

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.PlottingServices;
using Autodesk.AutoCAD.Interop;
using System;
using Ini;
//using System.Windows.Forms;

namespace QuickP
{
    class clsPlotting
    {
        public void SimplePlot(clsQuickP.plot_info pltfle)
        {
            bool plttofile = true;

            // Turn off all background plotting to make it faster
            Application.SetSystemVariable("BackGroundPlot", 0);

            IniFile iniset = new IniFile(Properties.Settings.Default.inifile);

            Document doc = Application.DocumentManager.MdiActiveDocument;
            //Editor ed = doc.Editor;
            Database db = doc.Database;

            string outfile = "";

            using (doc.LockDocument())
            {
                Transaction tr = db.TransactionManager.StartTransaction();
                using (tr)
                {
                    // We'll be plotting the current layout
                    BlockTableRecord btr = (BlockTableRecord)tr.GetObject(db.CurrentSpaceId, OpenMode.
ForRead);
                    Layout lo = (Layout)tr.GetObject(btr.LayoutId, OpenMode.ForRead);

                    // We need a PlotInfo object
                    // linked to the layout
                    PlotInfo pi = new PlotInfo();
                    pi.Layout = btr.LayoutId;

                    // We need a PlotSettings object
                    // based on the layout settings
                    // which we then customize
                    PlotSettings ps = new PlotSettings(lo.ModelType);
                    ps.CopyFrom(lo);

                    ps.ShadePlot = PlotSettingsShadePlotType.AsDisplayed;

                    // The PlotSettingsValidator helps
                    // create a valid PlotSettings object
                    PlotSettingsValidator psv = PlotSettingsValidator.Current;

                    // We'll plot the extents
                    psv.RefreshLists(ps);
                    psv.SetPlotType(ps, Autodesk.AutoCAD.DatabaseServices.PlotType.Extents);

                    //Halfsize is used to determine how much to multiply the dimscale
                    //Extents = 0
                    //Scale = 1
                    //Half-Scale = 2
                    //Third-Scale = 3
                }
            }
        }
    }
}
```

```
//Quarter-Scale = 4
//Extents is just ScaleToFit
if (pltfle.HalfSize == 0)
{
    psv.SetUseStandardScale(ps, true);
    psv.SetStdScaleType(ps, StdScaleType.ScaleToFit);
}
else
{
    CustomScale cps = new CustomScale((double)1,(double)((Convert.ToInt32(Application.
GetSystemVariable("DIMSCALE")) * (int)pltfle.HalfSize)));
    psv.SetUseStandardScale(ps, false);
    psv.SetCustomPrintScale(ps, cps);
    if ((Int16)Application.GetSystemVariable("LUNITS") == 2)
        psv.SetPlotPaperUnits(ps, PlotPaperUnit.Millimeters);
}

if (pltfle.Portrait)
    psv.SetPlotRotation(ps, PlotRotation.Degrees000);
else
    //We want the big border area to be on the left side of the paper
    //when printing landscape
    psv.SetPlotRotation(ps, PlotRotation.Degrees090);

psv.SetPlotCentered(ps, pltfle.CenterPlot);
if (!pltfle.CenterPlot)
{
    Point2d p2d = new Point2d(0, 0);
    psv.SetPlotOrigin(ps, p2d);
}

try
{
    psv.SetCurrentStyleSheet(ps, pltfle.CTBFfile);
}
catch
{
    psv.SetCurrentStyleSheet(ps, "");
}

switch (pltfle.printer)
{
    case "PLOTFILE":
        //Stuff for printing to plotfile goes here
        psv.SetPlotCentered(ps, true);
        psv.SetPlotConfigurationName(ps, "PLOTFILE.pc3", null);
        psv.SetCanonicalMediaName(ps, iniset.IniReadValue(pltfle.PaperSize, "PLOTFILE", 
"NONE"));
        outfile = pltfle.directory + pltfle.drawing + ".plt";
        break;
    case "PDF":
        //Stuff for printing to PDF goes here
        psv.SetPlotCentered(ps, true);
        psv.SetPlotConfigurationName(ps, "PDF.pc3", null);
        psv.SetCanonicalMediaName(ps, iniset.IniReadValue(pltfle.PaperSize, "PDF", 
"NONE"));
        string[] dwg = pltfle.drawing.Split('.');
        outfile = pltfle.directory + dwg[0] + ".pdf";
        break;
    default:
        //If we are printing to an actual printer
        psv.SetPlotConfigurationName(ps, iniset.IniReadValue("Printer", pltfle.printer, 
"NONE"), null);
        //psv.RefreshLists(ps);
        psv.SetCanonicalMediaName(ps, iniset.IniReadValue(pltfle.PaperSize, pltfle.
printer, "NONE"));
}
```

```

        Application.SetSystemVariable("PLOTROTMODE", 1);
        outfile = null;
        plttofile = false;
        break;
    }

    try
    {
        // We need to link the PlotInfo to the
        // PlotSettings and then validate it
        pi.OverrideSettings = ps;
        PlotInfoValidator piv = new PlotInfoValidator();
        piv.MediaMatchingPolicy = MatchingPolicy.MatchEnabled;
        piv.Validate(pi);
        //piv.Dispose();
    }
    catch
    {
        System.Windows.Forms.MessageBox.Show("Caught an error");
    }

    // A PlotEngine does the actual plotting
    // (can also create one for Preview)

    while (PlotFactory.ProcessPlotState != ProcessPlotState.NotPlotting)
        ;

    if (PlotFactory.ProcessPlotState == ProcessPlotState.NotPlotting)
    {

#ifndef DEBUG
        PlotEngine pe = PlotFactory.CreatePreviewEngine((int)PreviewEngineFlags.Plot);      ↵
#else
        PlotEngine pe = PlotFactory.CreatePublishEngine();
#endif
        try
        {
            using (pe)
            {
                // Create a Progress Dialog to provide info
                // and allow the user to cancel
                PlotProgressDialog ppd = new PlotProgressDialog(false, 1, true);
                using (ppd)
                {
                    ppd.set_PlotMsgString(PlotMessageIndex.DialogTitle, "Custom Plot      ↵
Progress");
                    ppd.set_PlotMsgString(PlotMessageIndex.CancelJobButtonMessage, "Cancel      ↵
Job");
                    ppd.set_PlotMsgString(PlotMessageIndex.CancelSheetButtonMessage,      ↵
"Cancel Sheet");
                    ppd.set_PlotMsgString(PlotMessageIndex.SheetSetProgressCaption, "Sheet      ↵
Set Progress");
                    ppd.set_PlotMsgString(PlotMessageIndex.SheetProgressCaption, "Sheet      ↵
Progress");

                    ppd.LowerPlotProgressRange = 0;
                    ppd.UpperPlotProgressRange = 100;
                    ppd.PlotProgressPos = 0;

                    // Let's start the plot, at last
                    ppd.OnBeginPlot();
                    ppd.Visible = true;
                    //ppd.Visible = false;
                    pe.BeginPlot(ppd, null);

                    // We'll be plotting a single document
                    //if ((pltfle.printer != "PLOTFILE") && (pltfle.printer != "PDF"))
                }
            }
        }
    }
}

```

