProperty = binding.GetType().GetProperty("ParameterDirection");

var dataTypeProperty = binding.GetType().GetProperty("DataType");

var valueProperty = binding.GetType().GetProperty("Value");

var DtsVariableName = binding.GetType().GetProperty("DtsVariableName");

// Add the parameter details to the list

taskParameters.Add(new TaskParameterInfo

{

ParameterName = nameProperty?.GetValue(binding)?.ToString(),

ParameterType = directionProperty?.GetValue(binding)?.ToString(),

DataType = dataTypeProperty?.GetValue(binding)?.ToString(),

Value = valueProperty?.GetValue(binding)?.ToString(),

DtsVariableName = DtsVariableName?.GetValue(binding)?.ToString()

});

String parameterName = (nameProperty?.GetValue(binding)?.ToString());

}

parametersUsed = taskParameters.Select(binding =>

$"Name: {binding.ParameterName}, Type: {binding.ParameterType}, DataType: {binding.DataType}, Value: {binding.Value},DtsVariableName: {binding.DtsVariableName}").ToList();

}

}

// Add other tasks that use parameters...

/\*foreach (var parameter in parametersUsed)

{

Console.WriteLine($"parametersUsed: {parameter}");

}\*/

return string.Join("| ", parametersUsed);

}

private List<DataFlowTaskInfo> ExtractDataFlowTask(TaskHost taskHost, string eventhandle)

{

// var DataFlowTaskdetails = new List<DataFlowTaskInfo>();

var Paperty = new List<PropertyInfo>();

var metadata = new PackageAnalysisResult

{

DataFlowTaskDetails = new List<DataFlowTaskInfo>(),

};

if (taskHost.InnerObject is MainPipe dataFlowTask)

{

foreach (IDTSComponentMetaData100 component in dataFlowTask.ComponentMetaDataCollection)

{

string CMCHECK = taskHost.Name + " : " + PackageName + " : " + PackagePath + " : " + component.Name;

if (ComponentNameCHeck.Contains(CMCHECK))

{

ComponentCount = ComponentCount+0;

}

else

{

ComponentNameCHeck.Add(CMCHECK);

ComponentCount = ComponentCount + 1;

}

string CompanaedName = $"Source Component: {component.Name}";

// Loop through the inputs (source columns)

foreach (IDTSInput100 input in component.InputCollection)

{

string CompanaedPropertyDetails = "";

foreach (IDTSInputColumn100 inputColumn in input.InputColumnCollection)

{

string columnDetails = $"Source Component: {component.Name}, " +

$"Source Column: {inputColumn.Name}, DataType: {inputColumn.DataType.ToString()}";

string columnPropertyDetails = "";

foreach (IDTSCustomProperty100 property in inputColumn.CustomPropertyCollection)

{

columnPropertyDetails += $"Property name: {property.Name}, value: {property.Value} , Exp: {property.ExpressionType} ";

}

//Console.WriteLine(inputColumn.ObjectType);

metadata.DataFlowTaskDetails.Add(new DataFlowTaskInfo

{

ColumnName = inputColumn.Name,

DataType = inputColumn.DataType.ToString(),

componentName = component.Name,

TaskName = taskHost.Name,

PackageName = PackageName,

PackagePath = PackagePath,

ColumnType = inputColumn.ObjectType.ToString(),

isEventHandler = eventhandle,

ColumnPropertyDetails= columnPropertyDetails,

});

}

foreach (IDTSCustomProperty100 property in input.CustomPropertyCollection)

{

CompanaedPropertyDetails +=

$"Property name: {property.Name}, value: {property.Value} , Exp: {property.ExpressionType}";

}

if (CompanaedPropertyDetails != "")

{

metadata.DataFlowTaskDetails.Add(new DataFlowTaskInfo

{

componentName = component.Name,

TaskName = taskHost.Name,

PackageName = PackageName,

PackagePath = PackagePath,

componentPropertyDetails = $"Companaed Type: {input.Name}, " + CompanaedPropertyDetails,

isEventHandler = eventhandle,

});

}

}

// Now check if there's a corresponding output column in the destination (or transformation)

foreach (IDTSOutput100 output in component.OutputCollection)

{

String CompanaedPropertyDetails = "";

foreach (IDTSOutputColumn100 outputColumn in output.OutputColumnCollection)

{

string columnDetails = $"Source Component: {component.Name}, " +

$"Source Column: {outputColumn.Name}, DataType: {outputColumn.DataType.ToString()}";

if (outputColumn.Name.Contains("Error") || outputColumn.Name.Contains("ErrorCode") || outputColumn.Name.Contains("ErrorColumn"))

{

//Console.WriteLine($"Skipping Error Column: {outputColumn.Name}");

continue;

}

string columnPropertyDetails = "";

foreach (IDTSCustomProperty100 property in outputColumn.CustomPropertyCollection)

{

columnPropertyDetails += $"Property name: {property.Name}, value: {property.Value} , Exp: {property.ExpressionType} ";

}

//Console.WriteLine(outputColumn.ObjectType);

// Match columns by their index (or manually defined mappings)

metadata.DataFlowTaskDetails.Add(new DataFlowTaskInfo

{

ColumnName = outputColumn.Name,

DataType = outputColumn.DataType.ToString(),

componentName = component.Name,

TaskName = taskHost.Name,

PackageName = PackageName,

PackagePath = PackagePath,

ColumnType = outputColumn.ObjectType.ToString(),

isEventHandler = eventhandle,

ColumnPropertyDetails= columnPropertyDetails,

});

}

foreach (IDTSCustomProperty100 property in output.CustomPropertyCollection)

{

CompanaedPropertyDetails +=

$"Property name: {property.Name}, value: {property.Value} , Exp: {property.ExpressionType}";

}

if (CompanaedPropertyDetails != "")

{

metadata.DataFlowTaskDetails.Add(new DataFlowTaskInfo

{

componentName = component.Name,

TaskName = taskHost.Name,

PackageName = PackageName,

PackagePath = PackagePath,

componentPropertyDetails = $"Companaed Type: {output.Name}, " + CompanaedPropertyDetails,

isEventHandler = eventhandle,

});

}

}

}

foreach (IDTSComponentMetaData100 component in dataFlowTask.ComponentMetaDataCollection)

{

// Check if it's a transformation (like Data Conversion)

if (component.Name.Contains("Data Conversion"))

{

foreach (IDTSInputColumn100 inputColumn in component.InputCollection[0].InputColumnCollection)

{

// Find the output column after conversion

string conversionDetails = $"Conversion: {component.Name}, Column: {inputColumn.Name}, From: {inputColumn.DataType}, To: {component.OutputCollection[0].OutputColumnCollection[0].DataType}";

//Console.WriteLine(inputColumn.ObjectType);

metadata.DataFlowTaskDetails.Add(new DataFlowTaskInfo

{

ColumnName = inputColumn.Name,

DataType = inputColumn.DataType.ToString(),

componentName = component.Name,

TaskName = taskHost.Name,

DataConversion = component.OutputCollection[0].OutputColumnCollection[0].DataType.ToString(),

PackageName = PackageName,

PackagePath = PackagePath,

ColumnType = "Data Conversion :" + inputColumn.ObjectType.ToString(),

isEventHandler = eventhandle,

});

}

foreach (IDTSOutputColumn100 outputColumn in component.OutputCollection[0].OutputColumnCollection)

{

if (outputColumn.Name.Contains("Error") || outputColumn.Name.Contains("ErrorCode") || outputColumn.Name.Contains("ErrorColumn"))

{

//Console.WriteLine($"Skipping Error Column: {outputColumn.Name}");

continue;

}

// Find the output column after conversion

string conversionDetails = $"Conversion: {component.Name}, Column: {outputColumn.Name}, From: {outputColumn.DataType}, To: {component.OutputCollection[0].OutputColumnCollection[0].DataType}";

//Console.WriteLine(inputColumn.ObjectType);

metadata.DataFlowTaskDetails.Add(new DataFlowTaskInfo

{

ColumnName = outputColumn.Name,

DataType = outputColumn.DataType.ToString(),

componentName = component.Name,

TaskName = taskHost.Name,

DataConversion = component.OutputCollection[0].OutputColumnCollection[0].DataType.ToString(),

PackageName = PackageName,

PackagePath = PackagePath,

ColumnType = "Data Conversion :" + outputColumn.ObjectType.ToString(),

isEventHandler = eventhandle,

});

}

}

}

}

/\*foreach (var dataflow in metadata.DataFlowTaskDetails)

{

Console.WriteLine($"Task Name: {dataflow.TaskName}, Column: {dataflow.ColumnName} componentName: {dataflow.componentName} ,");

}\*/

SaveDataFlowMetadata(metadata, DataFlowlFilePath);

return metadata.DataFlowTaskDetails;

}

static bool MatchColumns(IDTSInputColumn100 inputColumn, IDTSOutputColumn100 outputColumn)

{

// Check if the column names match

if (inputColumn.Name == outputColumn.Name)

{

return true;

}

// You can also match columns by other criteria, such as data type or index

if (inputColumn.DataType == outputColumn.DataType)

{

return true; // Data type match as an additional criteria

}

// If columns don't match by name or data type, you can try matching by index or other logic

return false;

}

private string ExtractExpressionsForTask(TaskHost taskHost)

{

{

var expressionsUsed = new List<string>();

String expressionDetails = "";

var Paperty = new List<PropertyInfo>();

Type taskType = taskHost.InnerObject.GetType();

try

{

// List of possible property names that might have expressions (adjust based on your task type)

List<string> propertyNames = new List<string>();

var task = taskHost.InnerObject;

if (taskHost.HasExpressions)

{

foreach (var propertyInfo in taskHost.Properties)

{

string expression = taskHost.GetExpression(propertyInfo.Name);

if (!string.IsNullOrEmpty(expression))

{

expressionsUsed.Add($"Property: {propertyInfo.Name}, Expression: {expression}");

}

}

}

if (taskHost.InnerObject is MainPipe dataFlowTask)

{

foreach (IDTSComponentMetaData100 component in dataFlowTask.ComponentMetaDataCollection)

{

foreach (IDTSCustomProperty100 customProperty in component.CustomPropertyCollection)

{

//Console.WriteLine(customProperty.Name);

// Expressions are typically stored in the CustomProperties with names like "Expression"

if (customProperty.Name == "Expression")

{

//Console.WriteLine($"Expression Name: {customProperty.Name}");

//Console.WriteLine($"Expression Value: {customProperty.Value}");

expressionsUsed.Add($"Expression Name: {customProperty.Name} Expression Value: {customProperty.Value} ");

}

}

}

}

else

{

// Loop through the list of property names

//foreach (var propertyName in propertyNames)

foreach (var propertyInfo in task.GetType().GetProperties())

{

string expression = taskHost.GetExpression(propertyInfo.Name);

// If an expression exists, add it to the list

if (!string.IsNullOrEmpty(expression))

{

expressionDetails = $"Property: {propertyInfo.Name}, Expression: {expression}";

expressionsUsed.Add($"Property: {propertyInfo.Name}, Expression: {expression}");

}

}

}

// Return the list of expressions as a joined string

return string.Join(", ", expressionsUsed);

}

catch (Exception ex)

{

Console.WriteLine($"Error while extracting expressions: {ex.Message}");

return string.Empty;

}

}

}

private List<ConnectionInfo> CountPackageConnections(Package package)

{

var connections = new List<ConnectionInfo>();

foreach (ConnectionManager conn in package.Connections)

{

String connectionDetails = "";

var expressionDetails = new List<string>();

foreach (DtsProperty property in conn.Properties)

{

try

{

// Check for any expression on the property

string expression = conn.GetExpression(property.Name);

if (!string.IsNullOrEmpty(expression))

{

expressionDetails.Add($"{property.Name}: {expression}");

}

}

catch (Exception ex)

{

Console.WriteLine($"Error accessing expression for property {property.Name}: {ex.Message}");

}

}

if (expressionDetails.Any())

{

connectionDetails += ("Expressions: " + string.Join(", ", expressionDetails));

}

else

{

connectionDetails = ("");

}

connections.Add(new ConnectionInfo

{

ConnectionName = conn.Name,

ConnectionString = conn.ConnectionString,

ConnectionExpressions = connectionDetails,

ConnectionType = conn.CreationName,

ConnectionID = conn.ID,

IsProjectConnection = "0"

});

}

/\*foreach (var connection in connections)

{

Console.WriteLine($"Connection Name: {connection.ConnectionName}, Connection Type: {connection.ConnectionType}");

}\*/

return connections;

}

private List<ContainerInfo> CountPackageContainers(Package package)

{

var containers = new List<ContainerInfo>();

String expressionDetails = "";

foreach (Executable executable in package.Executables)

{

if (executable is DtsContainer container && !(executable is TaskHost))

{

if (container is ForEachLoop foreachloop)

{

expressionDetails = GetForEachLoopExpressions(foreachloop);

}

else

{

expressionDetails = "";

}

// Add the base container

containers.Add(new ContainerInfo

{

ContainerName = container.Name,

ContainerType = container.GetType().Name,

ContainerExpression = expressionDetails

});

// Handle specific container types

if (container is Sequence sequenceContainer)

{

ProcessSequenceContainer(sequenceContainer, containers);

}

else if (container is ForEachLoop foreachLoop)

{

ProcessForEachLoopContainer(foreachLoop, containers);

}

else if (container is ForLoop forLoop)

{

ProcessForLoopContainer(forLoop, containers);

}

}

}

/\*foreach (var container in containers)

{

Console.WriteLine($"Container Name: {container.ContainerName}, Container Type: {container.ContainerType}");

}\*/

return containers;

}

public String GetForEachLoopExpressions(ForEachLoop foreachLoop)

{

// Assuming the enumerator is a ForEachFileEnumerator

var enumerator = foreachLoop.ForEachEnumerator;

var Expression = new List<string>();

// Get the properties of the enumerator

foreach (DtsProperty property in enumerator.Properties)

{

// Check if the property has an expression associated with it

string expression = string.Empty;

try

{

// Try to get the expression for the property

expression = enumerator.GetExpression(property.Name);

}

catch (Exception ex)

{

Console.WriteLine($"Error retrieving expression for {property.Name}: {ex.Message}");

}

// If there's an expression, log it

if (!string.IsNullOrEmpty(expression))

{

Expression.Add($"Property: {property.Name}, Expression: {expression}");

}

}

return string.Join("| ", Expression);

}

public String GetForEachLoopEnumerator(ForEachLoop foreachLoop)

{

var EnumeratorDetails = new List<string>();

try

{

var enumerator = foreachLoop.ForEachEnumerator;

if (enumerator != null)

{

if (enumerator is ForEachEnumeratorHost host)

{

foreach (DtsProperty customProperty in host.Properties)

{

var excludedProperties = new HashSet<string>

{

"ID",

"Description",

"CollectionEnumerator",

"CreationName",

"Name"

};

String enumname = customProperty.Name;

if (!excludedProperties.Contains(customProperty.Name))

{

object value = customProperty.GetValue(host);

EnumeratorDetails.Add($"EnumeratorName :{enumname} , EnumeratorValue: {value}");

}

}

}

}

}

catch (Exception ex)

{

Console.WriteLine($"An error occurred: {ex.Message}");

Console.WriteLine(ex.StackTrace);

}

return string.Join(" | ", EnumeratorDetails);

}

public String GetForLoopExpressions(ForLoop forLoop)

{

// Assuming the enumerator is a ForEachFileEnumerator

var enumerator = forLoop.EvalExpression;

var Expression = new List<string>();

// Get the properties of the enumerator

foreach (DtsProperty property in forLoop.Properties)

{

// Check if the property has an expression associated with it

string expression = string.Empty;

try

{

// Try to get the expression for the property

expression = forLoop.GetExpression(property.Name);

}

catch (Exception ex)

{

Console.WriteLine($"Error retrieving expression for {property.Name}: {ex.Message}");

}

// If there's an expression, log it

if (!string.IsNullOrEmpty(expression))

{

Expression.Add($"Property: {property.Name}, Expression: {expression}");

}

}

return string.Join(" | ", Expression);

}

public String GetForLoopEnumerator(ForLoop forLoop)

{

var EnumeratorDetails = new List<string>();

try

{

EnumeratorDetails.Add($"AssignExpression : {forLoop.AssignExpression} " +

$"| InitExpression : {forLoop.InitExpression} " +

$"| EvalExpression : {forLoop.EvalExpression} ");

}

catch (Exception ex)

{

Console.WriteLine($"An error occurred: {ex.Message}");

Console.WriteLine(ex.StackTrace);

}

return string.Join(" | ", EnumeratorDetails);

}

private List<TaskInfo> ProcessSequenceContainerDetails(Sequence container, List<ContainerInfo> containers, Package package)

{

var Seqtasks = new List<TaskInfo>();

var Paperty = new List<PropertyInfo>();

var DataFlowTaskDetails = new List<DataFlowTaskInfo>();

string ContainerName = container.Name;

string ContainerType = container.GetType().Name;

if (container.EventHandlers.Count > 0)

{

ExtractEventHandlersForSequence(container);

}

foreach (Executable executable in container.Executables)

{

if (executable is TaskHost taskHost)

{

ExtractTaskDetails(taskHost, "", "", "", "0", ContainerName, ContainerType, "", "");

Seqtasks.Add(new TaskInfo

{

SeqTaskName = taskHost.Name,

});

}

else if (executable is DtsContainer nestedContainer)

{

containers.Add(new ContainerInfo

{

ContainerName = nestedContainer.Name,

ContainerType = nestedContainer.GetType().Name

});

/\*foreach (var task in containers)

{

Console.WriteLine($"nestedContainer Name: {task.ContainerName}");

}\*/

if (nestedContainer is Sequence nestedSequence)

{

//Seqtasks.AddRange(ProcessSequenceContainerDetails(nestedSequence, containers,package));

Seqtasks.AddRange(CountSequenceContainerTasks(package));

ProcessSequenceContainer(nestedSequence, containers);

}

else if (nestedContainer is ForEachLoop nestedForeach)

{

Seqtasks.AddRange(CountForeacheContainerTasks(package));

//Seqtasks.AddRange(ProcessForEachLoopContainerDetails(nestedForeach, containers, package));

ProcessForEachLoopContainer(nestedForeach, containers);

}

else if (nestedContainer is ForLoop nestedForLoop)

{

//Foreachtasks.AddRange(ProcessForEachLoopContainerDetails(nestedForeach, containers,package));

Seqtasks.AddRange(CountForloopContainerTasks(package));

//Seqtasks.AddRange(ProcessForEachLoopContainerDetails(nestedForeach, containers, package));

ProcessForLoopContainer(nestedForLoop, containers);

}

}

}

containerTaskCount = containerTaskCount + Seqtasks.Count;

/\*foreach (var task in Seqtasks)

{

Console.WriteLine($"Task Name: {task.TaskName}, Task Type: {task.TaskType}, Task Query: {task.TaskSqlQuery}, Var: {task.Variables} EPD: {task.ExecuteProcessDetails}");

}\*/

return Seqtasks;

}

private List<TaskInfo> ProcessForEachLoopContainerDetails(ForEachLoop container, List<ContainerInfo> containers, Package package)

{

var Foreachtasks = new List<TaskInfo>();

var Paperty = new List<PropertyInfo>();

var DataFlowTaskDetails = new List<DataFlowTaskInfo>();

string expressionDetails = "";

expressionDetails = GetForEachLoopExpressions(container);

string ContainerName = container.Name;

string ContainerType = container.GetType().Name;

string enumeratorDetails = GetForEachLoopEnumerator(container);

if (container.EventHandlers.Count > 0)

{

ExtractEventHandlersForForEachLoop(container);

}

foreach (Executable executable in container.Executables)

{

if (executable is TaskHost taskHost)

{

ExtractTaskDetails(taskHost, "", "", "", "0", ContainerName, ContainerType, expressionDetails, enumeratorDetails);

Foreachtasks.Add(new TaskInfo

{

ForeachTaskName = taskHost.Name,

});

}

else if (executable is DtsContainer nestedContainer)

{

containers.Add(new ContainerInfo

{

ContainerName = nestedContainer.Name,

ContainerType = nestedContainer.GetType().Name

});

if (nestedContainer is Sequence nestedSequence)

{

Foreachtasks.AddRange(CountSequenceContainerTasks(package));

ProcessSequenceContainer(nestedSequence, containers);

}

else if (nestedContainer is ForEachLoop nestedForeach)

{

//Foreachtasks.AddRange(ProcessForEachLoopContainerDetails(nestedForeach, containers,package));

Foreachtasks.AddRange(CountForeacheContainerTasks(package));

//Seqtasks.AddRange(ProcessForEachLoopContainerDetails(nestedForeach, containers, package));

ProcessForEachLoopContainer(nestedForeach, containers);

}

else if (nestedContainer is ForLoop nestedForLoop)

{

//Foreachtasks.AddRange(ProcessForEachLoopContainerDetails(nestedForeach, containers,package));

Foreachtasks.AddRange(CountForloopContainerTasks(package));

//Seqtasks.AddRange(ProcessForEachLoopContainerDetails(nestedForeach, containers, package));

ProcessForLoopContainer(nestedForLoop, containers);

}

}

}

containerTaskCount = containerTaskCount + Foreachtasks.Count;

/\* foreach (var task in Foreachtasks)

{

Console.WriteLine($"for Loop Task Name: {task.ForeachTaskName}, Task Type: {task.ForeachTaskType}, , Task Query: {task.ForeachSqlQuery}");

}\*/

return Foreachtasks;

}

private List<TaskInfo> ProcessForLoopContainerDetails(ForLoop container, List<ContainerInfo> containers, Package package)

{

var ForLooptasks = new List<TaskInfo>();

var Paperty = new List<PropertyInfo>();

var DataFlowTaskDetails = new List<DataFlowTaskInfo>();

String expressionDetails = "";

string ContainerName = container.Name;

string ContainerType = container.GetType().Name;

expressionDetails = GetForLoopExpressions(container);

string enumeratorDetails = GetForLoopEnumerator(container);

if (container.EventHandlers.Count > 0)

{

ExtractEventHandlersForForLoop(container);

}

foreach (Executable executable in container.Executables)

{

if (executable is TaskHost taskHost)

{

ExtractTaskDetails(taskHost, "", "", "", "0", ContainerName, ContainerType, expressionDetails, enumeratorDetails);

ForLooptasks.Add(new TaskInfo

{

ForloopTaskName = taskHost.Name,

});

}

else if (executable is DtsContainer nestedContainer)

{

containers.Add(new ContainerInfo

{

ContainerName = nestedContainer.Name,

ContainerType = nestedContainer.GetType().Name

});

if (nestedContainer is Sequence nestedSequence)

{

ForLooptasks.AddRange(CountSequenceContainerTasks(package));

ProcessSequenceContainer(nestedSequence, containers);

}

else if (nestedContainer is ForEachLoop nestedForeach)

{

//Foreachtasks.AddRange(ProcessForEachLoopContainerDetails(nestedForeach, containers,package));

ForLooptasks.AddRange(CountForeacheContainerTasks(package));

//Seqtasks.AddRange(ProcessForEachLoopContainerDetails(nestedForeach, containers, package));

ProcessForEachLoopContainer(nestedForeach, containers);

}

else if (nestedContainer is ForLoop nestedForLoop)

{

//Foreachtasks.AddRange(ProcessForEachLoopContainerDetails(nestedForeach, containers,package));

ForLooptasks.AddRange(CountForloopContainerTasks(package));

//Seqtasks.AddRange(ProcessForEachLoopContainerDetails(nestedForeach, containers, package));

ProcessForLoopContainer(nestedForLoop, containers);

}

}

}

containerTaskCount = containerTaskCount + ForLooptasks.Count;

/\* foreach (var task in Foreachtasks)

{

Console.WriteLine($"for Loop Task Name: {task.ForeachTaskName}, Task Type: {task.ForeachTaskType}, , Task Query: {task.ForeachSqlQuery}");

}\*/

return ForLooptasks;

}

private int ProcessSequenceContainer(Sequence container, List<ContainerInfo> containers)

{

int taskCount = 0;

foreach (Executable executable in container.Executables)

{

if (executable is TaskHost taskHost)

{

taskCount++;

}

else if (executable is DtsContainer nestedContainer)

{

containerCount = containerCount + 1;

if (nestedContainer is Sequence nestedSequence)

{

taskCount += ProcessSequenceContainer(nestedSequence, containers);

// Add task count from nested sequence container

}

else if (nestedContainer is ForEachLoop nestedForeach)

{

ProcessForEachLoopContainer(nestedForeach, containers);

// Process nested ForEachLoop container (if any)

}

else if (nestedContainer is ForLoop nestedForLoop)

{

ProcessForLoopContainer(nestedForLoop, containers);

}

}

}

// Console.WriteLine($"Sequence Container '{container.Name}' has {taskCount} tasks.");

return taskCount;

}

private int ProcessForEachLoopContainer(ForEachLoop container, List<ContainerInfo> containers)

{

int taskCount = 0;

foreach (Executable executable in container.Executables)

{

if (executable is TaskHost taskHost)

{

taskCount++; // Increment taskCount for each task

}

else if (executable is DtsContainer nestedContainer)

{

containerCount = containerCount + 1;

if (nestedContainer is ForEachLoop nestedForeach)

{

taskCount += ProcessForEachLoopContainer(nestedForeach, containers);

}

else if (nestedContainer is Sequence nestedSequence)

{

ProcessSequenceContainer(nestedSequence, containers);

}

else if (nestedContainer is ForLoop nestedForLoop)

{

ProcessForLoopContainer(nestedForLoop, containers);

}

}

}

//Console.WriteLine($"foreach Container '{container.Name}' has {taskCount} tasks.");

return taskCount;

}

private int ProcessForLoopContainer(ForLoop container, List<ContainerInfo> containers)

{

int taskCount = 0;

foreach (Executable executable in container.Executables)

{

if (executable is TaskHost taskHost)

{

taskCount++; // Increment taskCount for each task

}

else if (executable is DtsContainer nestedContainer)

{

containerCount = containerCount + 1;

if (nestedContainer is ForLoop nestedForLoop)

{

taskCount += ProcessForLoopContainer(nestedForLoop, containers);

}

else if (nestedContainer is Sequence nestedSequence)

{

ProcessSequenceContainer(nestedSequence, containers);

}

else if (nestedContainer is ForEachLoop nestedForeach)

{

ProcessForEachLoopContainer(nestedForeach, containers);

}

}

}

//Console.WriteLine($"foreach Container '{container.Name}' has {taskCount} tasks.");

return taskCount;

}

private List<TaskInfo> ExtractTaskDetails(TaskHost taskHost, string EventHandlerName, string EventHandlerType, string EventType

, string Eventindicator, string ContainerName, string ContainerType, string ContainerExpression, string Enumdetails)

{

var metadata = new PackageAnalysisResult

{

ExtractTaskDetails = new List<TaskInfo>(),

};

string sqlQuery = "";

string ExecuteProcessDetails = "";

Type taskType = taskHost.InnerObject.GetType();

string SourcePath = "";

string DestinationPath = "";

string sourceComponentName = "";

string targetComponentName = "";

string sourceType = "";

string targetType = "";

string SQLTable = "";

string TargetSQLTable = "";

string SendMailTaskDetails = "";

string FTPTaskDetails = "";

string ScriptTaskDetails = "";

string ExecutePackageTaskDetails = "";

string taskTypeName = taskType.Name;

string ResultSet = "";

string ConnectionID = "";

string SourceConnectionID = "";

string TargetConnectionID = "";

string TaskComponentDetails = "";

string XMLTask = "";

string BulkInsertTask = "";

string ExpressionTask = "";

if (taskHost.EventHandlers.Count > 0)

{

ExtractEventHandlersForTask(taskHost);

}

if (taskHost.InnerObject is MainPipe dataFlowTask)

{

ExtractDataFlowTask(taskHost, Eventindicator);

taskTypeName = "DataFlowTask";

foreach (IDTSComponentMetaData100 component in dataFlowTask.ComponentMetaDataCollection)

{

// If it's a Source Component (e.g., OLE DB Source, ODBC Source)

if (component.Description.Contains("Source"))

{

sourceComponentName = component.Name;

sourceType = component.Description; // Get the source type

SourceConnectionID = component.RuntimeConnectionCollection[0].ConnectionManagerID;

if (component.Description.Contains("OLE DB"))

{

// Retrieve SQL query for OLE DB Source

foreach (IDTSCustomProperty100 customProperty in component.CustomPropertyCollection)

{

if (customProperty.Name == "SqlCommand")

{

string sqlCommand = customProperty.Value.ToString();

sqlQuery = sqlCommand;

}

if (customProperty.Name == "OpenRowset")

{

SQLTable = customProperty.Value.ToString();

}

}

if (sqlQuery == "")

{

sqlQuery = SQLTable;

}

}

// Check for ODBC Source

else if (component.Description.Contains("ODBC"))

{

// Retrieve SQL query for ODBC Source

foreach (IDTSCustomProperty100 customProperty in component.CustomPropertyCollection)

{

if (customProperty.Name == "SqlCommand")

{

string sqlCommand = customProperty.Value.ToString();

sqlQuery = sqlCommand;

}

if (customProperty.Name == "OpenRowset")

{

SQLTable = customProperty.Value.ToString();

}

}

if (sqlQuery == "")

{

sqlQuery = SQLTable;

}

}

}

// If it's a Destination Component (e.g., OLE DB Destination)

else if (component.Description.Contains("Destination"))

{

targetComponentName = component.Name;

targetType = component.Description; // Get the target type

TargetConnectionID = component.RuntimeConnectionCollection[0].ConnectionManagerID;

if (component.Description.Contains("OLE DB"))

{

// Retrieve SQL query for OLE DB Source

foreach (IDTSCustomProperty100 customProperty in component.CustomPropertyCollection)

{

/\* if (customProperty.Name == "SqlCommand")

{

string sqlCommand = customProperty.Value.ToString();

TargetSQLTable = sqlCommand;

}\*/

if (customProperty.Name == "OpenRowset")

{

TargetSQLTable = customProperty.Value.ToString();

}

}

}

// Check for ODBC Source

else if (component.Description.Contains("ODBC"))

{

// Retrieve SQL query for ODBC Source

foreach (IDTSCustomProperty100 customProperty in component.CustomPropertyCollection)

{

/\*if (customProperty.Name == "SqlCommand")

{

string sqlCommand = customProperty.Value.ToString();

TargetSQLTable = sqlCommand;

}\*/

if (customProperty.Name == "OpenRowset")

{

TargetSQLTable = customProperty.Value.ToString();

}

}

}

}

}

}

else if (taskType.FullName == "Microsoft.SqlServer.Dts.Tasks.ExecuteSQLTask.ExecuteSQLTask")

{

if (taskHost.InnerObject is ExecuteSQLTask executeSQLtask)

{

ConnectionID = executeSQLtask.Connection;

}

// Retrieve SQL query from the Execute SQL Task via reflection

PropertyInfo sqlStatementSourceProperty = taskType.GetProperty("SqlStatementSource");

if (sqlStatementSourceProperty != null)

{

sqlQuery = (string)sqlStatementSourceProperty.GetValue(taskHost.InnerObject);

}

else

{

sqlQuery = "";

}

Microsoft.SqlServer.Dts.Tasks.ExecuteSQLTask.ExecuteSQLTask sqlTask = taskHost.InnerObject as Microsoft.SqlServer.Dts.Tasks.ExecuteSQLTask.ExecuteSQLTask;

if (sqlTask.ResultSetBindings != null)

{

foreach (IDTSResultBinding binding in sqlTask.ResultSetBindings)

{

// Print the variable and column name from the binding

ResultSet += ($"Result Set Column: {binding.ResultName} | SSIS Variable: {binding.DtsVariableName} ");

}

}

}

else if (taskHost.InnerObject is FileSystemTask fileSystemTask)

{

SourcePath = fileSystemTask.Source;

DestinationPath = fileSystemTask.Destination;

TaskComponentDetails = ($"SourcePath: {SourcePath} | DestinationPath: {DestinationPath} ");

}

else if (taskHost.InnerObject is ExecuteProcess processTask)

{

// 1. Get executable and arguments

ExecuteProcessDetails = ($"Executable: {processTask.Executable} | Arguments: { processTask.Arguments} | WorkingDirectory: {processTask.WorkingDirectory}");

TaskComponentDetails = ExecuteProcessDetails;

}

else if (taskHost.InnerObject is SendMailTask mailTask)

{

ConnectionID = mailTask.SmtpConnection;

SendMailTaskDetails = ($"From: {mailTask.FromLine} | To: {mailTask.ToLine} " +

$"| CC: {mailTask.CCLine} BCC: {mailTask.BCCLine} | Subject: {mailTask.Subject} | " +

$"Body: {mailTask.MessageSource} | FileAttachments: {mailTask.FileAttachments} | Priority: {mailTask.Priority}");

TaskComponentDetails = SendMailTaskDetails;

}

else if (taskHost.InnerObject is FtpTask ftpTask)

{

ConnectionID = ftpTask.Connection;

FTPTaskDetails += ($"FTP Operation: {ftpTask.Operation} | ");

FTPTaskDetails += ($"LocalPath: {ftpTask.LocalPath} | ");

FTPTaskDetails += ($"RemotePath: {ftpTask.RemotePath} | ");

FTPTaskDetails += ($"OverwriteDestination: {ftpTask.OverwriteDestination} | ");

FTPTaskDetails += ($"IsLocalPathVariable: {ftpTask.IsLocalPathVariable} | ");

FTPTaskDetails += ($"IsRemotePathVariable: {ftpTask.IsRemotePathVariable} | ");

FTPTaskDetails += ($"IsTransferTypeASCII: {ftpTask.IsTransferTypeASCII} | ");

FTPTaskDetails += ($"StopOnOperationFailure: {ftpTask.StopOnOperationFailure} ");

TaskComponentDetails = FTPTaskDetails;

}

else if (taskHost.InnerObject is ScriptTask scriptask)

{

ScriptTaskDetails += ($"Script Language: {scriptask.ScriptLanguage} | ");

ScriptTaskDetails += ($"EntryPoint: {scriptask.EntryPoint} | ");

ScriptTaskDetails += ($"ReadOnlyVariables: {scriptask.ReadOnlyVariables} | ");

ScriptTaskDetails += ($"EntryPoint: {scriptask.ReadWriteVariables} | ");

ScriptTaskDetails += ($"ScriptProjectName: {scriptask.ScriptProjectName} | ");

foreach (DtsProperty property in taskHost.Properties)

{

if (property.Name == "CodePage") // CodePage

{

ScriptTaskDetails += ($"Code Page: {property.GetValue(scriptask)} | ");

}

}

TaskComponentDetails = ScriptTaskDetails;

}

else if (taskHost.InnerObject is ExecutePackageTask executepackage)

{

ExecutePackageTaskDetails = executepackage.PackageName;

taskTypeName = "ExecutePackageTask";

TaskComponentDetails = ExecutePackageTaskDetails;

}

else if (taskHost.InnerObject is XMLTask xmltask)

{

XMLTask += ($"Source: {xmltask.Source} | ");

XMLTask += ($"SourceType: {xmltask.SourceType} | ");

XMLTask += ($"DiffAlgorithm: {xmltask.DiffAlgorithm} | ");

XMLTask += ($"DiffGramDestination: {xmltask.DiffGramDestination} | ");

XMLTask += ($"DiffOptions: {xmltask.DiffOptions} | ");

XMLTask += ($"DiffGramDestinationType: {xmltask.DiffGramDestinationType} | ");

XMLTask += ($"FailOnDifference: {xmltask.FailOnDifference} | ");

XMLTask += ($"SaveDiffGram: {xmltask.SaveDiffGram} | ");

XMLTask += ($"OperationType: {xmltask.OperationType} | ");

XMLTask += ($"SaveOperationResult: {xmltask.SaveOperationResult} | ");

XMLTask += ($"SaveOperationResult: {xmltask.SaveOperationResult} | ");

XMLTask += ($"SecondOperand: {xmltask.SecondOperand} | ");

XMLTask += ($"SecondOperandType: {xmltask.SecondOperandType} ");

TaskComponentDetails = XMLTask;

}

else if (taskHost.InnerObject is BulkInsertTask bulkInsertTask)

{

SourceConnectionID = bulkInsertTask.SourceConnection;

TargetConnectionID = bulkInsertTask.DestinationConnection;

BulkInsertTask += ($"FormatFile: {bulkInsertTask.FormatFile} | ");

BulkInsertTask += ($"FieldTerminator: {bulkInsertTask.FieldTerminator} | ");

BulkInsertTask += ($"RowTerminator: {bulkInsertTask.RowTerminator} | ");

BulkInsertTask += ($"DestinationTableName: {bulkInsertTask.DestinationTableName} | ");

BulkInsertTask += ($"CodePage: {bulkInsertTask.CodePage} | ");

BulkInsertTask += ($"DataFileType: {bulkInsertTask.DataFileType} | ");

BulkInsertTask += ($"BatchSize: {bulkInsertTask.BatchSize} | ");

BulkInsertTask += ($"LastRow: {bulkInsertTask.LastRow} | ");

BulkInsertTask += ($"FirstRow: {bulkInsertTask.FirstRow} | ");

BulkInsertTask += ($"CheckConstraints: {bulkInsertTask.CheckConstraints} | ");

BulkInsertTask += ($"KeepNulls: {bulkInsertTask.KeepNulls} | ");

BulkInsertTask += ($"KeepIdentity: {bulkInsertTask.KeepIdentity} | ");

BulkInsertTask += ($"TableLock: {bulkInsertTask.TableLock} | ");

BulkInsertTask += ($"FireTriggers: {bulkInsertTask.FireTriggers} | ");

BulkInsertTask += ($"SortedData: {bulkInsertTask.SortedData} | ");

BulkInsertTask += ($"MaximumErrors: {bulkInsertTask.MaximumErrors} ");

TaskComponentDetails = BulkInsertTask;

}

else if (taskHost.InnerObject is ExpressionTask expressiontask)

{

ExpressionTask+= ($"Expression: {expressiontask.Expression} | ");

ExpressionTask += ($"ExecutionValue: {expressiontask.ExecutionValue} ");

TaskComponentDetails = ExpressionTask;

}

metadata.ExtractTaskDetails.Add(new TaskInfo

{

TaskName = taskHost.Name,

TaskType = taskTypeName,

TaskSqlQuery = sqlQuery,

Variables = ExtractVariablesForTask(taskHost),

FileSystemSourcePath = SourcePath,

FileSystemDestinationPath = DestinationPath,

Parameters = ExtractParametersForTask(taskHost),

Expressions = ExtractExpressionsForTask(taskHost),

ExecuteProcessDetails = ExecuteProcessDetails,

SourceComponent = sourceComponentName,

TargetComponent = targetComponentName,

SourceType = sourceType,

TargetType = targetType,

TargetTable = TargetSQLTable,

SendMailTask = SendMailTaskDetails,

ScriptTask = ScriptTaskDetails,

FTPTask = FTPTaskDetails,

ExecutePackage = ExecutePackageTaskDetails,

ResultSetDetails = ResultSet,

EventHandlerName = EventHandlerName,

EventHandlerType = EventHandlerType,

EventType = EventType,

ContainerName = ContainerName,

ContainerType = ContainerType,

ContainerExpression = ContainerExpression,

PackageName = PackageName,

PackagePath = PackagePath,

ContainerEnum = Enumdetails,

SourceConnectionName = SourceConnectionID,

TargetConnectionName = TargetConnectionID,

ConnectionName = ConnectionID,

TaskComponentDetails=TaskComponentDetails,

});

if (Eventindicator == "1")

{

SaveEventMetadata(metadata,PackageDetailsFilePath);

}

else

{

SavePackageTaskmetadata(metadata,PackageDetailsFilePath);

}

return metadata.ExtractTaskDetails;

}

private TimeSpan MeasurePackagePerformance(Package package)

{

DateTime startTime = DateTime.Now;

package.Execute();

return DateTime.Now - startTime;

}

public static bool DoesWorkbookExist(string filePath)

{

return File.Exists(filePath); // Returns true if the file exists, otherwise false

}

private void SavePackageMetadata(PackageAnalysisResult result, string AnalysisfilePath, string DetailsfilePath)

{

if (DataSaveType == "EXCEL")

{

bool workbookExists = DoesWorkbookExist(AnalysisfilePath);

string Complexcity = "";

int complexcitycount = result.Tasks.Count + result.Foreachtasks.Count + result.Seqtasks.Count +

result.Forlooptasks.Count + result.Containers.Count + containerCount + ComponentCount;

if (complexcitycount <= 5)

{

Complexcity = "Simple";

}

else if (complexcitycount > 5 && complexcitycount <= 10)

{

Complexcity = "Medium";

}

else if (complexcitycount > 10)

{

Complexcity = "Complex";

}

else

Complexcity = "Simple";

using (var workbook = workbookExists ? new XLWorkbook(AnalysisfilePath) : new XLWorkbook())

{

// Check if the common worksheet "ProjectParameters" already exists

var worksheet = workbook.Worksheets.FirstOrDefault(ws => ws.Name.Equals("PackageAnalysisResults", StringComparison.OrdinalIgnoreCase));

if (worksheet == null)

{

// Create the worksheet if it doesn't exist

worksheet = workbook.AddWorksheet("PackageAnalysisResults");

worksheet.Cell(1, 1).Value = "PackageName";

worksheet.Cell(1, 2).Value = "PackagePath";

worksheet.Cell(1, 3).Value = "TasksCount";

worksheet.Cell(1, 4).Value = "ConnectionsCount";

worksheet.Cell(1, 5).Value = "ContainerCount";

worksheet.Cell(1, 6).Value = "ComponentCount";

worksheet.Cell(1, 7).Value = "ExecutionTime";

//worksheet.Cell(1, 7).Value = "DTSXXML";

worksheet.Cell(1, 8).Value = "CreatedDate";

worksheet.Cell(1, 9).Value = "CreatedBy";

worksheet.Cell(1, 10).Value = "Complexcity";

}

var lastRow = worksheet.LastRowUsed(); // Get the last used row

int lastRowNumber = lastRow != null ? lastRow.RowNumber() : 0;

int row = lastRowNumber + 1; // If there's no existing data, start at row 2

worksheet.Cell(row, 1).Value = result.PackageName; // Package Name

worksheet.Cell(row, 2).Value = result.PackagePath;

worksheet.Cell(row, 3).Value = result.Tasks.Count + result.Foreachtasks.Count + result.Seqtasks.Count + result.Forlooptasks.Count;

worksheet.Cell(row, 4).Value = result.Connections.Count;

worksheet.Cell(row, 5).Value = result.Containers.Count + containerCount;

worksheet.Cell(row, 6).Value = ComponentCount;

worksheet.Cell(row, 7).Value = result.ExecutionTime;

// worksheet.Cell(row, 7).Value = result.DTSXXML;

worksheet.Cell(row, 8).Value = result.CreatedDate;

worksheet.Cell(row, 9).Value = result.CreatedBy;

worksheet.Cell(row, 10).Value = Complexcity;

// Save the Excel package (the file will be saved at filePath)

workbook.SaveAs(AnalysisfilePath);

}

bool workbookExists1 = DoesWorkbookExist(DetailsfilePath);

foreach (var variable in result.Variables)

{

using (var workbook = workbookExists1 ? new XLWorkbook(DetailsfilePath) : new XLWorkbook())

{

// Check if the common worksheet "ProjectParameters" already exists

var worksheet = workbook.Worksheets.FirstOrDefault(ws => ws.Name.Equals("PackageVariableParameterDetails", StringComparison.OrdinalIgnoreCase));

if (worksheet == null)

{

// Create the worksheet if it doesn't exist

worksheet = workbook.AddWorksheet("PackageVariableParameterDetails");

worksheet.Cell(1, 1).Value = "PackageName";

worksheet.Cell(1, 2).Value = "PackagePath";

worksheet.Cell(1, 3).Value = "VariableOrParameterName";

worksheet.Cell(1, 4).Value = "DataType";

worksheet.Cell(1, 5).Value = "Value";

worksheet.Cell(1, 6).Value = "IsParameter";

}

var lastRow = worksheet.LastRowUsed(); // Get the last used row

int lastRowNumber = lastRow != null ? lastRow.RowNumber() : 0;

int row = lastRowNumber + 1;

worksheet.Cell(row, 1).Value = result.PackageName;

worksheet.Cell(row, 2).Value = result.PackagePath;

worksheet.Cell(row, 3).Value = variable.Name;

worksheet.Cell(row, 4).Value = variable.DataType;

worksheet.Cell(row, 5).Value = variable.Value;

worksheet.Cell(row, 6).Value = variable.IsParameter;

workbook.SaveAs(DetailsfilePath);

}

}

foreach (var connectionInfo in result.Connections)

{

using (var workbook = workbookExists1 ? new XLWorkbook(DetailsfilePath) : new XLWorkbook())

{

// Check if the common worksheet "ProjectParameters" already exists

var worksheet = workbook.Worksheets.FirstOrDefault(ws => ws.Name.Equals("PackageConnectionDetails", StringComparison.OrdinalIgnoreCase));

if (worksheet == null)

{

// Create the worksheet if it doesn't exist

worksheet = workbook.AddWorksheet("PackageConnectionDetails");

worksheet.Cell(1, 1).Value = "PackageName";

worksheet.Cell(1, 2).Value = "PackagePath";

worksheet.Cell(1, 3).Value = "ConnectionName";

worksheet.Cell(1, 4).Value = "ConnectionType";

worksheet.Cell(1, 5).Value = "ConnectionExpressions";

worksheet.Cell(1, 6).Value = "ConnectionString";

worksheet.Cell(1, 7).Value = "ConnectionID";

worksheet.Cell(1, 8).Value = "IsProjectConnection";

}

var lastRow = worksheet.LastRowUsed(); // Get the last used row

int lastRowNumber = lastRow != null ? lastRow.RowNumber() : 0;

int row = lastRowNumber + 1;

worksheet.Cell(row, 1).Value = result.PackageName;

worksheet.Cell(row, 2).Value = result.PackagePath;

worksheet.Cell(row, 3).Value = connectionInfo.ConnectionName;

worksheet.Cell(row, 4).Value = connectionInfo.ConnectionType;

worksheet.Cell(row, 5).Value = connectionInfo.ConnectionExpressions;

worksheet.Cell(row, 6).Value = connectionInfo.ConnectionString;

worksheet.Cell(row, 7).Value = connectionInfo.ConnectionID;

worksheet.Cell(row, 8).Value = connectionInfo.IsProjectConnection;

workbook.SaveAs(DetailsfilePath);

}

}

}

else if (DataSaveType == "SQL")

{

using (SqlConnection connection = new SqlConnection(\_connectionString))

{

connection.Open();

string query = @"

INSERT INTO PackageAnalysisResults

(PackageName, CreatedDate, TaskCount, ConnectionCount, ExecutionTime, PackageFolder, ContainerCount, DTSXXML, CreatedBy, DataFlowTaskComponentCount)

VALUES

(@PackageName, @CreatedDate, @TaskCount, @ConnectionCount, @ExecutionTime, @PackagePath, @ContainerCount, @DTSXXML, @CreatedBy, @ComponentCount)";

using (SqlCommand cmd = new SqlCommand(query, connection))

{

cmd.Parameters.AddWithValue("@PackageName", result.PackageName);

cmd.Parameters.AddWithValue("@CreatedDate", result.CreatedDate);

cmd.Parameters.AddWithValue("@TaskCount", result.Tasks.Count + result.Foreachtasks.Count + result.Seqtasks.Count + result.Forlooptasks.Count);

cmd.Parameters.AddWithValue("@ConnectionCount", result.Connections.Count);

cmd.Parameters.AddWithValue("@ExecutionTime", result.ExecutionTime);

cmd.Parameters.AddWithValue("@PackagePath", result.PackagePath);

cmd.Parameters.AddWithValue("@ContainerCount", result.Containers.Count + containerCount);

cmd.Parameters.AddWithValue("@DTSXXML", result.DTSXXML);

cmd.Parameters.AddWithValue("@CreatedBy", result.CreatedBy);

cmd.Parameters.AddWithValue("@ComponentCount", ComponentCount);

cmd.ExecuteNonQuery();

}

foreach (var variable in result.Variables)

{

string taskQuery = @"

INSERT INTO PackageVariableParameterDetails (PackageName, VariableOrParameterName, DataType, Value, PackagePath, IsParameter)

VALUES (@PackageName, @VariableOrParameterName, @DataType, @Value, @PackagePath, @IsParameter)";

using (SqlCommand cmd = new SqlCommand(taskQuery, connection))

{

cmd.Parameters.AddWithValue("@PackageName", result.PackageName);

cmd.Parameters.AddWithValue("@VariableOrParameterName", variable.Name);

cmd.Parameters.AddWithValue("@DataType", variable.DataType ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@Value", variable.Value ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@PackagePath", result.PackagePath);

cmd.Parameters.AddWithValue("@IsParameter", variable.IsParameter);

cmd.ExecuteNonQuery();

}

}

// Insert connection details

foreach (var connectionInfo in result.Connections)

{

string connectionQuery = @"

INSERT INTO PackageConnectionDetails (PackageName, ConnectionName, ConnectionType, PackagePath,

ConnectionExpressions, ConnectionString, ConnectionDTSID, IsProjectConnection)

VALUES (@PackageName, @ConnectionName, @ConnectionType, @PackagePath, @ConnectionExpressions, @ConnectionString, @ConnectionID, @IsProjectConnection)";

using (SqlCommand cmd = new SqlCommand(connectionQuery, connection))

{

cmd.Parameters.AddWithValue("@PackageName", result.PackageName);

cmd.Parameters.AddWithValue("@ConnectionName", connectionInfo.ConnectionName);

cmd.Parameters.AddWithValue("@ConnectionType", connectionInfo.ConnectionType);

cmd.Parameters.AddWithValue("@PackagePath", result.PackagePath);

cmd.Parameters.AddWithValue("@ConnectionExpressions", connectionInfo.ConnectionExpressions);

cmd.Parameters.AddWithValue("@ConnectionString", connectionInfo.ConnectionString);

cmd.Parameters.AddWithValue("@ConnectionID", connectionInfo.ConnectionID);

cmd.Parameters.AddWithValue("@IsProjectConnection", connectionInfo.IsProjectConnection);

cmd.ExecuteNonQuery();

}

}

// If the workbook doesn't exist, create a new one

}

}

}

private void SaveDataFlowMetadata(PackageAnalysisResult result, string filePath)

{

if (DataSaveType == "EXCEL")

{

string Dataflowfile = "";

foreach (var dataflowtaskdetails in result.DataFlowTaskDetails)

{

Dataflowfile = filePath + dataflowtaskdetails.PackageName;

Dataflowfile = Dataflowfile.Replace(".dtsx", "\_DFM.xlsx");

bool workbookExists = DoesWorkbookExist(Dataflowfile);

using (var workbook = workbookExists ? new XLWorkbook(Dataflowfile) : new XLWorkbook())

{

string SheetName = "DataFlowTaskMappingDetails";

// Check if the common worksheet "ProjectParameters" already exists

var worksheet = workbook.Worksheets.FirstOrDefault(ws => ws.Name.Equals(SheetName, StringComparison.OrdinalIgnoreCase));

if (worksheet == null)

{

// Create the worksheet if it doesn't exist

worksheet = workbook.AddWorksheet(SheetName);

worksheet.Cell(1, 1).Value = "PackageName";

worksheet.Cell(1, 2).Value = "PackagePath";

worksheet.Cell(1, 3).Value = "TaskName";

worksheet.Cell(1, 4).Value = "ColumnName";

worksheet.Cell(1, 5).Value = "ColumnType";

worksheet.Cell(1, 6).Value = "DataType";

worksheet.Cell(1, 7).Value = "ComponentName";

worksheet.Cell(1, 8).Value = "DataConversion";

worksheet.Cell(1, 9).Value = "ComponentPropertyDetails";

worksheet.Cell(1, 10).Value = "ColumnPropertyDetails";

worksheet.Cell(1, 11).Value = "isEventHandler";

}

var rows = worksheet.RowsUsed(); // Get all rows that have data

bool recordExists = false;

foreach (var row1 in rows)

{

string existingPackageName = row1.Cell(1).GetString();

string existingPackagePath = row1.Cell(2).GetString();

string existingTaskName = row1.Cell(3).GetString();

string existingColumnName = row1.Cell(4).GetString();

string existingColumnType = row1.Cell(5).GetString();

string existingDataType = row1.Cell(6).GetString();

string existingComponentName = row1.Cell(7).GetString();

string existingDataConversion = row1.Cell(8).GetString();

string existingComponentPropertyDetails = row1.Cell(9).GetString();

string existingColumnPropertyDetails = row1.Cell(10).GetString();

string existingisEventHandler = row1.Cell(11).GetString();

if (existingPackageName == dataflowtaskdetails.PackageName &&

existingPackagePath == dataflowtaskdetails.PackagePath &&

existingTaskName == dataflowtaskdetails.TaskName &&

existingColumnName == dataflowtaskdetails.ColumnName &&

existingColumnType == dataflowtaskdetails.ColumnType &&

existingDataType == dataflowtaskdetails.DataType &&

existingComponentName == dataflowtaskdetails.componentName &&

existingDataConversion == dataflowtaskdetails.DataConversion &&

existingComponentPropertyDetails == dataflowtaskdetails.componentPropertyDetails &&

existingColumnPropertyDetails == dataflowtaskdetails.ColumnPropertyDetails &&

existingisEventHandler == dataflowtaskdetails.isEventHandler)

{

recordExists = true;

break; // No need to check further rows if the record is found

}

}

if (recordExists)

{

//Console.WriteLine("Record already exists. No insertion needed.");

}

else

{

var lastRow = worksheet.LastRowUsed(); // Get the last used row

int lastRowNumber = lastRow != null ? lastRow.RowNumber() : 0;

int row = lastRowNumber + 1;

worksheet.Cell(row, 1).Value = dataflowtaskdetails.PackageName;

worksheet.Cell(row, 2).Value = dataflowtaskdetails.PackagePath;

worksheet.Cell(row, 3).Value = dataflowtaskdetails.TaskName;

worksheet.Cell(row, 4).Value = dataflowtaskdetails.ColumnName;

worksheet.Cell(row, 5).Value = dataflowtaskdetails.ColumnType;

worksheet.Cell(row, 6).Value = dataflowtaskdetails.DataType;

worksheet.Cell(row, 7).Value = dataflowtaskdetails.componentName;

worksheet.Cell(row, 8).Value = dataflowtaskdetails.DataConversion;

worksheet.Cell(row, 9).Value = dataflowtaskdetails.componentPropertyDetails;

worksheet.Cell(row, 10).Value = dataflowtaskdetails.ColumnPropertyDetails;

worksheet.Cell(row, 11).Value = dataflowtaskdetails.isEventHandler;

workbook.SaveAs(Dataflowfile);

}

}

}

}

else if (DataSaveType == "SQL")

{

using (SqlConnection connection = new SqlConnection(\_connectionString))

{

connection.Open();

foreach (var dataflowtaskdetails in result.DataFlowTaskDetails)

{

string containerQuery = @"

INSERT INTO DataFlowTaskMappingDetails (PackageName, TaskName, ColumnName, DataType,

ComponentName, DataConversion, PackagePath, ColumnType, isEventHandler,ComponentPropertyDetails, ColumnPropertyDetails)

SELECT DISTINCT @PackageName, @TaskName, @ColumnName, @DataType, @ComponentName, @DataConversion, @PackagePath, @ColumnType, @isEventHandler,

@componentPropertyDetails, @ColumnPropertyDetails

WHERE NOT EXISTS(

SELECT 1 FROM DataFlowTaskMappingDetails

WHERE ISNULL(ColumnName,'') = ISNULL(@ColumnName,'') AND ISNULL(DataType,'') = ISNULL(@DataType,'') AND

ISNULL(PackageName,'') = ISNULL(@PackageName,'') AND ISNULL(PackagePath,'') = ISNULL(@PackagePath,'') AND

ISNULL(ColumnType,'') = ISNULL(@ColumnType,'') AND ISNULL(ComponentName,'') = ISNULL(@ComponentName,'') AND

ISNULL(TaskName,'') = ISNULL(@TaskName,'') AND ISNULL(ComponentPropertyDetails,'') = ISNULL(@componentPropertyDetails,'')

AND ISNULL(ColumnPropertyDetails,'') = ISNULL(@ColumnPropertyDetails,'')) ";

using (SqlCommand cmd = new SqlCommand(containerQuery, connection))

{

cmd.Parameters.AddWithValue("@PackageName", dataflowtaskdetails.PackageName);

cmd.Parameters.AddWithValue("@ColumnName", dataflowtaskdetails.ColumnName ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@DataType", dataflowtaskdetails.DataType ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@ComponentName", dataflowtaskdetails.componentName ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@DataConversion", dataflowtaskdetails.DataConversion ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@TaskName", dataflowtaskdetails.TaskName);

cmd.Parameters.AddWithValue("@PackagePath", dataflowtaskdetails.PackagePath);

cmd.Parameters.AddWithValue("@ColumnType", dataflowtaskdetails.ColumnType ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@isEventHandler", dataflowtaskdetails.isEventHandler ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@componentPropertyDetails", dataflowtaskdetails.componentPropertyDetails ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@ColumnPropertyDetails", dataflowtaskdetails.ColumnPropertyDetails ?? (object)DBNull.Value);

cmd.ExecuteNonQuery();

}

}

}

}

}

private void SavePrecedenceConstraintMetadata(PackageAnalysisResult result, string filePath)

{

if (DataSaveType == "EXCEL")

{

bool workbookExists = DoesWorkbookExist(filePath);

foreach (var precedenceConstraintDetails in result.PrecedenceConstraintDetails)

{

using (var workbook = workbookExists ? new XLWorkbook(filePath) : new XLWorkbook())

{

string SheetName = "PrecedenceConstraintDetails";

// Check if the common worksheet "ProjectParameters" already exists

var worksheet = workbook.Worksheets.FirstOrDefault(ws => ws.Name.Equals(SheetName, StringComparison.OrdinalIgnoreCase));

if (worksheet == null)

{

// Create the worksheet if it doesn't exist

worksheet = workbook.AddWorksheet(SheetName);

worksheet.Cell(1, 1).Value = "PackageName";

worksheet.Cell(1, 2).Value = "PackagePath";

worksheet.Cell(1, 3).Value = "PrecedenceConstraintFrom";

worksheet.Cell(1, 4).Value = "PrecedenceConstraintTo";

worksheet.Cell(1, 5).Value = "PrecedenceConstraintValue";

worksheet.Cell(1, 6).Value = "PrecedenceConstraintExpression";

worksheet.Cell(1, 7).Value = "PrecedenceConstraintLogicalAnd";

worksheet.Cell(1, 8).Value = "PrecedenceConstraintEvalOP";

worksheet.Cell(1, 9).Value = "ContainerName";

}

var rows = worksheet.RowsUsed(); // Get all rows that have data

bool recordExists = false;

foreach (var row1 in rows)

{

string existingPackageName = row1.Cell(1).GetString();

string existingPackagePath = row1.Cell(2).GetString();

string existingPrecedenceConstraintFrom = row1.Cell(3).GetString();

string existingPrecedenceConstraintTo = row1.Cell(4).GetString();

string existingPrecedenceConstraintValue = row1.Cell(5).GetString();

string existingPrecedenceConstraintExpression = row1.Cell(6).GetString();

string existingPrecedenceConstraintLogicalAnd = row1.Cell(7).GetString();

string existingPrecedenceConstraintEvalOP = row1.Cell(8).GetString();

string existingContainerName = row1.Cell(9).GetString();

if (existingPackageName == precedenceConstraintDetails.PackageName &&

existingPackagePath == precedenceConstraintDetails.PackagePath &&

existingPrecedenceConstraintFrom == precedenceConstraintDetails.PrecedenceConstraintFrom &&

existingPrecedenceConstraintTo == precedenceConstraintDetails.PrecedenceConstraintTo &&

existingPrecedenceConstraintValue == precedenceConstraintDetails.PrecedenceConstraintValue &&

existingPrecedenceConstraintExpression == precedenceConstraintDetails.PrecedenceConstraintExpression &&

existingPrecedenceConstraintLogicalAnd == precedenceConstraintDetails.PrecedenceConstraintLogicalAnd &&

existingPrecedenceConstraintEvalOP == precedenceConstraintDetails.PrecedenceConstraintEvalOP &&

existingContainerName == precedenceConstraintDetails.ContainerName

)

{

recordExists = true;

break; // No need to check further rows if the record is found

}

}

if (recordExists)

{

//Console.WriteLine("Record already exists. No insertion needed.");

}

else

{

var lastRow = worksheet.LastRowUsed(); // Get the last used row

int lastRowNumber = lastRow != null ? lastRow.RowNumber() : 0;

int row = lastRowNumber + 1;

worksheet.Cell(row, 1).Value = precedenceConstraintDetails.PackageName;

worksheet.Cell(row, 2).Value = precedenceConstraintDetails.PackagePath;

worksheet.Cell(row, 3).Value = precedenceConstraintDetails.PrecedenceConstraintFrom;

worksheet.Cell(row, 4).Value = precedenceConstraintDetails.PrecedenceConstraintTo;

worksheet.Cell(row, 5).Value = precedenceConstraintDetails.PrecedenceConstraintValue;

worksheet.Cell(row, 6).Value = precedenceConstraintDetails.PrecedenceConstraintExpression;

worksheet.Cell(row, 7).Value = precedenceConstraintDetails.PrecedenceConstraintLogicalAnd;

worksheet.Cell(row, 8).Value = precedenceConstraintDetails.PrecedenceConstraintEvalOP;

worksheet.Cell(row, 9).Value = precedenceConstraintDetails.ContainerName;

workbook.SaveAs(filePath);

}

}

}

}

else if (DataSaveType == "SQL")

{

using (SqlConnection connection = new SqlConnection(\_connectionString))

{

connection.Open();

foreach (var precedenceConstraintDetails in result.PrecedenceConstraintDetails)

{

string containerQuery = @"

INSERT INTO PrecedenceConstraintDetails (PackageName, PrecedenceConstraintFrom, PrecedenceConstraintTo,

PrecedenceConstraintValue, PrecedenceConstraintExpression, PrecedenceConstraintLogicalAnd, PrecedenceConstraintEvalOP, ContainerName, PackagePath)

SELECT DISTINCT @PackageName, @PrecedenceConstraintFrom, @PrecedenceConstraintTo,

@PrecedenceConstraintValue, @PrecedenceConstraintExpression, @PrecedenceConstraintLogicalAnd, @PrecedenceConstraintEvalOP,

@ContainerName, @PackagePath

WHERE NOT EXISTS(

SELECT 1 FROM PrecedenceConstraintDetails

WHERE PackageName = @PackageName AND PrecedenceConstraintFrom = @PrecedenceConstraintFrom

AND PrecedenceConstraintTo = @PrecedenceConstraintTo AND PrecedenceConstraintValue = @PrecedenceConstraintValue

AND ContainerName = @ContainerName AND PackagePath = @PackagePath

AND PrecedenceConstraintExpression=@PrecedenceConstraintExpression AND PrecedenceConstraintLogicalAnd= @PrecedenceConstraintLogicalAnd

AND PrecedenceConstraintEvalOP = @PrecedenceConstraintEvalOP) ";

using (SqlCommand cmd = new SqlCommand(containerQuery, connection))

{

cmd.Parameters.AddWithValue("@PackageName", precedenceConstraintDetails.PackageName);

cmd.Parameters.AddWithValue("@PrecedenceConstraintFrom", precedenceConstraintDetails.PrecedenceConstraintFrom ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@PrecedenceConstraintTo", precedenceConstraintDetails.PrecedenceConstraintTo ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@PrecedenceConstraintValue", precedenceConstraintDetails.PrecedenceConstraintValue ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@PrecedenceConstraintExpression", precedenceConstraintDetails.PrecedenceConstraintExpression ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@PrecedenceConstraintLogicalAnd", precedenceConstraintDetails.PrecedenceConstraintLogicalAnd ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@PrecedenceConstraintEvalOP", precedenceConstraintDetails.PrecedenceConstraintEvalOP ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@ContainerName", precedenceConstraintDetails.ContainerName ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@PackagePath", precedenceConstraintDetails.PackagePath);

cmd.ExecuteNonQuery();

}

}

}

}

}

private void SaveEventMetadata(PackageAnalysisResult result, String filePath)

{

if (DataSaveType == "EXCEL")

{

bool workbookExists = DoesWorkbookExist(filePath);

foreach (var task in result.ExtractTaskDetails)

{

using (var workbook = workbookExists ? new XLWorkbook(filePath) : new XLWorkbook())

{

string SheetName = "EventHandlerTaskDetails";

// Check if the common worksheet "ProjectParameters" already exists

var worksheet = workbook.Worksheets.FirstOrDefault(ws => ws.Name.Equals(SheetName, StringComparison.OrdinalIgnoreCase));

if (worksheet == null)

{

// Create the worksheet if it doesn't exist

worksheet = workbook.AddWorksheet(SheetName);

worksheet.Cell(1, 1).Value = "PackageName";

worksheet.Cell(1, 2).Value = "PackagePath";

worksheet.Cell(1, 3).Value = "EventHandlerName";

worksheet.Cell(1, 4).Value = "EventHandlerType";

worksheet.Cell(1, 5).Value = "EventType";

worksheet.Cell(1, 6).Value = "TaskName";

worksheet.Cell(1, 7).Value = "TaskType";

worksheet.Cell(1, 8).Value = "ContainerName";

worksheet.Cell(1, 9).Value = "ContainerType";

worksheet.Cell(1, 10).Value = "ContainerExpression";

worksheet.Cell(1, 11).Value = "TaskConnectionName";

worksheet.Cell(1, 12).Value = "SqlQuery";

worksheet.Cell(1, 13).Value = "Variables";

worksheet.Cell(1, 14).Value = "Parameters";

worksheet.Cell(1, 15).Value = "Expressions";

worksheet.Cell(1, 16).Value = "DataFlowDaskSourceName";

worksheet.Cell(1, 17).Value = "DataFlowTaskSourceType";

worksheet.Cell(1, 18).Value = "DataFlowTaskTargetName";

worksheet.Cell(1, 19).Value = "DataFlowTaskTargetType";

worksheet.Cell(1, 20).Value = "DataFlowTaskTargetTable";

worksheet.Cell(1, 21).Value = "DataFlowDaskSourceConnectionName";

worksheet.Cell(1, 22).Value = "DataFlowDaskTargetConnectionName";

worksheet.Cell(1, 23).Value = "SendMailTaskDetails";

worksheet.Cell(1, 24).Value = "ResultSetDetails";

worksheet.Cell(1, 25).Value = "TaskComponentDetails";

}

var rows = worksheet.RowsUsed(); // Get all rows that have data

bool recordExists = false;

if (recordExists)

{

//Console.WriteLine("Record already exists. No insertion needed.");

}

else

{

var lastRow = worksheet.LastRowUsed(); // Get the last used row

int lastRowNumber = lastRow != null ? lastRow.RowNumber() : 0;

int row = lastRowNumber + 1;

worksheet.Cell(row, 1).Value = task.PackageName;

worksheet.Cell(row, 2).Value = task.PackagePath;

worksheet.Cell(row, 3).Value = task.EventHandlerName;

worksheet.Cell(row, 4).Value = task.EventHandlerType;

worksheet.Cell(row, 5).Value = task.EventType;

worksheet.Cell(row, 6).Value = task.TaskName;

worksheet.Cell(row, 7).Value = task.TaskType;

worksheet.Cell(row, 8).Value = task.ContainerName;

worksheet.Cell(row, 9).Value = task.ContainerType;

worksheet.Cell(row, 10).Value = task.ContainerExpression;

worksheet.Cell(row, 11).Value = task.ConnectionName;

worksheet.Cell(row, 12).Value = task.TaskSqlQuery;

worksheet.Cell(row, 13).Value = task.Variables;

worksheet.Cell(row, 14).Value = task.Parameters;

worksheet.Cell(row, 15).Value = task.Expressions;

worksheet.Cell(row, 16).Value = task.SourceComponent;

worksheet.Cell(row, 17).Value = task.SourceType;

worksheet.Cell(row, 18).Value = task.TargetComponent;

worksheet.Cell(row, 19).Value = task.TargetType;

worksheet.Cell(row, 20).Value = task.TargetTable;

worksheet.Cell(row, 21).Value = task.SourceConnectionName;

worksheet.Cell(row, 22).Value = task.TargetConnectionName;

worksheet.Cell(row, 23).Value = task.SendMailTask;

worksheet.Cell(row, 24).Value = task.ResultSetDetails;

worksheet.Cell(row, 25).Value = task.TaskComponentDetails;

workbook.SaveAs(filePath);

}

}

}

}

else if (DataSaveType == "SQL")

{

using (SqlConnection connection = new SqlConnection(\_connectionString))

{

connection.Open();

foreach (var task in result.ExtractTaskDetails)

{

string taskQuery = @"

INSERT INTO EventTaskDetails (PackageName, TaskName, TaskType, SqlQuery, ContainerName, PackagePath, Variables,

Parameters, Expressions, DataFlowDaskSourceName, DataFlowTaskSourceType,

DataFlowTaskTargetName, DataFlowTaskTargetType, DataFlowTaskTargetTable, ResultSetDetails, ContainerType, ContainerExpression,

EventHandlerName, EventHandlerType, EventType, DataFlowDaskSourceConnectionName, DataFlowDaskTargetConnectionName,

TaskConnectionName, TaskComponentDetails)

VALUES (@PackageName, @TaskName, @TaskType, @SqlQuery, @ContainerName, @PackagePath, @Variables, @Parameters, @Expressions, @DataFlowDaskSourceName,

@DataFlowTaskSourceType, @DataFlowTaskTargetName, @DataFlowTaskTargetType,

@DataFlowTaskTargetTable, @ResultSetDetails, @ContainerType, @ContainerExpression,

@EventHandlerName, @EventHandlerType, @EventType, @DataFlowDaskSourceConnectionName,

@DataFlowDaskTargetConnectionName, @TaskConnectionName, @TaskComponentDetails)";

using (SqlCommand cmd = new SqlCommand(taskQuery, connection))

{

cmd.Parameters.AddWithValue("@PackageName", task.PackageName);

cmd.Parameters.AddWithValue("@TaskName", task.TaskName);

cmd.Parameters.AddWithValue("@TaskType", task.TaskType);

cmd.Parameters.AddWithValue("@SqlQuery", task.TaskSqlQuery);

cmd.Parameters.AddWithValue("@ContainerName", task.ContainerName ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@PackagePath", task.PackagePath);

cmd.Parameters.AddWithValue("@Variables", task.Variables ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@Parameters", task.Parameters ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@Expressions", task.Expressions ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@DataFlowDaskSourceName", task.SourceComponent ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@DataFlowTaskSourceType", task.SourceType ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@DataFlowTaskTargetName", task.TargetComponent ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@DataFlowTaskTargetType", task.TargetType ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@DataFlowTaskTargetTable", task.TargetTable ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@SendMailTaskDetails", task.SendMailTask ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@ResultSetDetails", task.ResultSetDetails ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@ContainerType", task.ContainerType ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@ContainerExpression", task.ContainerExpression ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@EventHandlerName", task.EventHandlerName ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@EventHandlerType", task.EventHandlerType ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@EventType", task.EventType ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@DataFlowDaskSourceConnectionName", task.SourceConnectionName ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@DataFlowDaskTargetConnectionName", task.TargetConnectionName ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@TaskConnectionName", task.ConnectionName ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@TaskComponentDetails", task.TaskComponentDetails ?? (object)DBNull.Value);

cmd.ExecuteNonQuery();

}

}

}

}

}

private void SavePackageTaskmetadata(PackageAnalysisResult result, string filePath)

{

if (DataSaveType == "EXCEL")

{

bool workbookExists = DoesWorkbookExist(filePath);

foreach (var task in result.ExtractTaskDetails)

{

if (!string.IsNullOrEmpty(task.TaskName))

{

using (var workbook = workbookExists ? new XLWorkbook(filePath) : new XLWorkbook())

{

string SheetName = "PackageTaskDetails";

// Check if the common worksheet "ProjectParameters" already exists

var worksheet = workbook.Worksheets.FirstOrDefault(ws => ws.Name.Equals(SheetName, StringComparison.OrdinalIgnoreCase));

if (worksheet == null)

{

// Create the worksheet if it doesn't exist

worksheet = workbook.AddWorksheet(SheetName);

worksheet.Cell(1, 1).Value = "PackageName";

worksheet.Cell(1, 2).Value = "PackagePath";

worksheet.Cell(1, 3).Value = "TaskName";

worksheet.Cell(1, 4).Value = "TaskType";

worksheet.Cell(1, 5).Value = "ContainerName";

worksheet.Cell(1, 6).Value = "TaskConnectionName";

worksheet.Cell(1, 7).Value = "SqlQuery";

worksheet.Cell(1, 8).Value = "Variables";

worksheet.Cell(1, 9).Value = "Parameters";

worksheet.Cell(1, 10).Value = "Expressions";

worksheet.Cell(1, 11).Value = "DataFlowDaskSourceName";

worksheet.Cell(1, 12).Value = "DataFlowTaskSourceType";

worksheet.Cell(1, 13).Value = "DataFlowTaskTargetName";

worksheet.Cell(1, 14).Value = "DataFlowTaskTargetType";

worksheet.Cell(1, 15).Value = "DataFlowTaskTargetTable";

worksheet.Cell(1, 16).Value = "DataFlowDaskSourceConnectionName";

worksheet.Cell(1, 17).Value = "DataFlowDaskTargetConnectionName";

worksheet.Cell(1, 18).Value = "ResultSetDetails";

worksheet.Cell(1, 19).Value = "TaskComponentDetails";

}

var rows = worksheet.RowsUsed(); // Get all rows that have data

bool recordExists = false;

foreach (var row1 in rows)

{

string existingPackageName = row1.Cell(1).GetString();

string existingPackagePath = row1.Cell(2).GetString();

string existingTaskName = row1.Cell(3).GetString();

string existingTaskType = row1.Cell(4).GetString();

string existingContainerName = row1.Cell(5).GetString();

if (existingPackageName == task.PackageName &&

existingPackagePath == task.PackagePath &&

existingTaskName == task.TaskName &&

existingTaskType == task.TaskType &&

existingContainerName == task.ContainerName

)

{

recordExists = true;

break; // No need to check further rows if the record is found

}

}

if (recordExists)

{

//Console.WriteLine("Record already exists. No insertion needed.");

}

else

{

var lastRow = worksheet.LastRowUsed(); // Get the last used row

int lastRowNumber = lastRow != null ? lastRow.RowNumber() : 0;

int row = lastRowNumber + 1;

worksheet.Cell(row, 1).Value = task.PackageName;

worksheet.Cell(row, 2).Value = task.PackagePath;

worksheet.Cell(row, 3).Value = task.TaskName;

worksheet.Cell(row, 4).Value = task.TaskType;

worksheet.Cell(row, 5).Value = task.ContainerName;

worksheet.Cell(row, 6).Value = task.ConnectionName;

worksheet.Cell(row, 7).Value = task.TaskSqlQuery;

worksheet.Cell(row, 8).Value = task.Variables;

worksheet.Cell(row, 9).Value = task.Parameters;

worksheet.Cell(row, 10).Value = task.Expressions;

worksheet.Cell(row, 11).Value = task.SourceComponent;

worksheet.Cell(row, 12).Value = task.SourceType;

worksheet.Cell(row, 13).Value = task.TargetComponent;

worksheet.Cell(row, 14).Value = task.TargetType;

worksheet.Cell(row, 15).Value = task.TargetTable;

worksheet.Cell(row, 16).Value = task.SourceConnectionName;

worksheet.Cell(row, 17).Value = task.TargetConnectionName;

worksheet.Cell(row, 18).Value = task.ResultSetDetails;

worksheet.Cell(row, 19).Value = task.TaskComponentDetails;

workbook.SaveAs(filePath);

}

}

}

if (!string.IsNullOrEmpty(task.ContainerName))

{

using (var workbook = workbookExists ? new XLWorkbook(filePath) : new XLWorkbook())

{

string SheetName = "PackageContainerDetails";

// Check if the common worksheet "ProjectParameters" already exists

var worksheet = workbook.Worksheets.FirstOrDefault(ws => ws.Name.Equals(SheetName, StringComparison.OrdinalIgnoreCase));

if (worksheet == null)

{

// Create the worksheet if it doesn't exist

worksheet = workbook.AddWorksheet(SheetName);

worksheet.Cell(1, 1).Value = "PackageName";

worksheet.Cell(1, 2).Value = "PackagePath";

worksheet.Cell(1, 3).Value = "ContainerName";

worksheet.Cell(1, 4).Value = "ContainerType";

worksheet.Cell(1, 5).Value = "ContainerExpressions";

worksheet.Cell(1, 6).Value = "ContainerEnumerator";

}

var rows = worksheet.RowsUsed(); // Get all rows that have data

bool recordExists = false;

foreach (var row1 in rows)

{

string existingPackageName = row1.Cell(1).GetString();

string existingPackagePath = row1.Cell(2).GetString();

string existingContainerName = row1.Cell(3).GetString();

string existingContainerType = row1.Cell(4).GetString();

string existingContainerExpressions = row1.Cell(5).GetString();

string existingContainerEnumerator = row1.Cell(6).GetString();

if (existingPackageName == task.PackageName &&

existingPackagePath == task.PackagePath &&

existingContainerName == task.ContainerName &&

existingContainerType == task.ContainerType &&

existingContainerExpressions == task.ContainerExpression &&

existingContainerEnumerator == task.ContainerEnum

)

{

recordExists = true;

break; // No need to check further rows if the record is found

}

}

if (recordExists)

{

// Console.WriteLine("Record already exists. No insertion needed.");

}

else

{

var lastRow = worksheet.LastRowUsed(); // Get the last used row

int lastRowNumber = lastRow != null ? lastRow.RowNumber() : 0;

int row = lastRowNumber + 1;

worksheet.Cell(row, 1).Value = task.PackageName;

worksheet.Cell(row, 2).Value = task.PackagePath;

worksheet.Cell(row, 3).Value = task.ContainerName;

worksheet.Cell(row, 4).Value = task.ContainerType;

worksheet.Cell(row, 5).Value = task.ContainerExpression;

worksheet.Cell(row, 6).Value = task.ContainerEnum;

workbook.SaveAs(filePath);

}

}

}

}

}

else if (DataSaveType == "SQL")

{

using (SqlConnection connection = new SqlConnection(\_connectionString))

{

connection.Open();

foreach (var task in result.ExtractTaskDetails)

{

if (!string.IsNullOrEmpty(task.TaskName))

{

//Console.WriteLine($"For Loop Container '{task.ForeachContainerName}' has {task.ForeachTaskName} tasks.");

string taskQuery = @"

INSERT INTO PackageTaskDetails (PackageName, TaskName, TaskType, SqlQuery, ContainerName, PackagePath, Variables, Parameters,Expressions,

DataFlowDaskSourceName, DataFlowTaskSourceType, DataFlowTaskTargetName, DataFlowTaskTargetType,DataFlowTaskTargetTable,

ResultSetDetails, DataFlowDaskSourceConnectionName,

DataFlowDaskTargetConnectionName, TaskConnectionName, TaskComponentDetails)

SELECT DISTINCT @PackageName, @TaskName, @TaskType, @SqlQuery, @ContainerName, @PackagePath,

@Variables, @Parameters,

@Expressions, @DataFlowDaskSourceName, @DataFlowTaskSourceType, @DataFlowTaskTargetName, @DataFlowTaskTargetType,

@DataFlowTaskTargetTable, @ResultSetDetails, @DataFlowDaskSourceConnectionName, @DataFlowDaskTargetConnectionName,

@TaskConnectionName, @TaskComponentDetails

WHERE NOT EXISTS (

SELECT 1 FROM PackageTaskDetails

WHERE ContainerName = @ContainerName AND PackageName=@PackageName

AND PackagePath= @PackagePath AND TaskName= @TaskName )";

using (SqlCommand cmd = new SqlCommand(taskQuery, connection))

{

cmd.Parameters.AddWithValue("@PackageName", task.PackageName);

cmd.Parameters.AddWithValue("@TaskName", task.TaskName);

cmd.Parameters.AddWithValue("@TaskType", task.TaskType);

cmd.Parameters.AddWithValue("@SqlQuery", task.TaskSqlQuery);

cmd.Parameters.AddWithValue("@ContainerName", task.ContainerName);

cmd.Parameters.AddWithValue("@PackagePath", task.PackagePath);

cmd.Parameters.AddWithValue("@Variables", task.Variables ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@Parameters", task.Parameters ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@Expressions", task.Expressions ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@DataFlowDaskSourceName", task.SourceComponent ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@DataFlowTaskSourceType", task.SourceType ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@DataFlowTaskTargetName", task.TargetComponent ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@DataFlowTaskTargetType", task.TargetType ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@DataFlowTaskTargetTable", task.TargetTable ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@ResultSetDetails", task.ResultSetDetails ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@DataFlowDaskSourceConnectionName", task.SourceConnectionName ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@DataFlowDaskTargetConnectionName", task.TargetConnectionName ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@TaskConnectionName", task.ConnectionName ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@TaskComponentDetails", task.TaskComponentDetails ?? (object)DBNull.Value);

cmd.ExecuteNonQuery();

}

}

if (!string.IsNullOrEmpty(task.ContainerName))

{

string containerQuery = @"

INSERT INTO PackageContainerDetails (PackageName, ContainerName, ContainerType, ContainerExpressions, ContainerEnumerator, PackagePath)

SELECT DISTINCT @PackageName, @ContainerName, @ContainerType, @ContainerExpressions, @ContainerEnumerator, @PackagePath WHERE NOT EXISTS (

SELECT 1 FROM PackageContainerDetails

WHERE ContainerName = @ContainerName AND ContainerType = @ContainerType AND PackageName=@PackageName AND PackagePath= @PackagePath

AND ContainerExpressions= ISNULL(@ContainerExpressions,'')AND ContainerEnumerator= ISNULL(@ContainerEnumerator,'') )";

using (SqlCommand cmd = new SqlCommand(containerQuery, connection))

{

cmd.Parameters.AddWithValue("@PackageName", task.PackageName);

cmd.Parameters.AddWithValue("@ContainerName", task.ContainerName);

cmd.Parameters.AddWithValue("@ContainerType", task.ContainerType);

cmd.Parameters.AddWithValue("@ContainerExpressions", task.ContainerExpression ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@ContainerEnumerator", task.ContainerEnum ?? (object)DBNull.Value);

cmd.Parameters.AddWithValue("@PackagePath", task.PackagePath);

cmd.ExecuteNonQuery();

}

}

}

}

}

}

private void SaveConnectionsmetadata(PackageAnalysisResult result, string filePath)

{

if (DataSaveType == "EXCEL")

{

bool workbookExists = DoesWorkbookExist(filePath);

foreach (var connectionInfo in result.Connections)

{

using (var workbook = workbookExists ? new XLWorkbook(filePath) : new XLWorkbook())

{

// Check if the common worksheet "ProjectParameters" already exists

var worksheet = workbook.Worksheets.FirstOrDefault(ws => ws.Name.Equals("PackageConnectionDetails", StringComparison.OrdinalIgnoreCase));

if (worksheet == null)

{

// Create the worksheet if it doesn't exist

worksheet = workbook.AddWorksheet("PackageConnectionDetails");

worksheet.Cell(1, 1).Value = "PackageName";

worksheet.Cell(1, 2).Value = "PackagePath";

worksheet.Cell(1, 3).Value = "ConnectionName";

worksheet.Cell(1, 4).Value = "ConnectionType";

worksheet.Cell(1, 5).Value = "ConnectionExpressions";

worksheet.Cell(1, 6).Value = "ConnectionString";

worksheet.Cell(1, 7).Value = "ConnectionID";

worksheet.Cell(1, 8).Value = "IsProjectConnection";

}

var lastRow = worksheet.LastRowUsed(); // Get the last used row

int lastRowNumber = lastRow != null ? lastRow.RowNumber() : 0;

int row = lastRowNumber + 1;

worksheet.Cell(row, 1).Value = result.PackageName;

worksheet.Cell(row, 2).Value = result.PackagePath;

worksheet.Cell(row, 3).Value = connectionInfo.ConnectionName;

worksheet.Cell(row, 4).Value = connectionInfo.ConnectionType;

worksheet.Cell(row, 5).Value = connectionInfo.ConnectionExpressions;

worksheet.Cell(row, 6).Value = connectionInfo.ConnectionString;

worksheet.Cell(row, 7).Value = connectionInfo.ConnectionID;

worksheet.Cell(row, 8).Value = connectionInfo.IsProjectConnection;

workbook.SaveAs(filePath);

}

}

}

else if (DataSaveType == "SQL")

{

using (SqlConnection connection = new SqlConnection(\_connectionString))

{

connection.Open();

foreach (var connectionInfo in result.Connections)

{

string connectionQuery = @"

INSERT INTO PackageConnectionDetails (PackageName, ConnectionName, ConnectionType, PackagePath,

ConnectionExpressions, ConnectionString, ConnectionDTSID, IsProjectConnection)

VALUES (@PackageName, @ConnectionName, @ConnectionType, @PackagePath, @ConnectionExpressions, @ConnectionString, @ConnectionID, @IsProjectConnection)";

using (SqlCommand cmd = new SqlCommand(connectionQuery, connection))

{

cmd.Parameters.AddWithValue("@PackageName", result.PackageName);

cmd.Parameters.AddWithValue("@ConnectionName", connectionInfo.ConnectionName);

cmd.Parameters.AddWithValue("@ConnectionType", connectionInfo.ConnectionType);

cmd.Parameters.AddWithValue("@PackagePath", result.PackagePath);

cmd.Parameters.AddWithValue("@ConnectionExpressions", connectionInfo.ConnectionExpressions);

cmd.Parameters.AddWithValue("@ConnectionString", connectionInfo.ConnectionString);

cmd.Parameters.AddWithValue("@ConnectionID", connectionInfo.ConnectionID);

cmd.Parameters.AddWithValue("@IsProjectConnection", connectionInfo.IsProjectConnection);

cmd.ExecuteNonQuery();

}

}

}

}

}

private void SaveProjectParametermetadata(PackageAnalysisResult result, string filePath)

{

if (DataSaveType == "EXCEL")

{

bool workbookExists = DoesWorkbookExist(filePath);

foreach (var ParameterInfo in result.ProjectParameterDetails)

{

if (!string.IsNullOrEmpty(result.PackageName))

{

using (var workbook = workbookExists ? new XLWorkbook(filePath) : new XLWorkbook())

{

// Check if the common worksheet "ProjectParameters" already exists

var worksheet = workbook.Worksheets.FirstOrDefault(ws => ws.Name.Equals("ProjectParameterDetails", StringComparison.OrdinalIgnoreCase));

if (worksheet == null)

{

// Create the worksheet if it doesn't exist

worksheet = workbook.AddWorksheet("ProjectParameterDetails");

worksheet.Cell(1, 1).Value = "ProjectPath";

worksheet.Cell(1, 2).Value = "ParameterName";

worksheet.Cell(1, 3).Value = "ParameterValue";

worksheet.Cell(1, 4).Value = "ParameterDataType";

}

var lastRow = worksheet.LastRowUsed(); // Get the last used row

int lastRowNumber = lastRow != null ? lastRow.RowNumber() : 0;

int row = lastRowNumber + 1;

worksheet.Cell(row, 1).Value = result.PackagePath;

worksheet.Cell(row, 2).Value = ParameterInfo.ParameterName;

worksheet.Cell(row, 3).Value = ParameterInfo.Value;

worksheet.Cell(row, 4).Value = ParameterInfo.DataType;

workbook.SaveAs(filePath);

}

}

}

}

else if (DataSaveType== "SQL")

{

using (SqlConnection connection = new SqlConnection(\_connectionString))

{

connection.Open();

foreach (var ParameterInfo in result.ProjectParameterDetails)

{

if (!string.IsNullOrEmpty(result.PackageName))

{

string connectionQuery = @"

INSERT INTO ProjectParameterDetails (ParameterName, ParameterValue, ParameterDataType, ProjectPath)

VALUES (@ParameterName, @ParameterValue, @ParameterDataType, @ProjectPath)";

using (SqlCommand cmd = new SqlCommand(connectionQuery, connection))

{

cmd.Parameters.AddWithValue("@ParameterName", ParameterInfo.ParameterName);

cmd.Parameters.AddWithValue("@ParameterValue", ParameterInfo.Value);

cmd.Parameters.AddWithValue("@ParameterDataType", ParameterInfo.DataType);

cmd.Parameters.AddWithValue("@ProjectPath", result.PackagePath);

cmd.ExecuteNonQuery();

}

}

}

}

}

}

private void SaveUdateConnectionName(String filePath)

{

if (DataSaveType == "EXCEL")

{

using (var workbook = new XLWorkbook(filePath))

{

// Get the sheets

string sheetName1 = "PackageTaskDetails";

string sheetName2 = "PackageConnectionDetails";

string sheetName3 = "EventHandlerTaskDetails";

var sheet1 = workbook.Worksheet(sheetName1);

var sheet2 = workbook.Worksheet(sheetName2);

var sheet3 = workbook.Worksheet(sheetName3);

// Find the last row in each sheet (assuming data starts at row 1)

int lastRowSheet1 = sheet1.LastRowUsed().RowNumber();

int lastRowSheet2 = sheet2.LastRowUsed().RowNumber();

int lastRowSheet3 = sheet3.LastRowUsed().RowNumber();

lastRowSheet1 = lastRowSheet1 + 1;

lastRowSheet2 = lastRowSheet2 + 1;

lastRowSheet3 = lastRowSheet3 + 1;

// Loop through rows of sheet1 to find matching values

for (int row1 = 1; row1 <= lastRowSheet2; row1++)

{

var cellValue1Sheet2 = sheet2.Cell(row1, 1).GetValue<string>();

var cellValue2Sheet2 = sheet2.Cell(row1, 2).GetValue<string>();

var cellValue3Sheet2 = sheet2.Cell(row1, 7).GetValue<string>();

var cellValue4Sheet2 = sheet2.Cell(row1, 3).GetValue<string>();

// Loop through rows of sheet2 to find matching values

for (int row2 = 1; row2 <= lastRowSheet1; row2++)

{

var cellValue1Sheet1 = sheet1.Cell(row2, 1).GetValue<string>();

var cellValue2Sheet1 = sheet1.Cell(row2, 2).GetValue<string>();

var cellValue3Sheet1 = sheet1.Cell(row2, 6).GetValue<string>();

var cellValue4Sheet1 = sheet1.Cell(row2, 16).GetValue<string>();

var cellValue5Sheet1 = sheet1.Cell(row2, 17).GetValue<string>();

// If a match is found, update the cell in sheet1

if (

cellValue2Sheet2 == cellValue2Sheet1 &&

cellValue3Sheet2 == cellValue3Sheet1)

{

// Update the specific cell in sheet1 (columnToUpdate) with the value from updateValue

sheet1.Cell(row2, 6).Value = cellValue4Sheet2;

}

else if (

cellValue2Sheet2 == cellValue2Sheet1 &&

cellValue3Sheet2 == cellValue4Sheet1)

{

// Update the specific cell in sheet1 (columnToUpdate) with the value from updateValue

sheet1.Cell(row2, 16).Value = cellValue4Sheet2;

}

else if (

cellValue2Sheet2 == cellValue2Sheet1 &&

cellValue3Sheet2 == cellValue5Sheet1)

{

// Update the specific cell in sheet1 (columnToUpdate) with the value from updateValue

sheet1.Cell(row2, 17).Value = cellValue4Sheet2;

}

}

for (int row3 = 1; row3 <= lastRowSheet3; row3++)

{

var cellValue1Sheet3 = sheet3.Cell(row3, 1).GetValue<string>();

var cellValue2Sheet3 = sheet3.Cell(row3, 2).GetValue<string>();

var cellValue3Sheet3 = sheet3.Cell(row3, 11).GetValue<string>();

var cellValue4Sheet3 = sheet3.Cell(row3, 21).GetValue<string>();

var cellValue5Sheet3 = sheet3.Cell(row3, 22).GetValue<string>();

// If a match is found, update the cell in sheet1

if (

cellValue2Sheet2 == cellValue2Sheet3 &&

cellValue3Sheet2 == cellValue3Sheet3)

{

// Update the specific cell in sheet1 (columnToUpdate) with the value from updateValue

sheet3.Cell(row3, 11).Value = cellValue4Sheet2;

}

else if (

cellValue2Sheet2 == cellValue2Sheet3 &&

cellValue3Sheet2 == cellValue4Sheet3)

{

// Update the specific cell in sheet1 (columnToUpdate) with the value from updateValue

sheet3.Cell(row3, 21).Value = cellValue4Sheet2;

}

else if (

cellValue2Sheet2 == cellValue2Sheet3 &&

cellValue3Sheet2 == cellValue5Sheet3)

{

// Update the specific cell in sheet1 (columnToUpdate) with the value from updateValue

sheet3.Cell(row3, 22).Value = cellValue4Sheet2;

}

}

}

// Save the workbook after updating

workbook.Save();

//Console.WriteLine("Cell updated successfully.");

}

}

else if (DataSaveType == "SQL")

{

using (SqlConnection connection = new SqlConnection(\_connectionString))

{

connection.Open();

string connectionQuery = @"Update task set task.TaskConnectionName= conn.ConnectionName

From PackageTaskDetails task INNER JOIN PackageConnectionDetails conn (NOLOCK) on

conn.PackagePath = task.PackagePath AND task.TaskConnectionName=conn.ConnectionDTSID

WHERE ISNULL(task.TaskConnectionName,'')<>'';

Update task set task.DataFlowDaskSourceConnectionName = sconn.ConnectionName ,

Task.DataFlowDaskTargetConnectionName = Tconn.ConnectionName

From PackageTaskDetails task INNER JOIN PackageConnectionDetails Sconn (NOLOCK) on

sconn.PackagePath = task.PackagePath AND task.DataFlowDaskSourceConnectionName=sconn.ConnectionDTSID

INNER JOIN PackageConnectionDetails Tconn (NOLOCK) on

Tconn.PackagePath = task.PackagePath AND task.DataFlowDaskTargetConnectionName=Tconn.ConnectionDTSID

WHERE ISNULL(task.DataFlowDaskSourceConnectionName,'')<> '';

Update task set task.TaskConnectionName= conn.ConnectionName

From EventTaskDetails task INNER JOIN PackageConnectionDetails conn (NOLOCK) on

conn.PackagePath = task.PackagePath AND task.TaskConnectionName=conn.ConnectionDTSID

WHERE ISNULL(task.TaskConnectionName,'')<>'';

Update task set task.DataFlowDaskSourceConnectionName = sconn.ConnectionName ,

Task.DataFlowDaskTargetConnectionName = Tconn.ConnectionName

From EventTaskDetails task INNER JOIN PackageConnectionDetails Sconn (NOLOCK) on

sconn.PackagePath = task.PackagePath AND task.DataFlowDaskSourceConnectionName=sconn.ConnectionDTSID

INNER JOIN PackageConnectionDetails Tconn (NOLOCK) on

Tconn.PackagePath = task.PackagePath AND task.DataFlowDaskTargetConnectionName=Tconn.ConnectionDTSID

WHERE ISNULL(task.DataFlowDaskSourceConnectionName,'')<> '';

UPDATE task set task.ONSuccessPrecedenceConstrainttoTask= '',task.ONSuccessPrecedenceConstraintExpression='',task.ONSuccessPrecedenceConstraintEvalOP='' ,

task.ONSuccessPrecedenceConstraintLogicalAnd= '' ,task.ONFailurePrecedenceConstrainttoTask= '',task.ONFailurePrecedenceConstraintExpression='',

task.ONFailurePrecedenceConstraintEvalOP='' ,task.ONFailurePrecedenceConstraintLogicalAnd= '', task.ONCompletionPrecedenceConstrainttoTask= '',

task.ONCompletionPrecedenceConstraintExpression='',task.ONCompletionPrecedenceConstraintEvalOP='' ,task.ONCompletionPrecedenceConstraintLogicalAnd= ''

FROM PackageTaskDetails task

UPDATE task set task.ONSuccessPrecedenceConstrainttoTask= PCD.PrecedenceConstraintto,

task.ONSuccessPrecedenceConstraintExpression=PrecedenceConstraintExpression,

task.ONSuccessPrecedenceConstraintEvalOP=PrecedenceConstraintEvalOP ,

task.ONSuccessPrecedenceConstraintLogicalAnd= PrecedenceConstraintLogicalAnd

FROM PackageTaskDetails task INNER JOIN PrecedenceConstraintDetails PCD(NOLOCK) ON

PCD.PrecedenceConstraintFrom=task.TaskName AND PCD.PackageName=task.PackageName

AND PCD.PackagePath=task.PackagePath AND ISNULL(PCD.ContainerName,'')=ISNULL(task.ContainerName,'')

WHERE PCD.PrecedenceConstraintValue='Success';

UPDATE task set task.ONFailurePrecedenceConstrainttoTask= PCD.PrecedenceConstraintto,

task.ONFailurePrecedenceConstraintExpression=PrecedenceConstraintExpression,

task.ONFailurePrecedenceConstraintEvalOP=PrecedenceConstraintEvalOP ,

task.ONFailurePrecedenceConstraintLogicalAnd= PrecedenceConstraintLogicalAnd

FROM PackageTaskDetails task INNER JOIN PrecedenceConstraintDetails PCD(NOLOCK) ON

PCD.PrecedenceConstraintFrom=task.TaskName AND PCD.PackageName=task.PackageName

AND PCD.PackagePath=task.PackagePath AND ISNULL(PCD.ContainerName,'')=ISNULL(task.ContainerName,'')

WHERE PCD.PrecedenceConstraintValue='Failure';

UPDATE task set task.ONCompletionPrecedenceConstrainttoTask= PCD.PrecedenceConstraintto,

task.ONCompletionPrecedenceConstraintExpression=PrecedenceConstraintExpression,

task.ONCompletionPrecedenceConstraintEvalOP=PrecedenceConstraintEvalOP ,

task.ONCompletionPrecedenceConstraintLogicalAnd= PrecedenceConstraintLogicalAnd

FROM PackageTaskDetails task INNER JOIN PrecedenceConstraintDetails PCD(NOLOCK) ON

PCD.PrecedenceConstraintFrom=task.TaskName AND PCD.PackageName=task.PackageName

AND PCD.PackagePath=task.PackagePath AND ISNULL(PCD.ContainerName,'')=ISNULL(task.ContainerName,'')

WHERE PCD.PrecedenceConstraintValue='Completion';

UPDATE PA set PA.Complexcity=CASE WHEN Final.TaskCount+Final.ContainerCount+Final.ComponentCount <5 THEN 'Simple'

WHEN Final.TaskCount+Final.ContainerCount+Final.ComponentCount >5 and Final.TaskCount+Final.ContainerCount+Final.ComponentCount<10 THEN 'Medium'

WHEN Final.TaskCount+Final.ContainerCount+Final.ComponentCount >10 THEN 'Complex' ELSE 'Simple' END

FROM PackageAnalysisResults PA LEFT JOIN (

SELECT PackageName,PackagePath ,SUm(TaskCount) TaskCount,Sum(ContainerCount ) ContainerCount,Sum(ComponentCount ) ComponentCount FROM (

SELECT PT.PackageName,Pt.PackagePath,Sum(Case When TaskType<>'ExecutePackageTask' Then 1 Else 0 End) 'TaskCount',

0'ContainerCount',0 as 'ComponentCount' FROM PackageTaskDetails PT GROUP BY PT.PackageName,Pt.PackagePath

UNION ALL

SELECT DISTINCT PT.PackageName,Pt.PackagePath ,1 'TaskCount',0 as 'ContainerCount',0 as 'ComponentCount'

FROM PackageTaskDetails PT WHERE TaskType='ExecutePackageTask'

UNION ALL

SELECT DISTINCT PC.PackageName,PC.PackagePath ,0 'TaskCount',1 as 'ContainerCount',0 as 'ComponentCount'

FROM PackageContainerDetails PC WHERE PC.ContainerType='Sequence'

UNION ALL

SELECT PC.PackageName,PC.PackagePath ,0 'TaskCount',Count(1) as 'ContainerCount',0 as 'ComponentCount'

FROM PackageContainerDetails PC WHERE PC.ContainerType<>'Sequence'

GROUP BY PackageName,PackagePath

UNION ALL

SELECT PC.PackageName,PC.PackagePath ,0 'TaskCount',0 as 'ContainerCount',Count(distinct ComponentName) as 'ComponentCount'

FROM DataFlowTaskMappingDetails PC

GROUP BY PackageName,PackagePath

) A

GROUP BY PackageName,PackagePath

) Final ON Final.PackageName=PA.PackageName AND Final.PackagePath=PA.PackageFolder;";

using (SqlCommand cmd = new SqlCommand(connectionQuery, connection))

{

cmd.ExecuteNonQuery();

}

}

}

}

private void TruncateTable()

{

if (DataSaveType == "SQL")

{

using (SqlConnection connection = new SqlConnection(\_connectionString))

{

connection.Open();

string connectionQuery = @"TRUNCATE TABLE PackageAnalysisResults

TRUNCATE TABLE PackageTaskDetails

TRUNCATE TABLE PackageConnectionDetails

TRUNCATE TABLE PackageContainerDetails

TRUNCATE TABLE ProjectParameterDetails

TRUNCATE TABLE PackageVariableParameterDetails

TRUNCATE TABLE DataFlowTaskMappingDetails

TRUNCATE TABLE PrecedenceConstraintDetails

TRUNCATE TABLE EventTaskDetails";

using (SqlCommand cmd = new SqlCommand(connectionQuery, connection))

{

cmd.ExecuteNonQuery();

}

}

}

}

private void LogError(string packagePath, Exception ex)

{

Console.WriteLine($"Error analyzing {packagePath}: {ex.Message}");

Console.WriteLine($"Error analyzing {packagePath}: {ex.Message}");

}

}

public class PackageAnalysisResult

{

public string PackageName { get; set; }

public DateTime CreatedDate { get; set; }

public string CreatedBy { get; set; }

public List<TaskInfo> Tasks { get; set; }

public List<TaskInfo> Seqtasks { get; set; }

public List<TaskInfo> Foreachtasks { get; set; }

public List<TaskInfo> Forlooptasks { get; set; }

public List<ConnectionInfo> Connections { get; set; }

public TimeSpan ExecutionTime { get; set; }

public string PackagePath { get; set; }

public List<ContainerInfo> Containers { get; set; }

public string DTSXXML { get; set; }

public List<TaskInfo> SequenceContainerTaskCount { get; set; }

public List<TaskInfo> ForeachContainerTaskCount { get; set; }

public List<TaskInfo> ForLoopContainerTaskCount { get; set; }

public List<VariableInfo> Variables { get; set; }

public List<DataFlowTaskInfo> DataFlowTaskDetails { get; set; }

public List<PrecedenceConstraintInfo> PrecedenceConstraintDetails { get; set; }

public List<TaskInfo> ExtractTaskDetails { get; set; }

public List<ProjectParameterInfo> ProjectParameterDetails { get; set; }

}

public class TaskInfo

{

public string PackageName { get; set; }

public string PackagePath { get; set; }

public string EventHandlerName { get; set; }

public string EventHandlerType { get; set; }

public string EventType { get; set; }

public string TaskName { get; set; }

public string TaskType { get; set; }

public string TaskSqlQuery { get; set; }

public string ContainerName { get; set; }

public string ContainerType { get; set; }

public string ContainerExpression { get; set; }

public string ContainerEnum { get; set; }

public string Variables { get; set; }

public string Parameters { get; set; }

public string Expressions { get; set; }

public string ExecuteProcessDetails { get; set; }

public string FileSystemSourcePath { get; set; }

public string FileSystemDestinationPath { get; set; }

public string SourceComponent { get; set; }

public string TargetComponent { get; set; }

public string SourceType { get; set; }

public string TargetType { get; set; }

public string TargetTable { get; set; }

public string SendMailTask { get; set; }

public string ScriptTask { get; set; }

public string FTPTask { get; set; }

public string ExecutePackage { get; set; }

public string ResultSetDetails { get; set; }

public string SeqTaskName { get; set; }

public string ForeachTaskName { get; set; }

public string ForloopTaskName { get; set; }

public string ConnectionName { get; set; }

public string SourceConnectionName { get; set; }

public string TargetConnectionName { get; set; }

public string TaskComponentDetails { get; set; }

}

public class ConnectionInfo

{

public string ConnectionName { get; set; }

public string ConnectionType { get; set; }

public string ConnectionString { get; set; }

public string ConnectionExpressions { get; set; }

public string ConnectionID { get; set; }

public string IsProjectConnection { get; set; }

}

public class ContainerInfo

{

public string ContainerName { get; set; }

public string ContainerType { get; set; }

public string ContainerExpression{ get; set; }

}

public class VariableInfo

{

public string Name { get; set; }

public string Value { get; set; }

public string DataType { get; set; }

public string Namespace { get; set; }

public int IsParameter { get; set; }

}

public class TaskParameterInfo

{

public string ParameterName { get; set; }

public string ParameterType { get; set; }

public string DataType { get; set; }

public string Value { get; set; }

public string DtsVariableName { get; set; }

}

public class DataFlowTaskInfo

{

public string ColumnName { get; set; }

public string ColumnType { get; set; }

public string DataType { get; set; }

public string TargetColumn { get; set; }

public string componentName { get; set; }

public string DataConversion { get; set; }

public string PackageName { get; set; }

public string PackagePath { get; set; }

public string TaskName { get; set; }

public string isEventHandler { get; set; }

public string componentPropertyDetails { get; set; }

public string ColumnPropertyDetails { get; set; }

}

public class PrecedenceConstraintInfo

{

public string PrecedenceConstraintFrom { get; set; }

public string PrecedenceConstraintTo { get; set; }

public string PrecedenceConstraintValue { get; set; }

public string PrecedenceConstraintLogicalAnd { get; set; }

public string PrecedenceConstraintEvalOP { get; set; }

public string PrecedenceConstraintExpression { get; set; }

public string ContainerName { get; set; }

public string PackageName { get; set; }

public string PackagePath { get; set; }

}

public class ProjectParameterInfo

{

public string ParameterName { get; set; }

public string DataType { get; set; }

public string Value { get; set; }

}

}