```
using System;
using System.IO;
using System.Data;
using System.Text;
using System.Drawing.Imaging;
using System.Drawing;
using System.Drawing.Printing;
using System.Collections.Generic;
using System.Windows.Forms;
using Microsoft.Reporting.WinForms;

using Sorveglianza.MyException;

/* PrintReport - v.1.0.1 - 12/10/2011
 * Alex Della Marra
 * Classe creata sulla base di modifiche al codice dell'articolo
 * http://msdn.microsoft.com/en-us/library/ms252091.aspx
 * Rilasciata sotto licenza CC-BY-SA : http://it.wikipedia.org/wiki/Licenze_Creative_Commons
 */
namespace Sorveglianza.Service
{
    public class PrintReport : IDisposable
    {
        private int m_currentPageIndex;
        private IList<Stream> m_streams;
        LocalReport i_report;

        public PrintReport(LocalReport report)
        {
            i_report = report;
        }

        // Routine to provide to the report renderer, in order to
        //    save an image for each page of the report.
        private Stream createStream(string name, string fileNameExtension, Encoding encoding, string mimeType, bool willSeek)
        {
            Stream stream = new MemoryStream();
            m_streams.Add(stream);
            return stream;
        }
```

```
// Export the given report as an EMF (Enhanced Metafile) file.
private void export(LocalReport report)
{
    ReportPageSettings rps = report.GetDefaultPageSettings();
    string deviceInfo = @"<DeviceInfo>";
    deviceInfo += @"<OutputFormat>EMF</OutputFormat>";
    deviceInfo += @"<PageWidth>" + ((decimal)rps.PaperSize.Width / 100) + "in</PageWidth>";
    deviceInfo += @"<PageHeight>" + ((decimal)rps.PaperSize.Height / 100) + "in</PageHeight>";
    deviceInfo += @"<MarginTop>" + ((decimal)rps.Margins.Top / 100) + "in</MarginTop>";
    deviceInfo += @"<MarginLeft>" + ((decimal)rps.Margins.Left / 100) + "in</MarginLeft>";
    deviceInfo += @"<MarginRight>" + ((decimal)rps.Margins.Right / 100) + "in</MarginRight>";
    deviceInfo += @"<MarginBottom>" + ((decimal)rps.Margins.Bottom / 100) + "in</MarginBottom>";
    deviceInfo += @"</DeviceInfo>";
    deviceInfo = deviceInfo.Replace(",", ".");

    Warning[] warnings;
    m_streams = new List<Stream>();
    report.Render("Image", deviceInfo, createStream, out warnings);
    foreach (Stream stream in m_streams)
        stream.Position = 0;
}
// Handler for PrintPageEvents
private void printPage(object sender, PrintPageEventArgs ev)
{
    Metafile pageImage = new Metafile(m_streams[m_currentPageIndex]);

    // Adjust rectangular area with printer margins.
    Rectangle adjustedRect = new Rectangle(
        ev.PageBounds.Left - (int)ev.PageSettings.HardMarginX,
        ev.PageBounds.Top - (int)ev.PageSettings.HardMarginY,
        ev.PageBounds.Width,
        ev.PageBounds.Height);

    // Draw a white background for the report
    ev.Graphics.FillRectangle(Brushes.White, adjustedRect);

    // Draw the report content
    ev.Graphics.DrawImage(pageImage, adjustedRect);

    // Prepare for the next page. Make sure we haven't hit the end.
    m_currentPageIndex++;
    ev.HasMorePages = (m_currentPageIndex < m_streams.Count);
}
```

```
        private void printReport()
        {
            if (m_streams == null || m_streams.Count == 0)
                throw new Exception("No stream in print");
            PrintDocument printDoc = new PrintDocument();
            if (!printDoc.PrinterSettings.IsValid)
            {
                throw new Exception("Impossibile trovare la stampante predefinita");
            }
            else
            {
                printDoc.PrintPage += new PrintPageEventHandler(printPage);
                m_currentPageIndex = 0;
                printDoc.Print();
            }
        }
        // Create a local report for rdlc, load the data,
        // export the report to a file, and print it.
        public void stampa(int numCopie)
        {
            for (int i = 0; i < numCopie; i++)
            {
                export(i_report);
                printReport();
            }
        }

        public void Dispose()
        {
            if (m_streams != null)
            {
                foreach (Stream stream in m_streams)
                    stream.Close();
                m_streams = null;
            }
        }
    }
}
```