**1. Export to excel-using EPPlus**

using OfficeOpenXml;

           DataTable dt = new DataTable();

            dt = GetDataTable();

            using (ExcelPackage p = new ExcelPackage())

            {

                ExcelWorksheet ws1 = p.Workbook.Worksheets.Add("IBC Issues");

                ws1.Cells["A1"].LoadFromDataTable(dt, true, OfficeOpenXml.Table.TableStyles.None);

                int argb = Int32.Parse("FFF333", NumberStyles.HexNumber);

                for(int i=0; i< dt.Columns.Count; i++)

                {

                    ws1.Cells[1,i+1].Style.Fill.PatternType = OfficeOpenXml.Style.ExcelFillStyle.Solid;

                    ws1.Cells[1, i + 1].Style.Fill.BackgroundColor.SetColor(Color.FromArgb(argb));

                }

                Byte[] bin = p.GetAsByteArray();

                HttpContext.Current.Response.Clear();

                HttpContext.Current.Response.AddHeader("content-disposition", "attachment; filename=IBCIssues.xlsx");

                HttpContext.Current.Response.ContentType = "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet";

                HttpContext.Current.Response.BinaryWrite(bin);

                HttpContext.Current.Response.End();

            }

**2. Export to excel-using EPPlus with custom functions**

ElectricalBAL electricalBAL = new ElectricalBAL();

DataTable electricaldt = new DataTable();

electricaldt = electricalBAL.GetSupplierProvidedInterCablesData(templateID, fileID);

electricaldt.Columns.Remove("Supplier\_Provided\_InterCables\_ID");

using (ExcelPackage p = new ExcelPackage())

{

ExcelWorksheet ws1 = p.Workbook.Worksheets.Add("Facility Power Connections");

ExcelWorksheet ws2 = p.Workbook.Worksheets.Add("Facility Signal Connections");

ExcelWorksheet ws3 = p.Workbook.Worksheets.Add("Customer-Provided Inter. Cables");

ExcelWorksheet ws4 = p.Workbook.Worksheets.Add("Supplier-Provided Inter. Cables");

ExcelWorksheet ws5 = p.Workbook.Worksheets.Add("Supplier-Provided Transformers");

ExcelWorksheet ws6 = p.Workbook.Worksheets.Add("Supplier-Provided GND Bus Connections");

ws4.View.FreezePanes(5, 6);

ws4.Cells["1:1"].Style.Font.Bold = true;

ws4.Cells["1:1"].Style.Font.Size = 12;

ws4.Cells["2:2"].Style.Font.Bold = true;

ws4.Cells["2:2"].Style.Font.Size = 12;

ws4.Cells["3:3"].Style.Font.Bold = true;

ws4.Cells["3:3"].Style.Font.Size = 12;

ws4.Cells["4:4"].Style.Font.Bold = true;

ws4.Cells["4:4"].Style.Font.Size = 12;

ws4.Row(4).Height = 50;

for (int ExcelColumnNo = 1; ExcelColumnNo <= 26; ExcelColumnNo++)

{

ws4.Column(ExcelColumnNo).Style.WrapText = true;

ws4.Column(ExcelColumnNo).Width = 25;

}

//ws4.Cells["A1:K30"].AutoFitColumns();

ws4.Cells[1, 1, 1, 5].Merge = true;

ws4.Cells[1, 6, 1, 26].Merge = true;

ws4.Cells[1, 6, 1, 26].Value = "SUPPLIER-PROVIDED INTERCONNECT CABLES";

ws4.Cells[1, 6, 1, 26].Style.HorizontalAlignment = ExcelHorizontalAlignment.Center;

ws4.Cells[1, 6, 1, 26].Style.VerticalAlignment = ExcelVerticalAlignment.Center;

ws4.Cells[2, 2, 2, 3].Merge = true;

ws4.Cells[2, 2, 2, 3].Value = "SOURCE";

ws4.Cells[2, 2, 2, 3].Style.HorizontalAlignment = ExcelHorizontalAlignment.Center;

ws4.Cells[2, 2, 2, 3].Style.VerticalAlignment = ExcelVerticalAlignment.Center;

ws4.Cells[2, 4, 2, 5].Merge = true;

ws4.Cells[2, 4, 2, 5].Value = "DESTINATION/LOAD";

ws4.Cells[2, 4, 2, 5].Style.HorizontalAlignment = ExcelHorizontalAlignment.Center;

ws4.Cells[2, 4, 2, 5].Style.VerticalAlignment = ExcelVerticalAlignment.Center;

ws4.Cells[2, 6, 2, 8].Merge = true;

ws4.Cells[2, 6, 2, 8].Value = "CIRCUIT DATA";

ws4.Cells[2, 6, 2, 8].Style.HorizontalAlignment = ExcelHorizontalAlignment.Center;

ws4.Cells[2, 6, 2, 8].Style.VerticalAlignment = ExcelVerticalAlignment.Center;

ws4.Cells[2, 9, 2, 25].Merge = true;

ws4.Cells[2, 9, 2, 25].Value = "CABLE/CONNECTION DATA";

ws4.Cells[2, 9, 2, 25].Style.HorizontalAlignment = ExcelHorizontalAlignment.Center;

ws4.Cells[2, 9, 2, 25].Style.VerticalAlignment = ExcelVerticalAlignment.Center;

ws4.Cells[3, 18, 3, 19].Merge = true;

ws4.Cells[3, 18, 3, 19].Style.HorizontalAlignment = ExcelHorizontalAlignment.Center;

ws4.Cells[3, 18, 3, 19].Style.VerticalAlignment = ExcelVerticalAlignment.Center;

ws4.Cells[3, 21, 3, 23].Merge = true;

ws4.Cells[3, 21, 3, 23].Style.HorizontalAlignment = ExcelHorizontalAlignment.Center;

ws4.Cells[3, 21, 3, 23].Style.VerticalAlignment = ExcelVerticalAlignment.Center;

//for (int colcount = 1; colcount <= 17; colcount++)

//{

// ws4.Cells[3, colcount, 4, colcount].Merge = true;

// ws4.Cells[3, colcount, 4, colcount].Style.HorizontalAlignment = ExcelHorizontalAlignment.Center;

// ws4.Cells[3, colcount, 4, colcount].Style.VerticalAlignment = ExcelVerticalAlignment.Center;

//}

//ws4.Cells[3, 20, 4, 20].Merge = true;

//ws4.Cells[3, 20, 4, 20].Style.HorizontalAlignment = ExcelHorizontalAlignment.Center;

//ws4.Cells[3, 20, 4, 20].Style.VerticalAlignment = ExcelVerticalAlignment.Center;

//ws4.Cells[3, 24, 4, 24].Merge = true;

//ws4.Cells[3, 24, 4, 24].Style.HorizontalAlignment = ExcelHorizontalAlignment.Center;

//ws4.Cells[3, 24, 4, 24].Style.VerticalAlignment = ExcelVerticalAlignment.Center;

//ws4.Cells[3, 25, 4, 25].Merge = true;

//ws4.Cells[3, 25, 4, 25].Style.HorizontalAlignment = ExcelHorizontalAlignment.Center;

//ws4.Cells[3, 25, 4, 25].Style.VerticalAlignment = ExcelVerticalAlignment.Center;

//ws4.Cells[3, 26, 4, 26].Merge = true;

//ws4.Cells[3, 26, 4, 26].Style.HorizontalAlignment = ExcelHorizontalAlignment.Center;

//ws4.Cells[3, 26, 4, 26].Style.VerticalAlignment = ExcelVerticalAlignment.Center;

ws4.Cells["A4"].LoadFromDataTable(electricaldt, true, OfficeOpenXml.Table.TableStyles.None);

ws4.Cells["A4"].Value = "CIRCUIT IDENTIFIER";

ws4.Cells["A4"].AddComment("Each item is either single conductor within a given circuit or a group of conductors in the same circuit with identical characteristics. Example:S001, S020, S009", " ");

ws4.Cells["B4"].Value = "CONNECTION LOCATIONS (EQUIPMENT LOCATION)";

ws4.Cells["B4"].AddComment("The equipment component within which the connection is made", " ");

ws4.Cells["C4"].Value = "CONNECTION LABEL";

ws4.Cells["C4"].AddComment("The physical label used to identify the temination device within the equipmen component", " ");

ws4.Cells["D4"].Value = "CONNECTION LOCATIONS (EQUIPMENT LOCATION)";

ws4.Cells["D4"].AddComment("The equipment component within which the connection is made", " ");

ws4.Cells["E4"].Value = "CONNECTION LABEL";

ws4.Cells["E4"].AddComment("The physical label used to identify the temination device within the equipmen component", " ");

ws4.Cells["F4"].Value = "NEC CLASS";

ws4.Cells["F4"].AddComment("The classification of the power source supplying this circuit as referenced in the NEC.1 = Class 1(see NEC Article 725) 1PL = Class 1 Power Limited(see NEC Article 725) 2 = Class 2(see NEC Article 725)3 = Class 3(see NEC Article 725)P = Power / RF G=Grond/Earth(see NEC Article 250) FO=Fiber Optic(see NEC Article 770) ", " ");

ws4.Cells["G4"].Value = "VOLTAGE (V)";

ws4.Cells["G4"].AddComment("Nominal circuit operating voltage(expressed in volts).If multiple circuits are present in a single cable, indicate the highest operating voltage within the cable.", " ");

ws4.Cells["H4"].Value = "UPSTREAM CIRCUIT BREAKER TRIP OR FUSE (A)";

ws4.Cells["H4"].AddComment("The upstream power source circuit breaker trip rating or fuse current rating.If no overcurrent protection is provided place '-' in the cell.", " ");

ws4.Cells["I4"].Value = "CONDUCTOR SIZE (AWG OR KCMIL)";

ws4.Cells["I4"].AddComment("The Size of the conductor(s) in AWG or kcmil", " ");

ws4.Cells["J4"].Value = "EQUIPMENT SUPPLIER PART NUMBER";

ws4.Cells["J4"].AddComment("This is the tool supplier's part number assigned to the cable or conductor.", " ");

ws4.Cells["K4"].Value = "NUMBER OF CONDUCTORS (EXCLUDING GROUNDS)";

ws4.Cells["K4"].AddComment("The total number of circuit conductors excluding any ground wires, drain wires, or shields.", " ");

ws4.Cells["L4"].Value = "NUMBER OF GROUND WIRES";

ws4.Cells["L4"].AddComment("The total number of ground wires, drain wires, and shields.", " ");

ws4.Cells["M4"].Value = "UL/NEC TYPE DESIGNATION";

ws4.Cells["M4"].AddComment("Enter the UL / NEC type designation from the cable/ conductor manufacturer's listing specifications.The UL / NEC Type identifies cable / conductor physical construction characteristics(e.g., single - conductor insulated wire, multi - conductor cable, presence o", " ");

ws4.Cells["N4"].Value = "INSULATION RATING (V)";

ws4.Cells["N4"].AddComment("This is the voltage rating of the conductor insulation(expressed in volts).Minimum insulation voltage ratings required per NEC circuit classification in compliance with NFPA 70, Article 725:Class 1 = 600V or more Class 2 = 150V or more", " ");

ws4.Cells["O4"].Value = "DIAMETER (IN)";

ws4.Cells["O4"].AddComment("The overall, nominal, outside diameter of the conductor or cable (expressed in inches).", " ");

ws4.Cells["P4"].Value = "BENDING RADIUS (IN)";

ws4.Cells["P4"].AddComment("This is the minimum bending radius of the cable/conductor (expressed in inches) per the cable manufacturer's specifications.", " ");

ws4.Cells["Q4"].Value = "TOTAL LENGTH (FT)";

ws4.Cells["Q4"].AddComment("This is the total length of the cable/conductor (expressed in feet) as furnished by the equipment supplier.", " ");

ws4.Cells["R3"].Value = "MAKE-UP LENGTH";

ws4.Cells["R3"].AddComment("The length of cable/conductor (expressed in feet) that is routed inside the equipment component from a penetration point on the outer surface of the component to the landing point location of the circuit.", " ");

ws4.Cells["R4"].Value = "SOURCE END (FT)";

ws4.Cells["S4"].Value = "DEST./LOAD END (FT)";

ws4.Cells["T4"].Value = "CAN CUSTOMER CUT CABLE/CONDUCTOR?";

ws4.Cells["T4"].AddComment("Indicate whether or not the cable/conductor can be cut to length in the field, after cable installation at the customer's facility.", " ");

ws4.Cells["U3"].Value = "CONNECTORS";

ws4.Cells["U4"].Value = "SOURCE CONNECTOR END CROSS-SECTION (IN)";

ws4.Cells["U4"].AddComment("Cross - sectional dimensions of the connector(expressed in inches) as viewed parallel to the cable axis.For circular connectors, provide a single dimension representing the connector overall, outside diameter.For square / rectangular connectors, provide ", " ");

ws4.Cells["V4"].Value = "DEST./LOAD CONNECTOR END CROSS-SECTION (IN)";

ws4.Cells["V4"].AddComment("Cross - sectional dimensions of the connector(expressed in inches) as viewed parallel to the cable axis.For circular connectors, provide a single dimension representing the connector overall, outside diameter.For square / rectangular connectors, provide ", " ");

ws4.Cells["W4"].Value = "IS CONNECTOR FIELD INSTALLED?";

ws4.Cells["W4"].AddComment("Indicate whether the connector is installed in the field, after cable installation at the customer's facility.", " ");

ws4.Cells["X4"].Value = "SUPPLIER BUNDLE IDENTIFIER";

ws4.Cells["X4"].AddComment("The unique identifier associated with a group of cables / conductors intended to be field - installed as a bundled unit.Examples: Multiple cables bundled in spiral wrap, zipper tube, split loom tube, ENT, etc.Note: Bundles should not contain mixed -class c", " ");

ws4.Cells["Y4"].Value = "BUNDLE DIAMETER (IN)";

ws4.Cells["Y4"].AddComment("This is the overall, outside diameter of the cable bundle, including any covering used (such as spiral wrap, zipper tube, split loom, etc.).", " ");

ws4.Cells["Z3"].Value = "COMMENTS";

for (int rowCount = 4; rowCount <= electricaldt.Rows.Count + 4; rowCount++)

{

for (int columnCount = 1; columnCount <= electricaldt.Columns.Count; columnCount++)

{

if (ws4.Cells[rowCount, columnCount] == null || ws4.Cells[rowCount, columnCount].Value == null || ws4.Cells[rowCount, columnCount].Value.ToString() == "")

ws4.Cells[rowCount, columnCount].Value = "-";

ws4.Cells[rowCount, columnCount].Style.HorizontalAlignment = ExcelHorizontalAlignment.Center;

ws4.Cells[rowCount, columnCount].Style.VerticalAlignment = ExcelVerticalAlignment.Center;

}

}

for (int rowCount = 1; rowCount <= electricaldt.Rows.Count + 4; rowCount++)

{

for (int columnCount = 1; columnCount <= electricaldt.Columns.Count; columnCount++)

{

ws4.Cells[rowCount, columnCount].Style.Border.Top.Style = ExcelBorderStyle.Thin;

ws4.Cells[rowCount, columnCount].Style.Border.Right.Style = ExcelBorderStyle.Thin;

ws4.Cells[rowCount, columnCount].Style.Border.Bottom.Style = ExcelBorderStyle.Thin;

ws4.Cells[rowCount, columnCount].Style.Border.Left.Style = ExcelBorderStyle.Thin;

}

}

Byte[] bin = p.GetAsByteArray();

HttpContext.Current.Response.Clear();

HttpContext.Current.Response.AddHeader("content-disposition", "attachment; filename=Electrical.xlsx");

HttpContext.Current.Response.ContentType = "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet";

HttpContext.Current.Response.BinaryWrite(bin);

}

HttpContext.Current.Response.End();

}

**3. Import Excel with Validations**

/// <summary>

/// method for importing excel to Supplier Provided InterCables Master Table

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

protected void btnMasterImport\_Click(object sender, EventArgs e)

{

try

{

if (ValidateFileUpload())

{

byte[] byteVal = uploadExcel.FileBytes;

Stream StreamValue = new MemoryStream(byteVal);

DataTable dtMasterData = new DataTable();

DataTable dtMasterHeaders = new DataTable();

DataTable dtErrors = new DataTable();

if (StreamValue != null)

{

ExcelPackage package = new ExcelPackage(StreamValue);

ExcelWorksheet worksheetMasterData = package.Workbook.Worksheets[package.Workbook.Worksheets[1].Name];

//Convert Excel Sheet values into Data Table.

dtMasterHeaders = WorksheetToDataTableWithHeaders(worksheetMasterData, Constants.ColumnCount\_FinalFile);

dtMasterData = WorksheetToDataTable(worksheetMasterData, Constants.ColumnCount\_FinalFile);

if (dtMasterData.Rows.Count > 0)

{

//validate the Data

dtErrors = ValidateExcelData(dtMasterData, dtMasterHeaders);

if (dtErrors.Rows.Count > 0)

{

//Display the Errors

GridViewErrors.Visible = true;

GridViewErrors.DataSource = dtErrors;

GridViewErrors.DataBind();

UploadStatusLabel.Text = "Import failed due to errors shown below. Rectify these and upload again.";

}

else

{

UploadStatusLabel.Text = string.Empty;

GridViewErrors.Visible = false;

adminBAL.ImportExcelSupplierProvidedInterCablesMaster(dtMasterData, hdnCurrentUser.Value);

}

}

}

}

}

catch (Exception ex)

{

if (ex.Message != "Thread was being aborted.")

{

ClientContext clientContext = amatContext.GetClientContext();

ErrorLogUtility.LogErrorException(clientContext, ex.StackTrace, ex.Message, ex.Source, "btnMasterImport\_Click-ElectricalAdmin");

}

}

}

protected bool ValidateFileUpload()

{

bool bStatus = true;

string strFileName = String.Empty;

string strFileExt = String.Empty;

if (!(uploadExcel.HasFile))

{

Page.ClientScript.RegisterStartupScript(this.GetType(), "uploadFile", "uploadFile()", true);

bStatus = false;

}

else

{

strFileName = uploadExcel.FileName;

// strFileExt = strFileName.Split('.')[1];

strFileExt = Path.GetExtension(strFileName);

if (!(strFileExt == ".xls" || strFileExt == ".xlsx"))

{

//Validate file extension

Page.ClientScript.RegisterStartupScript(this.GetType(), "uploadValidFile", "uploadValidFile()", true);

bStatus = false;

}

else

{

bStatus = true;

}

}

return bStatus;

}

protected DataTable ValidateExcelData(DataTable dtMasterData, DataTable dtMasterHeaders)

{

DataTable dtError = new DataTable();

StringBuilder strError = new StringBuilder();

//int lastcolumn = 0;

//Creating structure for Errors Table

DataColumn colErr1 = new DataColumn("Part\_Number", typeof(string));

DataColumn colErr2 = new DataColumn("Error", typeof(string));

dtError.Columns.Add(colErr1);

dtError.Columns.Add(colErr2);

//Getting array of headers to display name or error column in Error Table

var strHeaderArray = dtMasterHeaders.Rows[0].ItemArray.Select(x => x.ToString()).ToArray();

//validation begins

for (int i = 0; i < dtMasterData.Rows.Count; i++)

{

strError.Clear();

var partNumber = dtMasterData.Rows[i][0];

var cell = dtMasterData.Rows[i][0];

if (string.IsNullOrEmpty(Convert.ToString(cell)))

{

strError.Append("Column \"" + strHeaderArray[0] + "\" must have value and cannot be blank.<br/>");

}

else

{

if (!(Regex.IsMatch(Convert.ToString(cell), @"^[0-9]{4}-[0-9]{5}$")))

{

strError.Append("Column \"" + strHeaderArray[0] + "\" is not valid.<br/>");

}

}

for (int j = 5; j < 7; j++)

{

string numericCell = dtMasterData.Rows[i][j].ToString();

if (!string.IsNullOrEmpty(numericCell))

{

// numericCell = numericCell.Substring(0, numericCell.LastIndexOf("."));

numericCell = numericCell.Replace(@".00", "");

numericCell = numericCell.Replace(@",", "");

}

if (!(Regex.IsMatch(Convert.ToString(numericCell), @"^[0-9]\*$")))

{

strError.Append("Column \"" + strHeaderArray[j] + "\" must be in numeric format only.<br/>");

}

}

for (int j = 8; j < 9; j++)

{

string numericCell = dtMasterData.Rows[i][j].ToString();

if (!string.IsNullOrEmpty(numericCell))

{

//numericCell = numericCell.Substring(0, numericCell.LastIndexOf("."));

numericCell = numericCell.Replace(@".00", "");

numericCell = numericCell.Replace(@",", "");

}

if (!(Regex.IsMatch(Convert.ToString(numericCell), @"^[0-9]\*$")))

{

strError.Append("Column \"" + strHeaderArray[j] + "\" must be in numeric format only.<br/>");

}

}

for (int j = 9; j < 12; j++)

{

var numericCell = dtMasterData.Rows[i][j];

if (!(Regex.IsMatch(Convert.ToString(numericCell), @"^[0-9]\*(?:\.[0-9]\*)?$")))

{

strError.Append("Column \"" + strHeaderArray[j] + "\" must be in numeric format only.<br/>");

}

}

if (strError.Length > 0)

{

DataRow rowErr = dtError.NewRow();

rowErr["Part\_Number"] = Convert.ToString(partNumber);

rowErr["Error"] = Convert.ToString(strError);

dtError.Rows.Add(rowErr);

}

}

return dtError;

}

private DataTable WorksheetToDataTableWithHeaders(ExcelWorksheet ws, int columnCount, bool hasHeader = true)

{

DataTable dtSheettoDatableHeaders = new DataTable(ws.Name);

int totalCols = columnCount;

ExcelRange wsRow;

DataRow dr;

int startRow = 1;

foreach (var firstRowCell in ws.Cells[1, 1, 1, totalCols])

{

dtSheettoDatableHeaders.Columns.Add(hasHeader ? firstRowCell.Text : string.Format("Column {0}", firstRowCell.Start.Column));

}

for (int rowNum = startRow; rowNum < 2; rowNum++)

{

wsRow = ws.Cells[rowNum, 1, rowNum, totalCols];

dr = dtSheettoDatableHeaders.NewRow();

foreach (var cell in wsRow)

{

dr[cell.Start.Column - 1] = cell.Text;

}

dtSheettoDatableHeaders.Rows.Add(dr);

}

return dtSheettoDatableHeaders;

}

private DataTable WorksheetToDataTable(ExcelWorksheet ws, int columnCount, bool hasHeader = true)

{

DataTable dtSheettoDatableValue = new DataTable(ws.Name);

int totalCols = columnCount;

int totalRows = ws.Dimension.End.Row;

int startRow = hasHeader ? 2 : 1;

ExcelRange wsRow;

DataRow dr;

foreach (var firstRowCell in ws.Cells[1, 1, 1, totalCols])

{

dtSheettoDatableValue.Columns.Add(hasHeader ? firstRowCell.Text : string.Format("Column {0}", firstRowCell.Start.Column));

}

for (int rowNum = startRow; rowNum <= totalRows; rowNum++)

{

wsRow = ws.Cells[rowNum, 1, rowNum, totalCols];

dr = dtSheettoDatableValue.NewRow();

foreach (var cell in wsRow)

{

dr[cell.Start.Column - 1] = cell.Text;

}

dtSheettoDatableValue.Rows.Add(dr);

}

return dtSheettoDatableValue;

}