Ibex PDF Creator

.NET Programmers Guide

For Ibex version 6.11.1.3

Table of Contents

Chapter 1

Introduction

This manual describes the functionality and use of the Ibex PDF Creator. Ibex is a PDF creation component which can be used from the command line or, more usually, integrated into a larger application. It ships as a .Net assembly so it can be used both in stand-alone applications and in server-based applications using ASP and ASP.Net.

This chapter provides an overview of how Ibex works and the process involved in creating PDF files using Ibex.

1.1 The PDF creation process

Ibex creates PDF files which contain data from your application. You provide Ibex with your data in XML format and Ibex creates a PDF file containing that data. The format of the PDF file is specified using an XSL stylesheet which defines the layout of the document using the elements from the XSL Formatting Objects standard.

The XML you provide to Ibex can come from any source. Typically, it is extracted from a database or generated dynamically for each document.

The formatting objects (FO) standard defines elements such as table, row and cell which can be used to lay out your data on a page. It also defines objects for describing the overall shape and layout of the page, including parameters such as page size, number of columns and regions such as headers and footers where content will be placed on the page.

The process of creating a document in FO format is carried out using an XSLT stylesheet. The stylesheet transforms your XML data into standard FO syntax. Ibex then reads that XSL-FO and creates the PDF file. The actual execution of the XSLT translation can be done either by Ibex, which uses the .Net framework XSL translation objects, or externally to Ibex using any XSLT engine.

Figure 1-1 shows some XML data for creating a page which says "Hello world". The corresponding formatting objects from which the PDF is created are shown in Figure 1-2.

Figure 1-1: Simple XML

```
<?xml version="1.0" encoding="UTF-8"?>
<expression>hello world<expression>
```

Figure 1-2: Example formatting objects for hello world

```
<root xmlns="http://www.w3.org/1999/XSL/Format">
<layout-master-set>
<simple-page-master master-name="layout" page-width="8.5in" page-height="8in">
<region-body region-name="body" margin="2.5cm"/>
</simple-page-master>
</layout-master-set>
<page-sequence master-reference="layout">
<flow flow-name="body">
<block>Hello world</block>
</flow>
</page-sequence>
</root>
```

The process of getting from your data to the formatting objects is carried out using the XSLT stylesheet which you provide.

This approach to PDF creation has a number of advantages:

- the content and presentation are separated. The format of the PDF file is defined by the XSLT stylesheet which can be created and maintained externally to the application. Changing the stylesheet to alter the report layout does not require rebuilding your application;

- formatting elements such as report headers and footers can be maintained in a separate stylesheet which is included at runtime into many different reports;

- the formatting objects standard defines a powerful set of objects for creating page content. It supports single-column and multi-column pages, bookmarks, indexing, page headers and footers, complex page numbering, tables with optionally repeating headers and footers and many other features;

- formatting objects is a high-level approach which makes changing report layouts simple. For example to change the number of columns on a multi-column page you just need to change the column-count attribute on a single element. To do this using a lower level programmatic API is much more complex.

1.2 Terminology

Ibex uses the FO standard which defines formatting objects such as table and table-cell. FO makes a distinction between two types of objects:

block-level objects broadly speaking these are objects which are formatted vertically down the page. Block level objects are fo:block, fo:block-container, fo:table, fo:table-and-caption, and fo:list-block.

inline-level objects these are objects whose content is placed on lines within an fo:block object. Commonly used inline level objects are

fo:external-graphic, fo:inline, fo:leader, fo:page-number, fo:page-number-citation and fo:basic-link.

1.3 About this manual

This manual is split into two main sections. The first covers an introduction to the use of formatting objects and an overview of various formatting objects such as tables and lists. The second is a reference section listing all the available objects and attributes with many examples of their usage.

This manual was produced with the .Net version of Ibex, release 6.11.1.3 .

1.4 About Ibex

Ibex is developed entirely in C# and requires the Microsoft dotnet runtime to be installed. .Net Framework 4.8 and .Net 6.0 - 10.0 are supported. All operating systems which support dotnet are supported by Ibex.

Chapter 2

Installation

2.1 Ibex for .Net 6, .Net 7, .Net 8, .Net 9, .Net 10

The latest version of Ibex works on .Net 6, 7, 8 and 9. This version works on Windows and Linux operating systems.

Ibex is installed as from nuget.org as two separate components, the PDF creation library, and the command line interface.

2.1.1 Ibex.PDF.Creator

The Ibex.PDF.Creator package contains the library which you will link to your own application. This can be downloaded either using the "Manage Nuget Packages" menu option from within Visual Studio or using the "dotnet add package" command on the command line.

2.1.2 Ibex.CommandLine

The Ibex.CommandLine package is a console application to create PDF files from the command line, for use during testing and development.

The Ibex.CommandLine package is a [dotnet tool]. This is installed from a command shell like so:

```
dotnet tool install -g Ibex.CommandLine
```

An existing installation can be updated using this command:

```
dotnet tool update -g Ibex.CommandLine
```

An existing installation can be uninstalled using this command:

```
dotnet tool uninstall --global Ibex.CommandLine
```

The Ibex.CommandLine tool, when it is installed as a dotnet global tool, is accessed using the command "ibex". So for example if we had a file called 'test.fo' this command would created a PDF file from it:

```
ibex test.fo test.pdf
```

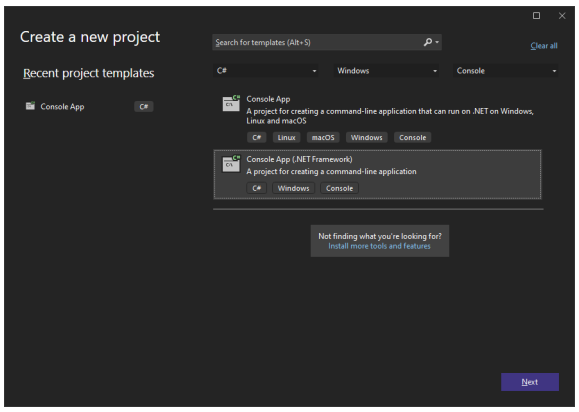
2.2 Installation for .Net Framework 4.8

The latest version of Ibex works on .Net Framework 4.8. This version works on Windows operating systems.

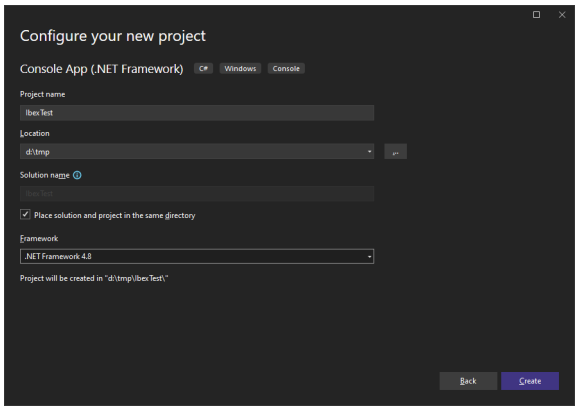
.Net Framework 4.8 does not support the "dotnet tool" command line approach used for .Net 6 and above. Instead here we show how to add the Ibex component to a program using Visual Studio 2022 or 2026.

2.2.1 Creating a Project

Open Visual Studio 2022 or 2026, use the File > New > Project menu to open the project creation dialog and select these options to create a .Net Framework based application:



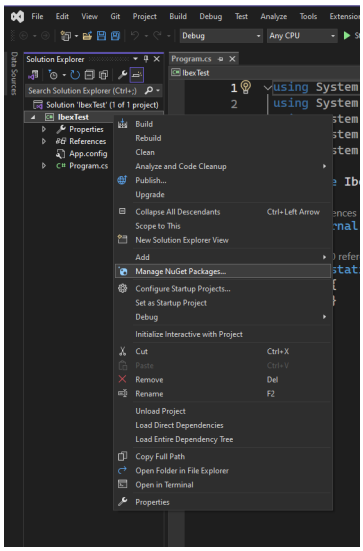
Press "Next" and then name the project and specify its location and make sure to choose ".Net Framework 4.8" as the framework.



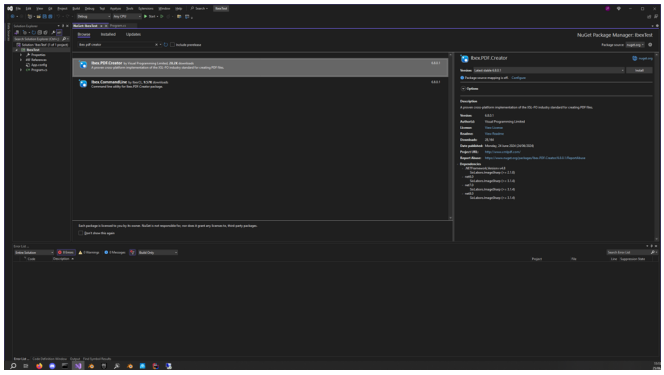
Press "Create" to create the project. You should see this window showing the project files in the Solution Explorer window (which might be on the left or the right side of the screen) and the contents of the file Progam.cs.

2.2.2 Adding the Ibex component

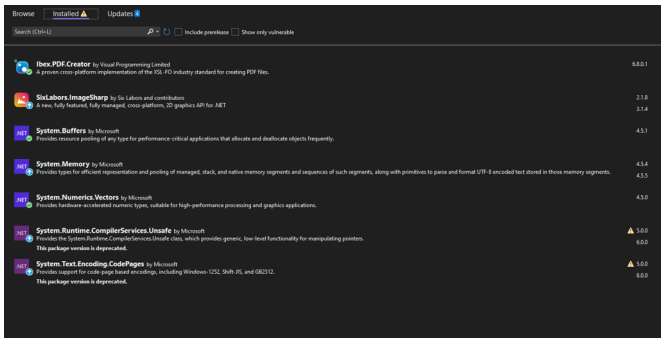
Right-click the project name (in this case "IbexTest") and select "Manage Nuget Packages..."



Click on the "Browse" tab at the top of the window and type "ibex pdf creator" in the search bar and press enter. This should show the Ibex.PDF.Creator package, click this and the click the Install button on the far right:



After a few seconds and possibly some confirm dialog boxes, if you click on the "Installed" tab you should see something like this, showing which components have been added to the project:



If you want you can update some of these packages to the latest versions. **Do not** update the version of SixLabors.ImageSharp to 3.1.4, this version only works with .Net 6 application it will not work with .Net Framework 4.8 applications.

2.2.3 Adding Code

Edit the file Program.cs and replace all its code with this code:

```
using ibex4;
using System.IO;
using System.Text;

public class test
{

    static void Main(string[] args)
    {
        string FO = "<fo:root xmlns:fo=\"http://www.w3.org/1999/XSL/Format\">"
            + "<fo:layout-master-set>"
            + "  <fo:simple-page-master master-name=\"page\" margin=\"36pt\"
page-height=\"11in\" page-width=\"8in\">"
            + "    <fo:region-body margin-top=\"78pt\"/>"
            + "  </fo:simple-page-master>"
            + "</fo:layout-master-set>"
            + "<fo:page-sequence master-reference=\"page\">"
            + "  <fo:flow font-family=\"Arial\" font-size=\"11pt\"
flow-name=\"xsl-region-body\">"
            + "    <fo:block border=\"1pt solid green\">Hello World</fo:block>"
            + "  </fo:flow>"
            + "</fo:page-sequence>"
            + "</fo:root>";


        byte[] byteArray = Encoding.UTF8.GetBytes(FO);
        MemoryStream inputStream = new MemoryStream(byteArray);
        inputStream.Seek(0, SeekOrigin.Begin);

        FODocument gen = new FODocument();
        using (FileStream pdfStream = new FileStream("test.pdf", FileMode.Create,
FileAccess.Write))
        {
            gen.generate(inputStream, pdfStream);
        }
    }
}
```

Compile this program.

2.2.4 Testing

You should now be able to run the program and create the file "test.pdf". Depending on the project setup this will appear in the current directory which might be for example in bin\Debug.

Chapter 3

Getting Started with Ibex

Although primarily intended for use as a part of a larger application, the Ibex installation includes command line programs which creates PDF files from XML, XSLT and XSL-FO files. We will use these programs to demonstrate the basics of PDF creation with Ibex.

Throughout this manual an XML file which uses the XSL formatting objects vocabulary is referred to as an FO file or just the FO.

The command line syntax for all versions is the same. In these examples we use ibex.exe.

3.1 Ibex command line program usage

To create a PDF file from a FO file specify the names of the FO and PDF files on the command line. For example to create hello.pdf from hello.fo you do this:

```
ibex hello.fo hello.pdf
```

If the names of the input and output files are the same (ignoring the extensions) you can abbreviate this to:

```
ibex hello.fo
```

and if the file extension of the input file is "fo" or "xml" you can abbreviate even further to:

```
ibex hello
```

3.2 Error logging

Any informational or error messages will be logged to the console. To send any error messages to a file as well, use the -logfile option. For example, to log errors to the file ibex.log the command becomes:

```
ibex -logfile ibex.log hello.fo hello.pdf
```

3.3 An example without XSLT translation

The Ibex command line program will create a PDF file from either (a) an FO file or (b) an XML file with an XSLT stylesheet. This section shows how to create a PDF file from an FO file.

This example uses the FO file hello.fo shown in Figure 3-1.

Figure 3-1: Hello World FO

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="http://www.w3.org/1999/XSL/Format">

  <layout-master-set>
    <simple-page-master master-name="page">
      <region-body margin="2.5cm" region-name="body"/>
    </simple-page-master>
  </layout-master-set>

  <page-sequence master-reference="page">
    <flow flow-name="body">
     <block>Hello World</block>
    </flow>
  </page-sequence>
</root>
```

Each of the elements and attributes used in the file is explained later in the manual. For now we just want to get started with using the Ibex command line program.

Using the command

```
ibex hello
```

creates the file hello.pdf containing the text "Hello World".

3.4 An example with XSLT translation

The Ibex command line program will create a PDF file from either (a) an FO file or (b) an XML file with an XSLT stylesheet. This section shows how to create a PDF file from an XML data file with an XSLT stylesheet.

Using Ibex without having Ibex do the XSLT transformation to create the FO is useful if you have created the FO using another tool or if you just want to manually change some FO to experiment with layout.

In practice XSLT is almost always part of the PDF creation process because XSL-FO lacks some simple features such as being able to sequentially number headings. The designers of XSL-FO presumed that XSLT would be used and so did not duplicate features already in XSLT.

Ibex gives you the flexibility of having Ibex do the XSLT translation or having some other tool do it. Internally Ibex uses the XSLT translation classes provided by .NET.

In this example we will translate some XML with an XSLT stylesheet and produce a PDF from the result of the translation.

We have some weather forecast data in the file weather.xml. This file contains the XML shown in Figure 3-2.

Figure 3-2: Weather Forecast Data

```
<?xml version="1.0" encoding="UTF-8"?>
<forecast>
   <city name="Wellington" temp="20"/>
</forecast>
```

We also have an XSLT stylesheet weather.xsl which contains the XSL shown in Figure 3-3.

Figure 3-3: Weather Forecast Stylesheet

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:fo="http://www.w3.org/1999/XSL/Format"
   xmlns:ibex="http://www.xmlpdf.com/2003/ibex/Format">

<xsl:strip-space elements="*"/>

<xsl:template match="forecast">

  <root xmlns="http://www.w3.org/1999/XSL/Format">

    <layout-master-set>
      <simple-page-master master-name="page-layout">
         <region-body margin="2.5cm" region-name="body"/>
      </simple-page-master>
    </layout-master-set>

    <page-sequence master-reference="page-layout">
       <flow flow-name="body">
           <xsl:apply-templates select="city"/>
       </flow>
    </page-sequence>

  </root>

</xsl:template>

<xsl:template match="city">
   <fo:block>
      <xsl:value-of select="@name"/>

      <xsl:value-of select="@temp"/>
   </fo:block>
</xsl:template>

</xsl:stylesheet>
```

This template outputs the root, layout-master-set and page-sequence elements. Then for each city record in the data outputs a block element using the template shown in Figure 3-4.

Figure 3-4: weather-data-xsl-subset

```
<xsl:template match="city">
   <block>
      <xsl:value-of select="@name"/>
       
      <xsl:value-of select="@temp"/>
   </block>
</xsl:template>
```

We can translate and format this example using the command:

```
ibex -xsl weather.xsl weather.xml weather.pdf
```

The result of this translation is the file weather.pdf

3.5 Required skills

To use Ibex you need know how to edit XSL stylesheets. Some familiarity with XSLT is required, although in depth knowledge is not. The Ibex website contains examples of using XSLT for common document related functions such as creating a table of contents.

Familiarity with XSL-FO is not required. This manual contains enough information to enable you to produce complex documents using Ibex.

Chapter 4

Introduction to XSL-FO

This chapter provides an overview of formatting objects and provides some suggestions on how to create PDF documents from XML files. We also look at the techniques for using XSLT transformation to create FO files.

4.1 Layout of an FO file

A very simple FO file is shown in Figure 4-1:

Figure 4-1: Simple FO file

```
<?xml version='1.0' encoding='UTF-8'?>
<root xmlns="http://www.w3.org/1999/XSL/Format">

   <layout-master-set>
      <simple-page-master master-name="simple">
         <region-body margin="2.5cm" region-name="body"
            background-color='#eeeeee'/>
      </simple-page-master>
   </layout-master-set>

   <page-sequence master-reference="simple">
      <flow flow-name="body">
         <block>Hello World</block>
      </flow>
   </page-sequence>

</root>
```

This file is logically in three parts, namely the root, layout-master-set and page-sequence parts. All FO files share this structure.

4.1.1 Namespaces

The examples used in this manual follow the style shown in Figure 4-1, where the XSL-FO namespace is set (on the root element) as the default namespace for the file. Namespace prefixes are not used for the FO elements such as block. Figure 4-2 shows the same FO as Figure 4-1 but without the default namespace. Each element has the "fo:" namespace prefix. The files shown in Figure 4-1 and Figure 4-2 both create the same output and are treated equally by Ibex. Using namespaces is a matter of preference, it does not effect performance.

Figure 4-2: Simple XML using the fo prefix

```
<?xml version='1.0' encoding='UTF-8'?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">

  <fo:layout-master-set>
  <fo:simple-page-master master-name="simple">
      <fo:region-body margin="2.5cm" region-name="body"
        background-color='#eeeeee'/>
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="simple">
    <fo:flow flow-name="body">
      <fo:block>Hello World</block>
    </fo:flow>
  </fo:page-sequence>

</fo:root>
```

4.1.2 The root element

The root element shown in Figure 4-3 contains the whole content of the file and establishes the XSL-FO namespace as the default namespace. This element is the same for all FO files.

Figure 4-3: The root element

```
<root xmlns="http://www.w3.org/1999/XSL/Format">
```

Additional namespaces can be added to the xml element as shown in Figure 4-4.

Figure 4-4: The root element with additional namespaces

```
<root xmlns="http://www.w3.org/1999/XSL/Format"
            xmlns:ibex="http://www.xmlpdf.com/2003/ibex/Format"
            xmlns:svg="xmlns="http://www.w3.org/2000/svg"
>
```

4.1.3 The layout-master-set element

The layout-master-set element show in Figure 4-5 defines the shape and layout of pages in the document. Within the layout-master-set we have a simple-page-master element which in turn contains the region-body element.

The simple-page-master defines the layout of one type of page and is uniquely identified by its master-name attribute. The region-body element defines an area of the page where content will be placed. A page can have more than one region so we give the region a unique name "body" using the region-name attribute. This value is used with flow elements to specify which content goes into which region on the page.

Figure 4-5: The master-layout element

```
<layout-master-set>
   <simple-page-master master-name="simple">
      <region-body margin="2.5cm" region-name="body"
         background-color="#eeeeee"/>
   </simple-page-master>
</layout-master-set>
```

A FO file contains one or more simple-page-master elements, each with a unique master-name. In this simple example we have only one. Each simple-page-master element creates a formatting object known as a *page master*.

An example of a more complex document is the Ibex manual. Each chapter begins with a page which has no header. This is followed by a page which has left-aligned footer, then a page with a right-aligned footer. Each of the three possible page layouts is defined by a different simple-page-master element.

4.1.4 The page-sequence element

The page-sequence element shown in Figure 4-6 defines a sequence of pages that will appear in the PDF document. The master-reference attribute is used to tie the content of the page-sequence to a particular page layout, in this case one defined previously using a simple-page-master. When Ibex finds a page-sequence element it looks at the list of known simple-page-master and page-sequence-master elements (we have no page-sequence-master elements in this example) and finds one with a master-name attribute which equals the master-reference attribute on the page-sequence. If Ibex does not find a matching page master the FO file is invalid and Ibex will throw an exception.

Figure 4-6: The page-sequence element

```
<page-sequence master-reference="simple">
  <flow flow-name="body">
     <block>Hello World</block>
  </flow>
</page-sequence>
```

Within the page-sequence element we have a flow element. This holds the content which will appear on one or more pages. A page can have multiple regions. To associate content with a region we use the flow-name attribute on the flow element. In order for the content contained in the flow to appear on the page, the flow-name of the flow should match a region-name of one of the regions (in this example the region-body) on the page.

If the flow-name of the flow does not match a region-name of one of the regions on the page the content is not displayed on that page. This is not an error. It is a useful feature and we show how to use it later in this chapter.

Looking at the FO in Figure 4-7, the underlined names must match each other, and the names in *italics* should match if you want the content to appear.

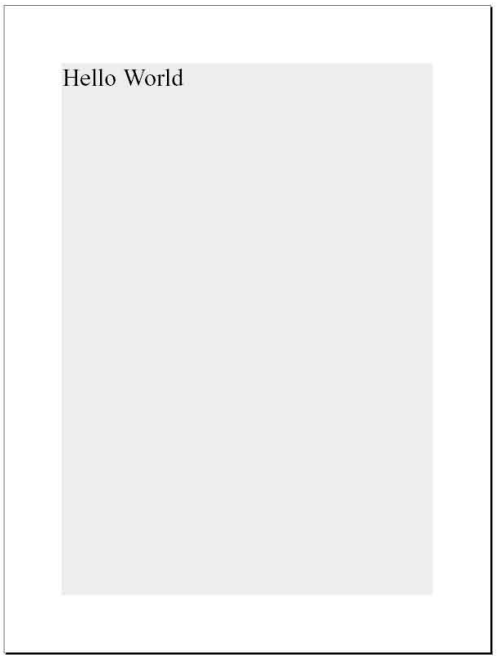
Figure 4-7: Matching master-name and master-reference

```
<layout-master-set>
  <simple-page-master master-name="simple">
     <region-body margin="2.5cm" region-name="body"
        background-color='#eeeeee'/>
  </simple-page-master>
</layout-master-set>

   <page-sequence master-reference="simple">
      <flow flow-name="body">
         <block>Hello World</block>
      </flow>
   </page-sequence>
```

Within the flow element we can have one or more "block level" elements. These are elements such as list, block and table which define the content to appear on the page. In this example we have a single block element containing the text "Hello World".

This produces a page like the one shown in Figure 4-8. The region created by the region-body element has a shaded background so you can see how large it is.

Figure 4-8: A basic page with a region-body and some text



4.2 Adding a footer region

In our example so far all the text contained in the flow element goes into the body region in the center of the page. To add a page footer we need to define a new region on the page and then define some new content to go into that region.

We define a footer region by adding a region-after element into the existing simple-page-master as shown in Figure 4-9.

Figure 4-9: Simple page master with footer region

```
<layout-master-set>
   <simple-page-master master-name="simple">
     ...
      <region-after extent='1cm' region-name="footer"
        background-color='#dddddd'/>
     ...
   </simple-page-master>
</layout-master-set>
```

The region-after element defines an area on the page which extends the full width of the page. If we had side regions (region-start and region-end) this might change, but in this example we have no side regions.

The height of the region created by the region-after element is defined by the extent attribute. In this example we have extent="1cm", so the region will be 1cm high and end at the bottom of the page.

Even without any content the footer region is still rendered on the page. Our page now looks like the one in Figure 4-10.

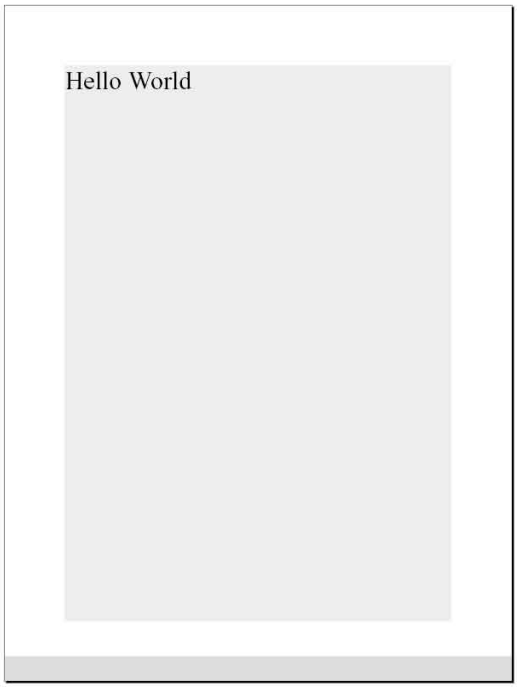


Figure 4-10: A basic page with a footer region

In its current position on the page the footer region will not print on most printers because they do not print right to the edge of the paper. We can define a margin around the whole page by setting the margin attribute on the simple-page-master element of the page-sequence as shown in Figure 4-11.

Figure 4-11: Simple page master with margin added

```
<layout-master-set>
  <simple-page-master master-name="simple"
          margin="2.5cm">
     <region-body margin="2.5cm" region-name="body"
        background-color="#eeeeee"/>
     <region-after extent="1cm" region-name="footer"
        background-color="#dddddd"/>
  </simple-page-master>
</layout-master-set>
```

The area inside the margins of the simple-page-master is called the "content area". The area covered by the regions (defined by the region-body and region-end) is measured from the inside of the page's content area, so when we add margins to the simple-page-master we reduce the size of the regions correspondingly.

Our page now appears as shown in Figure 4-12.

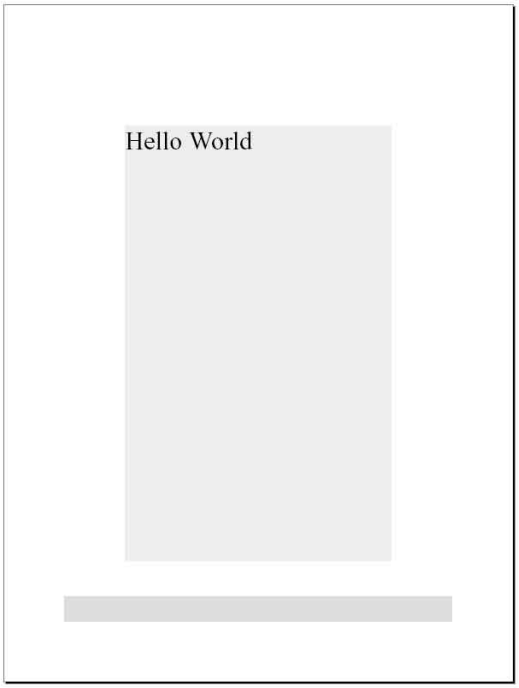


Figure 4-12: After adding margins to the simple-page-master

Now that we have some space on the sides of the body region we can remove the side margins from the body by changing the definition from that shown in Figure 4-13 to the one shown in Figure 4-14, resulting in the page layout shown in Figure 4-15.

Figure 4-13: Body with side margins

```
<region-body margin="2.5cm" region-name="body"
     background-color="#eeeeee"/>
```

Figure 4-14: Body without side margins

```
<region-body margin-top="2.5cm" margin-bottom="2.5cm"
   region-name="body"  background-color="#eeeeee"/>
```

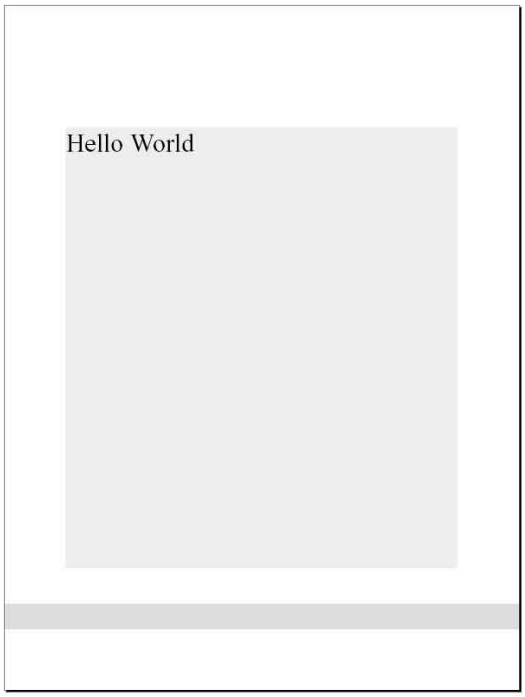


Figure 4-15: After removing the left and right margins from the region-body

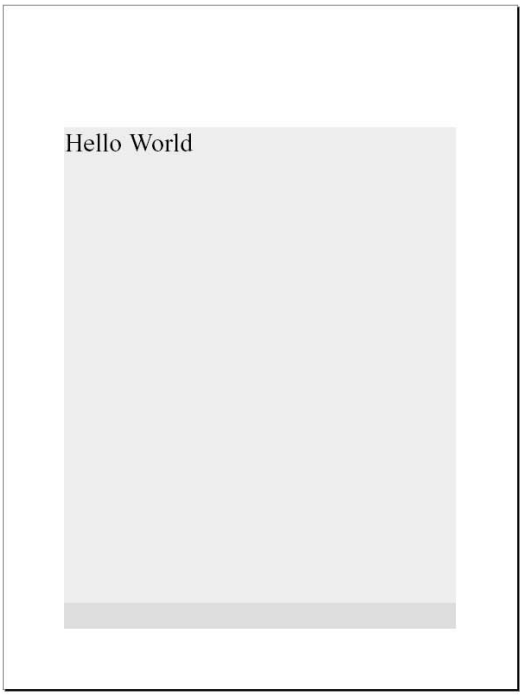
The last thing we need to do to get a working page layout is to make the footer region narrower by adding side regions. The left side region is created with a region-start element and the right side with a region-end element as in Figure 4-16. We can also specify the bottom-margin attribute of the body region so that it ends just where the footer starts, by setting margin-bottom= "1cm" on the region-body element.

Figure 4-16: side regions

```
<layout-master-set>
  <simple-page-master master-name="simple"
        margin='2.5cm'>
     <region-body margin="2.5cm" margin-bottom='1cm'
         region-name="body"
        background-color='#eeeeee'/>
     <region-after extent='1cm' region-name="footer"
        background-color='#dddddd'/>
     <region-start extent='2.5cm'/>
     <region-end extent='2.5cm'/>
  </simple-page-master>
</layout-master-set>
```

By default the side regions take precedence over the top and bottom regions so the top and bottom regions become narrower. This gives us the page layout shown in Figure 4-17, to which we can start adding some content.

Figure 4-17: With side regions to reduce the width of the footer



4.3 Attribute processing

The FO above also illustrates one of the ways in which XSL-FO handles attributes. We can specify a shorthand attribute such as "margin", which has the effect of setting the specific values margin-left, margin-right, margin-top and margin-bottom, and then override just the specific value we want (by setting margin-bottom="1cm"). The order in which the attributes are specified has no effect. A more specific setting will always override a more general one. So the two examples in Figure 4-18 and Figure 4-19 produce the same result.

Figure 4-18: Shorthand and specific attributes

```
<layout-master-set>
  <simple-page-master master-name="simple">
     <region-body margin="2.5cm" margin-bottom="1cm">
  </simple-page-master>
</layout-master-set>
```

Figure 4-19: Shorthand and specific attributes

```
<layout-master-set>
  <simple-page-master master-name="simple">
     <region-body margin-bottom="1cm" margin="2.5cm">
  </simple-page-master>
</layout-master-set>
```

4.4 Adding content to the footer

While content is added to the body of the page using the flow element, content is added to other regions using the static-content element. The "static" part of the static-content name refers to the fact that the content defined in this element stays within the region specified on this page. It does not flow from one page to the next. If the content exceeds the size of the region it will not flow to the next page.

The content of the static-content is repeated on every page which has a region with a matching flow-name (such as "footer"), and is typically different on every page as the page number changes.

To insert a simple footer with the words "XSL-FO Example" we add a static-content element as shown in Figure 4-20.

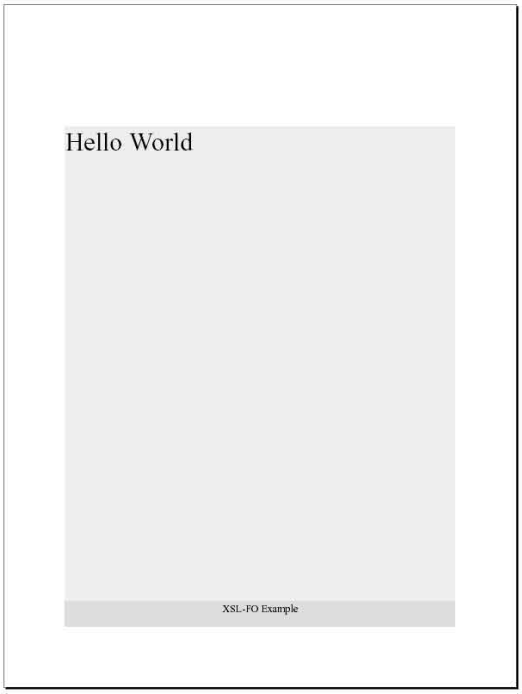
Figure 4-20: Adding a static-content element

```
<?xml version='1.0' encoding='UTF-8'?>
<root xmlns="http://www.w3.org/1999/XSL/Format">
   <layout-master-set>
      <simple-page-master master-name="simple"
           margin='2.5cm'>
         <region-body margin="2.5cm" margin-bottom='1cm'
            region-name="body"  background-color='#eeeeee'/>
         <region-after extent='1cm' region-name="footer"
            background-color='#dddddd'/>
         <region-start extent='2.5cm'/>
         <region-end extent='2.5cm'/>
      </simple-page-master>
   </layout-master-set>
   <page-sequence master-reference="simple">
      <static-content flow-name="footer">
          <block text-align='center'>
          XSL-FO Example</block>
      </static-content>
      <flow flow-name="body">
         <block>Hello World</block>
      </flow>
   </page-sequence>
</root>
```

Note that the order of the static-content and flow elements is important. All static-content elements must come before any flow elements.

This FO produces the page shown in Figure 4-21.

Figure 4-21: FO with static-content



Note that the flow-name of the static-content element and the region-name of the region-after element must match for the content to appear. This feature makes it

possible to have many static-content elements within the same page-sequence, and only those which match regions in the current simple-page-master will be rendered.

The Ibex manual has three different page layouts defined with three different simple-page-master elements. Each simple-page-master has a footer region with a different region-name. The main flow element contains three different static-content elements all containing footers. Only the footer whose flow-name matches the region-name for the currently active simple-page-master will be rendered.

4.5 Adding the page number to the footer

To insert the current page number into the document use the page-number element inside the static-content element as shown in Figure 4-22.

Figure 4-22: Adding a page number

```
<?xml version='1.0' encoding='UTF-8'?>
<root xmlns="http://www.w3.org/1999/XSL/Format">
   <layout-master-set>
      <simple-page-master master-name="simple"
           margin='2.5cm'>
         <region-body margin="2.5cm" margin-bottom='1cm'
            region-name="body"  background-color='#eeeeee'/>
         <region-after extent='1cm' region-name="footer"
            background-color='#dddddd'/>
         <region-start extent='2.5cm'/>
         <region-end extent='2.5cm'/>
      </simple-page-master>
   </layout-master-set>
   <page-sequence master-reference="simple">
      <static-content flow-name="footer">
          <block text-align='center'>
          XSL-FO Example, page <page-number/>
          </block>
      </static-content>
      <flow flow-name="body">
         <block>Hello World</block>
      </flow>
   </page-sequence>
</root>
```

This FO produces the page shown in Figure 4-23.

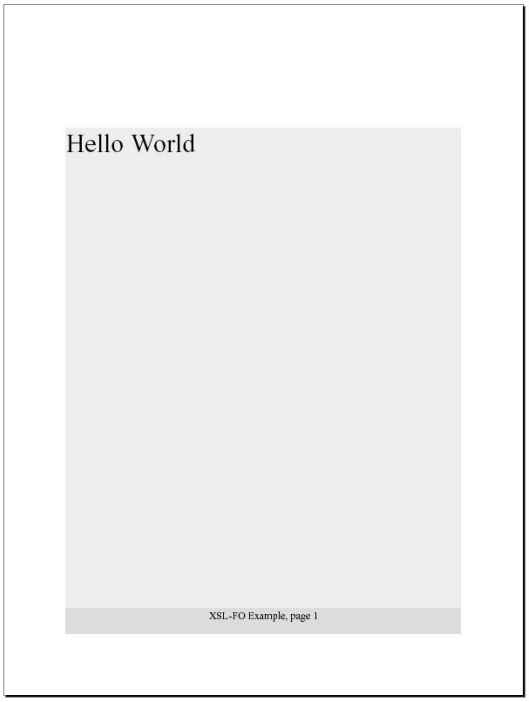


Figure 4-23: Page with page number

4.6 Adding the total page count to the footer

Adding the total page count (so we can have "page 3 of 5") is a two step process, based on the use of the "id" attribute which uniquely identifies an FO element. We place a block on the last page with the id of "last-page", and then we use the page-number-citation element to get *the number of the page on which that block appears* as our total number of pages.

Typically the block with the id of "last-page" is empty so a new page is not created at the end of the document.

The FO for the last block in the document is shown in Figure 4-24, and the FO to retrieve the last page number and put it in the footer is shown in Figure 4-25.

Figure 4-24: Block with id for last page

```
<block id="last-page"/>
```

Figure 4-25: FO to retrieve the page number of the identified block

```
<page-number-citation ref-id="last-page"/>
```

You can see how the id and ref-id values match. This is how Ibex associates the two elements and knows from which block to retrieve the page number.

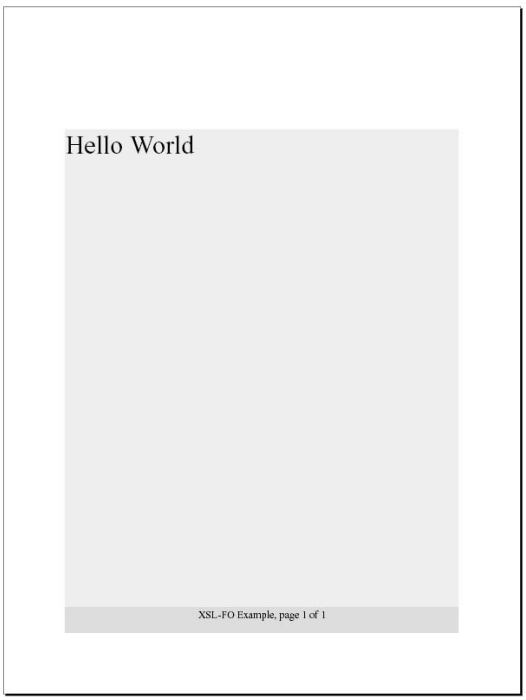
So bringing all these elements together we have the FO shown in Figure 4-26.

Figure 4-26: Complete FO to display total page count

```
<?xml version='1.0' encoding='UTF-8'?>
<root xmlns="http://www.w3.org/1999/XSL/Format">
   <layout-master-set>
      <simple-page-master master-name="simple"
            margin='2.5cm'>
         <region-body margin="2.5cm" margin-bottom='1cm'
            region-name="body"  background-color='#eeeeee'/>
         <region-after extent='1cm' region-name="footer"
            background-color='#dddddd'/>
         <region-start extent='2.5cm'/>
         <region-end extent='2.5cm'/>
      </simple-page-master>
   </layout-master-set>
   <page-sequence master-reference="simple">
      <static-content flow-name="footer">
          <block text-align='center'>
          XSL-FO Example, page <page-number/>
          of <page-number-citation ref-id='last-page'/>
          </block>
      </static-content>
      <flow flow-name="body">
         <block>Hello World</block>
         <block id='last-page'/>
      </flow>
   </page-sequence>
</root>
```

This FO produces the page shown in Figure 4-27.

Figure 4-27: Page with page number and total page count



4.7 Adding text content

Text is added to the body region of the page by using the block element. A block element can contain any amount of text and has attributes which define how the text will appear. These attributes are described in more detail later in the manual.

A block can contain text as shown in Figure 4-28.

Figure 4-28: Text in a block

```
<flow flow-name="body">
   <block>Hello World</block>
</flow>
```

A block element can also contain other block elements which in turn contain text or more nested elements. Figure 4-29 shows a block which contains another block with a different font, set using the font attribute.

Figure 4-29: Nested blocks

```
<flow flow-name="body">
     <block>
          Hello World
          <block font-size="16pt">
             this is a nested block
          </block>
     </block>
  </flow>
```

There is no limit to the nesting of block elements.

4.8 Using borders and padding

Many FO elements can have a border around the area they create on the page. If the border around an element is the same on all four sides then it can be defined with a single use of the border attribute. The space between a border and the content of the block (in this case the text) is controlled using the padding attribute. Figure 4-30 shows FO for a block with border and padding.

Figure 4-30: Block with border and padding

```
<flow flow-name="body">
  <block background-color='#eeeeee'>
    <block>
         Hello World
    </block>
    <block border='1pt solid red' padding='3pt'>
         Hello World
    </block>
  </block>
</flow>
```

This example has two block elements nested inside an outer element, with a background color set on the outer element to highlight the area created by the outer block. The block created from this FO is shown in Figure 4-31.

Figure 4-31: Default indentation of nested blocks

```
Hello World
Hello World
```

Ibex positions the content of the block (in this case the text) relative to the edge of the region. After positioning the content, the padding and borders are positioned relative to the position of the content. This places the padding and borders *outside* the content area of the block. The contents of the block are not indented, rather the padding and borders extend outside the block. This is the default behavior of XSL-FO formatters.

If you prefer Cascading Style Sheets (CSS) compatible behavior where adding a border to a block indents its content, you can specify the left-margin and right-margin attributes to force this to happen. Even if the left-margin and right-margin values are zero, CSS type indentation will still occur. The XML for this is shown in Figure 4-32 and the resulting output is shown in Figure 4-33.

Figure 4-32: Block with margins specified

```
<flow flow-name="body">
  <block background-color='#eeeeee'>
    <block>
      Hello World
    </block>
    <block border='1pt solid red' padding='3pt' margin-left="0" margin-right="0">
      Hello World
    </block>
  </block>
</flow>
```

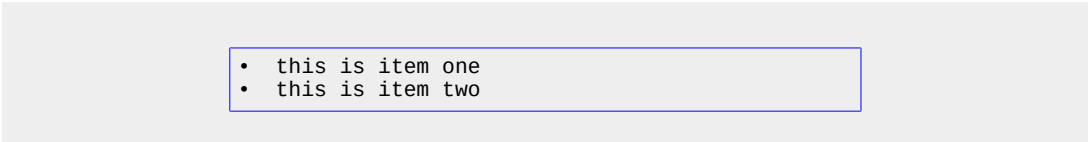
Figure 4-33: Default indentation of nested blocks

4.9 Creating lists

A list is content divided into two columns. Each item in the list is in two parts, called the *label* and the *body* respectively. A list is created with the list-block element. A list-block contains one or more list-item elements, each of which contains exactly one list-item-label element and one list-item-body element.

An example of a simple list is shown in Figure 4-34.

Figure 4-34: Example of list-block



This list was created with the FO shown in Figure 4-35:

Figure 4-35: FO for the list-block

```
<list-block
    margin-left='3cm' margin-right='3cm' padding='3pt'
    border='.1pt solid blue'
    provisional-distance-between-starts='0.5cm'
    provisional-label-separation='0.1cm'>
    <list-item>
        <list-item-label end-indent='label-end()'>
            <block>&#x2022;</block>
        </list-item-label>
        <list-item-body start-indent='body-start()'>
            <block>this is item one</block>
        </list-item-body>
    </list-item>

    <list-item>
        <list-item-label end-indent='label-end()'>
           <block>&#x2022;</block>
        </list-item-label>
        <list-item-body start-indent='body-start()'>
           <block>this is item two</block>
        </list-item-body>
    </list-item>
</list-block>
```

A list sets the two columns to widths specified using the attributes of the list-block elements. The provisional-distance-between-starts attribute specifies the distance between the start of the label column and the start of the body column. The provisional-label-separation attribute sets how much of the label column should be left empty to provide a blank space between the columns.

If we expand the above example and add more content to the first body we can see that the content is constrained to the column. If we add content to the first list-item-body as shown in Figure 4-36 we get the list shown in Figure 4-37.

Figure 4-36: FO for the list-block

```
<list-item-body start-indent='body-start()'>
  <block>
     If your Network Administrator has enabled it,
     Microsoft Windows can examine your network and
     automatically discover network connection settings.
  </block>
</list-item-body>
```

Figure 4-37: Output for the list-block

- If your Network Administrator has enabled it, Microsoft Windows can examine your network and automatically discover network connection settings.
- this is item two

For more information on lists see page 75.

4.10 Creating tables

A table is created using the table element. Within the table element there can be one table-header element, any number of table-body elements (which contain the rows) and one table-footer element. Each of the table-header, table-body and table-footer

elements contains one or more table-row elements, each containing one or more table-cell elements which in turn contain block-level elements such as block, table and list-block.

Since a table-cell can contain any block-level element you can easily create tables which contain text, nested tables or lists. Table headers and footers defined with the table-header and table-footer elements are automatically repeated at each page break, although this can be suppressed if required.

Figure 4-38 shows the FO for a simple table and Figure 4-39 shows the table created from the FO.

Figure 4-38: FO for a table

```
<table>
  <table-body>
    <table-row>
       <table-cell border='1pt solid blue' padding='2pt'>
          <block>row 1 column 1</block>
       </table-cell>
       <table-cell border='1pt solid blue' padding='2pt'>
          <block>row 1 column 2</block>
       </table-cell>
    </table-row>
       <table-cell border='1pt solid blue' padding='2pt'>
          <block>row 1 column 1</block>
       </table-cell>
       <table-cell border='1pt solid blue' padding='2pt'>
          <block>row 1 column 2</block>
       </table-cell>
    </table-row>
  </table-body>
</table>
```

This FO produces the table shown in Figure 4-39.

Figure 4-39: Simple table from the above FO

row 1 column 1	row 1 column 2
row 2 column 1	row 2 column 2

The padding and border attributes are not inherited from containing elements so must be defined on the table-cell elements.

4.10.1 Setting table column widths

The width of a table column is set using the table-column element. A table element contains zero or more table-column elements each of which defines properties such as width and background-color for a column in the table.

To make the first column 30% of the table width we would add table-column elements as shown in Figure 4-40, which creates the output shown in Figure 4-41.

Figure 4-40: Table with table-column elements

```
<table>
    <table-column column-width='30%' column-number='1'/>
    <table-column column-width='70%' column-number='2'/>
  <table-body>
    <table-row>
       <table-cell border='1pt solid blue' padding='2pt'>
          <block>row 1 column 1</block>
       </table-cell>
       <table-cell border='1pt solid blue' padding='2pt'>
          <block>row 1 column 2</block>
       </table-cell>
    </table-row>
    <table-row>
       <table-cell border='1pt solid blue' padding='2pt'>
          <block>row 1 column 1</block>
       </table-cell>
       <table-cell border='1pt solid blue' padding='2pt'>
          <block>row 1 column 2</block>
       </table-cell>
    </table-row>
  </table-body>
</table>
```

Figure 4-41: Rendered table with specified widths

row 1 column 1	row 1 column 2
row 2 column 1	row 2 column 2

For more information on tables see page 79.

Chapter 5

Using Ibex

This chapter describes how to call the Ibex API and how to use the accompanying command line program.

5.1 Ibex command line program

Although primarily intended to be used as a part of a larger application, Ibex ships with a command line program which can be used to create PDF files from FO files.

The command line programs shipped with Ibex are ibex10.exe (which uses .NET 1.0), ibex11.exe (which uses .NET 1.1) and ibex.exe (which uses .NET 2.0).

The command line syntax is the same for all programs. In these examples we use ibex.exe.

To create a PDF file from an FO file specify the file names on the command line. For instance to create hello.pdf from hello.fo, you do this:

```
ibex hello.fo hello.pdf
```

5.1.1 XSLT translation

The command line program will accept XML data and an XSLT stylesheet as inputs. The XML will be translated to FO by the stylesheet and the results then formatted to PDF. The command line syntax is:

```
ibex -xsl xsl-file xml-file pdf-file
```

So to create a PDF file from the files book.xml and book.xsl, the command is:

```
ibex -xsl book.xsl book.xml book.pdf
```

XSLT parameters can be passed to the stylesheet by adding them as name-value pairs to the command line. For instance, if we want to define the parameter called "age" to the value "30" we use a command like this:

```
ibex -xsl book.xsl book.xml hello.pdf "age=30"
```

The use of the double quotes around the name-value pair is necessary on some operating systems to force them to come through as a single parameter to the Ibex program.

5.1.2 Logging from the command line

Any informational or error messages will be logged to the console. To send error messages to a file as well, use the -logfile option. For example to log errors to the file ibex.log, you would do this:

```
ibex -logfile ibex.log hello.fo hello.pdf
```

5.1.3 Listing available fonts

You can also list the fonts which are available (based on what fonts are installed on your system) by using the -fonts option like this:

```
ibex -fonts
```

The list of fonts is produced as a FO file to the standard output. This can be redirected to a file and then used as input to Ibex to create a PDF file containing a table which looks like this:

	file	usage	example
minion	c:\windows\fonts\MOB_____.TTF	10pt minion	**10pt minion**
	c:\windows\fonts\MOB_____.TTF	bold 10pt minion	**bold 10pt minion**
	c:\windows\fonts\MOI_____.TTF	italic 10pt minion	*italic 10pt minion*
	c:\windows\fonts\MOBI____.TTF	bold italic 10pt minion	***bold italic 10pt minion***

The list of fonts can be limited to fonts whose name contains a specified string by passing the string on the command line. For instance if we wanted to see what versions of "arial" are installed, we can use the command:

```
ibex -fonts arial
```

5.2 The Ibex API

A PDF document is generated using the FODocument object which is in the ibex4 namespace.

First you create a new FODocument object and then calling the generate() method on that object. The generate() method has various versions which take different parameters depending on whether the input is from files or streams and whether XSLT translation should occur.

The FODocument object is not thread safe. A new FODcoument should be created for each PDF file to be created. Ibex does support creating multiple PDF files concurrently on multiple threads, as long as each PDF file is associated with a unique FODocument instance.

Example C# code to convert the file "manual.fo" to "manual.pdf" the code is shown in Figure 5-1, the equivalent VB.NET code is in Figure 5-2.

Figure 5-1: C# code to create a PDF from an FO file

```
using System;
using ibex4;

public class Simple {

   static void Main( string[] args ) {

      FODocument doc = new FODocument();

      gen.generate( "manual.fo", "manual.pdf" );
   }

}
```

Figure 5-2: VB.NET code to create a PDF from an FO file

```
Imports System
Imports ibex4

Module Module1

    Sub Main()

        Dim doc As New FODocument

        doc.generate("manual.fo", "manual.pdf")

    End Sub

End Module
```

Projects need to have a reference to the ibex DLL.

5.2.1 Generating to File

```
public void generate( string fo_file_name, string pdf_file_name )
```

This will read the FO contained in the file named in pdf_file_name and create the PDF file named in pdf_file_name.

5.2.2 Generating using streams

```
public void generate( Stream fo_stream, Stream pdf_stream )

public void generate( Stream fo_stream, Stream pdf_stream, bool close_stream )
```

This will read the FO from the System.IO.Stream called fo_stream and create the PDF file into the System.IO.Stream pdf_stream. These streams can be anything derived from System.IO.Stream, such as System.IO.FileStream or System.IO.MemoryStream.

If close_stream is true the PDF stream will be closed after the PDF file is generated, if false it will not. By default the stream is closed. Not closing the stream is useful if you are generating to a MemoryStream object as the bytes cannot be read from the MemoryStream if it has been closed.

5.2.3 Generating a PDF from XML and XSL

These methods take XML, an XSLT stylesheet, and a stream to write the resulting PDF file to.

```
public void generate( Stream xml_stream, Stream xsl_stream, Stream pdf_stream )

public void generate( Stream xml_stream, Stream xsl_stream, Stream pdf_stream, bool
closeStream )
```

Ibex uses the .NET XSLT processor to transform the XML using the specified stylesheet and passes the resulting FO to the PDF creation routines. XSLT transformation is faster or more efficient in .NET 2.0 and later and we recommend using this version or later if possible.

5.2.4 Generate a PDF from XML and XSL with parameters

These methods are similar to the ones in the previous section but take an additional hashtable which (if not null) should contain name-value pairs which are then passed as arguments to the XSLT translation process.

```
public void generate( Stream xml_stream, Stream xsl_stream, Stream pdf_stream,
                      bool close_stream, Hashtable params )
```

Chapter 6

Error Handling & Logging

This chapter describes error handling using the Ibex API.

Ibex associates an error handler with the library as a whole. Generally this error handler will log a message and not throw an exception.

The Ibex Logger object is a singleton which is retrieved using a call to the ibex4.logging.Logger.getLogger() method. Typically you would import the ibex4.logging namespace and then access the logger as shown in Figure 6-1.

Figure 6-1: Clearing existing error handlers

```
using ibex4.logging;

void sumfunc() {
  Logger.getLogger().clearHandlers();
}
```

The default error handler writes messages to the console. Messages are displayed in various circumstances including:

- when an invalid attribute is found;
- when a reference is made to a font or image file which cannot be found;
- when a formatting error occurs, such as defining the widths of columns in table that exceed the available width.

As the Ibex Logger is a singleton object, **logging should be configured once at the start of an application, not on a per-document basis.**

6.1 Error severity

To change the level of information logged you can set the level on the logging object to one of the values defined in the ibex4.logging.Level object. Possible levels of logging which can be set are:

SEVERE WARNING INFO CONFIG FINE FINER FINEST

An example of how to set the logger to log only messages which are WARNING or worse is shown in Figure 6-2.

Figure 6-2: Setting the error level

```
using System;

using ibex4;
using ibex4.logging;

public class Create {

  public static void Main( string[] args ) {

    PDFDocument doc = new PDFDocument();

    Logger.getLogger().setLevel( Level.WARNING );
```

6.2 Logging to a file

To log messages to a file, create an ibex4.logging.FileHandler object and then tell the logger to log to this object. The example in Figure 6-3 logs to the file "log.txt", but any valid file name can be used.

Figure 6-3: Logging to a file

```
using System;
using ibex4;
using ibex4.logging;

public class Create {

  public static void Main( string[] args ) {

    Logger.getLogger()
      .setLevel( Level.SEVERE )
      .clearHandlers()
      .addHandler(
        new FileHandler("log.txt") );
```

The FileHandler object synchronises access to the log file.

If you omit the clearHandlers() call shown in the above example, log records will be written to the default console handler and also to the file handler. You will see error messages on the console and they will also be written to the file.

6.3 Logging to a stream

Ibex can log messages to a stream created by the caller. The stream is any object which implements the System.IO.Stream interface.

To log messages to a stream, create an ibex4.logging.StreamHandler object and then tell the logger to log to this object. The example in Figure 6-4 logs to a MemoryStream, but any valid stream can be used.

Figure 6-4: Logging to a stream

```
using System;
using System.IO;
using ibex4;
using ibex4.logging;

public class Create {

  public static void Main( string[] args ) {

    Logger.getLogger().clearHandlers();
    MemoryStream stream = new MemoryStream();
    StreamHandler h = new StreamHandler( stream );
    Logger.getLogger().addHandler( h )
```

If you omit the clearHandlers() call shown in the above example log records will be written to the default console handler and to the stream handler as well.

6.4 Logging to multiple destinations

Errors can be logged to any number of handlers. The example in Figure 6-5 logs to a file called "xslfo.log", to a memory stream and to the console.

Figure 6-5: Logging to multiple destinations

```
using System;
using System.IO;

using ibex4;
using ibex4.logging;

public class Create {

public static void Main( string[] args ) {

   MemoryStream stream = new MemoryStream();

   Logger.getLogger()
        .addHandler( new ConsoleHandler() )
        .addHandler( new StreamHandler( stream ) )
        .addHandler( new FileHandler("xslfo.log") );
   }
}
```

Chapter 7

Page Layout

This chapter describes how to configure the size of a page and position the regions in which content appears.

7.1 Using one layout for all pages

The first element in any FO file is the root element which contains the whole FO tree defining the document and declares the XML namespaces used. Figure 7-1 shows a simple FO file.

Figure 7-1: Simple FO file

```
<root xmlns="http://www.w3.org/1999/XSL/Format">
  <layout-master-set>
    <simple-page-master master-name="layout" page-width="8.5in" page-height="8in">
      <region-body region-name="body" margin="2.5cm"/>
    </simple-page-master>
  </layout-master-set>
  <page-sequence master-reference="layout">
    <flow flow-name="body">
      <block>Hello world</block>
    </flow>
  </page-sequence>
</root>
```

The first FO element within the root element is the layout-master-set element. This contains one or more simple-page-master elements which define the layout of a page, including the width and height.

The simple-page-master element is like a template for a page, defining the page size and the areas on the page into which content will be placed. As content is read from a flow, Ibex decides which simple-page-master to use as the basis for creating the current page. If there is only one simple-page-master then it is always used. If there are several simple-page-masters then a selection process is used to see which one applies to the current page.

The simple-page-master element contains region elements such as region-body which define an area on the page which can be filled with text or image content.

There can be any number of simple-page-master elements provided each has a unique master-name attribute.

Figure 7-2 shows an example of a layout-master-set.

Figure 7-2: Example layout-master-set

```
<layout-master-set>
  <simple-page-master master-name="front-page">
    <region-body margin-right="2.5cm"
           margin-left="4cm"
           margin-bottom="4cm"
           margin-top="4cm" region-name="body"
     background-color="#eeeeee"/>
    <region-after extent="3cm" region-name="footer"
       background-color='#dddddd'/>
  </simple-page-master>
</layout-master-set>
```

This shows a layout-master-set which contains a single simple-page-master with a master-name of "front-page".

This simple-page-master defines a page which has two regions on which content can be printed. A page defined with this layout appears in the examples at the end of this chapter, on page 45. For the purposes of this example the regions have background-colors defined to show them clearly. More complex layouts showing five regions appear in the examples on page 45.

Having defined a page layout which has a name, (defined by its master-name attribute) we then use the page-sequence element to define the content of the document. The page-sequence element has a master-name attribute which should match the master-name defined for a simple-page-master (or a page-sequence-master, more of which later).

A page-sequence for printing "Hello World" is shown in Figure 7-3.

Figure 7-3: page-sequence for hello world

```
<page-sequence master-reference="front-page">
  <flow flow-name="body">
    <block>Hello World</block>
  </flow>
</page-sequence>
```

A key thing to note is that the content of the page-sequence is contained in a flow element. For content of the flow to appear on the PDF page *the flow-name attribute of the flow element must match the region-name of a region on the page master specified by the master-reference on the page-sequence.* If the flow-name does not match a region-name, none of the content of this flow will appear in the output document.

It is important to understand this feature. It means that a page-sequence can contain multiple flow and static-content elements each containing a flow element with a different flow-name. Only flow elements whose flow-name attribute matches a region-name defined in the current page sequence will appear. This is how we produce different formats for odd and even pages.

Figure 7-4 shows in matching colors the attributes which should match for content to appear.

Figure 7-4: Matching flow and region names

```
<root xmlns="http://www.w3.org/1999/XSL/Format">
  <layout-master-set>
    <simple-page-master master-name="front-page">
      <region-body margin-right="2.5cm"
           margin-left="4cm"
           margin-bottom="4cm"
           margin-top="4cm" region-name="body"/>
      <region-after extent="3cm" region-name="footer"/>
    </simple-page-master>
  </layout-master-set>

  <page-sequence master-reference="front-page">
    <flow flow-name="body">
      <block>Hello World</block>
    </flow>
  </page-sequence>
</root>
```

7.2 Using different layouts for different pages

It is possible to define different page layouts for different pages. This can be done in two possible ways, either by assigning different page masters to different page sequences, or by using a page-master-alternatives element which chooses from a set of simple-page-master elements based on criteria such as the current page number.

7.2.1 Using different page masters for each page sequence

Using a different page master for each page sequence is useful when you can clearly divide the document into distinct sections. For example, this manual has a different page master for the front cover and for the pages in the table of contents. The page masters for this are shown in Figure 7-5.

Figure 7-5: Two page masters

```
<layout-master-set>

  <simple-page-master master-name="front-page" margin="1.5cm" page-height="297mm"
page-width="210mm">
    <region-body region-name="body" margin="0.75cm 0.5cm 0.75cm 3cm"/>
    <region-before region-name="header" extent="2.5cm"/>
    <region-after region-name="footer" extent="1cm"/>
    <region-start extent="1cm" background-color="#eeeeee"/>
  </simple-page-master>


  <simple-page-master master-name="toc-page" margin="1.5cm" >
    <region-body column-count="1" region-name="body" margin="0.75cm 0.5cm 1cm 3cm"
margin-left="2cm" margin-right="1.5cm" />
    <region-before region-name="header" extent="1cm"/>
    <region-after region-name="footer" extent="0.75cm"/>
    <region-start extent="2cm" />
    <region-end region-name="end" extent="1.5cm" />
  </simple-page-master>

</layout-master-set>
```

Content is allocated to the two sections of the document using two separate page-sequences, as shown in Figure 7-6.

Figure 7-6: Allocating content to two page masters

```
<page-sequence master-reference="front-page">
   <flow flow-name="body">
      <block>
        content that appears in the body of the front page
      </block>
  </flow>
</page-sequence>

<page-sequence master-reference="toc-page">
   <flow flow-name="body">
      <block>
        content that appears in the table of contents
      </block>
  </flow>
</page-sequence>
```

When using this approach content from one flow always appears on pages with the same layout. Flowing content across different page layouts is described in the next section.

7.2.2 Using page master alternatives

Often it is desirable to have content flow continuously across pages with different layouts. This is done in the Ibex manual, where the pages are laid out like this:

first page of chapter	has no page header page number is on the right of the footer
even numbered page	has a page header page number is on the left of the footer
odd numbered page	has a page header page number is on the right of the footer

The three page masters are shown in Figure 7-7.

Figure 7-7: Page masters for three different layouts

```
<simple-page-master  master-name="chapter-odd-no-header">
  <region-body region-name="body" margin="2.5cm 2.5cm 2.5cm 4.0cm"/>
  <region-after region-name="footer-odd" extent="1.5cm" display-align="before"/>
</simple-page-master>

<simple-page-master master-name="chapter-even">
  <region-body region-name="body" margin="2.5cm 2.5cm 2.5cm 4.0cm" column-count="1"/>
  <region-before region-name="header-even" extent="1.5cm" display-align="after"/>
  <region-after region-name="footer-even" extent="1.5cm" display-align="before"/>
</simple-page-master>

<simple-page-master  master-name="chapter-odd">
  <region-body region-name="body" margin="2.5cm 2.5cm 2.5cm 4.0cm"/>
  <region-before region-name="header-odd" extent="1.5cm" display-align="after"/>
  <region-after region-name="footer-odd" extent="1.5cm" display-align="before"/>
</simple-page-master>
```

To make content from a single flow element span multiple pages with different page layouts we use a page-sequence-master element as shown in Figure 7-8. This element contains a repeatable-page-master-alternatives element, which in turn contains a set of conditional-page-master-reference elements.

When formatting content from a page-sequence which has flow-name="chapter", Ibex looks at each of the conditional-page-master-reference elements and chooses which one will be active for the current page. This is done by evaluating conditions specified with the page-position attribute. As a page is created, each conditional-page-master-reference is considered in turn, starting from the first one. The first one found whose conditions are satisfied will determine the page master for the current page. **Since alternatives are considered in the order in which they appear in the FO, the order in which the alternatives are listed is important.**

When the first page of the chapter is being created, the page-position="first" condition is true, so the first conditional-page-master-reference will be chosen because it has page-position = "first". This has master-reference = "chapter-odd-no-header", so the simple-page-master with master-name = "chapter-odd-no-header" becomes the active page master for the first page of the chapter.

When the second page of the chapter is being created, the page-position="first" is no longer true so the conditions on the next conditional-page-master-reference will be evaluated.

Although not shown in this example, other attributes such as blank-or-not-blank can be used to control the selection of one of the alternatives.

Figure 7-8: The page-sequence-master element

```
<page-sequence-master master-name="chapter" >
  <repeatable-page-master-alternatives>

    <conditional-page-master-reference page-position="first"
             master-reference="chapter-odd-no-header"/>

    <conditional-page-master-reference odd-or-even="odd"
             master-reference="chapter-odd"/>

    <conditional-page-master-reference odd-or-even="even"
             master-reference="chapter-even"/>

  </repeatable-page-master-alternatives>
</page-sequence-master>
```

region-before

region-body

This page layout is created with the XML below. Note that by default the region-start and region-end regions extend the full height of the page and the region-before and region-after regions are narrowed so as not to overlap the side regions. See the following page for an example where the precedence attribute is used to change this.

```
<simple-page-master master-name="region-example-1">

    <region-body margin="2.5cm" region-name="body"
       background-color="#eeeeee"/>

    <region-before extent="2.5cm" region-name="header"
       background-color="#dddddd"/>

    <region-after extent="2.5cm" region-name="footer"
       background-color="#dddddd"/>

    <region-start extent="2.5cm" region-name="start"
       background-color="#aaaaaa"/>

    <region-end extent="2.5cm" region-name="end"
       background-color="#aaaaaa"/>

</simple-page-master>
```

region-after

region-before region-example-1-margins

region-body

This page layout is created with the XML below. Note that by default the region-start and region-end regions extend the full height of the page and the region-before and region-after regions are narrowed so as not to overlap the side regions. See the following page for an example where the precedence attribute is used to change this.
This layout differs from the previous page in that the simple-page-master has the margin attribute set to "2.5cm". This creates a margin of 2.5cm around the entire page, and regions are positioned with respect to the rectangle created by the margins, not with respect to the edges of the paper.

```
<simple-page-master
master-name="region-example-1M" margin="2.5cm">

    <region-body margin="2.5cm"
region-name="body"
       background-color="#eeeeee"/>

    <region-before extent="2.5cm"
region-name="header"
       background-color="#dddddd"/>

    <region-after extent="2.5cm"
region-name="footer"
       background-color="#dddddd"/>

    <region-start extent="2.5cm"
region-name="start"
       background-color="#aaaaaa"/>

    <region-end extent="2.5cm"
region-name="end"
       background-color="#aaaaaa"/>

</simple-page-master>
```

region-after

region-before

region-body

region-after

This page layout is created with the XML below. Note that the region-before and region-after regions have precedence="true" so they extend the full width of the page and the side regions are reduced in height to the regions do not overlap.

```
<simple-page-master master-name="region-example-1">

    <region-body margin="2.5cm" region-name="body"
       background-color="#eeeeee"/>

    <region-before extent="2.5cm" region-name="header"
       precedence="true" background-color="#dddddd"/>

    <region-after extent="2.5cm" region-name="footer"
       precedence="true" background-color="#dddddd"/>

    <region-start extent="2.5cm" region-name="start"
       background-color="#aaaaaa"/>

    <region-end extent="2.5cm" region-name="end"
       background-color="#aaaaaa"/>

</simple-page-master>
```

Chapter 8

Text Formatting

Text is created in the output document using the block element.

The simplest possible block is shown in Figure 8-1.

Figure 8-1: A simple block

```
<block>hello world</block>
```

This creates a paragraph in the output document which has the default font (which is helvetica) and the default alignment (which is left).

The sections below describe elements and attributes used to control the formatting of text.

8.1 Using the font attribute

The quickest way to get the font you require is to use the `font` attribute, as shown in Figure 8-2.

Figure 8-2: Using the font attribute

```
<block font="bold 12pt garamond">hello world</block>
```

Using the font attribute is simpler than specifying all the individual attributes such as font-weight and font-size, but does need some care. When using the font attribute the order of the words is important. The font style (normal, italic) and the font weight (bold, normal) must come before the font size. The font name must come after the font size. If the font name contains spaces, it must be enclosed in quotes, as shown in Figure 8-3.

Figure 8-3: A font name with spaces

```
<block font="bold 12pt "times new roman"">
      hello world
</block>
```

The full syntax of the font attribute is shown in Figure 8-4.

Figure 8-4: Syntax of font attribute

```
[ [ <font-style> || <font-variant> || <font-weight> ]?
<font-size> [ / <lineheight>]?
<font-family> ]
```

8.2 Using the font-family attribute

The font-family attribute is used to specify the name of the font to use. More than one font name can be listed. These names can be specific font names such as "times roman" or "garamond", or generic names such as "monospace". Ibex will use the first name in the list which matches a font on your system. Font names are separated by a comma.

The ability to list multiple font names derives from the CSS standard. It is designed to support the creation of a web page which will be rendered on a computer that may not have the same fonts installed as the page's author. In practice when you generate a PDF file you know what fonts you have installed, so you will probably just specify one font.

8.3 Italic text

Text is made italic using the font-style attribute.

The font style can be "normal" or "italic". Other font values such as the font-family are inherited from the current font, as shown in Figure 8-5. The output created by the FO in Figure 8-5 is shown in Figure 8-6.

Figure 8-5: Using font-style

```
<block font-family="arial">
  hello <inline font-style="italic">world</inline>
</block>
```

Figure 8-6: Using the font-style attribute

hello *world*

8.4 Bold text

Text is made bold using the font-weight attribute.

The font weight can be "normal" or "bold", as shown in Figure 8-7. The output created by the FO in Figure 8-7 is shown in Figure 8-8.

Figure 8-7: Using the font-weight attribute

```
<block font-family="arial">
  hello <inline font-weight="bold">world</inline>
</block>
```

Figure 8-8: Using font-weight

hello **world**

8.5 Text size

The size of text is set using the font-size attribute.

The font size specifies the size of the font and can be specified in a number of ways listed below.

A numeric size

The most common approach is to specify the size you want in points, for example font-size="12pt" or font-size="30pt".

An absolute size

Attribute Value	Size
xx-small	7.0pt
x-small	8.3pt
small	10.0pt
medium	12.0pt
large	14.4pt
x-large	17.4pt
xx-large	20.7pt

A relative size

This sets the font size based on the size of the prevailing font.

Attribute Value	Size
smaller	existing size / 1.2
larger	existing size * 1.2

Another way of setting the font size relative to the current font size is to use the "em" unit. "1.0em" is the current font size, so "2.0em" specifies a size which is twice as big as the current size.

8.6 Underlining text

Text is underlined using the text-decoration attribute.

Specifying text-decoration="underline" will cause text to be underlined, like this.

8.7 Striking out text

You can strike out text using the text-decoration attribute.

Specifying text-decoration="line-through" will cause text to be underlined, like ~~this~~.

8.8 Horizontal alignment

Horizontal alignment is specified using the text-align attribute. The default alignment is left.

Valid values for text-align are shown in the table below.

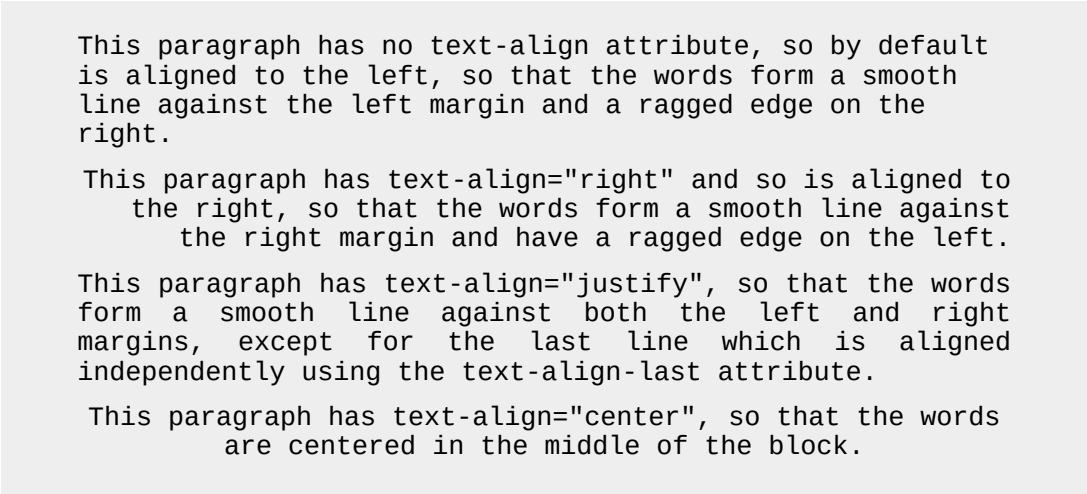
Value	Effect
left	text is aligned against the left edge of the block
right	text is aligned against the right edge of the block

Value	Effect
center	text is centered in the middle of the block
justify	text is aligned against both the left and right edges of the block. Space is inserted between words to achieve this effect. Setting text-align = "justify" does not align the last line of the paragraph, this is done using text-align-last = "justify".
start	text is aligned against the start edge, which for a block that is not rotated, with the default left-to-right writing direction, is the left edge.
end	text is aligned against the end edge, which for a block that is not rotated, with the default left-to-right writing direction, is the right edge.
inside	assuming the document is to be bound as a book, text is aligned against the edge which is nearest the binding. For an odd-numbered page this will be the left edge, for an even numbered page it will be the right edge.
outside	assuming the document is to be bound as a book, text is aligned against the edge which is furtherest from the binding. For an odd-numbered page this will be the right edge, for an even numbered page it will be the left edge.

For text-align values of "inside" and "outside" the page number is used to determine the binding edge, which is assumed to be the left hand edge of odd-numbered pages and the right hand edge of even-numbered pages.

The effect of some of the text-align values is shown in Figure 8-9.

Figure 8-9: Effects of text-align values



8.8.1 Justifying the last line of a paragraph

Specifying text-align="justify" will justify all lines of a paragraph except the last. This is because a justified paragraph typically looks like the one in Figure 8-10, with the last line not being justified.

Figure 8-10: Paragraph without the last line justified

```
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nunc mollis, turpis
vehicula aliquam auctor, metus turpis tempus justo, eu gravida nisl nibh vitae nisl.
Cras a nisl. Integer et metus vitae dui placerat egestas. Duis rutrum. Nulla in
enim. Suspendisse vel massa in mauris sagittis pharetra. Etiam hendrerit euismod
velit. Ut laoreet lectus nec nisl.
```

The text-align-last attribute controls the alignment of the last line of a paragraph. Values include are shown in the table below:

Value	Effect
relative	if text-align is "justify", align the last line against the start edge (normally the left edge), otherwise use the setting if the text-align attribute.
left	text is aligned against the left edge of the block
right	text is aligned against the right edge of the block
start	text is aligned against the start edge, which for a block that is not rotated, with the default left-to-right writing direction, is the left edge.
end	text is aligned against the end edge, which for a block that is not rotated, with the default left-to-right writing direction, is the right edge.
inside	assuming the document is to be bound as a book, text is aligned against the edge which is nearest the binding. For an odd-numbered page this will be the left edge, for an even numbered page it will be the right edge.
outside	assuming the document is to be bound as a book, text is aligned against the edge which is furtherest from the binding. For an odd-numbered page this will be the right edge, for an even numbered page it will be the left edge.
justify	justify the last line across the whole width of page.

8.9 Left and right margins

The margins of a block are specified using the margin-left and margin-right attributes.

The margin properties indent the edge of the paragraph by the specified amount from the edge of the containing area.

The FO for a block with a 2.5cm left margin is shown in Figure 8-11.

Figure 8-11: Setting the left margin

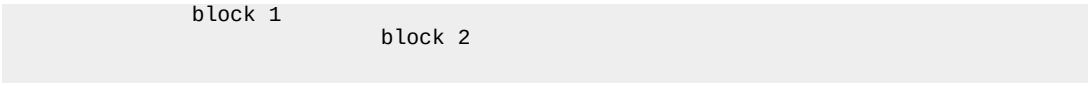
```
<block margin-left="2.5cm">hello world</block>
```

If we nest another block inside this one, as shown in Figure 8-12, the margins are cumulative. The output from this FO is shown in Figure 8-13.

Figure 8-12: Nested blocks

```
<block margin-left="2.5cm">
  block 1
  <block margin-left="2.5cm">
    block 2
  </block>
</block>
```

Figure 8-13: Output from the above FO



Putting background colors on the blocks shows this more clearly. The FO is in Figure 8-14 and the output is in Figure 8-15.

Figure 8-14: Nested blocks with background color

```
<block margin-left="2.5cm" background-color="#777777">
  block 1
  <block margin-left="2.5cm"  background-color="#999999">
    block 2
  </block>
</block>
```

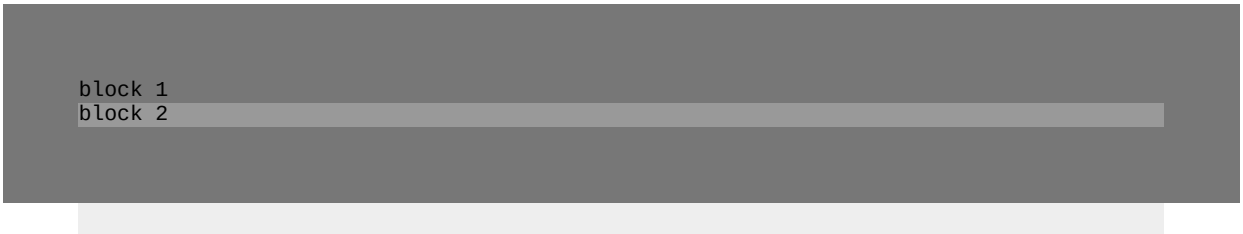
Figure 8-15: Output from above FO

The approach to indentation defined in the XSL-FO standard is that *the content* of two nested blocks *which do not specify a margin* have the same left edge. The edges of the content (which in our example is the text) are aligned, and any borders and padding are placed outside those edges. Figure 8-16 shows the FO for two nested blocks with no margin attributes. The text will be vertically aligned and the background colors will be placed outside the text. Figure 8-17 shows the resulting output.

Figure 8-16: Nested blocks with no margins specified

```
<block padding="1cm" background-color="#777777">
  block 1
  <block padding="1cm" background-color="#999999">
    block 2
  </block>
</block>
```

Figure 8-17: Output from nested blocks with no margins



In XSL-FO terms, both areas have the same start-indent and hence the same content rectangle, and the padding on the outer block extends outside its content rectangle. This may seem counter-intuitive to some developers used to the CSS model. You can invoke the CSS nested areas model by specifying a margin-left value, even "opt".

8.10 Spacing between letters

The amount of space between two letters is dependent on the font used. Ibex reads the TrueType or Type 1 font file and loads the width of each character. Kerning information which specifies adjustments to the gaps between particular pairs of characters is also read from the font file and used in the text formatting process.

The spacing between letters can be changed using the letter-spacing attribute. Any value specified using this attribute is *added* to the default spacing specified by the font.

Figure 8-18 shows the FO to increase the letter spacing of some text. The resulting text is shown in Figure 8-19.

Figure 8-18: Using letter-spacing

```
<block letter-spacing="0.2em">WELLINGTON NEW ZEALAND</block>
```

Figure 8-19: Text formatted using letter-spacing

```
W E L L I N G T O N  N E W  Z E A L A N D
```

It is possible to make letters closer than normal using a negative value for letter-spacing. Example FO for this is shown in Figure 8-20 and the result in Figure 8-21.

Figure 8-20: Moving letters closer together

```
<block letter-spacing="-0.1em">WELLINGTON NEW ZEALAND</block>
```

Figure 8-21: Text formatted using negative letter-spacing

```
WELLINGTON NEW ZEALAND
```

8.11 Spacing before and after words

Spacing before and after text is specified using the space-start and space-end attributes on the inline element.

The space-start attribute specifies space to appear before text, space-end specifies space to appear after the text.

Figure 8-22 shows how to specify a gap between two words. This FO produces a 3cm gap between the words as shown in Figure 8-23 .

Figure 8-22: Using space-start

```
<block>
  hello <inline space-start="3cm">world</inline>
</block>
```

Figure 8-23: Output using space-start

```
hello                    world
```

Space between words is collapsed (i.e. merged) by default. If a word has space-end="1.0cm" and the following word has space-start="0.5cm", the gap between

the two words will be the larger of the two spaces (i.e. 1.0cm), not the sum. FO showing this is in Figure 8-24 and the output is in Figure 8-25.

Figure 8-24: FO showing merging of spaces

```
<block>
  <inline space-end="1cm">hello</inline>
  <inline space-start="0.5cm">world</inline>
</block>
```

Figure 8-25: The resulting 1.0cm space

```
hello      world
```

8.12 Forcing a line break

You can cause a line break in normal text by inserting an empty block element. Figure 8-26 shows an FO example which does this and Figure 8-27 shows the resulting output.

Figure 8-26: Forcing a line break

```
<block>
  this will be line one <block/>this will be line two
</block>
```

Figure 8-27: Line break created with an empty block

```
this will be line one
this will be line two
```

8.13 Space at the start of a line

Space specified with the space-start attribute is normally discarded at the start of the line. To force it to be retained use the space-start.conditionality attribute.

Figure 8-28 shows two blocks which create two lines. The first block will have no space at the start of the word. The second block has space-start.conditionality="retain" so the space specified by the space-start="1cm" will be retained. The output created by this FO is shown in Figure 8-29.

Figure 8-28: Using retain

```
<block background-color="#eeeeee">
  <inline space-start="1cm">
  discard
  </inline>
</block>
<block background-color="#eeeeee">
  <inline space-start="1cm" space-start.conditionality="retain">
  retain
  </inline>
</block>
```

Figure 8-29: Output from using retain

```
discard
     retain
```

8.14 Vertical alignment

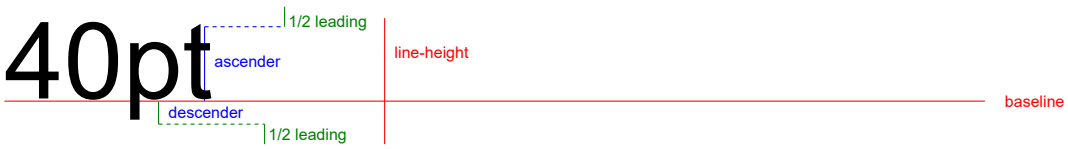
The vertical alignment of blocks of text within a containing flow or block is controlled by the display-align attribute.

The vertical alignment of words on a line is controlled by the vertical-align attribute.

Text on a line is positioned relative to the baseline, which is shown in Figure 8-30.

By default text sits on the baseline. In the terms of the XSL-FO specification, this is the *alphabetic* baseline.

Figure 8-30: The baseline



The height of the font above the baseline is the *ascender*. The height of the font below the baseline is the *descender*. Adding the ascender and descender values for the font (not for individual characters) gives the font size. The *leading* is the space above and below the characters, and is the difference between the line-height and the font-size.

The XSL-FO specification refers to the ascender value as the *text-altitude* and the descender as the *text-depth*. Together these two values add up to the *allocation rectangle* height. In these terms:

leading = (line-height - text-altitude - text-depth)

so

1/2 leading = (line-height - text-altitude - text-depth) / 2

By default the line height is 1.2em. The em unit is proportional to the size of the current font, so as the font size increases so does the line height. This can be changed by setting the Settings.LineHeightNormal value. For instance to make the line height larger and so space text out more vertically you could use the code in Figure 8-31.

Figure 8-31: Changing the default line height

```
FODocument doc = new FODocument();
doc.Settings.LineHeightNormal = "1.4em";
```

8.14.1 The effect of subscript and superscript text on line spacing

When calculating the largest characters on this line, we really mean those whose ascender and descender values are greatest (i.e. futherest from the baseline). When making this calculation, the value of the line-height-shift-adjustment attribute is considered. If text is a subscript or superscript and so has a baseline-shift value which changes its position vertically, this will also change its effective ascender and descender values. If line-height-shift-adjustment = "consider-shifts" (the default value) then the baseline-shift amount is taken into account when working out the greatest ascender and descender. If line-height-shift-adjustment = "disregard-shifts" then the effect of the baseline-shift is ignored. Setting line-height-shift-adjustment = "disregard-shifts" makes lines stay the same distance apart regardless of subscript and superscript elements.

The effect line-height-shift-adjustment is shown in Figure 8-32; the first two lines are in a block which has line-height-shift-adjustment= "consider-shifts" and so are further apart than the second two which are in a block which has line-height-shift-adjustment = "disregard-shifts":

Figure 8-32: Effect of disregard-shifts

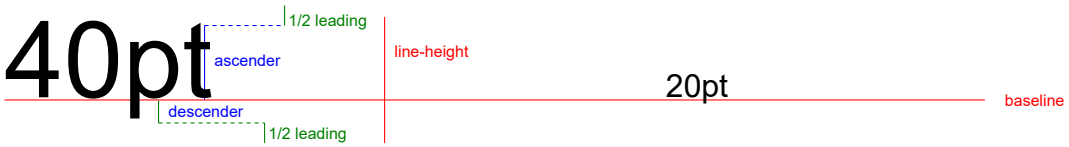
Specifies a string on which content of cells in a table column will align (see the section, in the CSS2 Recommendation[2]).

Specifies a string on which content of cells in a table column will align (see the section, in the CSS2 Recommendation[2]).

8.14.2 The baseline

The baseline is below the top of the text block a distance equal to 1/2 leading + max(ascender), which places the baseline in the same place for all text elements. This means that normally text rests on the same baseline regardless of the font size, as shown in Figure 8-33.

Figure 8-33: Text on the baseline

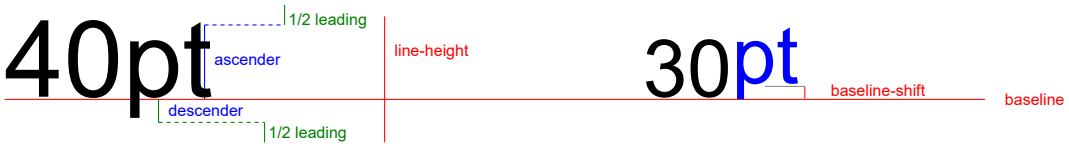


8.14.3 Subscript and superscript

Subscripted and superscripted text is created by using the baseline-shift attribute on an inline element.

The effect of the baseline shift is shown in Figure 8-34, where the "pt" characters are in an inline element with baseline-shift = "5pt".

Figure 8-34: Effect of baseline shift



The FO to move a word above the current baseline by 5 points is shown in Figure 8-35 with the resulting output appears in Figure 8-36.

Figure 8-35: FO showing baseline-shift

```
<block
  hello
  <inline color="red" baseline-shift="5pt">
    super
  </inline>
</block>
```

Figure 8-36: Output from the above FO

```
hello super
```

Font files contain default baseline shift values for superscripted and subscripted text. Rather than specifying baseline-shift="5pt", you can use the values "super" and "sub". The FO to move a word above the current baseline by the default amount for the current font is shown in Figure 8-37 with the resulting output in Figure 8-38. Using the "sub" and "super" values is preferable to using specific measurements because it means (a) if you change the font size of the paragraph you do not have to change all the baseline-shift values and (b) you get the baseline sift the font designer intended.

Figure 8-37: Using the default superscript

```
<block
  hello
  <inline color="red" baseline-shift="super">
    super
  </inline>
</block>
```

Figure 8-38: Output from the above FO

```
    hello super
```

8.15 Line stacking strategies

XSL-FO uses the line-stacking-strategy attribute to determine how lines are stacked vertically on a page. The default value of this attribute is "max-height". When the "max-height" strategy is used the height of a line depends on the height of the characters or images on that line. The information which follows assumes that this default value is used. The other values for line-stacking-strategy, namely "font-height" and "line-height" will produce different results, since the height of the line using these strategies does not change when the content of the line changes.

The leading value is calculated from the line-height and font-size specified for the block element which contains the text. It is constant for the whole block and is not affected by other values specified on contained within the block.

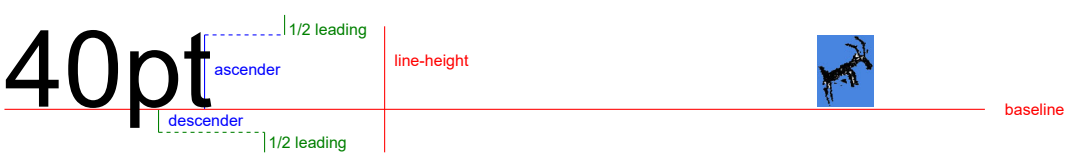
The height the line is calculated using "largest" characters found on the line, i.e. the sum of the max(ascender) and max(descender) values.

8.16 Aligning images

An inline element such as external-graphic is treated similarly to a text element. The height of the image is used as the ascender value. The descender value is zero.

This means that by default an image will be positioned on the baseline, as shown in Figure 8-39 .

Figure 8-39: An image on the baseline



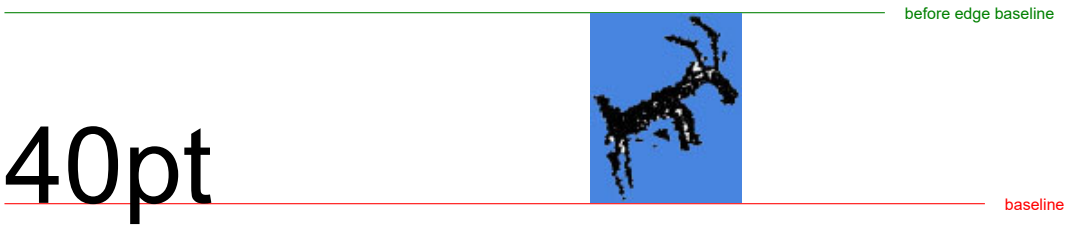
A large image will contribute a large ascender value to the baseline placement calculation, but will still sit on that baseline as shown in Figure 8-40.

Figure 8-40: Large image on baseline

8.16.1 The before-edge baseline

By default an element has an alignment-baseline value of "baseline" and so sits on the baseline shown in the above diagrams. For a given line, the largest thing on that line which has alignment-baseline = "baseline" establishes the position of the *before edge baseline*. This is shown in Figure 8-41.

Figure 8-41: Image aligned to before-edge baseline



To align another object with the before edge baseline, either set vertical-align = "top" or alignment-baseline = "before-edge".

Figure 8-42 shows a second smaller image with default alignment, which positions it on the baseline.

Figure 8-42: Differently aligned images

By specifying vertical-align="top" on the external-graphic for the second image, we can align this image to the before edge baseline and get the layout shown in Figure 8-43.

Figure 8-43: Two images aligned using vertical-align

If all the elements on the line have vertical-align = "top", then the *before edge baseline* cannot be calculated, so the *text before edge baseline* is used. This is the top of the ascender for the font specified for the block which contains the elements.

Chapter 9

Fonts

Ibex supports TrueType and Type 1 (Postscript) fonts. Font information is read from the registry at runtime, no configuration of fonts is required.

Information on how to list the fonts which Ibex can use can be found in the usage chapter on page 32.

Ibex reads the registry to see which fonts are available. Specifically the entries under "HKLM\software\microsoft\windows nt\currentversion\fonts" list available fonts, and those under "HKLM\software\microsoft\windows nt\currentversion\fontsubstitutes" list translations from font names to existing fonts. Any of the font names listed in these two places can be used.

In addition Type 1 font names are read from "HKLM\software\microsoft\windows nt\currentversion\type 1 installer\type 1 fonts". Only Type 1 fonts that come as a PFM (metrics) and PFB (binary) pair of files are supported.

9.1 How Ibex uses fonts

Your FO file contains a series of letters. Each of which is stored in the file as a one or two byte *code point* such as 65 for 'A' or 0x8226 for the bullet character.

Ibex reads the TrueType or Type 1 font file and looks in the font to see if the font supports that particular code point. If it does, then the font maps that code point to a glyph, which is what gets displayed.

Not all fonts support all code points. For example arial.ttf is 370 KB in size, whereas arialuni.ttf is 23,000 KB, because arialuni has glyphs for a many more code points that arial.ttf.

Not all fonts map a code point to the same glyph. Some fonts map code points they do not support to a glyph such as the square box one.

Chapter 10

Floats

The float element can be used to position an image or other elements to the side or top of the page and cause text to flow around that image.

The paragraph in Figure 10-1 uses two float elements to make the image appear on the left and right sides, with the text flowing around the images below them.

Figure 10-1:
Left and right floats

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis in dolor quis lectus cursus condimentum. Vestibulum id nunc vitae dui semper ultrices. Mauris a mi. Phasellus eu lacus. Pellentesque eu ligula mattis odio faucibus faucibus. Aliquam sit amet nunc laoreet tellus ullamcorper malesuada. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Cras nec mauris. Proin cursus tincidunt leo. Maecenas metus lacus, imperdiet fermentum, blandit at, sollicitudin eu, sem. Duis elementum libero vitae lacus. Curabitur justo. Aliquam erat volutpat. Maecenas nec nulla in massa consectetuer volutpat. Aenean turpis nisl, rutrum a, posuere sit amet, varius in, enim. Praesent risus.

Nam volutpat enim eget neque. Maecenas a dui ac felis nonummy sollicitudin. Proin iaculis. Vestibulum in eros sit amet libero mollis convallis. nunc laoreet tellus ullamcorper malesuada. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Cras nec mauris. Proin cursus tincidunt leo. Maecenas metus lacus, imperdiet fermentum, blandit at, sollicitudin eu, sem. Duis elementum libero vitae lacus. Curabitur justo. Aliquam erat volutpat. Maecenas nec nulla in massa consectetuer volutpat. Aenean turpis nisl, rutrum a, posuere sit amet, varius in, enim. Praesent risus. Nam volutpat enim eget neque. Maecenas a dui ac felis nonummy sollicitudin. Proin iaculis. Vestibulum in eros sit amet libero mollis convallis.

Figure 10-2: FO for float example

```
<block font-size="1.0em" text-align="justify">
    <float float="left">
      <block-container inline-progression-dimension="2.5cm">
        <block text-align="center">
          <external-graphic src="url(ibexorange.jpg)" content-width="50%"
               padding="3pt"/>
        </block>
      </block-container>
    </float>
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis in dolor quis lectus
cursus condimentum. Vestibulum id nunc vitae
dui semper ultrices. Mauris a mi. Phasellus eu lacus. Pellentesque eu ligula mattis
odio faucibus faucibus. Aliquam sit amet
    <float float="right">
      <block-container inline-progression-dimension="2.5cm">
        <block text-align="center">
          <external-graphic src="url(ibexorange.jpg)" content-width="50%"
              padding="3pt"/>
        </block>
      </block-container>
    </float>
nunc laoreet tellus ullamcorper malesuada. Vestibulum ante ipsum primis in faucibus
orci luctus et ultrices posuere cubilia Curae; Cras nec mauris. Proin cursus
tincidunt leo. Maecenas metus lacus, imperdiet fermentum, blandit at,
sollicitudin eu, sem. Duis elementum libero vitae lacus. Curabitur justo. Aliquam
erat volutpat. Maecenas nec nulla in massa consectetuer volutpat. Aenean turpis
nisl, rutrum a, posuere sit amet, varius in, enim. Praesent risus. Nam volutpat
enim eget neque. Maecenas a dui ac felis nonummy sollicitudin. Proin iaculis.
Vestibulum in eros sit amet libero mollis convallis. nunc laoreet tellus
ullamcorper malesuada. Vestibulum ante ipsum primis in faucibus orci luctus et
ultrices posuere cubilia Curae; Cras nec mauris. Proin cursus tincidunt leo.
Maecenas metus lacus,imperdiet fermentum, blandit at, sollicitudin eu, sem.
Duis elementum libero vitae lacus. Curabitur justo. Aliquam erat volutpat.
Maecenas nec nulla in massa consectetuer volutpat. Aenean turpis nisl, rutrum a,
posuere sit amet, varius in, enim. Praesent risus. Nam volutpat enim eget neque.
Maecenas a dui ac felis nonummy sollicitudin. Proin iaculis. Vestibulum in eros
sit amet libero mollis convallis.
</block>
```

This effect is achieved by having a block which contains the text and two float elements. The float elements in turn contain a block-container element which has a inline-progression-dimension attribute defining the width of the float area. Any elements inside the block-container will be in the float area. If a block-container is not used within the float and the width of the float cannot be determined, a default configurable value is used.

The FO for creating the above is show in Figure 10-2. Figure 10-2 is itself contained inside a float with float = "before", which will make it appear at the top of the following page. This technique is used in this manual when we do not want a large example to be split across page breaks or to interrupt the content. When a float has float = "before", its position in the PDF file is not the same as its position in the FO file, in that it will be moved to the top of the next page and the blocks before and after the float will flow as if the float was not there.

The side on which the float occurs is specified using the float attribute. This can be set to "left" or "right" to position the float at the side of the page. It can also be set to "before" to position the float at the start of the next page.

Side floats (with float = "left" or float = "right") are closely tied to the block which contains the float element. If the float element does not fit on the page, then the float and some or all of the containing block will be moved to the following page. This ensures that the text in the block does not refer to (for example) an image in the float which is not on the same page as the text.

10.1 How the float width is calculated

Ibex looks at the content of the float element to try and determine how wide the float should be. If a block-container element is found directly below the float element, and this block-container has a width attribute, then that determines the width of the float. If no width can be found, then the width of the float is calculated from by multplying the containing block width by Settings.SideFloatDefaultWidthPercentage, which defaults to 30%.

Chapter 11

Space Handling

XSL-FO defines various attributes for managing whitespace in FO. These allow you to control how linefeeds and whitespace are output.

11.1 Linefeeds and carriage returns

A linefeed is a character with ASCII code 10, or Unicode code point U+000A. This is different to a carriage return which has ASCII code 13. Ibex acts on linefeeds, not on carriage returns. Carriage returns are ignored during PDF creation.

11.2 Default treatment of linefeeds and spaces

By default linefeeds and whitespace preceding and following linefeeds are removed during formatting. Figure 11-1 shows FO which has linefeeds at the end of each line. The resulting output shown in Figure 11-2 has neither linefeeds nor spaces around the text. This is the default treatment for text in XSL-FO.

Figure 11-1: Text with linefeeds and spaces

```
<block margin='2cm'>To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
</block>
```

Figure 11-2: Output with default handling

```
To be, or not to be: that is the question: Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune, Or to take arms against a sea of
troubles,
```

11.3 Using linefeeds to break text

The linefeed-treatment attribute is used to specify the treatment of linefeeds in text. This defaults to "ignore" causing linefeeds to be ignored. We can retain the linefeeds by setting the linefeed-treatment attribute to "preserve". Figure 11-3 shows our example with this attribute added. Figure 11-4 shows the output from this FO.

Figure 11-3: Using linefeed-treatment

```
<block linefeed-treatment="preserve">To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
</block>
```

Figure 11-4: Output with linefeeds preserved

```
To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
```

11.4 Retaining spaces

The white-space-treatment and white-space-collapse attributes are used to control the handling of spaces.

If we want to put some formatted code in our documen, Figure 11-5 shows FO for this.

Figure 11-5: Code example

```
<block linefeed-treatment="preserve">
private void swap_byte( ref byte x, ref byte y ) {
    byte t = x;
    x = y;
    y = t;
}
</block>
```

Setting linefeed-treatment = "preserve" we get the output show in Figure 11-6. We have preserved the linefeeds but all formatting spaces have gone.

Figure 11-6: Code with linefeeds but no spacing

```
private void swap_byte( ref byte x, ref byte y ) {
byte t = x;
x = y;
y = t;
}
```

The white-space-collapse attribute controls whether Ibex compresses adjacent white space characters into a single space. By default any number of adjacent spaces are compressed into a single space.

The white-space-treatment attribute controls whether Ibex ignores spaces adjacent to linefeeds. Setting white-space-treatment = "preserve" makes Ibex retain white space which appears adjacent to linefeeds.

If we set white-space-treatment to "preserve", and white-space-collapse to "false" we will retain the white spaces around the linefeeds. The FO for this is shown in Figure 11-7, and the formatted output is shown in Figure 11-8.

Figure 11-7: FO to retain spaces and linefeeds

```
<block
   linefeed-treatment="preserve"

   white-space-treatment="preserve"
   white-space-collapse="false"
      >
private void swap_byte( ref byte x, ref byte y ) {
    byte t = x;
    x = y;
    y = t;
}
</block>
```

Figure 11-8: Output with linefeeds but no spacing

```
private void swap_byte( ref byte x, ref byte y ) {
    byte t = x;
    x = y;
    y = t;
}
```

11.5 Non-breaking spaces

Unicode defines the code point U+00A0 called NO-BREAK SPACE. This can be used to insert a space between words without allowing a line break to occur between the words. Ibex treats two words separated by a U+00A0 as a single word.

The non-breaking space can be inserted into XML using the   entity.

The example in Figure 11-9 shows a block used in a table header. It contains the three words "Score per 100". The default formatting is shown in Figure 11-10. If we want to move the word "per" to the next line to keep it with the "100", we replace the space between "per" and "100" with a non-breaking space. This will prevent Ibex breaking the line between the "per" and "100" words.

Figure 11-11 shows the FO with a non-breaking space and Figure 11-12 shows the resulting output.

Figure 11-9: FO without a non-breaking space

```
<block-container width="2.8cm">
  <block border="1pt solid black"
     padding="3pt" text-align="center">
    Score per 100
  </block>
</block-container>
```

Figure 11-10: Output without a non-breaking space

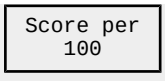
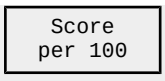


Figure 11-11: FO with non-breaking space

```
<fo:block-container width="2.8cm">
  <fo:block border="1pt solid black"
       padding="3pt" text-align="center">
    Score per 100
  </fo:block>
</fo:block-container>
```

Figure 11-12: Output with a non-breaking space



Chapter 12

Colors

XSL-FO defines various attributes for managing color. By default a block is displayed with the foreground color (that is the text) being black and the background color being white.

Colors are most commonly expressed using the RGB color scheme, where there are three parts to a color: red, green and blue. Ibex also supports the CMYK color scheme commonly used in the printing industry.

12.1 Text color

The color of text is specified using the color attribute. Figure 12-1 shows a simple example of some FO to make text blue. The output is shown in Figure 12-2.

Figure 12-1: FO for blue text

```
<block color="blue">
  To be, or not to be: that is the question:
</block>
```

Figure 12-2: Blue text

```
The resulting text will be blue like this
```

12.2 Background color

The background color of any element is defined using the background-color attribute. Figure 12-3 shows FO for a block with a gray background. The output from this is shown in Figure 12-4.

Figure 12-3: FO for gray background

```
<block background-color="gray">
    To be, or not to be: that is the question:
</block>
```

Figure 12-4: Gray background

```
The resulting text will have a gray background like this
```

12.3 Available colors

The value used for the color and background-color attributes can be a predefined color such as "red", an RGB color defined using a hex value such as "#eeffdd" or a CMYK color.

12.3.1 Predefined colors

XSL-FO uses the list of colors defined for HTML 4.0, which contains these values:

aqua	ibex
black	ibex
blue	ibex
fuchsia	ibex
gray	ibex
green	ibex
lime	ibex
maroon	ibex
navy	ibex
olive	ibex
purple	ibex
red	ibex
silver	ibex
teal	ibex
white	
yellow	ibex

12.3.2 Hexadecimal RGB colors

A color can be defined as a string of six digits preceded by a "#" character. The first two digits define the red component of the color, in a range from 0 to 255. The second two digits define the green component and the last two digits define the blue component. This is the same scheme for defining colors as is used in HTML.

12.3.3 CMYK colors

CMYK colors are four-part colors using values for cyan, magenta, yellow and black respectively. The CMYK system is *subtractive*, meaning that higher values mean less color, unlike RGB where higher values mean more color. CMYK colors are used in the printing industry to define a color which will appear the same across all media. Typically a color defined using RGB will not appear exactly the same on the screen and on a

printed page, or even on two different computer screens. CMYK colors are used to ensure that colors are the same on screen and on the printed page.

PDF files are usually created with single color scheme. You would not usually mix CMYK and RGB colors in one document. Note that when creating a CMYK PDF file any images included in the document should be in CMYK format.

A CMYK color is defined using the rgb-icc() function. This takes eight parameters. The first three define the red, green and blue components of a fallback RGB color, the fourth defines the color profile name, and the last four define the four parts of the CMYK color. The color profile must have been declared in the declarations formatting object using a color-profile element.

Figure 12-5 shows an example of the rgb-icc() function.

Figure 12-5: The rgb-icc function

```
<block color="rgb-icc( 0, 0, 0, cmyk, 0.7,0.3,0.3,0.4 )">
   in cmyk .5,.5,.5,0
</block>
```

In Figure 12-5 the three components of the fallback RGB color are zero. This is normal because we are creating a CMYK PDF file and will not be using any fallback RGB colors. The color profile name is "cmyk". Ibex requires that the color profile name be "cmyk" when creating a CMYK color.

A complete document using the CMYK color space is shown in Figure 12-6. This shows how to use the declarations and color-profile elements to define a color profile.

Figure 12-6: FO for a CMYK PDF file

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="http://www.w3.org/1999/XSL/Format">
 <layout-master-set>
  <simple-page-master master-name="page">
   <region-body margin="1in"
       region-name="body"/>
  </simple-page-master>
 </layout-master-set>

 <declarations>
     <color-profile src="src"
            color-profile-name="cmyk"/>
 </declarations>

 <page-sequence master-reference="page">
  <flow flow-name="body">
   <block color="rgb-icc( 0, 0, 0, cmyk, 0.7,0.3,0.3,0.4 )">
   in cmyk .5,.5,.5,0
   </block>
  </flow>
 </page-sequence>
</root>
```

12.3.4 PDF/X color profiles

Ibex can create PDF files which conform to the PDF/X standard. These files can include embedded color profiles, used to define a common color scheme across different devices.

Color profiles are loaded from files on disk and included in the PDF file. Some color profiles are very large (i.e. > 500k) and can result in large PDF files.

Loading a color profile from a file on disk is an Ibex extension. The name of the color profile file is specified using the color-profile-file-name attribute of the ibex:pdfx element, as shown in Figure 12-7 below.

Figure 12-7: FO for a PDF/X showing the loading of a color profile

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="http://www.w3.org/1999/XSL/Format"
xmlns:ibex="http://www.xmlpdf.com/2003/ibex/Format">
 <layout-master-set>
  <simple-page-master master-name="page" page-width="20cm">
   <region-body region-name="body" margin="3cm" reference-orientation='0'/>
  </simple-page-master>
 </layout-master-set>

 <ibex:pdfx color-profile-file-name="colorprofiles\USWebCoatedSWOP.icc"
output-condition="TR001 SWOP/CGATS"/>

 <page-sequence master-reference="page">
  <flow flow-name="body">

   <block font="10pt arial">
    hello world
   </block>
 </flow>


 </page-sequence>
</root>
```

Chapter 13

Lists

Lists are created using the list-block element. A list-block in XSL-FO is an area of content divided into two columns.

A simple list-block is shown in Figure 13-1. The list created by this FO is shown in Figure 13-2.

Figure 13-1: FO for a list

```
<list-block provisional-distance-between-starts=".5cm"
   provisional-label-separation="0.1cm">
  <list-item>
     <list-item-label end-indent="label-end()">
        <block font='8pt arial'>&#x25CF;</block>
     </list-item-label>
     <list-item-body start-indent="body-start()">
          <block>
              item one
          </block>
      </list-item-body>
  </list-item>
  <list-item>
     <list-item-label end-indent="label-end()">
        <block font='8pt arial'>&#x25CF;</block>
     </list-item-label>
     <list-item-body start-indent="body-start()">
          <block>
              item two
          </block>
      </list-item-body>
 </list-item>
</list-block>
```

Figure 13-2: A list

- item one
- item two

Features of lists include:

- the list-block is a block-level element which contains the whole list.
- the provisional-distance-between-starts attribute on the list-block defines the distance between the start of the label and the start of the body.
- the provisional-label-separation attribute on the list-block defines the size of the gap between the end of the label and the start of the body. This gap is created by reducing the size of the label. For example, if provisional-distance-between-starts is 5cm and the provisional-label-separation is 1cm, then the start edges of the label and body will be 5cm apart, and the label will be 4cm (5cm - 1cm) wide.

- each item in the list is contained in a list-item element.
- each list-item must contain both a list-item-label and a list-item-body. The list-item-label must come first.
- the list-item-label should have the end-indent attribute set to "label-end()". This is a special function which returns a value derived from provisional-distance-between-starts and provisional-label-separation.
- the list-item-body should have the start-indent attribute set to "body-start()". This is a special function which returns a value derived from provisional-distance-between-starts and provisional-label-separation.
- both the list-item-label and list-item-body contain one or more block-level elements, so a list-item-label or list-item-body can contain other block-level elements such as block, table and list-block.

13.1 Bulleted lists

The example in Figure 13-1 also shows how to insert a Unicode character into the FO, using the syntax ●.

This table shows some common bullet types for lists:

Unicode	Result
&#x2022;	•
&#x2023;	‣
&#x25CF;	●
&#x25CB;	○
&#x25A0;	■
&#x25A1;	□
&#x25C6;	◆
&#x25C7;	◇

Note that what is displayed in the document depends on whether the font you are using contains the specified character. If the font does not contain the specified character you will see a warning message like the one in Figure 13-3.

```
warning:380 No glyph index found for code point 2023 in font ArialMT
```

Figure 13-3: Error message if bullet not in font

Chapter 14

Tables

A table in XSL-FO is an area of content divided into rows and columns. A table is created with the table element.

A FO for a simple table is shown in Figure 14-1 and the output it creates is shown in Figure 14-2. This shows the basic structure of a table element containing table-body, table-row and table-cell elements.

Figure 14-1: FO for a simple 2 x 2 table

```
<table>
  <table-body>
    <table-row>
      <table-cell border="1pt solid blue" padding="2pt">
            <block>row 1 column 1</block>
      </table-cell>
      <table-cell border="1pt solid blue" padding="2pt">
            <block>row 1 column 2</block>
      </table-cell>
    </table-row>
    <table-row>
       <table-cell border="1pt solid blue" padding="2pt">
             <block>row 2 column 1</block>
       </table-cell>
       <table-cell border="1pt solid blue" padding="2pt">
             <block>row 2 column 2</block>
       </table-cell>
    </table-row>
</table-body>
```

Figure 14-2: The simple 2 x 2 table

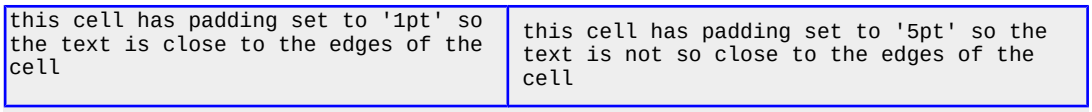
row 1 column 1	row 1 column 2
row 2 column 1	row 2 column 2

The padding and border attributes are not inherited from containing elements, so are best defined on the table-cell elements.

14.1 Cell padding

Padding is the amount of space that appears between the inside edge of the border of a cell and the outside edge of the content of the cell. Padding is specified by the padding attribute. The default amount of padding is '0pt'. Figure 14-3 shows a table with two cells. The first cell has padding="1pt" and the second has padding="5pt". Padding is almost always used to avoid having the content too close to the cell borders.

Figure 14-3: FO showing cells with different padding



The padding attribute sets padding for all four sides of the cell. Individual sides can be set using the padding-left, padding-right, padding-top and padding-bottom attributes.

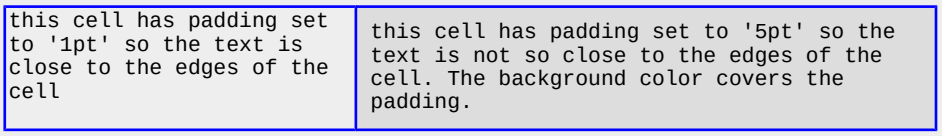
The padding attribute also supports a shorthand format where:

- if one value is specified (padding="2pt") the same value will apply to all four sides;
- if two values are specified (padding="2pt 3pt") the first value will apply to the top and bottom edges, the second value to the left and right edges;
- if three values are specified (padding="2pt 3pt 1pt") the first value will apply to the top edge, the second to the left and right edges, and the third to bottom edge;
- if four values are specified (padding="2pt 3pt 1pt 0pt") these will apply to top, right, bottom and left edges in that order.

14.2 Cell background color

The background color of a cell is specified using the background-color attribute. This supports the same predefined colors as CSS and the use of hex values such as "#33ffcc". The background color of the cell extends to the inside edge of the border, which means that the area specified by the padding attribute is colored by the background color. This is shown in Figure 14-4 where the second cell has the attribute background-color = "#dddddd".

Figure 14-4: Cell with background color set



If you do not want the background to extend to the edge of the padding, specify the background-color attribute on the contents of the cell (i.e. the block elements) rather than on the table-cell. An example FO for this is shown in Figure 14-5 and the resulting output appears in Figure 14-6.

Figure 14-5: FO setting the background color on a block

```
<table>
  <table-body>
    <table-row>
      <table-cell border='1pt solid blue' padding='1pt'>
            <block>
            this cell has padding set to '1pt' so the text is close to the edges of
the cell
            </block>
      </table-cell>
      <table-cell border='1pt solid blue' padding='5pt'
            background-color='#dddddd'>
            <block background-color='#dddddd'>
            this cell has padding set to '5pt' so the text is not so close to the
edges of the cell
            </block>
      </table-cell>
    </table-row>
</table-body>
```

Figure 14-6: Cell with background color on the block element

this cell has padding set to '1pt' so the text is close to the edges of the cell	this cell has padding set to '5pt' so the text is not so close to the edges of the cell

14.3 Cell background images

An image can be used as the background to a cell by specifying the background-image element, as shown in Figure 14-7. This produces the output shown in Figure 14-8.

Figure 14-7: FO for using an image as a cell background

```
<table>
  <table-body>
    <table-row>
      <table-cell border='1pt solid blue' padding='1pt'>
            <block>
            this cell has padding set to '1pt' so the text is close to the edges of
the cell
            </block>
      </table-cell>
      <table-cell border='1pt solid blue' padding='5pt'
            background-image='url(ibex.jpg)'>
            <block>
            this cell has a background image
            </block>
      </table-cell>
    </table-row>
</table-body>
```

Figure 14-8: Cell with image background

this cell has padding set to '1pt' so the text is close to the edges of the cell	this cell has a background image

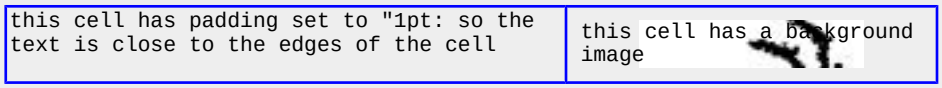
As the above example shows, by default the image will be repeated if it is less than the width of the cell. This can be changed using the background-repeat attribute. If this is set to "no-repeat" the output changes to that shown in Figure 14-9.

Figure 14-9: Using background-repeat = 'no-repeat'

this cell has padding set to '1pt' so the text is close to the edges of the cell	this cell has a background image

The background image can be positioned in the cell using the background-position-horizontal and background-position-vertical attributes. The cell in Figure 14-10 example has background-position-horizontal set to "50%".

Figure 14-10: Centering the background image



14.4 Implicit and explicit rows

Usually FO files use the table-row element to define which cells are in which rows, as shown in Figure 14-11.

Figure 14-11: Tables with cells contained in rows

```
<table>
  <table-body>
    <table-row>
      <table-cell border="1pt solid blue" padding="2pt">
            <block>row 1 column 1</block>
      </table-cell>
      <table-cell border="1pt solid blue" padding="2pt">
            <block>row 1 column 2</block>
      </table-cell>
    </table-row>
    <table-row>
       <table-cell border="1pt solid blue" padding="2pt">
             <block>row 2 column 1</block>
       </table-cell>
       <table-cell border="1pt solid blue" padding="2pt">
             <block>row 2 column 2</block>
       </table-cell>
    </table-row>
</table-body>
```

It is possible to dispense with the table-row element and have the table-body contain table-cell elements directly. In this case any cell can have the ends-row attribute set to "true", which causes a new row to be started containing the next cell. This approach is sometimes easier to use when generating the FO using XSLT.

Figure 14-12 shows what the above FO would look like if we changed it to use implicit rows. The output from this appears in Figure 14-13 below.

Figure 14-12: FO for a table with implicit rows

```
<table>
  <table-body>
      <table-cell border='1pt solid blue' padding='2pt'>
            <block>row 1 column 1</block>
      </table-cell>
      <table-cell border='1pt solid blue' padding='2pt'
            ends-row='true'>
            <block>row 1 column 2</block>
      </table-cell>
       <table-cell border='1pt solid blue' padding='2pt'>
             <block>row 2 column 1</block>
       </table-cell>
       <table-cell border='1pt solid blue' padding='2pt'>
             <block>row 2 column 2</block>
       </table-cell>
</table-body>
```

Figure 14-13: Table with implicit rows

row 1 column 1	row 1 column 2
row 2 column 1	row 2 column 2

14.5 Table columns

The table-column element is used to set the column width and other characteristics of a table column. A table-column element has an associated column number which determines which column the table-column element refers to. This column number is either implied (with the first table-column element applying to the first column, the second to the next etc.), or explicitly set using the column-number attribute.

A single table-column element can be used to define the style of multiple columns by using the number-columns-spanned attribute.

Figure 14-14 shows the FO for a table with two table-column elements, which apply to the first and second columns. In this case they set the column widths (to 30% and 70%), and the give the second column a shaded background. The output created from the FO appears in Figure 14-15.

Figure 14-14: FO using table-column elements

```
<table>
    <table-column column-width='30%'/>
      <table-column column-width='70%'
         background-color='#dddddd'/>
  <table-body>
    <table-row>
      <table-cell border='1pt solid blue' padding='2pt'>
            <block>row 1 column 1</block>
      </table-cell>
      <table-cell border='1pt solid blue' padding='2pt'>
            <block>row 1 column 2</block>
      </table-cell>
    </table-row>
    <table-row>
       <table-cell border='1pt solid blue' padding='2pt'>
             <block>row 2 column 1</block>
       </table-cell>
       <table-cell border='1pt solid blue' padding='2pt'>
             <block>row 2 column 2</block>
       </table-cell>
    </table-row>
</table-body>
```

Figure 14-15: Table with defined column widths

row 1 column 1	row 1 column 2
row 2 column 1	row 2 column 2

Some cell attributes such as background color are determined using attributes from the cell itself and from the other elements of the table structure. The order of precedence in determining cell characteristics such as background-color is table-cell, table-row, table-body, table-column and finally table.

14.6 Proportional column widths

Columns can be allocated widths which are proportional to the widths of other columns. For example, if we have two columns and want to give the first column twice the width

of the second, we can specify column widths using the proportional-column-width() function as shown in Figure 14-16. The total of the values used in the proportional-column-width() functions is 3 (2+1), so the first column will gave 2/3 of the width and the second 1/3. The output from this FO appears in Figure 14-17.

Figure 14-16: FO using proportional column widths

```
<table>
  <table-column
        column-width='proportional-column-width(2)'/>
  <table-column
        column-width='proportional-column-width(1)'
        background-color='#dddddd'/>
  <table-body>
    <table-row>
      <table-cell border='1pt solid blue' padding='2pt'>
            <block>row 1 column 1</block>
      </table-cell>
      <table-cell border='1pt solid blue' padding='2pt'>
            <block>row 1 column 2</block>
      </table-cell>
    </table-row>
    <table-row>
       <table-cell border='1pt solid blue' padding='2pt'>
             <block>row 2 column 1</block>
       </table-cell>
       <table-cell border='1pt solid blue' padding='2pt'>
             <block>row 2 column 2</block>
       </table-cell>
    </table-row>
</table-body>
```

Figure 14-17: Output from proportional width example

row 1 column 1	row 1 column 2
row 2 column 1	row 2 column 2

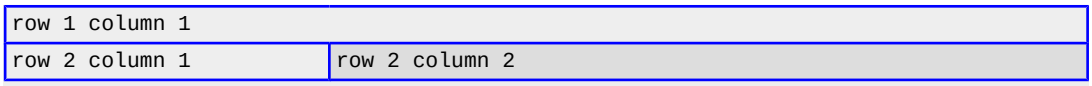
14.7 Spanning columns and rows

The number of columns which a cell spans is set by the number-columns-spanned attribute. An example FO for this is shown in Figure 14-18. In this example the first cell of the first row spans two columns. The output from this FO appears in Figure 14-19.

Figure 14-18: FO for cell spanning 2 columns

```
<table>
  <table-column column-width="30%"/>
  <table-column column-width="70%"
        background-color="#dddddd"/>
  <table-body>
    <table-row>
      <table-cell border="1pt solid blue" padding="2pt"
          number-columns-spanned="2">
            <block>row 1 column 1</block>
      </table-cell>
    </table-row>
    <table-row>
       <table-cell border="1pt solid blue" padding="2pt">
             <block>row 2 column 1</block>
       </table-cell>
       <table-cell border="1pt solid blue" padding="2pt">
             <block>row 2 column 2</block>
       </table-cell>
    </table-row>
</table-body>
```

Figure 14-19: Cell spanning two columns

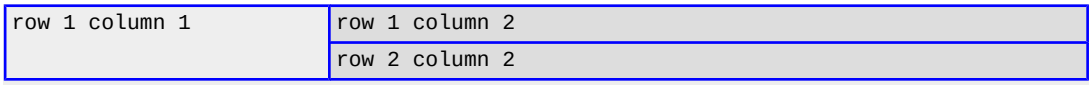


The number of rows which a cell spans is set by the number-rows-spanned attribute. Example FO for this is shown in Figure 14-20. In this example the first cell of the first row spans two rows. The output from this FO appears in Figure 14-21.

Figure 14-20: FO for cell spanning two rows

```
<table>
    <table-column column-width='30%'/>
    <table-column column-width='70%'
         background-color='#dddddd'/>
    <table-body>
    <table-row>
      <table-cell border='1pt solid blue' padding='2pt'
          number-rows-spanned='2'>
            <block>row 1 column 1</block>
      </table-cell>
       <table-cell border='1pt solid blue' padding='2pt'>
             <block>row 1 column 2</block>
       </table-cell>
    </table-row>
    <table-row>
       <table-cell border='1pt solid blue' padding='2pt'>
             <block>row 2 column 2</block>
       </table-cell>
    </table-row>
</table-body>
```

Figure 14-21: Output for cell spanning two rows



14.8 Cell separation

XSL-FO has two ways of processing the borders of adjacent cells depending on the value of the border-collapse attribute on the table.

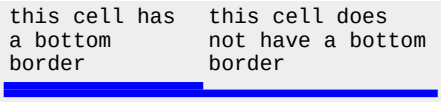
If border-collapse="collapse", which is the default, there is no gap between cells and the borders of adjacent cells are merged (or "collapsed") to get a single border shared by both cells. The rules for combining borders are explained in the XSL-FO specification. Broadly speaking the widest border will be used. This is called the collapsed border model.

If border-collapse="separate" adjacent borders are not merged. A gap can be inserted between adjacent borders using the border-spacing attribute. The border-spacing attribute can have one or two values. If one value is specified (for instance border-spacing="1mm") the vertical and horizontal spacing between cells is set to this value. If two values are specified separated by a space (for instance border-spacing='1mm 3mm') the horizontal separation is set to the first value and the vertical separation is set to the second. This is called the separated border model.

The following examples use a table with one row containing two cells. The first cell has a bottom border, the second does not. The table also has a bottom border.

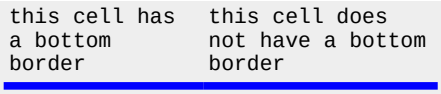
In the separate border model the border from the first cell will be drawn before the border of the table as shown in Figure 14-22.

Figure 14-22: Cells with separate borders



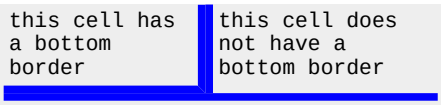
In the collapsed border model the border from the first cell will be merged with the border of the table and a single border will be drawn as shown in Figure 14-23.

Figure 14-23: Cell border collapsed with table border



If we add an inner border to each cell we can see this with the separate model, as shown in Figure 14-24.

Figure 14-24: Separate cell and table borders



With the collapsed border model the border between the two cells will be half the width it is in the separate model, as shown in Figure 14-25.

Figure 14-25: Collapsed borders

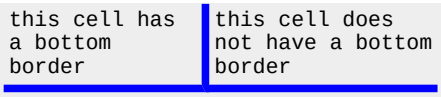


Figure 14-26 shows an example of a table with separate borders. Note how the border-spacing on the previous table sets the space between cells only, not the space between the cell and the table border. This space can be set using padding. If we add padding="2mm" to the table we get the layout shown in Figure 14-27.

Figure 14-26: Table with separate borders

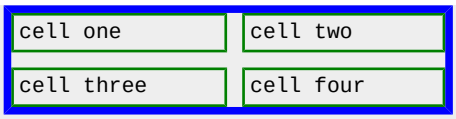
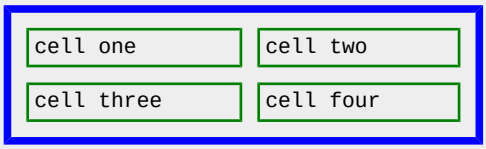


Figure 14-27: Cells separated from the table borders by padding



14.9 Table headers

Table headers are created using the table-header element. The table-header should appear inside the table element after any table-column elements and before any table-body elements. The table-header element is similar in structure to a table-body element in that it contains table-row elements.

This section describes the behavior of table headers which do not change. Headers which can have different content on different pages are described later in this chapter in the section on continuation markers on page 89.

Figure 14-28 shows the FO for a simple table with a one row header and two content rows. The output created by the FO appears in Figure 14-29.

Figure 14-28: Simple table with header

```
<table>
 <table-column column-width="100%"/>
 <table-header>
  <table-row>
    <table-cell border="1pt solid black" padding="5pt">
     <block>Heading</block>
    </table-cell>
  </table-row>
 </table-header>
 <table-body>
  <table-row>
    <table-cell border="1pt solid black" padding="5pt">
     <block>row 1</block>
    </table-cell>
  </table-row>
  <table-row border="1pt solid black" padding="5pt">
    <table-cell>
     <block>row 2</block>
    </table-cell>
  </table-row>
 </table-body>
</table>
```

Figure 14-29: Table with simple header

Heading
row 1
row 2

Table headers are repeated at the top of the table after each page break. This is the default. To prevent the table header appearing on pages after the first, specify table-omit-header-at-break = "true" on the table element.

14.10 Table footers

Table footers are created using the table-footer element. The table-footer should appear inside the table element after any table-column and table-header elements and before any table-body elements. The table-footer element is similar in structure to a table-body element in that it contains table-row elements.

It is a common error to place the table-footer element at the end of the table, after the table-body elements. It must be placed before the table-body elements because Ibex may start rendering the table to PDF before the whole table has been read from the FO file.

This section describes the behavior of table footers which do not change. Footers which can have different content on different pages are described later in this chapter in the section on continuation markers on page 89.

Figure 14-30 shows the FO for a simple table with a one row header and footer and two content rows. The output created by the FO appears in Figure 14-31.

Figure 14-30: FO for simple table with header and footer

```
<table>
 <table-column column-width="100%"/>
 <table-header>
  <table-row>
    <table-cell border="1pt solid black" padding="5pt">
     <block>Heading</block>
    </table-cell>
  </table-row>
 </table-header>
 <table-footer>
  <table-row>
    <table-cell border="1pt solid black" padding="5pt">
     <block>Footer</block>
    </table-cell>
  </table-row>
 </table-footer>
 <table-body>
  <table-row>
    <table-cell border="1pt solid black" padding="5pt">
     <block>row 1</block>
    </table-cell>
  </table-row>
  <table-row border="1pt solid black" padding="5pt">
    <table-cell>
     <block>row 2</block>
    </table-cell>
  </table-row>
 </table-body>
</table>
```

Figure 14-31: Table with simple header and footer

Heading
row 1
row 2
Footer

Table footers are repeated at the bottom of the table before each page break. This is the default. To prevent the table footer appearing on pages other than the last, specify table-omit-footer-at-break = "true" on the table element.

14.11 Behavior at page breaks

14.11.1 Repeating headers

Table headers are repeated at the top of the table after each page break. This is the default. To prevent the table header appearing on pages after the first, specify table-omit-header-at-break = "true" on the table element.

14.11.2 Repeating footers

Table footers are repeated at the bottom of the table before each page break. This is the default. To prevent the table footer appearing on pages other than the last, specify table-omit-footer-at-break = "true" on the table element.

14.11.3 Repeating table borders

Table borders by default do not repeat at a break in the table, so the top border of a table is rendered only on the first page the table is on and the bottom border is rendered only on the last page.

To make the table bottom border repeat at each page break it is necessary to specify border-after-width.conditionality = "retain" on the table element.

To make the table top border repeat at each page break it is necessary to specify border-before-width.conditionality = "retain" on the table element.

14.12 Table continuation markers

Table continuation markers provide a way of dynamically changing the header and footer on a table so that different content can be displayed on different pages. A typical use of this feature is to put the words "continued on next page" in the footer of a table on all pages except the last.

Here we examine how the "continued on next page" requirement can be satisfied using Ibex. The approach taken by XSL-FO has two parts, implemented using the marker and retrieve-table-marker elements. First a retrieve-table-marker element is added to the footer. When the PDF is created this element will be replaced by the contents of one of the marker elements which has the same class name. The marker element which appears in the footer depends on the values of the attributes on the retrieve-table-marker.

The footer for this example is shown in Figure 14-32. As the PDF file is created the contents of the marker element with marker-class-name = "continued" will be located and inserted into the table-footer element. *The content of the marker element must be valid FO elements for their location in the table-footer*. In this example the retrieved elements go directly under the table-footer element, so the elements retrieved must be table-row elements.

Figure 14-32: FO for retrieve-table-marker

```
<table-footer>
  <retrieve-table-marker
        retrieve-class-name="continued"
        retrieve-position-within-table="first-starting"
        retrieve-boundary-within-table="page"/>
</table-footer>
```

Typically, there will be more than one marker element which has the marker-class-name = "continued". If this is not the case then the footer content will never change. The retrieve-position attribute specifies which marker to retrieve. In this example we want the first marker which appears on the page, so we use retrieve-position =

"first-starting-within-page". We also specify retrieve-boundary = "table" so any marker from any part of the table which has been output to PDF can be retrieved. Other options are detailed later in this section.

Conceptually, Ibex looks at every row in the table which has been output to the PDF file (including rows on the current page), collects all the markers associated with each of those rows and selects one to go into the footer. Markers associated with rows which are not on either the current page or prior pages are not considered. It is possible to have a different marker associated with every row in the table. This is useful for situations such as like rendering a running total.

The second part of the process is to define one or more marker elements. In this case our marker elements are associated with table-row elements. The first table-row has a marker element which specifies the "continued on next page" text. The contents of this marker will be retrieved for all pages except the last.

The last row of the table has an empty marker element. The content of this (that is to say no rows) will be what appears in the footer on the last page of the table. The marker from the first row is shown in Figure 14-33 and the marker from the last row is shown in Figure 14-34.

Figure 14-33: FO for marker in the first table row

```
<table-row>
 <marker marker-class-name="continued">
  <table-row>
   <table-cell>
    <block>continued on next page/<block>
   </table-cell>
  </table-row>
 </marker>

 <table-cell>
  <block>row 1 cell 1 /<block>
 </table-cell>
</table-row>
```

Figure 14-34: FO for marker in the last table row

```
<table-row>
 <marker marker-class-name="continued"/>
 <table-cell>
  <block>row (last) cell 1 /<block>
 </table-cell>
<table-row>
```

14.13 Aligning columns at the decimal point

Ibex can align the contents of cells in a column on the decimal point by specifying text-align="." on each fo:table-cell in the column. This can be done explicity on each fo:table-cell, or to make things easier to maintain it can be done by specifying text-align="." on the fo:table-column and text-align="from-table-column" on each fo:table-cell.

Example FO for aligning columns is shown in Figure 14-35 and the resulting output is shown in Figure 14-36.

Figure 14-35: FO for decimal point alignment

```
<table font="10pt arial">
  <table-column column-width="50%" />
  <table-column column-width="50%" text-align="."/>
  <table-body>
    <table-row>
      <table-cell border="1pt solid black" padding="3pt" >
        <block>ibexdls</block>
      </table-cell>
      <table-cell border="1pt solid black" padding="3pt"
         text-align="from-table-column()">
        <block>499.02</block>
      </table-cell>
    </table-row>
    <table-row>
      <table-cell border="1pt solid black" padding="3pt" >
        <block>Total</block>
      </table-cell>
      <table-cell border="1pt solid black" padding="3pt"
         text-align="from-table-column()" font-size="18pt">
        <block>499.00</block>
      </table-cell>
    </table-row>
  </table-body>
</table>
```

Figure 14-36: Output for decimal point alignment

ibexdls	499.02
Total	499.00

Chapter 15

Images

Images are added to the document using either the external-graphic or instream-foreign-object elements. The external-graphic element is used to include a file in JPEG, GIF, TIFF, BMP, SVG or PNG formats. The instream-foreign-object element is used to include an image defined in Scalable Vector Graphics (SVG) format where the image SVG is contained within the FO.

The properties used to format the external-graphic and instream-foreign-object elements are the same.

The size of the image is distinct from the size of the area in which the image is placed.

The height and width attributes on the external-graphic or instream-foreign-object element specify the size of the area into which the graphic will be placed. If these properties are not specified they default to an area large enough to contain the image.

The content-width and content-height attributes control the size of the image. These can be values such as "3cm" or percentages such as "120%". If content-width and content-height not specified the image defaults to the size in pixels specified in the image file itself. This means that if you do not specify any of the above attributes the image will be as large as specified in the image file, and will be placed in an area the same size.

The dots per inch (dpi) value of the image makes a difference to the image size. Two images can have the same dimensions in pixels but appear different sizes in the PDF file. This is because Ibex uses the dpi value to work out the size of the image. An image which is 300 pixels wide and stored at 300 dpi will be 1 inch wide. The same image stored at 100 dpi will be 3 inches wide.

An image is an inline element, so for formatting purposes it can be placed in a sentence surrounded by text and is treated as a single word.

15.1 Image basics

The external-graphic element is used to include an image which is in a file external to the FO file. The name of the file to be included is specified using the src attribute.

The src attribute is called a *uri-specification* and must follow the following rules:

> A sequence of characters that is "url(", followed by optional white space, followed by an optional single quote (') or double quote (") character, followed by a URI reference as defined in [RFC2396], followed by an optional single quote (') or double quote (") character, followed by optional white space, followed by ")". The two quote characters must be the same and must both be present or absent. If the URI reference contains a single quote, the two quote characters must be present and be double quotes.

This means the following are all valid values for the src attribute:

uri(ibex.jpg)

uri("ibex.jpg")

uri('ibex.jpg')

url(http://www.xmlpdf.com/images/download2.gif)

As the src attribute is a URL, an image which exists on a web server can be downloaded automatically by Ibex as the PDF file is created. This is common in real estate and catalog applications and means you do not need to make a copy of an existing image just to get it into the PDF file. The FO shown in Figure 15-1 will fetch the file download2.gif from www.xmlpdf.com. The resulting image is shown in Figure 15-2.

Figure 15-1: FO to insert an image from a web server

```
<block space-before="6pt">
   <external-graphic border="1pt solid black"
     src="url(http://www.xmlpdf.com/images/download2.gif)"
   content-width="200%" content-height="200%"/>
</block>
```

Figure 15-2: Image included from web server

The external-graphic element can be used to include image files in PNG, JPEG, TIFF, BMP and GIF formats. It can also be used to include SVG images held in external files.

The inline-foreign-object is used for loading images from SVG content that is contained inline in the FO. See SVG Images on page 105.

15.2 Making an image fit a specified space

To make an image fit a specified size use the height and width attributes to specify the size of the external-graphic element, and then use the content-width and content-height to fit the image to that size.

For example to fit an image into an area 2cm x 2cm, set the width and height attributes to "2cm" and set the content-width and content-height attributes to "scale-to-fit", as shown in Figure 15-3.

Figure 15-3: Scaling an image

```
<fo:external-graphic src="url(image.jpg)"
                     height="2in" width="2in"
                     content-height="scale-to-fit"
                     content-width="scale-to-fit"/>
```

If you only want the image reduced in size to fit the specified area and do not want it increased in size if it is smaller, specify content-width="scale-down-to-fit". This also applies to content-height.

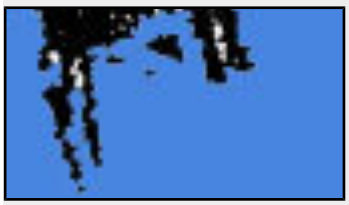
If you only want the image enlarged to fit the specified area and do not want it reduced in size if it is larger, specify content-width="scale-up-to-fit". This also applies to content-height.

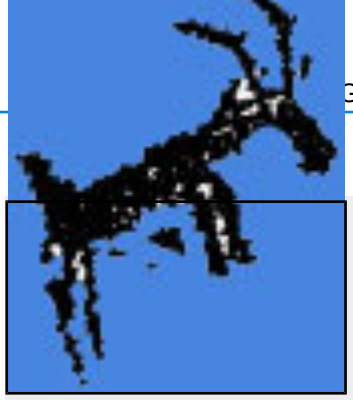
15.3 Clipping

If the image is larger than the area in which it is contained then the image *may* be clipped. Figure 15-4 shows an image at its natural size, based on the pixels and dpi values read from the image file. If we specify the height of the external-graphic element as 2.5cm and specify overflow="hidden", the image will be clipped to this height, as shown in Figure 15-5.

Figure 15-4: Image at natural size

Figure 15-5: Clipped image



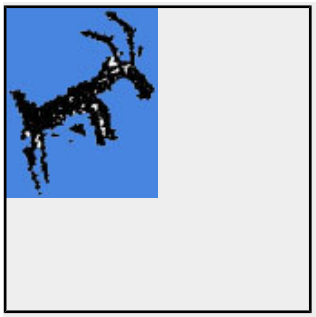


If we specify the height of the external-graphic element as 2.5cm and do not specify overflow="hidden", the image will not be clipped to this height, and will overwrite other content as shown to the right. Because the image is positioned on the same baseline as text, the overflow will be at the top of the area containing the image.

15.4 Image size and alignment

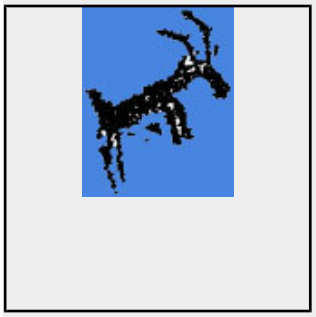
If an image is smaller than the containing area we can control where it appears in that area using the display-align and text-align attributes. The display-align attribute controls the vertical alignment, text-align controls the horizontal alignment. By default the image appears in the top left corner of the inline area created by the external-graphic or instream-foreign-object element, as shown in Figure 15-6.

Figure 15-6: Default alignment of an image



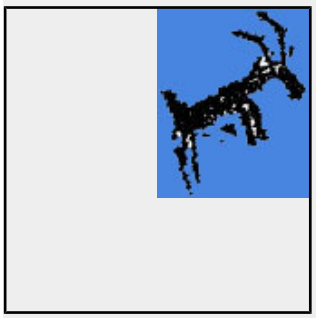
If we specify text-align="center" the image will move to the horizontal center of the inline area, as shown in Figure 15-7.

Figure 15-7: Using text-align = 'center'



If we specify text-align="right" the image will move to the right of the inline area as shown in Figure 15-8.

Figure 15-8: Right aligned image



If we specify text-align="center" and display-align="center" the image will move to the horizontal and vertical center of the inline area, as shown in Figure 15-9.

Figure 15-9: Vertically and horizontally centered image

15.4.1 Leading

Looking at the image in Figure 15-10 you can see a gap between the top of the image and the border. This is the leading, which appears because the image is treated as a text element and sits on the baseline. The amount of leading is derived from the font size, so you can reduce it to zero by setting the font size to zero, by specifying font-size="0pt" on the containing block element. This has been done in Figure 15-11.

Figure 15-10: Image with leading above it

Figure 15-11: Using with leading removed

15.5 Image resolution

The resolution of an image in dots per inch (dpi) can be set using the dpi attribute on the external-graphic element. Setting this attribute overrides the dpi value read from the image file.

Setting the dpi to a lower value than the one specified in the image will result in smaller image of lower quality than the original. This is often done to reduce the size of the image in the PDF file and can result in massive decreases in PDF file size. If you have an image which is stored at 2400 dpi, and your end user will display it on a 96 dpi screen or print it on 600 dpi printer, reducing the image dpi to 600 will not effect the appearance of the image.

Setting the dpi to a value higher than the value read from the image file has no effect.

If for example if we wanted to store an image in the PDF file at 1200 dpi, we would use the FO shown in Figure 15-12.

Figure 15-12: FO to set image dpi

```
<block space-before="6pt">
  <external-graphic border="1pt solid black"
      src="url(http://www.xmlpdf.com/images/download2.gif)"
      content-width="200%" content-height="200%"
      dpi="1200"/>
</block>
```

The dpi attribute is an Ibex extension. It is not part of the XSL-FO standard.

15.6 Image anti-aliasing

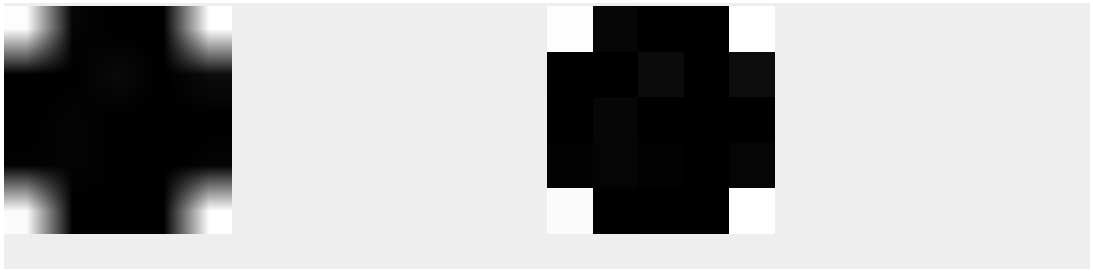
Images are anti-aliased by default. This can be disabled using the ibex:anti-alias attribute as shown in figure Figure 15-13.

Figure 15-13: FO to disable anti-aliasing

```
<block space-before="6pt">
  <external-graphic
      src="url(http://www.xmlpdf.com/images/download2.gif)"
      ibex:anti-alias="false"
      dpi="1200"/>
</block>
```

Figure 15-14 shows two images, the left right one has anti-aliasing disabled so the edges of the image appear more clearly.

Figure 15-14: Images with and without anti-aliasing



The ibex:anti-alias attribute is an Ibex extension. It is not part of the XSL-FO standard.

15.7 Loading an image from memory

Ibex has the facility to load an image which is stored in memory. This permits an application to dynamically generate an image or to load an image from a database for inclusion in the PDF file.

The image must be passed to Ibex in a byte array or a Stream (from the System.IO namespace).

The image must be given a unique identifier by which it can be retrieved during the PDF creation process. This is done using the addNamedImage() method on the FODocument object. This method takes two parameters; (1) a string which identifies the image and (2) the stream or array which contains the image itself.

For example if we had an image in a byte array called "image" and we wanted to give it the identifier "1029" we would use the code shown in Figure 15-15 to do this.

Figure 15-15: C# code to load an image from memory

```
byte[] image = ... dynamically create

FODocument document = new FODocument();

document.addNamedImage( "1029", image );
```

This must be done before calling generate() to create the PDF file.

Within the FO file the image is retrieved from memory using the syntax shown in Figure 15-16

Figure 15-16: FO to load an image from memory

```
<external-graphic src="url(data:application/ibex-image,1029)"/>
```

The value of the src attribute must be the string "url(data:application/ibex-image," followed by the unique identifier which was passed to addNamedImage().

This syntax for the url attribute conforms to RFC 2397 - The "data" URL scheme (which can be found at http://www.faqs.org/rfcs/rfc2397.html).

15.8 Transparent Images

15.8.1 Transparent GIF images

GIF images which have transparent areas are supported. The FO in Figure 15-17 places the same transparent GIF image on two different backgrounds. The output from this FO is shown in Figure 15-18.

Figure 15-17: FO for transparent image

```
<block background-color="blue">
   <external-graphic src="url(ibm-logo.gif)" content-height="2cm"/>
</block>
<block background-color="black">
   <external-graphic src="url(ibm-logo.gif)" content-height="2cm"/>
</block>
```

Figure 15-18: Transparent GIF images

15.8.2 Transparent PNG images

PNG images which have transparent areas are supported. The FO in Figure 15-19 places a transparent PNG image on a white backgrounds. The output from this FO is shown in Figure 15-20.

Figure 15-19: FO for transparent image

```
<block background-color="white">
   <external-graphic src="url(RedbrushAlpha-0.25.png)" content-height="2cm"/>
</block>
```

Figure 15-20: Transparent PNG image

15.8.3 Transparent images using masking

Ibex can use *image masking* to make some parts of an image appear transparent. This is an extension to the XSL-FO standard.

Image masking works by defining colors from the image which should not be displayed. The PDF viewer will compare each pixel in the image with the mask and not display

pixels which match the mask, effectively making these pixels transparent and so leaving visible the content behind the image.

The image mask is defined using the <ibex:mask> element, which must be contained within an external-graphic element, as shown in Figure 15-21.

Figure 15-21: FO to mask an image

```
<external-graphic src="url(ixsl.jpg)"  z-index='10'>
 <ibex:mask
        red-min="255" red-max="255"
        green-min="255" green-max="255"
        blue-min="255" blue-max="255"/>
</external-graphic>
```

To use the ibex:mask element you must reference the ibex namespace in your FO as shown in Figure 15-22.

Figure 15-22: Referencing the ibex namespace

```
<root
  xmlns:fo="http://www.w3.org/1999/XSL/Format"
  xmlns:ibex="http://www.xmlpdf.com/2003/ibex/Format">
```

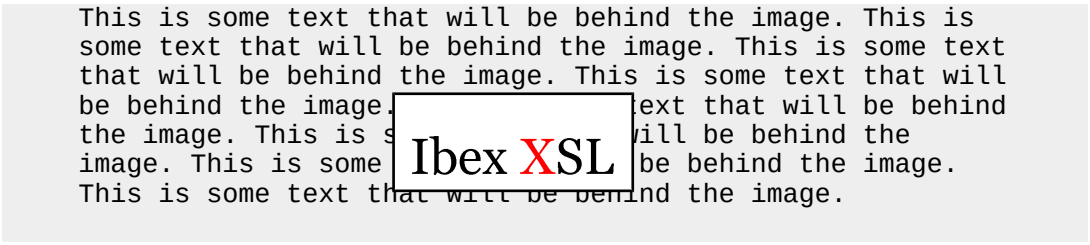
The mask defines the minimum and maximum values for each of the red, green and blue components of an image. A mask using these values is applicable only to images which are in RGB format with 24 bits per pixel.

For CMYK images, attributes called c-min, c-max, m-min, m-max, y-min, y-max, k-min and k-max define the minimum and maximum values for each of the cyan, magenta, yellow and black components of the image.

The image mask shown above causes any pixel which has red=255, green=255 and blue=255 to not be rendered. As a pixel with red, green and blue all equal to 255 is white, this means any white pixels will not be rendered.

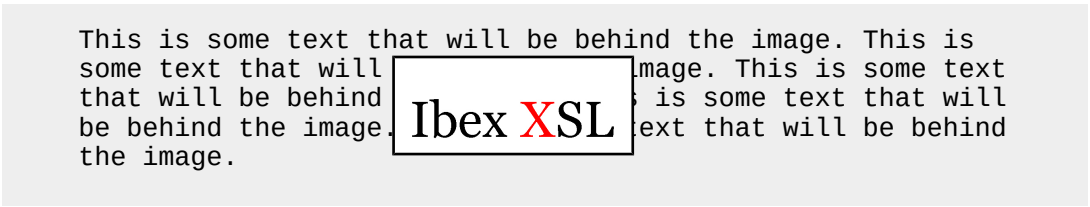
Figure 15-23 shows some text over which we have placed an image with red and black letters on a white background.

Figure 15-23: Image placed over text



If we add a mask to eliminate white pixels the image then appears as shown in Figure 15-24.

Figure 15-24: Image with masking



15.8.4 Transparent Images using SVG

Transparent images can also be implemented by placing a SVG image over the top of other content. This approach uses the vector SVG renderer introduced in Ibex 2.1.2 and is

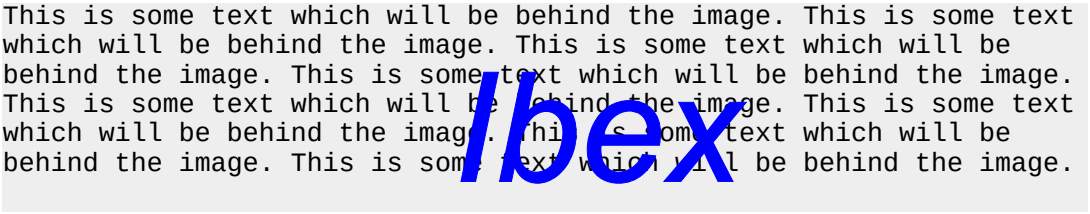
only available when using .NET 1.1 or higher. This is the best approach for transparent images because (a) there is no background on the SVG image so the best clarity is achieved, and (b) SVG uses a vector renderer which creates a smaller PDF file than you would get using a bitmap image.

Figure 15-25 shows the FO to put the word "ibex" over some text. The resulting output is shown in Figure 15-26.

Figure 15-25: FO using SVG to place content over text

```
<block-container>
  <block-container space-before="6pt"
    absolute-position="absolute" top="-1.6cm"
        left="5cm">
    <block>
       <instream-foreign-object z-index="30">
          <svg width="315" height="100"
              xmlns="http://www.w3.org/2000/svg">
           <text  x="30" y="60" fill="blue" stroke="blue"
              font-size="61pt" font-style="italic"
              style="font-family:arial;stroke-width:0.5">
              Ibex
           </text>
          </svg>
       </instream-foreign-object>
    </block>
  </block-container>
</block-container>
```

Figure 15-26: Text overlaid using SVG



15.9 Network credentials used in accessing images

Ibex can retrieve images from HTTP servers as shown in Figure 15-27 below. By default Ibex will do this using the credentials of the process which is creating the PDF file. If Ibex is running in an ASP.NET server then the default configuration is that ASP runs as the ASPNET user. This user does not have permissions to access other servers and so will not be able to retrieve images from other servers.

Figure 15-27: FO to insert an image from an HTTP server

```
<block space-before="6pt">
   <external-graphic border="1pt solid black"
     src="url(http://www.xmlpdf.com/images/download2.gif)"
   content-width="200%" content-height="200%"/>
</block>
```

The FODocument object supports the setNetworkCredentials() method. This method takes 4 parameters, as shown in Figure 15-28.

Figure 15-28: The setNetworkCredentials method

```
public void setNetworkCredentials(
    string prefix,
    string username,
    string password,
    string domain )
```

The parameters are:

prefix — The start of a URL, such as "http://www.xmlpdf.com". Any image URL which starts with this prefix will use these credentials.

username — the username passed to the remote server

password — the password passed to the remote server

domain — the domain name passed to the remote server

Each call to setNetworkCredentials() is passed a prefix which is compared with image URLs to see which set of credentials to use.

For example if your application accesses two HTTP servers using different credentials your code might look like the code in Figure 15-29. Obviously you would get the username and password information from somewhere in your application rather than hard coding them.

Figure 15-29: Setting credentials for different servers

```
FODocument doc = new FODocument()

doc.setNetworkCredentials( "http://www.xmlpdf.com","user1","password1","domain1" );
doc.setNetworkCredentials( "http://www.ibex4.com","user2","password2","domain2" );
```

Internally Ibex uses the System.Net.WebRequest and System.Net.NetworkCredential objects to pass credentials to the remote server. If credentials have been passed to Ibex using the setNetworkCredentials() method a new NetworkCredential object is created when creating the WebRequest object. SO the actual forwarding of the credentials to the remote server is all done by the .NET framework.

Calls to setNetworkCredentials() should be made before the generate() method is called.

15.10 Multi-page TIFF image processing

Ibex has an extension attribute "ibex:page" which is used to specify which page of a multi-page TIFF image should be included in the PDF file.

FO to place the third page of a multi-page TIFF image is shown in Figure 15-30.

Figure 15-30: Specifying the page of a multi-page TIFF image

```
<block>
  <external-graphic src="url('7pages.tif')" ibex:page="3"/>
</block>
```

Chapter 16

Scalable Vector Graphics (SVG) images

Ibex supports the inclusion of Scalable Vector Graphics (SVG) images in the PDF file. SVG images retain their vector format inside the PDF file, so will remain precise under high magnification unlike bitmap images which will be come pixellated.

SVG images are inserted into the document using either the <fo:external-graphic> or <fo:instream-foreign-object> elements. Images can be included inline like this:

```
<fo:block border="1pt solid red">
  <fo:instream-foreign-object>
    <svg xmlns="http://www.w3.org/2000/svg" width="20" height="20">
      <rect width="10" height="10" fill="green"/>
    </svg>
  </fo:instream-foreign-object>
</fo:block>
```

or from an external file like this:

```
<fo:block border="1pt solid red">
  <fo:external-graphic src="url(file.svg)"/>
</fo:block>
```

where the external file contains the SVG image like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" width="20" height="20">
  <rect width="10" height="10" fill="green"/>
</svg>
```

If an image occurs more than once in the document it should be loaded from an external file so that it is only store in the PDF once.

16.2 Namespaces

The SVG image must begin with the <svg> element in the namespace "http://www.w3.org/2000/svg". Images which do not declare the namespace will not be included in the PDF.

16.3 Image size

The size of the image should be specified using the width and height attributes the outer <svg> element. These can be absolute measurements such as "3cm" or scalar values such as "400". Scalar values are assumed to be pixels and are converted to inches based on 1 pixel = 1/96 inch. Percentages cannot be effectively used; the size of the block containing the image is determined from the size of the image, at the time the image is processed the size of the containing block is unknown.

16.4 Summary of supported elements

This section briefly documents the degree to which SVG elements are supported in Ibex. It is not an SVG manual. Information on the SVG specification can be found at http://www.w3.org/TR/SVG11/expanded-toc.html

Animation of SVG elements using javascript is not supported.

16.4.1 <svg>

The <svg> element is used to define the size and shape of the image (using the width and height attributes) and to establish a new coordinate system using the viewBox attribute.

16.4.2 <g>

The <g> element used to move the coordinate system using the transform attribute. Supported transform operations are:

Operation	Effect
translate(x,y)	translate the coordinate system x units horizontally and y units vertically
translate(x)	translate the coordinate system x units horizontally and zero units vertically
matrix(a,b,c,d,e,f)	multiply the current transformation matrix by the one specified
scale(x,y)	scale the coordinate system x units horizontally and y units vertically
rotate(angle)	rotate the coordinate system angle degrees about the origin
rotate(angle,x,y)	rotate the coordinate system angle degrees about the point x,y
skewX(angle)	skew the coordinate system angle degrees along the X axis
skewY(angle)	skew the coordinate system angle degrees units along the Y axis

Multiple transformations can be performed by placing them one after the other in the transform attribute, like this:

```
<g transform="translate(10,20) scale(2,3) rotate(30)">
```

Transforms will be applied in the order in which they appear.

16.4.3 <defs>

The <defs> element is supported as a container for other elements. See <symbol> below for an example.

16.4.4 <desc>

The <desc> element is ignored.

16.4.5 <title>

The <title> element is ignored.

16.4.6 <symbol>

The <symbol> element is supported. The following image shows an example of defininig a system using <symbol> and retrieving it using <use>.

```
<?xml version="1.0" standalone="yes"?>
<svg width="10cm" height="3cm" viewBox="0 0 100 30"
      xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <defs>
    <symbol id="MySymbol" viewBox="0 0 20 20">
      <rect x="1" y="1" width="8" height="8"/>
      <rect x="11" y="1" width="8" height="8"/>
      <rect x="1" y="11" width="8" height="8"/>
      <rect x="11" y="11" width="8" height="8"/>
    </symbol>
  </defs>

  <use x="45" y="10" width="10" height="10" xlink:href="#MySymbol" fill="blue" />

</svg>
```

The <use> element will find the symbol element with id="#MySymbol" and display the content of this element, which should look like this:

16.4.7 <use>

The <use> element is supported, see above for an example. Note that as this element uses the xlink:href attribute it is necessary to declare the xmlns:xlink="http://www.w3.org/1999/xlink" namespace.

16.4.8 <image>

The <image> element is supported. This element embeds an image inside the SVG image. For example this image will display a rectangle and on top of that display the image held in the file "use_symbol.svg":

```
<?xml version="1.0"?>
<svg width="4cm" height="2cm" viewBox="0 0 200 100"
xmlns:xlink="http://www.w3.org/1999/xlink"
      xmlns="http://www.w3.org/2000/svg" version="1.1" preserveAspectRatio="none">

  <rect width="300" height="150" stroke="red" stroke-width="1" fill="silver"/>

  <image x="20" y="20" xlink:href="use_symbol.svg" width="100" height="100"/>

</svg>
```

16.4.9 <switch>

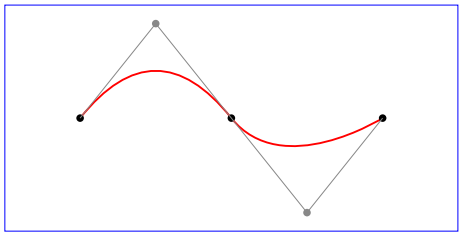
The <switch> element is ignored.

16.4.10 <path>

The <path> element is supported. Internally PDF does not support quadratic Bézier curves so they are converted to cubic Bézier curves. The following SVG draws a simple curve with marked end points:

```
<?xml version="1.0" standalone="no"?>
<svg width="6cm" height="5cm" viewBox="0 0 1200 600"
xmlns="http://www.w3.org/2000/svg">
  <rect x="1" y="1" width="1198" height="598" fill="none" stroke="blue"
stroke-width="1" />

  <path d="M200,300 Q400,50 600,300 T1000,300" fill="none" stroke="red"
stroke-width="5"  />
  <!-- End points -->
  <g fill="black" >
    <circle cx="200" cy="300" r="10"/>
    <circle cx="600" cy="300" r="10"/>
    <circle cx="1000" cy="300" r="10"/>
  </g>
  <!-- Control points and lines from end points to control points -->
  <g fill="#888888" >
    <circle cx="400" cy="50" r="10"/>
    <circle cx="800" cy="550" r="10"/>
  </g>
  <path d="M200,300 L400,50 L600,300
    L800,550 L1000,300"
        fill="none" stroke="#888888" stroke-width="2" />
</svg>
```

The curve looks like this:



16.4.10.1 Path line join shapes

The shape where a path changes direction is set with the stroke-linejoin attribute. Possible values are:

Value	Shape
stroke-linejoin="miter"	
stroke-linejoin="bevel"	
stroke-linejoin="round"	

16.4.11 <style>

The <style> element is currently implemented to some extent in .Net. In .Net the class attribute can be used in conjunction with a style to apply that style to an element.

16.4.12 <rect>

The <rect> element is supported. A simple rectangle can be drawn like this:

```
<svg xmlns="http://www.w3.org/2000/svg" width="400" height="120" >
  <rect x="10" y="10" width="100" height="100" fill="none" stroke="red"/>
</svg>
```

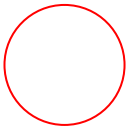
resulting in this image:

16.4.13 <circle>

The <circle> element is supported. A simple circle can be drawn like this:

```
<svg xmlns="http://www.w3.org/2000/svg" width="400" height="120" >
    <circle cx="50" cy="50" r="30" fill="none" stroke="red"/>
</svg>
```

resulting in this image:

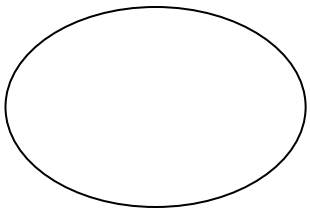


16.4.14 <ellipse>

The <ellipse> element is supported. A simple ellipse can be drawn like this:

```
<svg xmlns="http://www.w3.org/2000/svg" width="400" height="200" >
  <ellipse cx="100" cy="100" rx="75" ry="50" fill="none" stroke="black"/>
</svg>
```

resulting in this image:



16.4.15 <line>

The <line> element is supported. A simple line can be drawn like this:

```
<svg xmlns="http://www.w3.org/2000/svg" width="400" height="400" >
   <line x1="10" y1="10" x2="100" y2="10" stroke="blue" stroke-width="4"/>
</svg>
```

resulting in this image:

16.4.15.1 Line end shapes

The shape of the end of a line is set with the stroke-linecap attribute. Possible values are:

Value	Shape
stroke-linecap="butt"	
stroke-linecap="round"	
stroke-linecap="square"	The end of the line is the same shape as the default stroke-linecap="butt" but projects further beyond the end coordinate.

16.4.15.2 Dashed lines

Dashed lines are supported using the stroke-dasharray attribute. A dashed line can be drawn like this:

```
<svg xmlns="http://www.w3.org/2000/svg" width="400" height="400" >
  <line x1="10" y1="10" x2="100" y2="10" stroke="blue" stroke-width="4"
stroke-dasharray="3 2"/>
</svg>
```

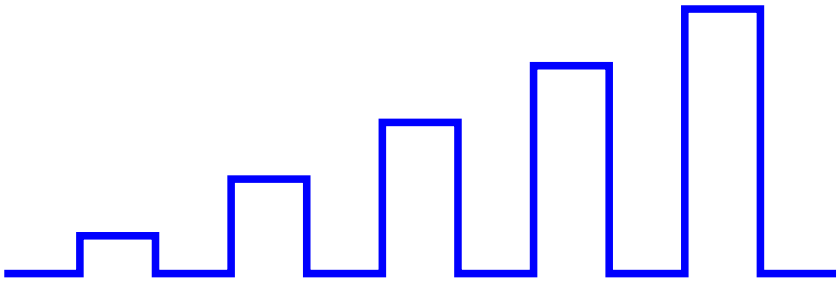
resulting in this image:

16.4.16 <polyline>

The <polyline> element is supported. A simple polyline can be drawn like this:

```
<svg xmlns="http://www.w3.org/2000/svg" width="12cm" height="4cm"
  viewBox="0 0 1200 400">
  <polyline fill="none" stroke="blue" stroke-width="10"
            points="50,375
                    150,375 150,325 250,325 250,375
                    350,375 350,250 450,250 450,375
                    550,375 550,175 650,175 650,375
                    750,375 750,100 850,100 850,375
                    950,375 950,25 1050,25 1050,375
                    1150,375" />

</svg>
```

resulting in this image:

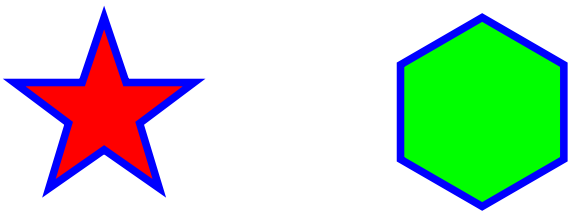


16.4.17 <polygon>

The <polygon> element is supported. A simple polygon can be drawn like this:

```
<svg xmlns="http://www.w3.org/2000/svg" width="12cm" height="4cm"
      viewBox="0 0 1200 400">
        <polygon fill="red" stroke="blue" stroke-width="10"
                    points="350,75  379,161 469,161 397,215
                    423,301 350,250 277,301 303,215
                    231,161 321,161" />
        <polygon fill="lime" stroke="blue" stroke-width="10"
                  points="850,75  958,137.5 958,262.5
                    850,325 742,262.6 742,137.5" />
</svg>
```

resulting in this image:



16.4.18 <text>

The <text> element is supported.

16.4.19 <tspan>

The <tspan> element is not implemented.

16.4.20 <textpath>

The <textpath> element is not implemented.

16.4.21 <pattern>

The <pattern> element is not implemented.

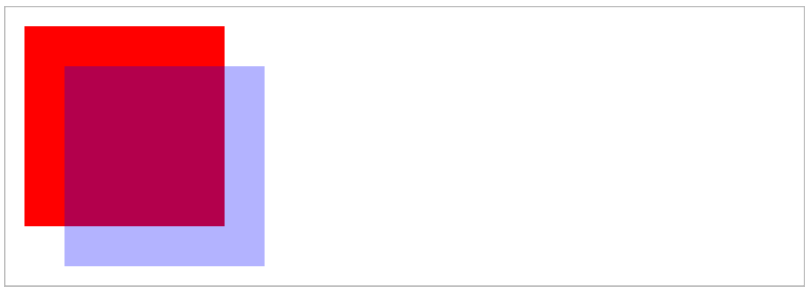
16.5 Opacity

The attributes stroke-opacity and fill-opacity are supported. Using the group opacity attribute to apply opacity to a group of elements is not supported, instead the opacity value is applied as if stroke-opacity and fill-opacity has been specified.

This example shows a transparent blue rectangle drawn over an opaque red rectangle.

```
<svg xmlns="http://www.w3.org/2000/svg" width="400" height="140" >
  <rect width="400" height="140" fill="none" stroke="silver"/>
  <g transform="translate(10,10)">
    <rect width="100" height="100" fill="red"/>
  </g>
  <g transform="translate(30,30)">
    <rect width="100" height="100" fill="blue" stroke-width="1" fill-opacity="0.3" />
  </g>
</svg>
```

resulting in this image:

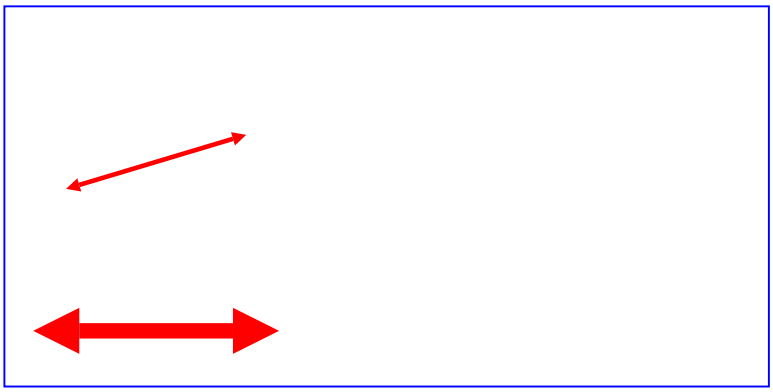


16.6 Markers

Markers are supported at the start and end of <line> and <path> elements. The <marker> element contains a separate drawing which can be reused. This example shows an arrowhead which is drawn at the each end of a line:

```
<?xml version="1.0" standalone="no"?>
<svg width="4in" height="2in"
     viewBox="0 0 4000 2000" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
  <defs>
    <marker id="RedTriangle" viewBox="0 0 10 10" refX="0" refY="5"
      markerUnits="strokeWidth"
      markerWidth="4" markerHeight="3"
      orient="auto" fill="red">
      <path d="M 0 0 L 10 5 L 0 10 z" />
    </marker>
  </defs>
  <rect x="10" y="10" width="3980" height="1980"
       fill="none" stroke="blue" stroke-width="10" />

  <g transform="translate(400,1700) scale(.8)">
    <line x1="0" x2="1000" y1="0" y2="0" stroke="red" stroke-width="100"
marker-end="url(#RedTriangle)"
          marker-start="url(#RedTriangle)"/>
  </g>

  <g transform="translate(400,700) scale(.8)">
    <line x1="0" x2="1000" y1="300" y2="0" stroke="red" stroke-width="30"
marker-end="url(#RedTriangle)"
          marker-start="url(#RedTriangle)"/>
  </g>

</svg>
```

In this example the arrowhead appears once in the SVG, and is rendered four times. Each time it is rendered its rotaton and size are changed to match the rotation and size of the line.



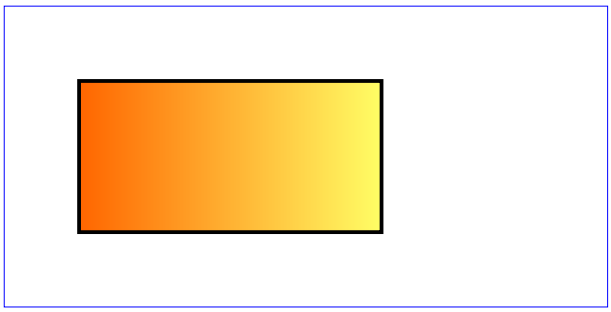
16.7 Linear gradients

Linear gradients are supported. This example produces a gradient from red to yellow horizontally:

```
<?xml version="1.0" standalone="no"?>
<svg width="8cm" height="4cm" viewBox="0 0 800 400" version="1.1"
      xmlns="http://www.w3.org/2000/svg">

  <g>
    <defs>
      <linearGradient id="MyGradient"
           x1="100" x2="500" gradientUnits="userSpaceOnUse">
        <stop offset="5%" stop-color="#F60" />
        <stop offset="95%" stop-color="#FF6" />
      </linearGradient>
    </defs>

    <rect fill="none" stroke="blue"
```

```
          x="1" y="1" width="798" height="398"/>

    <rect fill="url(#MyGradient)" stroke="black" stroke-width="5"
          x="100" y="100" width="600" height="200"/>
  </g>
</svg>
```

producing this image:



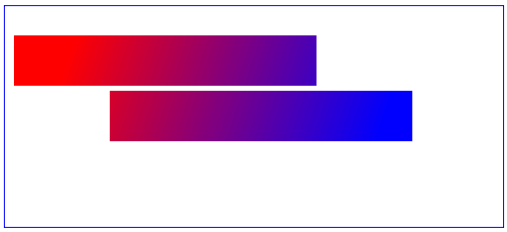
The interpretation of the values specified for the coordinates x1/x2/y1/y2 of the linearGradient element changes depending on value specified for gradientUnits.

When gradientUnits="userSpaceOnUse" the specified values are in "user space", which is the space defined by the prevailing <g> element. The specified coordinates are relative to the prevailing <g> element, so two elements which use the same gradient as their fill color will appear differently if they are placed in different locations on the page.

This SVG image shows rectangles using the same gradient in conjunction with gradientUnits="userSpaceOnUse"

```
<?xml version="1.0" standalone="no"?>
<svg width="8cm" height="3cm" viewBox="0 0 1000 450" version="1.1"
xmlns="http://www.w3.org/2000/svg">
  <g>
    <defs>
      <linearGradient id="linear_userSpaceOnUse" gradientUnits="userSpaceOnUse"
x1="100" y1="100" x2="700" y2="300">
          <stop offset="5%" stop-color="#ff0000" />
          <stop offset="95%" stop-color="#0000ff" />
      </linearGradient>
    </defs>

    <rect fill="none" stroke="blue" x="1" y="1" width="990" height="440"/>

    <g transform="translate(10,50)">
       <rect fill="url(#linear_userSpaceOnUse)" x="10" y="10" width="600"
height="100"/>
       <rect fill="url(#linear_userSpaceOnUse)" x="200" y="120" width="600"
height="100"/>
    </g>

  </g>
</svg>
```

producing this image:

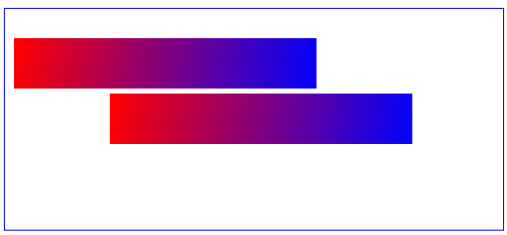


When gradientUnits="objectBoundingBox" the specified values are relative to the bounding box of the element being filled, and should be expressed as fractions of the dimensions of the element being filled. The values for coordinates should be in the range [0..1], so for example specifying x1="0" starts the gradient at the left hand edge of the element being filled, and specifying x1="0.2" starts the gradient at 20% of the width of that element. As the gradient is positioned relative to the element being filled, two element using the same gradient will appear the same regardless of the position of the element.

This SVG image shows rectangles using the same gradient in conjunction with gradientUnits="objectBoundingBox"

```
<?xml version="1.0" standalone="no"?>
<svg width="8cm" height="3cm" viewBox="0 0 1000 450" version="1.1"
xmlns="http://www.w3.org/2000/svg">
  <g>
    <defs>
      <linearGradient id="linear_objectBoundingBox" x1="0" y1="0" x2="1" y2="1">
        <stop offset="5%" stop-color="#ff0000" />
        <stop offset="95%" stop-color="#0000ff" />
      </linearGradient>

    </defs>

    <rect fill="none" stroke="blue" x="1" y="1" width="990" height="440"/>

    <g transform="translate(10,50)">
       <rect fill="url(#linear_userSpaceOnUse)" x="10" y="10" width="600"
height="100"/>
       <rect fill="url(#linear_userSpaceOnUse)" x="200" y="120" width="600"
height="100"/>
    </g>

  </g>
</svg>
```

producing this image:



16.8 Radial gradients

Radial gradients are supported from version 5.7.6 onwards.

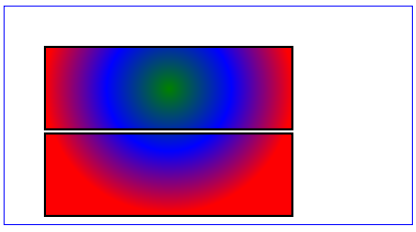
The interpretation of the values specified for the coordinates cx/cy/r/fx/fy of the radialGradient element changes depending on value specified for gradientUnits.

When gradientUnits="userSpaceOnUse" the specified values are in "user space", which is the space defined by the prevailing <g> element. The specified coordinates are relative to the prevailing <g> element, so two elements which use the same gradient as their fill color will appear differently if they are placed in different locations on the page.

This SVG image shows rectangles using the same gradient in conjunction with gradientUnits="userSpaceOnUse"

```
<?xml version="1.0" standalone="no"?>
<svg width="8cm" height="3cm" viewBox="0 0 1000 550" version="1.1"
xmlns="http://www.w3.org/2000/svg">

  <g>
    <defs>
      <radialGradient id="radial_userSpaceOnUse" gradientUnits="userSpaceOnUse"
cx="400" cy="200" r="300" fx="400" fy="200">
        <stop offset="0%" stop-color="green" />
        <stop offset="50%" stop-color="blue" />
        <stop offset="100%" stop-color="red" />
      </radialGradient>
    </defs>

    <rect fill="none" stroke="blue" x="1" y="1" width="990" height="530"/>
    <rect fill="url(#radial_userSpaceOnUse)" stroke="black" stroke-width="5" x="100"
y="100" width="600" height="200"/>
    <rect fill="url(#radial_userSpaceOnUse)" stroke="black" stroke-width="5" x="100"
y="310" width="600" height="200"/>
  </g>
</svg>
```

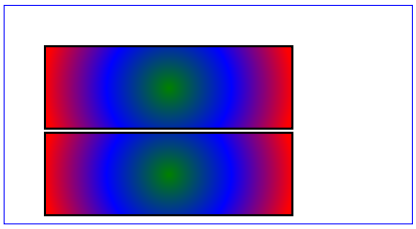
producing the image below, in which you can clearly see the gradient circles are centered within the first rectangle.



When gradientUnits="objectBoundingBox" the specified values are relative to the bounding box of the element being filled, and should be expressed as fractions of the dimensions of the element being filled. The values for coordinates should be in the range [0..1], so for example specifying x1="0" starts the gradient at the left hand edge of the element being filled, and specifying x1="0.2" starts the gradient at 20% of the width of that element. As the gradient is positioned relative to the element being filled, two element using the same gradient will appear the same regardless of the position of the element.

This SVG image shows rectangles using the same gradient in conjunction with gradientUnits="userSpaceOnUse"

producing the image below.



Chapter 17

Absolute Positioning

Content can be positioned anywhere on the page by placing the content in a block-container element and setting the absolute-position attribute.

If the absolute-position attribute is set to "fixed", the content will then be positioned on the page *relative to the page* area which contains the block-container element.

If the absolute-position attribute is set to "absolute", the content will be positioned on the page *relative* to the *reference area* which contains the block-container element. The reference area is not the containing block, it is the containing region, table-cell, block-container, inline-container or table-caption. In XSL-FO 1.0, the specification was ambiguous and the block-container was positioned relative to the containing area, but in XSL 1.1 this has been clarified to mean the containing reference area.

17.1 Positioning block-containers

It is important to realise that block-containers are not positioned relative to the containing block. Figure 17-1 shows FO with two absolutely positioned block containers. Both block-containers will be positioned relative to the containing region, because the region is the containing reference area. As they both have the same top attribute they will both be positioned in the same place.

Figure 17-1: Badly positioned block containers

```
<flow flow-name="body">
  <block>
    some text
    <block-container absolute-position="absolute" height="2cm" top="3cm">
       <block>
         in block-container one
       </block>
    </block-container>
  </block>

  <block>
    some more text
    <block-container absolute-position="absolute" height="2cm" top="3cm">
       <block>
         in block-container two
       </block>
    </block-container>
  </block>
</flow>
```

The simplest way to position a block-container is to place it inside another block-container which does not have the absolute-position attribute. FO for doing this is shown in Figure 17-2. The outer block-container is not absolutely positioned and will be

placed in the normal flow of content. The inner block-container is absolutely positioned relative to the outer one.

Figure 17-2: Positioned a block-container using another block-container

```
<flow flow-name="body">
  <block>
    some text
    <block-container>
      <block-container absolute-position="absolute" height="2cm" top="3cm">
         <block>
           in block-container one
         </block>
      </block-container>
    </block-container>
  </block>

  <block>
    some more text
    <block-container>
      <block-container absolute-position="absolute" height="2cm" top="3cm">
         <block>
           in block-container two
         </block>
      </block-container>
    </block-container>
  </block>
</flow>
```

17.2 Positioning and sizing block containers

A block-container with absolute-position = "absolute" is positioned relative to its containing reference area.

The distance between the left edge of the block-container and the left edge of the containing reference area is set by the left attribute. This attribute specifies the offset of the block-container's left edge from the containing reference area's left edge. The default value is "opt", which causes the two edges to be in the same place. Positive values of left move the left edge of the block-container to the right, making the block-container smaller.

The distance between the right edge of the block-container and the right edge of the containing reference area is set by the right attribute. This attribute specifies the offset of the block-container's right edge from the containing reference area's right edge. The default value is "opt", which causes the two edges to be in the same place. Positive values of right move the right edge of the block-container to the left, making the block-container smaller.

The distance between the top edge of the block-container and the top edge of the containing reference area is set by the top attribute. This attribute specifies the offset of the block-container's top edge from the containing reference area's top edge. The default value is "opt", which causes the two edges to be in the same place. Positive values of top move the top edge of the block-container downwards, making the block-container smaller.

The distance between the bottom edge of the block-container and the bottom edge of the containing reference area is set by the bottom attribute. This attribute specifies the offset of the block-container's bottom edge from the containing reference area's bottom edge. The default value is "opt", which causes the two edges to be in the same

place. Positive values of bottom move the bottom edge of the block-container upwards, making the block-container smaller.

If none of the left, right, top or bottom attributes is specified the block-container will be the same size as the reference area which contains it. This is because the offsets all default to "opt" so the edges of the block-container are the same as the edges of its containing reference area. This means a block-container with absolute-position= "absolute" which is placed in a region will by default fill that region.

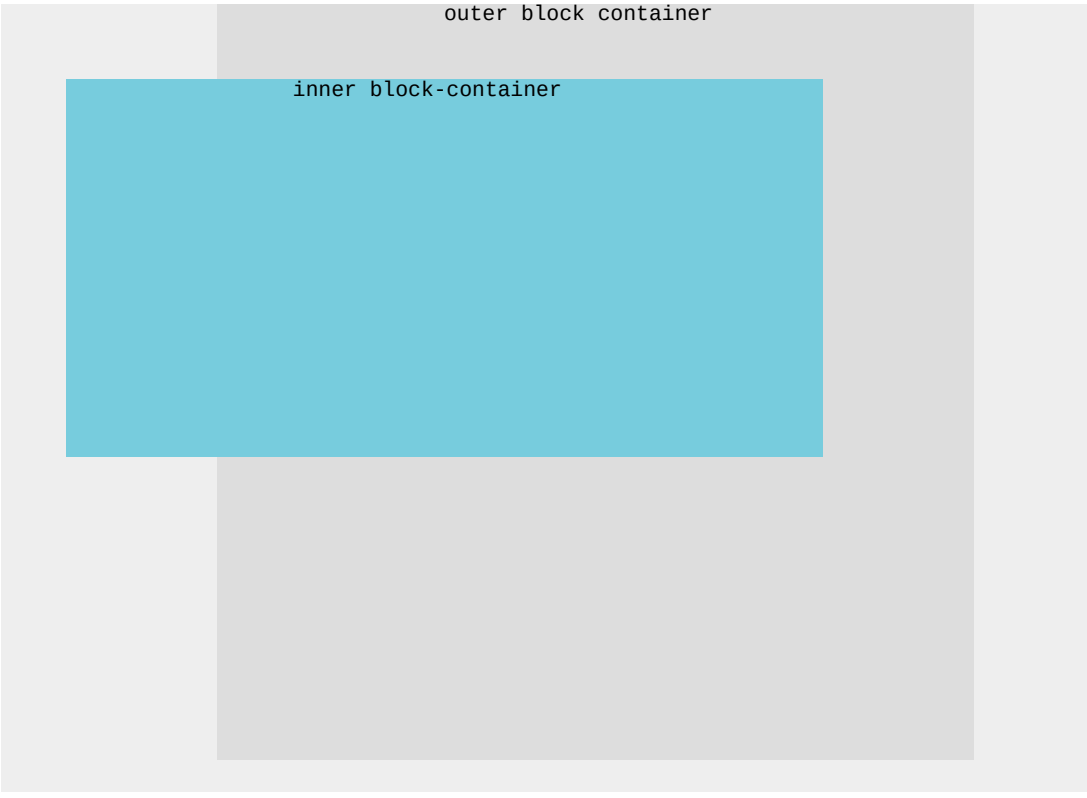
The height of a block-container can be specified with the height attribute, and the width with the width attribute.

Figure 17-3 shows the FO for a block container with height and width of 10cm, and an inner block-container which is offset from the outer one, including a negative offset on the left side. The output from this FO appears in Figure 17-4.

Figure 17-3: block-containers positioned and sized

```
<flow flow-name="body">
  <block>
    <block-container height="10cm" width="10cm" margin-left="3cm"
       background-color="#dddddd">
      <block>outer block container</block>
      <block-container absolute-position="absolute"
        top="1cm"
        right="2cm"
        left="-2cm"
        bottom="4cm"
        background-color="#77ccdd"
       >
         <block>
           inner block-container
         </block>
      </block-container>
    </block-container>
  </block>
</flow>
```

Figure 17-4: block-containers positioned and sized



The example in Figure 17-4 shows how using a negative left value will position the content to the left of the containing reference area. This is the technique used in this manual to place the labels next to each example.

Chapter 18

Columns

XSL-FO allows us to define a page which has multiple columns, just like this one.

This can only be done for whole pages, not for partial pages. However if we are in a region which has multiple columns we can treat it as a single-column region and place output across the whole width of the multi-column page by setting span="all" on block-level elements which appear immediately below the flow element.

Columns are defined by setting the column-count attribute of a body region element to a value greater than 1, and optionally setting the column-gap attribute to define a gap between the columns.

The page master for this is similar to the one shown in Figure 18-1.

Figure 18-1: The page master for a multi-column page

```
<simple-page-master
master-name="chapter-2col-odd">
    <region-start extent='2.5cm'/>
    <region-end extent='2.5cm'/>
    <region-body column-count='2'
       region-name="body"
margin='2.5cm'/>
    <region-after
region-name="footer-odd" extent="2.5cm"/>
    <region-before
region-name="header-odd" extent="2.5cm"/>
</simple-page-master>
```

All the blocks above, including this one, have span="all" set so that they span the whole page.

This block does not have span="all", so it will be fitted into the first column in the page. Text will flow to the bottom of this page and then start at the top of the next column.

If there are blocks above this one on the page which have span="all" (as there are) then they will remain in place and the text which is in only one column will be placed in the next column, below the span="all" blocks.

We deliberately repeat the paragraph to demonstrate this wrapping. This block does not have span="all", so it will be fitted into the first column in the page. Text will flow to the bottom of this page and then start at the top of the next column. If there are blocks above this one on the page (as there are) which have span="all" then they will remain in place and the text which is in only one column will be placed in the next column, below the span="all" blocks.

It is also possible to have a page start with content in two columns (like this).

When a block-level object is encountered which has span="all" the content already on the page is pushed up to the top, and the block with span="all" is spread over the two columns.

Chapter 19

Bookmarks

Bookmarks are the entries which appear on the right in a PDF file in Adobe Acrobat. They are used to navigate directly to locations within the document. They also have a hierarchical structure, where one bookmark can contain a set of child bookmarks which in turn can themselves contain other bookmarks.

The bookmark-tree element is optional. If used it should be placed under the root element, after the layout-master-set and declarations elements and before any page-sequence or page-sequence-wrapper elements.

The structure of a bookmark tree is shown in Figure 19-1.

Figure 19-1: A bookmark tree

```
<bookmark-tree>
  <bookmark internal-destination="section-1">
    <bookmark-title>Chapter 1</bookmark-title>
    <bookmark internal-destination="section-1-1">
      <bookmark-title>Section 1</bookmark-title>
    </bookmark>
    <bookmark internal-destination="section-1-2">
      <bookmark-title>Section 2</bookmark-title>
    </bookmark>
  </bookmark>
  <bookmark internal-destination="section-2">
    <bookmark-title>Chapter 2</bookmark-title>
    <bookmark internal-destination="section-2-1">
      <bookmark-title>Section 1</bookmark-title>
    </bookmark>
  </bookmark>
</bookmark-tree>
```

We can see the following from the structure shown in Figure 19-1.

- The bookmarks are contained in a bookmark-tree element.
- A bookmark element has an internal-destination attribute identifying where in the document it links to. The value for this attribute should be used as the id attribute on the destination element.
- A bookmark element can contain other bookmark elements.
- The text which appears in the bookmark is contained within a bookmark-title element. Ibex supports using Unicode text in bookmarks.

The example above creates bookmarks like the ones in the Ibex manual.

The bookmarks which have child bookmark elements appear in the PDF file in a closed state, so the user can click the '+' next to them to display the child elements. If you specify starting-state="show" on a bookmark or bookmark-tree element it's immediate children will be visible when the PDF file is opened.

Chapter 20

Configuration

All configuration of Ibex is done using the Settings class which is accessed from the ibex4.FODocument object. This class has many properties which can be changed to configure the operation of Ibex.

Properties of the Settings class should be changed prior to calling the generate() method on the FODocument object. The fact that the Settings object is a property of the FODocument object means that different documents can have different Settings values. For example Figure 20-1 shows how to set the default line height to 1.4em.

Figure 20-1: Example usage of the Settings object

```
using System;

using ibex4;

public class IbexTester {

   public static void Main(string[] args) {

      FODocument doc = new FODocument()

      doc.Settings.LineHeightNormal = "1.4em";

      using( Stream xml =
             new FileStream( args[0], FileMode.Open, FileAccess.Read ) ) {
          using ( Stream output =
              new FileStream( args[1], FileMode.Create, FileAccess.Write ) ) {
                    doc.generate( xml, output );
               }
           }
   }
}
```

20.1 Fonts

The following properties on the Settings change the way fonts a processed. By default each absolute font size (small, medium, large etc.) is 1.2 times larger than the previous size.

Property	Type	Default	Notes
XX_Small	string	7.0pt	Must end in 'pt'.
X_Small	string	8.3pt	Must end in 'pt'.
Small	string	10.0pt	Must end in 'pt'.
Medium	string	12.0pt	Must end in 'pt'.
Large	string	14.4pt	Must end in 'pt'.
X_Large	string	17.4pt	Must end in 'pt'.
XX_Large	string	20.7pt	Must end in 'pt'.

Property	Type	Default	Notes
Smaller	string	0.8em	Must end in 'em'.
Larger	string	1.2em	Must end in 'em'.

20.2 Line height

The following properties on the Settings change the default line height. Ideally Settings.LineHeightNormal should end in 'em' to make line height proportional to the font height.

Property	Type	Default	Notes
LineHeightNormal	string	1.2em	

20.3 Page size

The following properties on the Settings change the default page size.

Property	Type	Default	Notes
PageHeight	string	297mm	
PageWidth	string	210mm	

20.4 Include paths

The following properties on the Settings effect retrieving XML or XSL files.

Property	Type	Default	Notes
BaseURI_XML	string		This value sets the base URI for including images and other XML files. When an external-graphic element specifies a relative path, Settings.BaseURI_XML is the base URI used in accordance with the rfc2396 URI Specification. When an XML file uses an entity to include another XML file, Settings.BaseURI_XML is the base URI used when Ibex searches for the included XML file.
BaseURI_XSL	string		This value sets the base URI for including other XSL files. When an xsl:include element is used to include another XSL stylesheet, Settings.BaseURI_XSL can be used to specify the location the included stylesheet should be loaded from.

20.5 Images

The following properties on the Settings effect retrieving images specified using the external-graphic element.

Property	Type	Default	Notes
BaseURI_XML	string		This value sets the base URI for including images and other XML files. When an external-graphic element specifies a relative path, Settings.BaseURI_XML is the base URI used in accordance with the rfc2396 URI Specification. When an XML file uses an entity to include another XML file, Settings.BaseURI_XML is the base URI used when Ibex searches for the included XML file.
WebRequestTimeoutMs	int	300	When an external-graphic element specifies an image is retrieved from a web server, this is the timeout used for the call to the web server. Units are milliseconds.

20.6 Border widths

The following properties on the Settings change the values for border widths specified with constants like 'thin'.

Property	Type	Default	Notes
BorderWidthThin	string	1pt	
BorderWidthMedium	string	2pt	
BorderWidthThick	string	3pt	

20.7 Layout

The following properties on the Settings change the appearance of content in the PDF file.

Property	Type	Default	Notes
OverflowIsVisible	bool	true	By default a region has overflow='auto', leaving it up the Ibex to decide whether content which overflows the bottom edge of a region is displayed or discarded. If Settings.OverflowIsVisible is true, the content will be displayed, if false it will be discarded. This property applies only if the XSL-FO attribute 'overflow' is not set or is set to 'auto'.

20.8 Leaders

The following properties on the Settings change the values for leader formatting objects.

Property	Type	Default	Notes
LeaderDot	char	.	When leader-pattern='dots', this is the character used as the dot

Chapter 21

Extensions

This chapter details Ibex-specific extensions to XSL-FO. Typically these extensions implement functionality such as password protecting a document which is not part of the XSL-FO standard.

The Ibex extensions have a namespace which is specified using the xmlns attribute as shown in Figure 21-1.

21.1 Document security

Ibex supports encryption of PDF documents and the setting of various document permissions. This is done using the ibex:security element as shown in Figure 21-1.

Figure 21-1: FO using the ibex:security element

```
<root xmlns="http://www.w3.org/1999/XSL/Format"
 xmlns:ibex="http://www.xmlpdf.com/2003/ibex/Format">

<ibex:security deny-print='true' deny-extract='true'
 deny-modify='true' user-password='user' owner-password='owner'/>

...
```

Two levels of encryption are available, 40 bit and 128 bit. When using 40 bit encryption available permissions which can be set including deny-print, deny-extract and deny-modify. When using 128 bit encyption additional permissions can be set including deny-assembly and deny-print-high-resolution. These options are details in the sections below.

The level of encryption is specified using the bits attribute of the ibex:security element. This defaults to "40", so specify 128 bit encryption specify bits="128".

If used the ibex:security element **must** occur before any page-sequence elements.

21.1.1 40 bit encryption security options

When the number of bits of encryption is set to 40 or not specified, the attributes of the ibex:security element are:

Attribute	Values	Meaning
user-password		Specifies a password required to open the document in Acrobat. Once the document is opened with the correct user password, access is limited to permissions given using the attributes below.
owner-password		Specifies a password required to get all rights to the document in Acrobat. Once the document is opened with the correct owner password the user has total control of the document.
deny-print	true false	If this is set to true a user who opens the document with the user password will not be able to print the document.
deny-extract	true false	If this is set to true a user who opens the document with the user password will not be able to use cut-and-paste functionality to copy part of the document.
deny-modify	true false	If this is set to true a user who opens the document with the user password will not be able to modify the document.

Setting any of the attributes listed above will cause Ibex to encrypt the document.

Specifying the user-password but not the owner-password will set the owner-password to the same value as the user-password. This means anyone who can open the document using the user password has complete control of the document.

Specifying the owner-password but not the user-password is common usage. This means the user can open the document with limited rights without needing a password, but cannot then change or exceed those rights without knowing the owner password.

21.1.2 128 bit encryption security options

When the number of bits of encryption is set to 128, the attributes of the ibex:security element are:

Attribute	Values	Meaning
user-password		Specifies a password required to open the document in Acrobat. Once the document is opened with the correct user password, access is limited to permissions given using the attributes below.
owner-password		Specifies a password required to get all rights to the document in Acrobat. Once the document is opened with the correct owner password the user has total control of the document.
deny-print	true false	If this is set to true a user who opens the document with the user password will not be able to print the document.
deny-print-high-resolution	true false	If this is set to true a user who opens the document with the user password will not be able to print a high resolution copy of the document. They will only be able to print a low resolution (150dpi) version. If deny-print="true" this attribute has no effect and the document cannot be printed.
deny-extract	true false	If this is set to true a user who opens the document with the user password will not be able to use cut-and-paste functionality to copy part of the document.
deny-modify	true false	If this is set to true a user who opens the document with the user password will not be able to modify the document but can still "assemble" it. See deny-assembly below.
deny-assembly	true false	If deny-modify="true" and deny-assembly="false" then the user cannot change the document, but can "assemble" it, which means insert, rotate or delete pages and create bookmarks or thumbnail images. Setting deny-modify="true" and deny-assembly="true" prevents assembly.

Setting any of the attributes listed above will cause Ibex to encrypt the document.

Specifying the user-password but not the owner-password will set the owner-password to the same value as the user-password. This means anyone who can open the document using the user password has complete control of the document.

Specifying the owner-password but not the user-password is common usage. This means the user can open the document with limited rights without needing a password, but cannot then change or exceed those rights without knowing the owner password.

21.2 Standard document properties

Ibex allows you to set the various properties associated with a PDF document. These properties can be viewed in Acrobat by using the File | Document Properties | Summary menu option or just pressing control-d.

Figure 21-2 shows FO for setting the document properties using the ibex:properties element.

Figure 21-2: FO using ibex:properties

```
<root xmlns="http://www.w3.org/1999/XSL/Format"
  xmlns:ibex="http://www.xmlpdf.com/2003/ibex/Format">

<ibex:properties
     title="Ibex User Manual"  subject="Ibex"
     author="visual programming limited"
     keywords="xml,pdf" creator="xtransform" />
...
```

If used the ibex:security element **must** occur before any page-sequence elements.

The attributes of the ibex:properties element are:

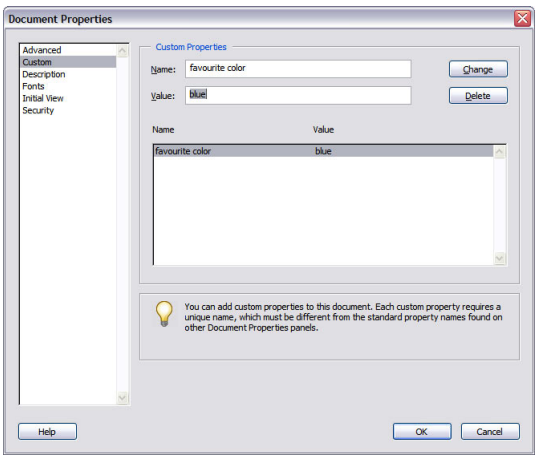
Attribute	Values	Meaning
title		Specifies a string which becomes the title property of the document.
subject		Specifies a string which becomes the subject property of the document.
author		Specifies a string which becomes the author property of the document.
keywords		Specifies a string which becomes the keywords property of the document. Separate individual keywords with commas.
creator		Specifies a string which becomes the creator property of the document. This should be the name of the application which created the XSL-FO document from which the PDF file was created.

Attribute	Values	Meaning
page-mode		Specifies how Acrobat will display the document when it is first opened. If set to 'bookmarks' then if the document has bookmarks they will be displayed. If set to 'thumbs' then the thumbnails tab in Acrobat will be displayed. If set to 'fullscreen' the document will be displayed without any toolbar, border etc.

Following the PDF standard, the document creator property should be the name of the product which converted the content to PDF format, so this is always Ibex. Other document properties such as creation and modification date are populated automatically by Ibex.

21.3 Custom Document Properties

Acrobat supports the display and editing of custom document properties. These properties are a set of name value pairs stored within the PDF file. In Acrobat 6.0 these properties can be viewed by using the File | Document Properties menu option and clicking on the "Custom" entry in the list box to display a screen like this:



These custom properties are inserted into the PDF using the ibex:custom element as shown in Figure 21-3.

Figure 21-3: FO using the ibex:custom element

```
<root xmlns="http://www.w3.org/1999/XSL/Format"
  xmlns:ibex="http://www.xmlpdf.com/2003/ibex/Format">

<ibex:properties title="Ibex User Manual">
    <ibex:custom name="favourite color" value="blue"/>
</ibex:properties>
...
```

Each property must have a name and value attribute.

21.4 Image processing

21.4.1 Image resolution

Ibex adds the dpi attribute to the external-graphic element to permit managing the dots per inch resolution of images. See Image resolution on page 98.

21.4.2 Anti-aliasing

Ibex adds the ibex:anti-alias attribute to the external-graphic element to permit disabling anti-aliasing in order to achieve clearer images. See Image anti-aliasing on page 98.

21.4.3 Multi-page TIFF image processing

Ibex adds the ibex:page attribute to the external-graphic element to specify which page of a muti-page TIFF image should be included in the PDF file. See Multi-page TIFF images on page 103.

21.5 Bookmarks

XSL-FO 1.0 had no support for creating bookmarks in the PDF file. XSL 1.1 now has this feature so the ibex:bookmark element is no longer supported.

The XSL 1.1 implementation of bookmarks is described on page 125.

21.6 Document base URL

The PDF format supports setting a base URL for links created with a basic-link element. This base URL is prepended to the destination specified with an external-destination attribute if (and only if) the specified destination does not start with a '/' character.

Figure 21-4 shows FO which creates a document with "http://www.xmlpdf.com" as the base URL and a link to the page "index.html". When the user clicks on the link in the PDF file, it will go to "http://www.xmlpdf.com/index.html".

Figure 21-4: FO setting the document base URL

```
<ibex:document-base-url value="http://www.xmlpdf.com"/>
..
<block>
   <basic-link external-destination='url(index.html)'>
      index.html
   </basic-link>
</block>
```

The base URL is a document-wide property and can be set only once.

This property should not be confused with the Settings.BaseURI value which specifies a base URI to be used when Ibex retrieves images, stylesheets and XML during creation of the PDF file.

21.7 Ibex version

The ibex:version element inserts the version number of Ibex used to create the PDF file. This is an inline element which inserts characters into the document. Figure 21-5 shows FO which uses this element and the output appears in Figure 21-6.

Figure 21-5: FO using ibex:version

```
<block xmlns:ibex="http://www.xmlpdf.com/2003/ibex/Format">
  created with Ibex version <ibex:version/>
</block>
```

Figure 21-6: Output from ibex:version

```
created with Ibex version 6.11.1.3
```

21.8 PDF/X

Ibex can create PDF files which comply with the PDF/X standard. This is described in detail on page 151.

21.9 Viewer Preferences

Ibex can set flags on the PDF file which control how the viewer application, such as Acrobat Reader, will display the PDF file.

These flags are set using the ibex:viewer-preferences element as shown in Figure 21-7.

Figure 21-7: FO using the ibex:viewer-preferences element

```
<root xmlns="http://www.w3.org/1999/XSL/Format"
 xmlns:ibex="http://www.xmlpdf.com/2003/ibex/Format">
<ibex:viewer-preferences hide-toolbar="true"/>
...
```

The attributes for the ibex:viewer-preferences element are:

Attribute	Values	Meaning
hide-toolbar	true false	Set to true to hide the viewer application's tool bars
hide-menubar	true false	Set to true to hide the viewer application's menu bar
hide-window-ui	true false	Set to true to hide the UI and just display the document content
fit-window	true false	Set to true to resize the viewer window to fit the document page size

Attribute	Values	Meaning
center-window	true false	Set to true to center the viewer window on the screen
display-doc-title	true false	Set to true to have the viewer display the document title in the viewer frame rather than the file name. The document title is set using the title attribute of the ibex:properties element as detailed on page 134.

Chapter 22

Accessiblity and PDF/UA

Ibex now supports the PDF Universal Accessibility Standard and WCAG

For information on the standard see PDF/UA

22.1 Enabling PDF/UA Creation

To create a PDF/UA compliant PDF the FO file needs to have three things:

(1) a declaration of the ibex namespace on the <fo:root> element like this:

```
<fo:root
    xmlns:fo="http://www.w3.org/1999/XSL/Format"
    xmlns:ibex="http://www.xmlpdf.com/2003/ibex/Format"
    ...
    >
```

(2) a language declaration, which can be done on the <fo:root> element like this:

```
<fo:root
  xml:lang="en-US"
  ...
  >
```

(3) the FO file needs to include metadata surrounded by <ibex:pdfua> tags like this:

```
<ibex:pdfua>
    <x:xmpmeta xmlns:x="adobe:ns:meta/"
        x:xmptk="Adobe XMP Core 5.6-c01591.163280, 2018/06/22-11:31:03">
        <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
        <rdf:Description rdf:about="" xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:pdfuaid="http://www.aiim.org/pdfua/ns/id/">
            <dc:title>
            <rdf:Alt>
                <rdf:li xml:lang="en">PDF/UA Document</rdf:li>
            </rdf:Alt>
            </dc:title>
            <pdfuaid:part>1</pdfuaid:part>
        </rdf:Description>
        </rdf:RDF>
    </x:xmpmeta>
</ibex:pdfua>
```

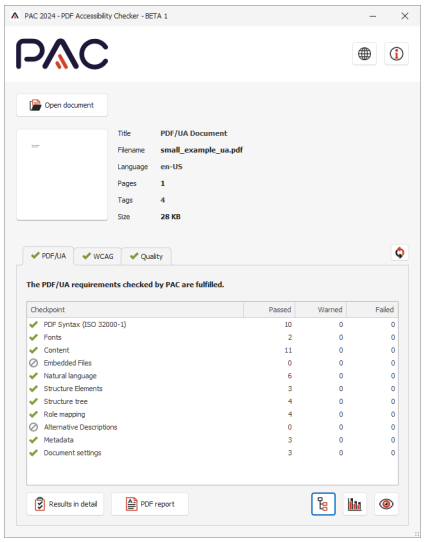
This metadata includes the title "PDF/UA Document", change that to your own document title.

Once the three items above are included in the FO file Ibex will produce a PDF/UA compliant file.

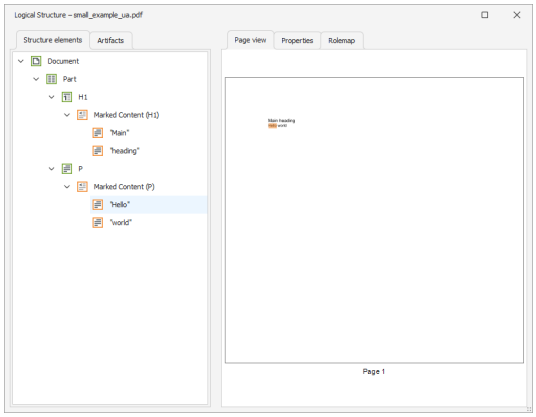
A complete test file with one paragraph looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<fo:root
  xmlns:fo="http://www.w3.org/1999/XSL/Format"
  xmlns:ibex="http://www.xmlpdf.com/2003/ibex/Format"
  xml:lang="en-US"
  >

  <ibex:pdfua>
    <x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="Adobe XMP Core 5.6-c01591.163280,
2018/06/22-11:31:03">
      <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
        <rdf:Description rdf:about="" xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:pdfuaid="http://www.aiim.org/pdfua/ns/id/">
          <dc:title>
            <rdf:Alt>
              <rdf:li xml:lang="x-default">PDF/UA Document</rdf:li>
              <rdf:li xml:lang="en">PDF/UA Document</rdf:li>
            </rdf:Alt>
          </dc:title>
          <pdfuaid:part>1</pdfuaid:part>
        </rdf:Description>
      </rdf:RDF>
    </x:xmpmeta>
  </ibex:pdfua>

  <fo:layout-master-set>
    <fo:simple-page-master master-name="page" margin="1.5cm" page-height="297mm"
page-width="310mm">
      <fo:region-body column-count="1" region-name="body" margin="2.75cm 0.5cm 1cm
3cm" />
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:bookmark-tree>
    <fo:bookmark internal-destination="header1" starting-state="show">
      <fo:bookmark-title>Heading One</fo:bookmark-title>
    </fo:bookmark>
  </fo:bookmark-tree>

  <fo:page-sequence master-reference="page" initial-page-number="1" format="i"
font="12pt arial">
    <fo:flow font="11pt arial" flow-name="body">
      <fo:block font-size="larger" role="H1" id="header1">
         Main heading
      </fo:block>
      <fo:block>
        Hello world
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

The file created from the FO can be validated using the free PAC program. This tests various aspects of compliance and shows the results:



The tagged pdf tree structure can be viewed:



Note that the contents of the fo:page-sequence have been placed in a "Part" structure element. This is optional, controlled by the Settings. PDFUA_PutPageSequenceAreasInPartElements flag.

22.2 Headers

As shown in the above example any fo:block can have the "role" property set. To create a header use H1, H2 .. H6 as standards-compliant heading roles, like this:

```
<fo:block font-size="larger" role="H1" id="header1">
    Main heading
</fo:block>
```

To disable the use of the "role" property when creating structured elements specify the 'ignore-role-attributes' property on the <ibex:pdfua> node like this:

```
<ibex:pdfua ignore-role-attributes="true">
```

22.3 Tables

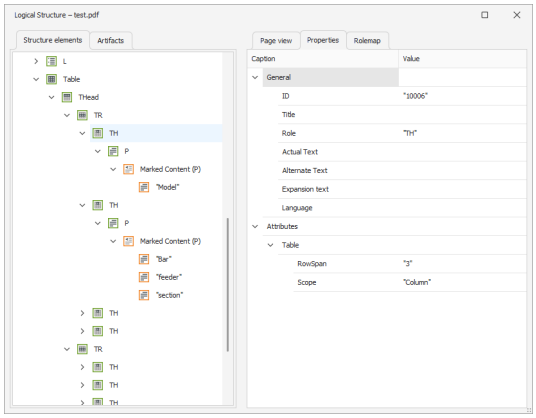
Table elements are automatically tagged according to the following table:

Element	Tag
fo:table	Table
fo:table-caption	Caption
fo:table-header	THead
fo:table-body	TBody
fo:table-footer	TFoot
fo:table-row	TR
fo:table-cell	TD or TH

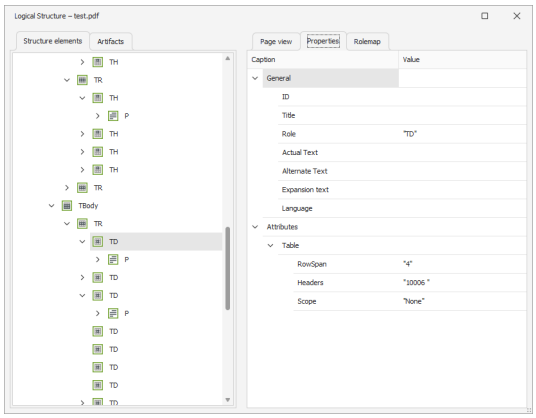
Table cells inside a table header as tagged as TH. In addition:

- cells in table headers are given an "ID" property to identify them
- cells in the table body automatically have a "Headers" property which identifies which header cell(s) are relevant headings. There might be multiple if the cell spans multiple columns
- where are header has multiple rows, the cells in the lower rows have "Headers" properties which reference the cells in higher rows which cover the same columns

In practice this looks like the element tree shown below, where are TH element has "ID", "Role" and "Rowspan" properties:



And a TD cell element in the table body has a "Headers" property which matches the "id" property of the cell above it in the header:



22.4 Lists

List elements are automatically tagged according to the following table:

Element	Tag
fo:list-block	L
fo:list-item-label	Lbl
fo:list-item-body	Lbody

22.5 Static Content

The contents of fo:static-content elements is marked as "Artifact".

22.6 Image Alt Tags

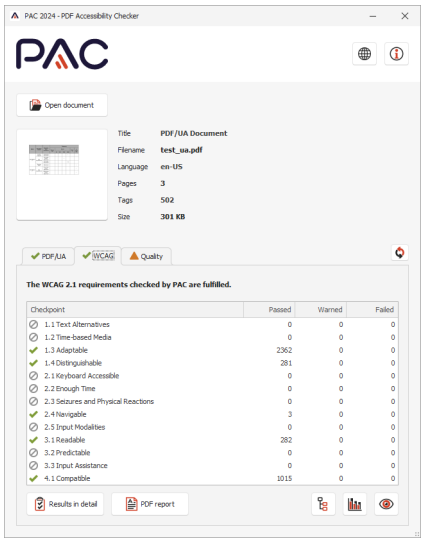
You can specify an Alt tag to describe images with text using the "ibex:alt" property like this:

```
<fo:external-graphic src="RedbrushAlpha-0.25.png" ibex:alt="picture of tree"/>
```

22.7 WCAG Requirements

PDF/UA documents created by Ibex support the Web Content Accessibility Guidelines 2.2 standard.

The PAC program supports viewing WCAG compilance using the WCAG tab on the main screen:



Each of the WCAG 2.2 requirements which relate to PDF files are explained below. The descriptions of the requirements come from www.w3.org

22.7.1 PDF1: Applying text alternatives to images with the Alt entry in PDF documents

This requirement is satisfied. Ibex supports Alt entries for images using the ibex:alt property on each <fo:external-graphic> elements like this:

```
<fo:external-graphic src="RedbrushAlpha-0.25.png" ibex:alt="picture of tree"/>
```

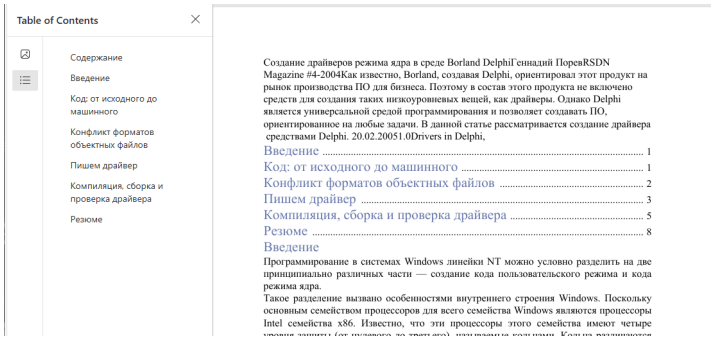
Support for the same thing on the <fo:instream-foreign-object> element will be added soon.

22.7.2 PDF2: Creating bookmarks in PDF documents

This requirement is satisfied. Ibex supports creating bookmarks using the <fo:bookmark-tree> and <fo:bookmark> elements like this:

```
<fo:bookmark-tree>
  <fo:bookmark internal-destination="CONTENTS67104136">
   <fo:bookmark-title>„1‡6†4†5^0‡Øf0‡Þ0ßark-title>
  </fo:bookmark>
    ...
```

This creates a bookmark tree in the PDF file like this:



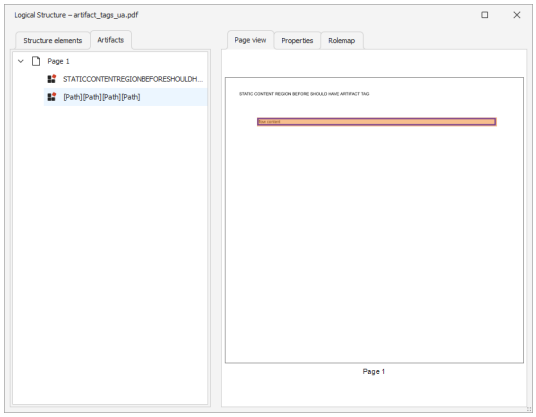
22.7.3 PDF3: Ensuring correct tab and reading order in PDF documents

This requirement is satisfied. The order of text in the PDF follows the order used in the input formatting objects document.

22.7.4 PDF4: Hiding decorative images with the Artifact tag in PDF documents

This requirement is satisfied. The Artifact tag is applied to page headers and footers (specifically the contents of <fo:static-content> elements) and other elements such as table cell borders and backgrounds.

Elements marked with the Artifact tag can be viewed on the Artifacts tab of the Logical Structure view in PAC program:



22.7.5 PDF5: Indicating required form controls in PDF forms

This requirement is not applicable as Ibex does not support creation of PDF forms as they are not part of the XSL Formatting Objects specification.

22.7.6 PDF6: Using table elements for table markup in PDF Documents

This requirement is satisfied. Table elements are automatically tagged according to the following table:

Element	Tag
fo:table	Table
fo:table-caption	Caption
fo:table-header	THead
fo:table-body	TBody
fo:table-footer	TFoot
fo:table-row	row
fo:table-cell	TD or TH

22.7.7 PDF7: Performing OCR on a scanned PDF document to provide actual text

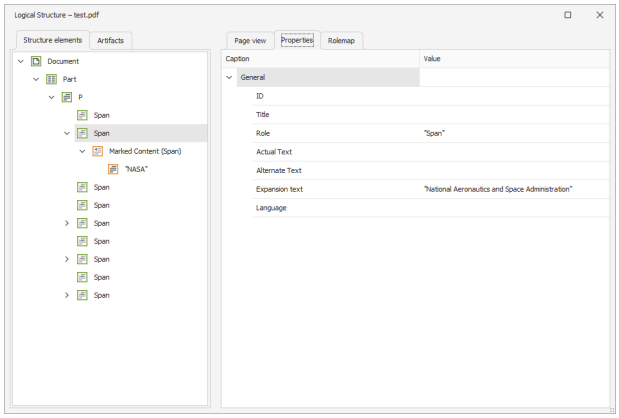
This requirement is satisfied. Ibex generates actual text rather than images of text.

22.7.8 PDF8: Providing definitions for abbreviations via an E entry for a structure element

This requirement is satisfied. The definition of an abbreviation can be specified using the ibex:abbrev property like this:

```
<fo:block>
  <fo:inline ibex:abbrev="National Aeronautics and Space
Administration">NASA</fo:inline>
  goes to moon
</fo:block>
```

This can be viewed using the PAC program like so:



22.7.9 PDF9: Providing headings by marking content with heading tags in PDF documents

This requirement is satisfied.

22.7.10 PDF10: Providing labels for interactive form controls in PDF documents

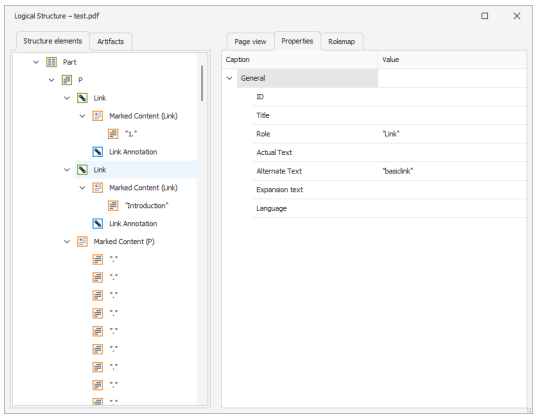
This requirement is not applicable as Ibex does not support creation of PDF forms as they are not part of the XSL Formatting Objects specification.

22.7.11 PDF11: Providing links and link text using the Link annotation and the /Link structure element in PDF documents

This requirement is satisfied. A link annotation is created automatically when using a <fo:basic-link> element

```
<fo:block text-align="justify" text-align-last="justify" space-after="3pt" >
       <fo:basic-link internal-destination="id8">1. Introduction
       ...
```

This can be viewed using the PAC program like so:



22.7.12 PDF12: Providing name, role, value information for form fields in PDF documents

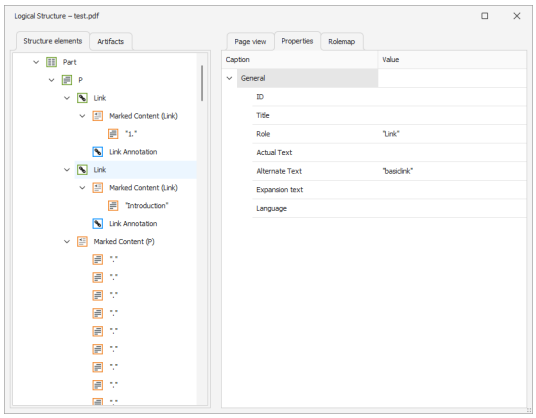
This requirement is not applicable as Ibex does not support creation of PDF forms as they are not part of the XSL Formatting Objects specification.

22.7.13 PDF13: Providing replacement text using the /Alt entry for links in PDF documents

This requirement is satisfied. Atlernative text can be provided using the ibex:alt property of the <fo:basic-link< element

```
<fo:block text-align="justify" text-align-last="justify" space-after="3pt" >
       <fo:basic-link internal-destination="id8" ibex:alt="basiclink">1.
Introduction
        ...
```

This can be viewed using the PAC program like so:



22.7.14 PDF14: Providing running headers and footers in PDF documents

This requirement is satisfied using the <fo:table-header> and <fo:table-footer> elements.

22.7.15 PDF15: Providing submit buttons with the submit-form action in PDF forms

This requirement is not applicable as Ibex does not support creation of PDF forms as they are not part of the XSL Formatting Objects specification.

22.7.16 PDF16: Setting the default language using the /Lang entry in the document catalog of a PDF document

This requirement is satisfied using the xml:lang attribute on the <fo:root> element as shown here:

```
<fo:root
  xmlns:fo="http://www.w3.org/1999/XSL/Format"
  xmlns:ibex="http://www.xmlpdf.com/2003/ibex/Format"
  xml:lang="en-US"
  >
```

22.7.17 PDF17: Specifying consistent page numbering for PDF documents

This requirement is satisfied using:

22.7.18 PDF18: Specifying the document title using the Title entry in the document information dictionary of a PDF document

This requirement is satisfied.

The XML from which a PDF/UA document is created contains an entry like this:

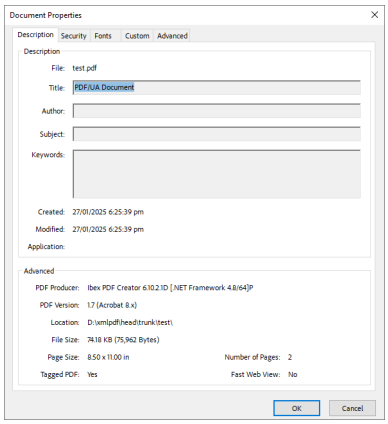
```
<ibex:pdfua>
  <x:xmpmeta xmlns:x="adobe:ns:meta/"
    x:xmptk="Adobe XMP Core 5.6-c01591.163280, 2018/06/22-11:31:03">
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
      <rdf:Description rdf:about="" xmlns:dc="http://purl.org/dc/elements/1.1/"
        xmlns:pdfuaid="http://www.aiim.org/pdfua/ns/id/">
        <dc:title>
          <rdf:Alt>
            <rdf:li xml:lang="x-default">PDF/UA Document</rdf:li>
            <rdf:li xml:lang="en">PDF/UA Document</rdf:li>
          </rdf:Alt>
        </dc:title>
        <pdfuaid:part>1</pdfuaid:part>
      </rdf:Description>
    </rdf:RDF>
  </x:xmpmeta>
</ibex:pdfua>
```

This entry is necessary for Ibex to identify the document as a PDF/UA document.

The <dc:title> element contains one or more document titles.

This XML is copied to the created PDF.

Adobe Acrobat will interpret this XML and display the title from the <dc:title> element like this:



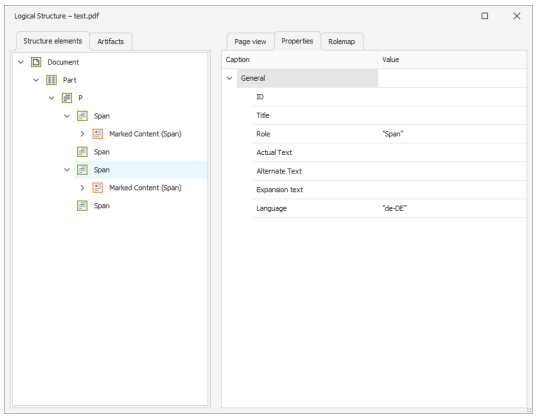
but other PDF viewers do not do this, so in addition to adding the <dc:title> element to the PDF file Ibex copies it to the catalog document properties.

22.7.19 PDF19: Specifying the language for a passage or phrase with the Lang entry in PDF documents

This requirement is satisfied. Block and inline elements can have their language specified using the xml:lang property like this:

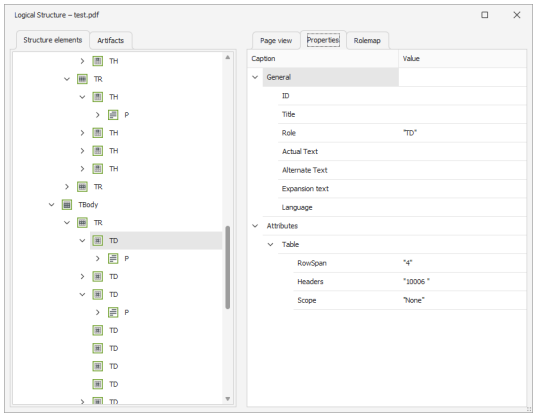
```
<fo:block xml:lang="en-GB">
  this block is "en-GB"
  <fo:inline xml:lang="de-DE">but this sentence is "de-DE"</fo:inline>
  this block is "erun-GB"
</fo:block>
```

Where there are multiple languages used in a single paragraph this creates span elements in the document structure:



22.7.20 PDF20: Using Adobe Acrobat Pro's Table Editor to repair mistagged tables

This requirement is satisfied. The table elements and tags are shown here using the PAC program:



22.7.21 PDF21: Using List tags for lists in PDF documents

This requirement is satisfied. List elements are automatically tagged according to the following table:

Element	Tag
fo:list-block	L
fo:list-item	LI
fo:list-item-label	Lbl
fo:list-item-body	Lbody

22.7.22 PDF22: Indicating when user input falls outside the required format or values in PDF forms

This requirement is not applicable as Ibex does not support creation of PDF forms as they are not part of the XSL Formatting Objects specification.

22.7.23 PDF23: Providing interactive form controls in PDF documents

This requirement is not applicable as Ibex does not support creation of PDF forms as they are not part of the XSL Formatting Objects specification.

Chapter 23

PDF/X

This chapter details Ibex-specific extensions to the XSL-FO XML to support creation of PDF files which conform to the PDF/X standard.

Ibex implements the PDF/X standard using the ibex:pfdx element as shown in Figure 23-1.

Figure 23-1: PDF/X

```
<root xmlns="http://www.w3.org/1999/XSL/Format"
 xmlns:ibex="http://www.xmlpdf.com/2003/ibex/Format">

<ibex:pdfx color-profile-file-name="WideGamutRGB.icc"/>

...
```

The Ibex extensions have a namespace which is specified using the xmlns attribute as shown above.

The ibex:pdfx element **must** occur before any output is generated.

Using the ibex:pdfx element will automatically set the document color space to CMYK.

The existence of the ibex:pdfx element causes Ibex to create a PDF/X compatible PDF file. The field Settings.PDFXMode used in earlier releases has been removed.

The attributes of the ibex:pdfx element are:

Attribute	Values	Meaning
color-profile-file-name		Full or relative path to a ICC color profile file
registry-name		Registry Name used in the PDF OutputIntents structure. If not specified this defaults to "http://www.color.org".
info		Optional text which will become the Info value in the first OutputIntents array entry.
output-condition-identifier		Optional text which will become the OutputConditionIdentifier value in the first OutputIntents array entry. This defaults to "Custom"

Attribute	Values	Meaning
output-condition		Optional text which will become the OutputCondition value in the first OutputIntents array entry. This defaults to "Custom". Acrobat proposes values such as "TR001 SWOP/CGATS".

The color profiles is read from the specified ICC file, compressed, and embedded in the PDF file.

23.1 Media box

The MediaBox size within the PDF file will be set to the size of the page as specified on the simple-page-master for that page.

23.2 Bleed box

The BleedBox size defaults to the MediaBox size. The BleedBox can be specified as a change from the MediaBox by specifying the ibex:bleed-width attribute on the simple-page-master. This attribute specifies the distance by which the BleedBox is smaller than the MediaBox as shown in Figure 23-2.

Figure 23-2: Setting the bleed box size

```
<simple-page-master page-height="313mm" page-width="226mm"
     master-name="page" ibex:bleed-width="3mm">
```

If one value is used it applies to all sides of the page, if two values are used the top and bottom edges use the first value and the left and right edges use the second. If there are three values the top is set to the first value, the sides are set to the second value, and the bottom is set to the third value. If there are four values, they apply to the top, right, bottom and left edges in that order.

The following attributes can be specified to set each side explicitly: bleed-top-width, bleed-bottom-width, bleed-right-width, bleed-left-width.

23.3 Trim box

The TrimBox size defaults to the BleedBox size. The TrimBox can be specified as a change from the BleedBox by specifying the ibex:trim-width attribute on the simple-page-master. This attribute specifies the distance by which the TrimBox is smaller than the BleedBox as shown in Figure 23-3.

Figure 23-3: Setting the trim box size

```
<simple-page-master page-height="313mm" page-width="226mm"
    master-name="page" ibex:trim-width="3mm">
```

If one value is used it applies to all sides of the page, if two values are used the top and bottom edges use the first value and the left and right edges use the second. If there are three values the top is set to the first value, the sides are set to the second value, and the bottom is set to the third value. If there are four values, they apply to the top, right, bottom and left edges in that order.

The following attributes can be specified to set each side explicitly: trim-top-width, trim-bottom-width, trim-right-width, trim-left-width.

23.4 Overprint

Overprint mode can be enabled for the entire page by specifying the ibex:ibex-overprint-stroking, ibex:overprint-nonstroking and ibex:overprint-mode attributes as shown in Figure 23-4.

Figure 23-4: Setting the overprint mode

```
<simple-page-master page-height="313mm" page-width="226mm"
  master-name="page" ibex:overprint-stroking="true"
  ibex:overprint-nonstroking="true" ibex:overprint-mode="1">
```