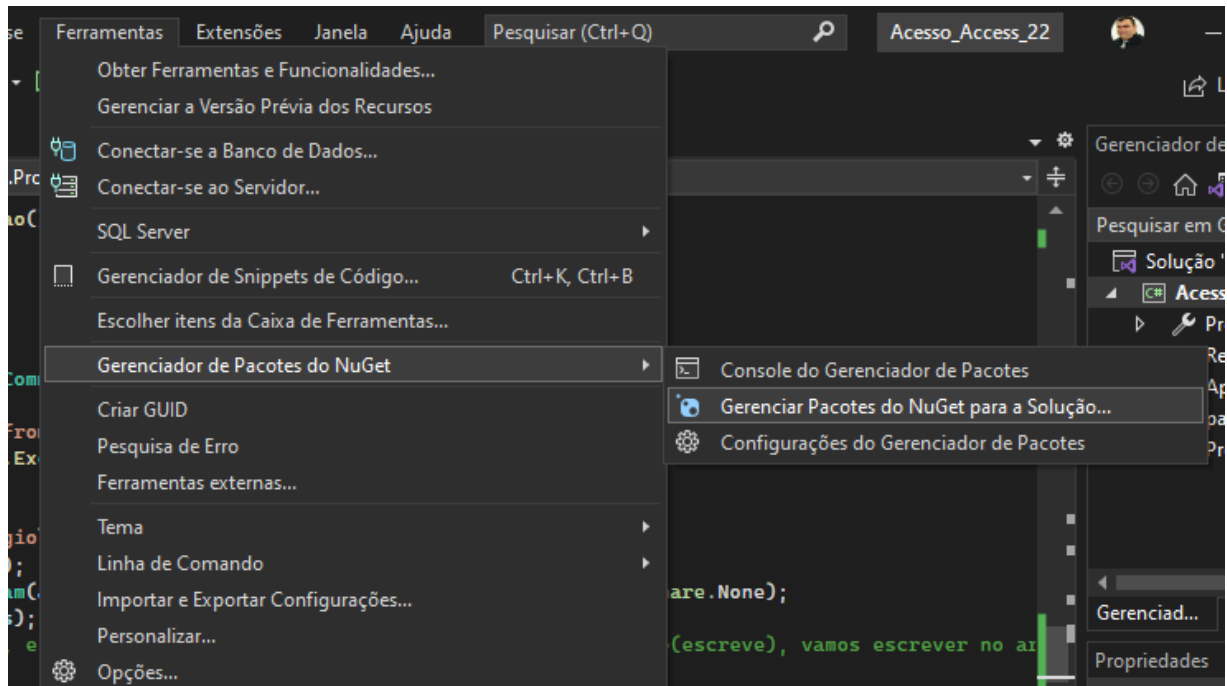


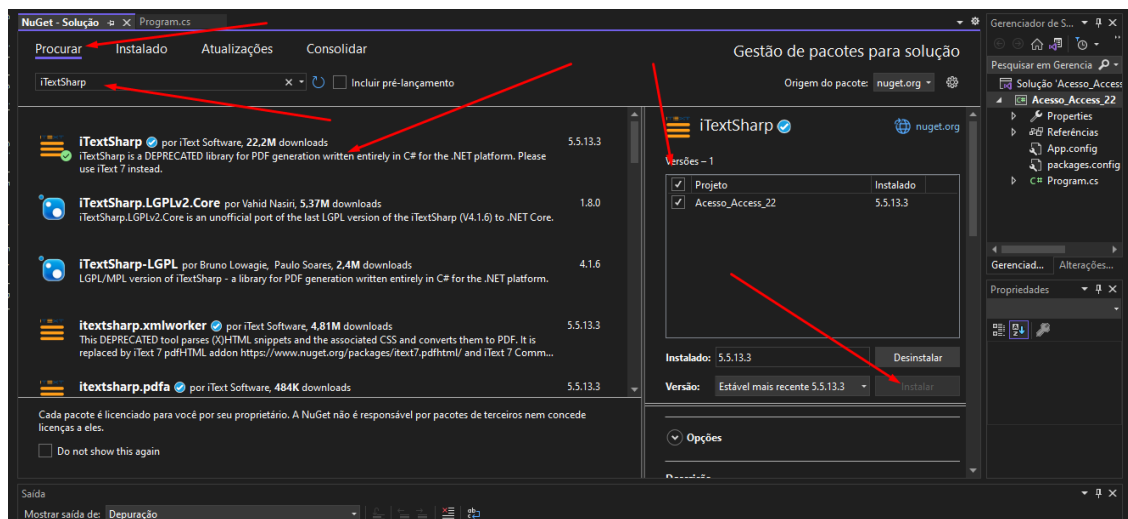
ACESSO AO BANCO DE DADOS – PARTE 3 – RELATÓRIOS

Relatórios em PDF com o iTextSharp

1 - Clique em Ferramentas, selecione Gerenciador de Pacotes do NuGet e Gerenciar Pacotes do NuGet para a Solução...



2 - Clique em Procurar, Digite iTextSharp, selecione o iTextSharp 5.5.13.3, clique em Projeto e clique em instalar



1 - Vamos colocar a chamada das funções da DLL lá na seção using, no início. E para a criação de um arquivo em PDF, também precisaremos da DLL de escrita e leitura de arquivos. Então adicione

```
using iTextSharp.text;
```

```
using iTextSharp.text.pdf;
```

```
using System.IO;
```

```
using System.Diagnostics;
```

2 - Vamos criar um método para imprimir todo o conteúdo do banco na classe AcessoAoAccess

```
public static void Gera_PDFs()
{
    OleDbConnection conexao = AbrirConexao();
    Cliente cliente = new Cliente();

    try
    {
        conexao.Open();
        OleDbCommand comando = new OleDbCommand();
        comando.Connection = conexao;
        comando.CommandText = "Select * from Clientes";
        OleDbDataReader reader = comando.ExecuteReader();
        if (reader.HasRows)
        {
            string arquivoPDF = "d:\\Sergio\\relatorio.pdf";
            Document doc = new Document();
            FileStream fs = new FileStream(arquivoPDF, FileMode.Create,
            FileAccess.Write, FileShare.None);
            PdfWriter.GetInstance(doc, fs); //cria o arquivo
            doc.Open(); //abre o arquivo, e como o FileMode é Creator (criar) e
            o acesso é Write(escreve), vamos escrever no arquivo
            while (reader.Read())
            {
                doc.Add(new Paragraph("Id: " + reader["id"].ToString()));
                doc.Add(new Paragraph("Nome: " + reader["nome"].ToString()));
                doc.Add(new Paragraph("E-mail: " +
            reader["email"].ToString()));
                doc.Add(new Paragraph("-----"));
            }
            doc.Close();
            conexao.Close();
            Console.WriteLine("Arquivo gerado com sucesso em " + arquivoPDF);
            Process.Start(arquivoPDF);
        }
        else
        {
            Console.WriteLine("Nenhum registro retornado!");
        }

        conexao.Close();
    }
}
```

```

    }
    catch (OleDbException err)
    {
        Console.WriteLine(err.Message);
    }
}

```

Apêndice - Comandos do iTextSharp - retirado da documentação original:

Working with Page Size of PDF Document:

Creating a Page of specified size, we must have to create a **iTextSharp.text.Rectangle** object and Passing the size as argument to its constructor. There are a few way to define Page Size:

- First Way to define Page Size:

Creating Page Size by Pixels or Inch. NOTE: In iTextSharp library, unit is 'point'. 72 points = 1 inch. Suppose we want to create a PDF File of width = 2 inch & height = 10 inch, then we need 144pt for 2 inch & 72pt for 10 inch. Lets see, how to do this:

[Hide](#) [Copy Code](#)

```
Rectangle rec = new Rectangle(144, 720);
```

- Second Way to define Page Size:

Taking Page Size from in-built **iTextSharp.text.PageSize** class:

[Hide](#) [Copy Code](#)

```
Rectangle rec2 = new Rectangle(PageSize.A4);
```

The following are the Supported in-built Page Size. Read the full [Documentation of Page Size](#):

◦ **_11X17** ◦ **A0** ◦ **A1** ◦ **A10** ◦ **A2** ◦ **A3** ◦ **A4** ◦ **A4_LANDSCAPE** ◦ **A5** ◦ **A6** ◦ **A7** ◦ **A8** ◦ **A9** ◦ **ARCH_A** ◦ **ARCH_B** ◦ **ARCH_C** ◦ **ARCH_D** ◦ **ARCH_E** ◦ **B0** ◦ **B1** ◦ **B10** ◦ **B2** ◦ **B3** ◦ **B4** ◦ **B5** ◦ **B6** ◦ **B7** ◦ **B8** ◦ **B9**
 ◦ **CROWN_OCTAVO** ◦ **CROWN_QUARTO** ◦ **DEMY_OCTAVO** ◦ **DEMY_QUARTO**
 ◦ **EXECUTIVE** ◦ **FLSA** ◦ **FLSE** ◦ **HALFLETTER** ◦ **ID_1** ◦ **ID_2** ◦ **ID_3** ◦ **LARGE_CROWN_OCTAVO** ◦ **LARGE_CROWN_QUARTO** ◦ **LEDGER**
 ◦ **LEGAL** ◦ **LEGAL_LANDSCAPE** ◦ **LETTER** ◦ **LETTER_LANDSCAPE** ◦ **NOTE** ◦ **PENGUIN_LARGE_PAPERBACK** ◦ **PENGUIN_SMALL_PAPERBACK** ◦ **POSTCARD** ◦ **ROYAL_OCTAVO** ◦ **ROYAL_QUARTO** ◦ **SMALL_PAPERBACK** ◦ **TABLOID**

- Third Way to define Page Size:

Rotating Document i.e. height becomes width & vice-versa:

[Hide](#) [Copy Code](#)

```
Rectangle rec3 = new Rectangle(PageSize.A4.Rotate());
```

Now, just pass this **iTextSharp.text.Rectangle** object (any one) i.e. either 'rec', or 'rec2' or 'rec3' to the **iTextSharp.text.Document**'s constructor during object creation like below:

```
Document doc = new Document(rec);
```

Setting Background Color of PDF Document:

There are a few ways to set background color:

- First Way to Set Background Color:

It takes the object of **iTextSharp.text.BaseColor**. BaseColor constructor takes in-built System.Drawing.Color object Or you can pass RGB values to the constructor in different forms:

Hide Copy Code

```
rec.BackgroundColor = new BaseColor(System.Drawing.Color.WhiteSmoke);
```

- Second Way to Set Background Color:

It takes the object of **iTextSharp.text.pdf.CMYKColor**. CMYKColor constructor takes only CMYK values in different forms:

Hide Copy Code

```
rec2.BackgroundColor = new CMYKColor(25, 90, 25, 0);
```

Setting Page Margins of PDF Document:

Margins can be set during **Document** object creation like Page Size Suppose we set the margins as below:

- Left Margin: 0.5 inch
- Right Margin: 1 inch
- Top Margin: 1.5 inch
- Bottom Margin: 2.5 inch

So, we need to do set the following points for the Left, Right, Top, Bottom Margins respectively as we already know that **iTextSharp** library only understand points where 72 points = 1 inch.

- Left Margin: 36pt => 0.5 inch
- Right Margin: 72pt => 1 inch
- Top Margin: 108pt => 1.5 inch
- Bottom Margin: 180pt => 2.5 inch

Lets implement:

Hide Copy Code

```
Document doc = new Document(PageSize.A4, 36, 72, 108, 180);
```

Setting Text Alignment in PDF Document:

Alignment is one of the property of **iTextSharp.text.Paragraph**'s object. **iTextSharp** Library provides various types of Alignments. These Alignments can be access through **iTextSharp.text.Element** class. The following are Alignment types provides iTextSharp:

- `ALIGN_BASELINE[^]`
- `ALIGN_BOTTOM[^]`
- `ALIGN_CENTER[^]`
- `ALIGN_JUSTIFIED[^]`
- `ALIGN_JUSTIFIED_ALL[^]`
- `ALIGN_LEFT[^]`
- `ALIGN_MIDDLE[^]`
- `ALIGN_RIGHT[^]`
- `ALIGN_TOP[^]`
- `ALIGN_UNDEFINED[^]`

As we already see the `iTextSharp.text.Document`'s constructor takes `iTextSharp.text.Paragraph`'s object during Document creation. So, after creating the Paragraph object and setting Alignment property, we can pass this object to `iTextSharp.text.Document`'s constructor during Document ceration. Lets implement:

[Hide](#) [Copy Code](#)

```
Paragraph para = new Paragraph("Hello World Hello World Hello World Hello World Hello World Hello World  
Hello World Hello World Hello World Hello World Hello World");  
// Setting paragraph's text alignment using iTextSharp.text.Element class  
para.Alignment = Element.ALIGN_JUSTIFIED; // Adding this 'para' to the  
Document object doc.Add(para);
```

Setting Meta Information or Properties of a PDF Document:

The following Meta information of PDF Document, you can set through `iTextSharp.text.Document`'s methods by creating its object i.e. `doc` here:

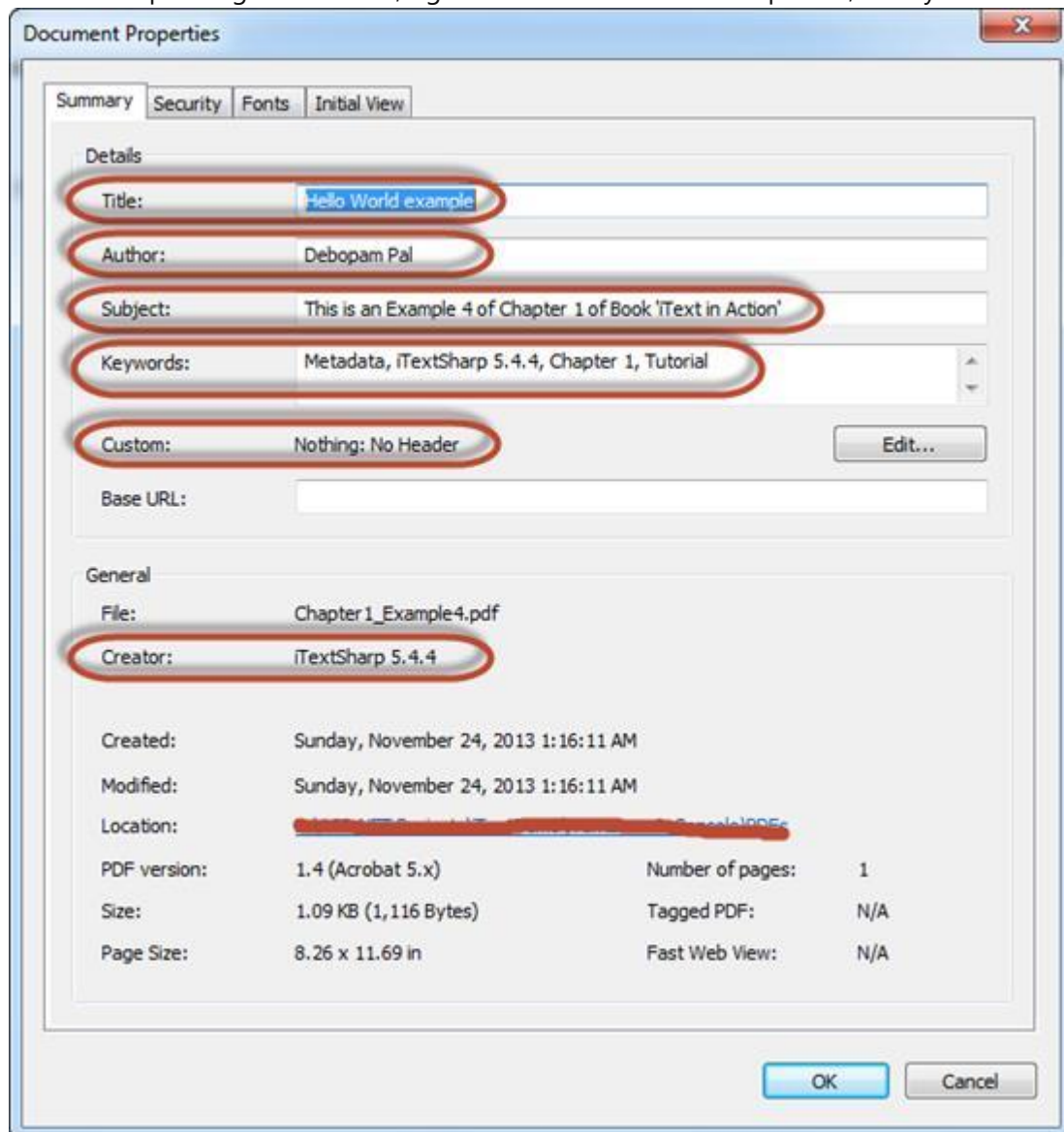
- `Author Name[^]`
- `Creation Date[^]`
- `Creator Name[^]`
- `Header Name & Content[^]`
- `Keywords[^]`
- `Langugae[^]`
- `Producer[^]`
- `Subject[^]`
- `Title[^]`

Lets implement a few of them:

[Hide](#) [Copy Code](#)

```
// Setting Document properties e.g.
// 1. Title
// 2. Subject
// 3. Keywords
// 4. Creator
// 5. Author //
6. Header
doc.AddTitle("Hello World example");
doc.AddSubject("This is an Example 4 of Chapter 1 of Book 'iText in Action'");
doc.AddKeywords("Metadata, iTextSharp 5.4.4, Chapter 1, Tutorial");
doc.AddCreator("iTextSharp 5.4.4"); doc.AddAuthor("Debopam Pal");
doc.AddHeader("Nothing", "No Header");
```

Now after opening this PDF file, right click over it and click Properties, then you'll see the following:



Creating a Multipage Document:

We can create a new page through `iTextSharp.text.Document`'s `NewPage()` method by creating its object. Lets add five pages in PDF Document:

Hide Copy Code

```
for (int i = 1; i <= 5; i++)
{
    doc.NewPage();
    doc.Add(new Paragraph(string.Format("This is a page {0}", i))); } }
```

Creating a New PDF Document from an existing PDF Document:

We can read from a PDF Document using `iTextSharp.text.pdf.PdfReader`'s object and write it to another PDF Document using `iTextSharp.text.pdf.PdfStamper`'s object. Lets implement:

Hide Copy Code

```
string originalFile = "Original.pdf"; string
copyOfOriginal = "Copy.pdf";
using (FileStream fs = new FileStream(originalFile, FileMode.Create, FileAccess.Write, FileShare.None))
using (Document doc = new Document(PageSize.LETTER)) using (PdfWriter writer =
PdfWriter.GetInstance(doc, fs))
{
    doc.Open();
    doc.Add(new Paragraph("Hi! I'm Original"));
doc.Close();
}
PdfReader reader = new PdfReader(originalFile);
using (FileStream fs = new FileStream(copyOfOriginal, FileMode.Create, FileAccess.Write,
FileShare.None))
// Creating iTextSharp.text.pdf.PdfStamper object to write
// Data from iTextSharp.text.pdf.PdfReader object to FileStream object
using (PdfStamper stamper = new PdfStamper(reader, fs)) { }
```

Adding Watermark to PDF Document using Layer:

Watermark can be add after creating the PDF Document in `iTextSharp` Library. Here I'll add Watermark to existing PDF Document i.e. `Original.pdf`, through creating a `iTextSharp.text.pdf.PdfLayer` object. Lets implement:

Hide Shrink ▲ Copy Code

```

string watermarkedFile = "Watermarked.pdf";
// Creating watermark on a separate layer
// Creating iTextSharp.text.pdf.PdfReader object to read the Existing PDF Document PdfReader
reader1 = new PdfReader(originalFile);
using (FileStream fs = new FileStream(watermarkedFile, FileMode.Create, FileAccess.Write,
FileShare.None))
// Creating iTextSharp.text.pdf.PdfStamper object to write Data from iTextSharp.text.pdf.PdfReader
object to FileStream object
using (PdfStamper stamper = new PdfStamper(reader1, fs)) {
    // Getting total number of pages of the Existing Document
    int pageCount = reader1.NumberOfPages;

    // Create New Layer for Watermark
    PdfLayer layer = new PdfLayer("WatermarkLayer", stamper.Writer);
    // Loop through each Page
    for (int i = 1; i <= pageCount; i++)
    {
        // Getting the Page Size
        Rectangle rect = reader1.GetPageSize(i);

        // Get the ContentByte object
        PdfContentByte cb = stamper.GetUnderContent(i);
        // Tell the cb that the next commands should be "bound" to this new Layer
        cb.BeginLayer(layer);
        cb.SetFontAndSize(BaseFont.CreateFont(
            BaseFont.HELVETICA, BaseFont.CP1252, BaseFont.NOT_EMBEDDED), 50);
        PdfGState gState = new PdfGState();
        gState.FillOpacity = 0.25f;          cb.SetGState(gState);

        cb.SetColorFill(BaseColor.BLACK);
        cb.BeginText();

        cb.ShowTextAligned(PdfContentByte.ALIGN_CENTER, watermarkText, rect.Width / 2, rect.Height / 2,
45f);
        cb.EndText();

        // Close the Layer
        cb.EndLayer();
    }
}

```

The created PDF Document is below:

Hi! I'm Original

This is a Test

Removing Watermark from the just created Watermarked Document by Removing Layer:

Whenever we add Layer in PDF Document, then the content of the Layer resides under **OCG** Group. So if I remove this Layer we can remove the content of the Layer also e.g. here it is Watermark Text. To remove all the Layers from PDF Document, you have to remove OCG Group completely from the Document using `reader.Catalog.Remove(PdfName.OCPROPERTIES)`. Now follow the Steps below to remove the Watermark Text from Layer:

- Read the existing watermarked document using `iTextSharp.text.pdf.PdfReader`'s object
- Taking each Page in the `iTextSharp.text.pdf.PdfDictionary`'s object using `GetPageN(int pageNumber)` method of `iTextSharp.text.pdf.PdfReader`'s object.
- Taking the Content of the Page in the `iTextSharp.text.pdf.PdfArray`'s object using `GetAsArray(PdfName.CONTENTS)` method of `iTextSharp.text.pdf.PdfDictionary`'s object
- Loop through this array and Get each element as `iTextSharp.text.pdf.PRStream`'s object using `GetAsStream(int arrayIndex)` method of `iTextSharp.text.pdf.PdfArray`'s object
- Convert each stream into Bytes using Static method `GetStreamBytes(PRStream stream)` of `iTextSharp.text.pdf.PdfReader` class
- Convert these Bytes into String using `System.Text.Encoding.ASCII.GetString(byte[] bytes)` method
- Search for the String `"/OC"` and also the **Watermarked Text**. If found then remove it by giving it zero length and zero data using two methods: `Put()` & `SetData()` of `iTextSharp.text.pdf.PRStream` class

- Write this modified document exists in the **reader** to a new document using **iTextSharp.text.pdf.PdfStamper**'s object Lets Implement it:

Hide Shrink ▲ Copy Code

```
// Removing the layer created above
// 1. First we bind a reader to the watermarked file
// 2. Then strip out a branch of things
// 3. Finally use a simple stamper to write out the edited reader
PdfReader reader2 = new PdfReader(watermarkedFile);

// NOTE: This will destroy all layers in the Document, only use if you don't have any additional layers
// Remove the OCG group completely from the Document: reader2.Catalog.Remove(PdfName.OCPROPERTIES);

// Clean up the reader, optional reader2.RemoveUnusedObjects();

// Placeholder variables
PRStream stream;
string content;
PdfDictionary page;
PdfArray contentArray;

// Get the number of pages
int pageCount2 = reader2.NumberOfPages;

// Loop through each page
for (int i = 1; i <= pageCount2; i++)
{
    // Get the page
    page = reader2.GetPageN(i);

    // Get the raw content
    contentArray = page.GetAsArray(PdfName.CONTENTS);

    if (contentArray != null)
    {
        // Loop through content
        for (int j = 0; j < contentArray.Size; j++)
        {
            stream = (PRStream)contentArray.GetAsStream(j);
            // Convert to a String, NOTE: you might need a different encoding here
            content = System.Text.Encoding.ASCII.GetString(PdfReader.GetStreamBytes(stream));
            //Look for the OCG token in the stream as well as our watermarked text
            if (content.IndexOf("/OC") >= 0 && content.IndexOf(watermarkText) >= 0)
            {
                //Remove it by giving it zero length and zero data
                stream.Put(PdfName.LENGTH, new PdfNumber(0));
                stream.SetData(new byte[0]);
            }
        }
    }
}

// Write the content out
using (FileStream fs = new FileStream(unwatermarkedFile, FileMode.Create, FileAccess.Write,
FileShare.None)) using (PdfStamper stamper = new PdfStamper(reader2, fs)) { }
```

Adding Watermark to each Page during Creation:

Now, we already know that, watermark cannot be add during Page creation, it have to add after document creation. So, I've created a class **PDFWriterEvents** which implements the interface **iTextSharp.text.pdf.IPdfPageEvent** and modify the event **OnStartPage**. This interface contains a set of events from the Opening & to Closing the PDF Document. The events are following:

- `public void OnOpenDocument(PdfWriter writer, Document document)`
- `public void OnCloseDocument(PdfWriter writer, Document document)`
- `public void OnStartPage(PdfWriter writer, Document document)`
- `public void OnEndPage(PdfWriter writer, Document document)`
- `public void OnParagraph(PdfWriter writer, Document document, float paragraphPosition)`
- `public void OnParagraphEnd(PdfWriter writer, Document document, float paragraphPosition)`
- `public void OnChapter(PdfWriter writer, Document document, float paragraphPosition, Paragraph title)`
- `public void OnChapterEnd(PdfWriter writer, Document document, float paragraphPosition)`
- `public void OnSection(PdfWriter writer, Document document, float paragraphPosition, int depth, Paragraph title)`
- `public void OnSectionEnd(PdfWriter writer, Document document, float paragraphPosition)`
- `public void OnGenericTag(PdfWriter writer, Document document, Rectangle rect, String text)`

You may modify other events accroding to your needs which occured against a particular action. Let see the which I've created:

Hide Shrink ▲ Copy Code

```
// Creating Watermark inside OnStartPage Event by implementing IPdfPageEvent interface
// So that, dusring Page Creation, Watermark will be create class
PDFWriterEvents : IPdfPageEvent
{
    string watermarkText;
    float fontSize = 80f;
    float xPosition = 300f;
    float yPosition = 800f;
    float angle = 45f;

    public PDFWriterEvents(string watermarkText, float fontSize = 80f,
float xPosition = 300f, float yPosition = 400f, float angle = 45f)
    {
        this.watermarkText = watermarkText;
    }
}
```

```

        this.xPosition = xPosition;
this.yPosition = yPosition;          this.angle
= angle;
    }
    public void OnOpenDocument(PdfWriter writer, Document document) { }
public void OnCloseDocument(PdfWriter writer, Document document) { }
public void OnStartPage(PdfWriter writer, Document document)
    {
try
    {
        PdfContentByte cb = writer.DirectContentUnder;
        BaseFont baseFont = BaseFont.CreateFont(BaseFont.HELVETICA, BaseFont.WINANSI,
BaseFont.EMBEDDED);          cb.BeginText();
        cb.SetColorFill(BaseColor.LIGHT_GRAY);
cb.SetFontAndSize(baseFont, fontSize);
        cb.ShowTextAligned(PdfContentByte.ALIGN_CENTER, watermarkText, xPosition, yPosition, angle);
cb.EndText();
    }
    catch (DocumentException docEx)
    {
        throw docEx;
    }
}
    public void OnEndPage(PdfWriter writer, Document document) { }
    public void OnParagraph(PdfWriter writer, Document document, float paragraphPosition) { }
public void OnParagraphEnd(PdfWriter writer, Document document, float paragraphPosition) { }
    public void OnChapter(PdfWriter writer, Document document, float paragraphPosition, Paragraph title)
{ }    public void OnChapterEnd(PdfWriter writer, Document document, float paragraphPosition) { }
public void OnSection(PdfWriter writer, Document document, float paragraphPosition, int depth,
Paragraph title) { }
    public void OnSectionEnd(PdfWriter writer, Document document, float paragraphPosition) { }
public void OnGenericTag(PdfWriter writer, Document document, Rectangle rect, String text) { } }

```

Lets see how & when you use the object of this class:

Hide Copy Code

```

using (FileStream fs = new FileStream(
    "Watermark_During_Page_Creation.pdf", FileMode.Create, FileAccess.Write, FileShare.None))
using (Document doc = new Document(PageSize.LETTER)) using (PdfWriter writer =
PdfWriter.GetInstance(doc, fs))
{
    writer.PageEvent = new PDFWriterEvents("This is a Test");
doc.Open();
    doc.Add(new Paragraph("This is a page 1"));
doc.Close();
}

```

See, **OnStartPage** event called during adding a new paragraph. So I don't need to add watermark later 😊

Export/Print/Output the PDF File directly to the Client without saving it to the Disk:

We can create PDF File in memory by creatig **System.IO.MemorySystem**'s object. Lets see:

Hide Copy Code

```

using (MemoryStream ms = new MemoryStream())

```

```
using(Document document = new Document(PageSize.A4, 25, 25, 30, 30)) using(PdfWriter
writer = PdfWriter.GetInstance(document, ms))
{
    document.Open();
    document.Add(new Paragraph("Hello
World"));    document.Close();
writer.Close();    ms.Close();
    Response.ContentType = "pdf/application";
    Response.AddHeader("content-disposition", "attachment;filename=First_PDF_document.pdf");
    Response.OutputStream.Write(ms.GetBuffer(), 0, ms.GetBuffer().Length); }
}
```

Setting Viewer Preferences of a PDF Document:

The values of the different **ViewerPreferences** were originally stored in **iTextSharp.text.pdf.PdfWriter** class as an integer constant. You can set the **ViewerPreferences** by following two ways:

- By setting property **ViewerPreferences** of **iTextSharp.text.pdf.PdfWriter** class. To know all the **ViewerPreferences** and its purpose, please read [this](#) first. E.g.-

[Hide](#) [Copy Code](#)

```
writer.ViewerPreferences = PdfWriter.HideMenubar;
```

- By calling method **AddViewerPreference(PdfName key, PdfObject value)** of **iTextSharp.text.pdf.PdfWriter**'s object. To know which **value** is appropriate for which **key**, read [this](#) first. E.g.-

[Hide](#) [Copy Code](#)

```
writer.AddViewerPreference(PdfName.HIDEMENUBAR, new PdfBoolean(true));
```

Encrypting a PDF Document:

By **SetEncryption()** method of **iTextSharp.text.pdf.PdfWriter**'s object, we can encrypt a PDF document. Read full documentation of this method [here](#). To know all the encryption types, click [here](#). E.g.-

[Hide](#) [Copy Code](#)

```
writer.SetEncryption(PdfWriter.STRENGTH40BITS, null, null, PdfWriter.ALLOW_COPY);
```