XMLmind XSL-FO Converter - User's Guide

Jean-Yves Belmonte

Hussein Shafie
XMLmind Software
<xfc-support@xmlmind.com>

XMLmind XSL-FO Converter - User's Guide

Jean-Yves Belmonte
Hussein Shafie
XMLmind Software
<xfc-support@xmlmind.com>

Publication date December 27, 2024

Abstract

This guide describes how to install the XMLmind XSL-FO Converter engine and use its command-line executables. It also explains how to integrate this software component into your application.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

XMLmind XSL-FO Converter (**XFC** for short) is an XSL-FO processor similar to Apache FOP, RenderX XEP or Antenna House XSL Formatter. Unlike the aforementioned processors which all renders XSL-FO as PDF and PostScript®, XMLmind XSL-FO Converter converts XSL-FO v1.0 to the following formats:

- RTF (Word 2000+),
- WordprocessingML (Word 2003+),
- Office Open XML (`.docx`, Word 2007+),
- OpenOffice (`.odt`, OpenOffice/LibreOffice 2+).

That is, XMLmind XSL-FO Converter *translates* one format, XSL-FO v1.0, to the file formats of the two most commonly used word processors, Microsoft Word and OpenOffice.org Writer.

Working at a higher level than the other XSL-FO processors, XMLmind XSL-FO Converter has intrinsic limitations which are detailed in Section 2, “Limitations” [16]. Despite these limitations, XMLmind XSL-FO Converter allows to process very elaborate XSL-FO files. In practice, you should be able to reuse *as is* the XSLT style sheets (which generate XSL-FO) that you have developed to convert your XML documents to PDF.

About Evaluation Edition

Do not be surprised because XMLmind XSL-FO Converter Evaluation Edition generates output containing random duplicate letters. Of course, this does not happen with Professional Edition!

The target audience of this document is a developer or an integrator, that is, a technical person and not an end user. End users, that is persons who need to convert XML documents to a variety of formats, are more likely to use XMLmind XSL Utility, a handy graphical tool, which is available in a separate, self-contained, distribution.

Chapter 2. Installing XMLmind XSL-FO Converter

1. System requirements

A .NET Framework 4.0+ is required to run the XMLmind XSL-FO Converter engine, .NET Edition. Mono 5.1+ is also supported on Linux.

XMLmind XSL-FO Converter is officially supported on Windows 7/8/10/11 (32-bit or 64-bit) and thanks to Mono, also on Linux.

2. Installation

Simply unzip the distribution somewhere. Example:

```
C:\> unzip temp\xfc_pro_dotnet-6_5_0.zip
C:\> dir xfc_pro_dotnet-6_5_0
...
bin     <DIR> ...
doc     <DIR> ...
legal   <DIR> ...
samples <DIR> ...
```

This means that uninstalling XMLmind XSL-FO Converter simply consists in deleting the directory created by unzipping its distribution.

3. Contents of the installation directory

`bin/fo2rtf.exe`, `fo2wml.exe`, `fo2docx.exe`, `fo2odt.exe`

Executable files used to run XMLmind XSL-FO Converter (**XFC** for short).

`bin/xfc.dll`

The .NET 4.0 assembly containing the XMLmind XSL-FO Converter engine. Reference it in your project if you are integrating XMLmind XSL-FO Converter in your application.

`bin/IKVM*.dll`

The IKVM.NET runtime needed by `xfc.dll`.

`bin/gac_install_all.bat`, `gac_uninstall_all.bat`

Simples scripts allowing to install all the above DLLs in the Global Assembly Cache (**GAC**). Script `gac_uninstall_all.bat` may be used to uninstall all the DLLs installed using `gac_install_all.bat`.

`doc/index.html`

Points to copies of this document in HTML, PDF, RTF, WordprocessingML, Office Open XML and OpenOffice formats.

Also points to the reference manual of the API of XMLmind XSL-FO Converter.

`legal.txt`, `legal/`

Contains legal information about **XFC** and about third-party components used in **XFC**.

`samples/`

A few XSL-FO sample files, in case you want to test the installation of XMLmind XSL-FO Converter by running `samples/make_samples.bat`.

Chapter 3. Command-line executables

About Evaluation Edition

Do not be surprised because XMLmind XSL-FO Converter Evaluation Edition generates output containing random duplicate letters. Of course, this does not happen with Professional Edition!

Four command-line executables are provided: `fo2rtf`, `fo2wml`, `fo2docx` and `fo2odt`, to convert an XSL-FO file to RTF, WML, Open XML (`.docx`) and OpenDocument (`.odt`) respectively. The general syntax of a command line is:

```
fo2rtf [<options>] <input> [<output>]
```

where *<input>* is the input XSL-FO file name and *<output>* the output file name. If no output file is specified the conversion output is written to the standard output stream. Available options are described below.

Commonly used options:

*/e**encoding***

Specifies the output encoding. Supported values depend on the target output format:

- For RTF output, supported encodings are `US-ASCII`, `windows-1250` (Windows Eastern European), `windows-1251` (Windows Cyrillic) and `windows-1252` (Windows Latin-1). The default value is `windows-1252`.
- For WML output, all the encodings available in the runtime are supported. The default value is `windows-1252` (Windows Latin-1).
- For Open XML output (`.docx`), supported encodings are `UTF-8` and `UTF-16`. The default value is `UTF-8`.
- For OpenDocument output (`.odt`), all the encodings available in the runtime are supported. The default value is `UTF-8`.

*/r**DPI***

Default image resolution in DPI. A positive integer. Used to compute the intrinsic size of an image, but only when an image file does not contain resolution or absolute size information.

Default value: 96.

`/pi`

Prescale images to minimize output document size. By default, the original size of images is preserved and scaling directives are inserted in the output document.

Note that:

- This option will never create an image which has larger dimensions than the original image. It can only create an image which has smaller dimensions than the original image.

- The `/pi` option is honored only for true raster graphics. Vector graphics (WMF, EMF) are never prescaled. Pre-rasterized vector graphics (SVG, MathML) are always prescaled (by the competent renderer, e.g. Batik or JEuclid, not by XMLmind XSL-FO Converter itself).

/fmap

May be used to map the generic font families `serif`, `sans-serif`, `monospace`, `fantasy` and `cursive` to actual font families.

Syntax:

```
map -> entry [',' entry]*

entry -> generic_family '=' actual_family

generic_family -> 'serif' | 'sans-serif' | 'monospace'
                  | 'cursive' | 'fantasy'
```

Example: `/f "fantasy=Impact,cursive=Comic Sans MS"`.

The default mapping depends on the output format: the generic font families `serif`, `sans-serif`, `monospace` are mapped to `"Times New Roman"`, `Arial`, `"Courier New"` for RTF, WML and Open XML (`.docx`) and to `"DejaVu Serif"`, `"DejaVu Sans"`, `"DejaVu Sans Mono"` for OpenDocument (`.odt`).

Note that by default, generic font families `fantasy` and `cursive` are not mapped.

`/s`

Specifies single-sided page layout. By default RTF, WML and Open XML (`.docx`) output documents are given a double-sided page layout regardless of the input document properties. This option may be used to force a single-sided page layout.

/styURL_or_filename

Specifies the location of an XML file containing the set of *user styles* to be used during the conversion. More information about user styles in Chapter 6, *XSL-FO extension for generating named styles* [44].

This location is an URL in its string form (e.g. `"file:///C:/My%20Folder/styles.xfc"`) or a filename (e.g. `"C:\My Folder\styles.xfc"`). A relative filename is relative to the current working directory.

The XML file must conform to the `styles.xsd` schema.

By default, XMLmind XSL-FO Converter generates only direct formatting (RTF, WordprocessingML, `.docx`) or automatic styles (`.odt`).

Rarely used options:

/eafmap

May be used to map East Asian font families to Western font families. Such East Asian fonts are used to render mainly **CJK** (Chinese Japanese Korean) text, possibly mixed with Western text. More information in Section 4.10, “Special support for East Asian fonts” [41].

Syntax:

```
map -> entry [',' entry]*

entry -> east_asian_family '=' western_family
```

Example: `/eaf "MS UI Gothic=Times New Roman,Meiryo=Calibri"`.

For compatibility with previous versions of XMLmind XSL-FO Converter, the default value of this property is "`Arial Unicode MS=Arial`".

This property is supported by the ODT, WML and DOCX output formats, but not by the RTF output format.

`/l`

List supported encodings.

`/rr`*`DPI`*

Default image resolution in DPI. A positive integer. Used to compute the intrinsic size of an image, *according to the image renderer (that is, MS-Word or OpenOffice)*, when an image file does not contain resolution or absolute size information.

The default value depends on the output format. Generally 96. *It is strongly recommended to use this default value*.

`/sr`*`DPI`*

Screen resolution in DPI. A positive integer. Used to convert `px` lengths to other units (`in`, `mm`, `cm`, `pt`, etc).

Default value: 96.

`/b`*`URL`*

Specifies the base URL of relative paths in attribute values (typically the `src` attribute of the `external-graphic` element). By default, paths are taken relative to the input source URL.

`/w`

Specifies MS-Word as target RTF viewer. This option may be needed to circumvent an obscure bug in the RTF loader of MS-Word, which does not handle table cell padding tags correctly. When this option is used, XFC will swap top and left padding values in table cells to work around this bug.

`/vml`

Specifies that images contained in Office Open XML (`.docx`) files must be represented using the *deprecated* VML markup rather than the DrawingML markup.

By default, images contained in Office Open XML (`.docx`) files are represented using DrawingML markup.

`/variant` *`MS-Word_major_version`* [`strict`]?

Examples: `/variant 14`, `/variant 15`, `/variant 15strict`.

Marks generated DOCX file as being compatible with MS-Word having specified major version. Any major version other than `14` (MS-Word 2010), `15` (MS-Word 2013), `16` (MS-Word 2016) is currently ignored.

Moreover suffix "`strict`" (supported only when *`MS-Word_major_version`* >= 15) may be used to generate DOCX files marked as being "Strict Open XML".

Default: None. The generated DOCX files are not marked as being compatible with a specific version of MS-Word.

Specifying `/variant 15` suppresses the "`[Compatibility Mode]`" text appearing in the title bar of MS-Word 2013 and 2016.

Specifying `/variant 15` does not prevent the generated DOCX file from being opened in MS-Word 2007 and 2010. However specifying `/variant 15strict` generates "Strict Open XML" files which are not supported by MS-Word 2007 and 2010.

`/png`

Specifies that input JPEG images must be converted to PNG in the output file.

`/nolist`

Do *not* attempt to create proper lists by inferring the numbering style of the list from the label of its first item. (By default, XFC attempts to create proper lists by inferring he numbering style of the list from the label of its first item.)

Note that even when this option is used, it's still possible to instruct XFC to create proper lists by specifying extension attribute `xfc:label-format` [29] in the XSL-FO input file.

`/meta`*`metadata_name``metadata_value`*

Specifies a metadata to be added to the document information section of the generated document. More information in Section 4.7, “Adding metadata to the documents created by **XFC**” [35].

Examples: "`/meta lastModifiedBy john@acme.com`", "`/meta xfc:final true`".

`/prot`*`restrictions`*

Specifies how the generated document is to be restricted in terms of editing and/or formatting. Restrictions syntax is:

```
'unrestricted' | 'limited-formatting' |
('read-only'|'comments-only'|'fill-forms-only'|'tracked-changes-only'
 [ '+limited-formatting' ]?)
```

Examples: "`/prot comments-only`", "`/prot limited-formatting`", "`/prot tracked-changes-only+limited-formatting`".

Use "`""`" or "`unrestricted`" to discard any existing edit restriction. More information in Section 4.8, “Restricting editing in the documents created by **XFC**” [39].

`/password`*`password`*

This clear text password lets the user of the word processor remove the edit restrictions. By default, the document protection is not enforced using a password.

Use "`""`" to discard any existing password. More information in Section 4.8, “Restricting editing in the documents created by **XFC**” [39].

Chapter 4. Integrating XMLmind XSL-FO Converter into your application

1. Compiling and running the code samples

All the code samples used to illustrate this document are found in the samples/dotnet/ subdirectory.

- Compile the three samples by executing **`nmake`** in the `samples/dotnet/` directory.

If you are using Mono on Linux rather a Microsoft .NET framework on Windows, please run `make -f mono.mak` and not `nmake`.

- Run the first sample by executing **`nmake`** `tsample1` in the `samples/dotnet/` directory.
- Run the second sample by executing **`nmake`** `tsample2` in the `samples/dotnet/` directory.
- Run the third sample by executing **`nmake`** `icographic` in the `samples/dotnet/` directory.

2. Converting an XSL-FO file to RTF

This first sample consists in a single step: invoke XMLmind XSL-FO Converter to convert the input XSL-FO file to RTF.

Note that converting XSL-FO to other formats is simply a matter of changing the value of the OutputFormat property. The possible values for this property are: `OutputFormat.Rtf`, `OutputFormat.Wml`, `OutputFormat.Docx`, `OutputFormat.Odt`.

Excerpts of samples/dotnet/Sample1.cs:

```
using XmlMind.FoConverter;

...
            Converter converter = new Converter();❶
            converter.OutputFormat = OutputFormat.Rtf;❷
            converter.OutputEncoding = "windows-1252";
            converter.ImageResolution = 120;

            String inUri = ToUri(inPath);
            converter.SetInput(inUri);❸
            converter.SetOutput(outPath);❹
            converter.Convert();❺
...
```

❶ Create a new Converter object.

❷ Parameterize the `Converter` using some of its properties.

Note that specifying property `OutputEncoding` is really useful only in the case of the RTF format. All the other formats are XML-based and thus, the default value of `OutputEncoding`, generally `UTF-8`, should work fine in all cases.

❸ Specify the input source of the `Converter` using Converter.SetInput.

Here we use the most high-level specification: we specify a (%-escaped) URI. In production, you'll generally specify a `Stream`, `TextReader` or `XMLReader`. Note that when you'll specify a `Stream`, `TextReader` or `XMLReader`, the `Converter` will not automatically close it at the end of the conversion. You'll have to do that yourself. The rule here is: the code which has opened a `Stream`, `TextReader` or `XMLReader` has the responsibility to close it.

`ToURI` is a simple helper function implemented as follows:

```
private static string ToUri(String fileName)
{
    Uri uri = new Uri(Path.GetFullPath(fileName));
    return uri.AbsoluteUri;
}
```

❹ Specify the output destination of the `Converter` using Converter.SetOutput.

Here we use the most high-level specification: we specify a file path. In production, you'll generally specify a `Stream` or a `TextWriter`. As explained before, when you'll specify a `Stream` or a `TextWriter`, the `Converter` will not automatically close it at the end of the conversion.

❺ Perform the conversion by invoking Converter.Convert.

3. Converting an XML document to RTF

This second sample consists in three steps:

1. Compile the XSLT style sheet for all subsequent uses.
2. Invoke the XSLT engine to convert the input XML document to XSL-FO.
3. Invoke XMLmind XSL-FO Converter to convert the temporary XSL-FO file generated by second step to RTF.

Excerpts of samples/dotnet/Sample2.cs:

```
using System.Xml.Xsl;
using XmlMind.FoConverter;

...
            XslCompiledTransform transform = new XslCompiledTransform();
            transform.Load(ToUri(xslPath));❶

            string xmlUri = ToUri(xmlPath);
            foPath = Path.GetTempFileName();
```

```
            transform.Transform(xmlUri, foPath);❷

            Converter converter = new Converter();❸
            converter.OutputFormat = OutputFormat.Rtf;
            converter.OutputEncoding = "windows-1252";
            converter.ImageResolution = 72;
            converter.BaseUrl = xmlUri;❹

            converter.SetInput(ToUri(foPath));
            converter.SetOutput(rtfPath);
            converter.Convert();❺
...
```

❶ Compile the XSLT style sheet.

About the thread safety of XMLmind XSL-FO Converter

A `Converter` instance must not be shared between different threads.

❷ Transform the XML input file to a temporary output file created in the system-dependant temporary file directory.

❸ Create and parameterize a Converter object as explained in Section 2, “Converting an XSL-FO file to RTF” [8].

❹ Setting the BaseUrl property to the URL of the XML input file is really needed in our case:

If the XML input file references graphics files using relative URLs (example: `images/screen-shot1.png`), then the generated XSL-FO file is likely to contain `fo:external-graphic` objects referencing the same graphics files using the same relative URLs. The problem is that, in our case, the XSL-FO file is not generated in the same directory as the XML input file. Therefore, without the `BaseUrl` property, these relative URLs would be resolved incorrectly by XMLmind XSL-FO Converter.

An advanced alternative to specifying a `BaseUrl` property, is to specify an IUriResolver object using Converter.SetUriResolver.

❺ Perform the conversion by invoking Converter.Convert.

4. Implementing a custom `IGraphicFactory` and registering it with XMLmind XSL-FO Converter

We'll use the support of `.ico` files — Windows native icons — as an example of extending the graphic capabilities of XMLmind XSL-FO Converter.

Implementing a IGraphicFactory is straightforward. You just need to implement 4 methods: `GetInput-Formats`, `GetOutputFormats`, `CreateGraphic` and `ConvertGraphic`.

Excerpts of samples/dotnet/IcoGraphicFactory.cs:

```
...
using XmlMind.FoConverter;

public class IcoGraphicFactory : IGraphicFactory
{
    private static readonly string[] inputFormats = {
        "image/vnd.microsoft.icon"
    };
    private static readonly string[] outputFormats = {
        "image/png"
    };

    public string[] GetInputFormats()❶
    {
        return inputFormats;
    }

    public string[] GetOutputFormats()❷
    {
        return outputFormats;
    }
    ...
```

❶ GetInputFormats returns the list of the media types (AKA MIME types) that the `IGraphicFactory` can read.

❷ GetOutputFormats returns the list of the media types that the `IGraphicFactory` can write. In order to be useful to XMLmind XSL-FO Converter, a factory must return one or more of "`image/png`", "`image/x-wmf`", "`image/x-emf`".

```
    ...
    public IGraphic CreateGraphic(string location, string format,
                                  object clientData, IGraphicEnv env)❶
    {
        Image image = LoadImage(location);

        double xRes = 0;
        double yRes = 0;
        if ((image.Flags & ((int) ImageFlags.HasRealDpi)) != 0) {
            xRes = image.HorizontalResolution;
            yRes = image.VerticalResolution;
        }

        return new Graphic(location, format, image.Width, image.Height,
                           xRes, yRes, GraphicType.Raster, clientData);❷
    }

    private static Image LoadImage(String location) {
        Image image = null;

        Stream stream = GraphicUtil.OpenStream(location);❸
        try {
```

```
            image = Image.FromStream(stream);
        } finally {
            stream.Close();
        }

        return image;
    }
    ...
```

❶ Method CreateGraphic basically needs to parse the image file found at absolute URI *`location`* and having *`format`* as its media type. This method then returns an implementation of interface IGraphic which represents the parsed image file.

Note that argument *`format`* is guaranteed to be one the media types listed by `GetInputFormats`.

❷ Class Graphic is a simple implementation of interface `IGraphic`.

❸ In order to obtain the dimension of the image (`width` and `height` in pixels and possibly `xResolution` and `yResolution` in DPI), we have chosen in this simple example to fully load the image into memory. To do that, we use the GraphicUtil.OpenStream helper function. This helper not only supports ``normal URIs" (that is, starting with "`file:`", "`http:`", etc) but also "`data:`" URIs.

```
    ...
    public IGraphic ConvertGraphic(IGraphic graphic, string format,
                                   double xScale, double yScale,
                                   object clientData, IGraphicEnv env)❶
    {
        int width = graphic.GetWidth();
        int height = graphic.GetHeight();

        double xRes = graphic.GetXResolution();
        double yRes = graphic.GetYResolution();

        Image image = LoadImage(graphic.GetLocation());

        if (xScale != 1) {
            width = (int) Math.Round(width * xScale);
        }
        if (yScale != 1) {
            height = (int) Math.Round(height * yScale);
        }

        Bitmap bitmap = new Bitmap(image, width, height);

        if (xRes > 0 && yRes > 0) {
            bitmap.SetResolution((float) xRes, (float) yRes);
        }

        string outPath = env.CreateTempFile(".png");❷
        bitmap.Save(outPath, ImageFormat.Png);

        return new Graphic(GraphicUtil.FilenameToLocation(outPath), format,❸
```

```
                        width, height, xRes, yRes,
                        GraphicType.Raster, clientData);
    }
    ...
```

❶ Method ConvertGraphic is invoked to convert its *graphic* argument (previously created using `CreateGraphic`) to the *format* media type. This method then returns an implementation of interface IGraphic which represents the converted image file.

Note that argument *format* is guaranteed to be one the media types listed by `GetOutputFormats`.

❷ The converted image file must be stored in a temporary file created using method IGraphicEnv.CreateTempFile. Such temporary files are automatically deleted when no longer needed.

❸ Class GraphicUtil contains several useful helper functions, among them `FilenameToLocation` which converts a filename to a "`file:`" URI.

```
    public static int Main(string[] args)
    {
        ...
        GraphicFactories.Register(new IcoGraphicFactory());❶
        ...
            converter.SetInput(inUri);
            converter.SetOutput(outPath);
            converter.Convert();
        ...
```

❶ For an implementation of `IGraphicFactory` to be used by XMLmind XSL-FO Converter, this class must be registered using GraphicFactories.Register. In this simple example, we do that in the `Main` method, prior to invoking `Converter.Convert`.

Chapter 5. Support of the XSL-FO v1.0 standard

1. Features

XFC preserves the structure of source documents, as well as most of the presentation information. Below is a list of key features of **XFC**.

- Paragraph attributes

  Most paragraph attributes (e.g. indentation) are supported. Vertical spacing is handled reasonably in most cases.

- Font attributes

  Most font attributes (family, size, weight, etc) are supported.

About the `font-family` property

When the `font-family` property contains a list of several font families, it's *always the first* font family which is used by **XFC**. Example: `font-family= "'FF Trixie', 'Andale Mono',monospace"`, the font used by **XFC** is "`FF Trixie`" (a very uncommon font indeed).

What happens when this font family is absent from the platform where the file generated by **XFC** is used? The answer is: the word processor will automatically substitute another font for it. However for this font substitution to work well, the font family being referenced in the generated file must have been properly declared.

XFC uses the generic font family name (`serif`, `sans-serif`, `monospace`, `fantasy`, `cursive`) possibly found in the list to properly declare the font being used.

In the above example, the font used by **XFC** is "`FF Trixie`" and because the list contains `monospace`, "`FF Trixie`" is declared to be a “modern” font having a fixed pitch.

Note that when the `font-family` property does not contain any generic font family name, **XFC** will nevertheless try to properly declare the font being used. It does so by searching its own internal set of known fonts for the font being used. For example, **XFC** knows that "`Andale Mono`" is equivalent to a `monospace` font and as such, it will declare it as being a “modern” font having a fixed pitch.

- Lists

  XFC automatically tries to infer the numbering style from the label of the first list item. Both bulleted and numbered lists are supported. Nested lists are supported.

  When the heuristics used by **XFC** are insufficient to infer the type of a list, it's still possible to explicitly specify this type by adding an `xfc:label-format` proprietary attribute [29] to the `fo:list-block`.

  When the heuristics used by **XFC** are insufficient to infer the type of a list and the `xfc:label-format` attribute is absent from the `fo:list-block`, then list items are output as plain paragraphs. That is,

the list items look as expected, but will not behave as proper list items when edited in MS-Word or OpenOffice.org.

- Tables

  XFC supports both the fixed and automatic table layout, as well as the two border models defined in the W3C recommendation. The implementation of the collapsing border model does not strictly conforms to the CSS2 specification, but should give the expected result in most cases.

- Images

  Out of the box, **XFC** supports WMF, EMF, BMP (only .NET version and Java™ 1.5+), TIFF (only .NET version and Java™ with `jai_imageio.jar` in the `CLASSPATH`), GIF, JPEG and PNG graphics.

  Implementing the public, documented, `Graphic` and `GraphicFactory` interfaces (`IGraphic` and `IGraphicFactory` for the .NET version) allows third-party programmers to add support for even more graphic formats.

- Embedded foreign XML

  The XML content of a `fo:instream-foreign-object` element is now passed to the proper `GraphicFactory`. For this to work, the `fo:instream-foreign-object` element must have a `content-type` attribute containing a *media type* supported by a registered `GraphicFactory`.

  Note that `content-type` ``sniffing" is implemented only for SVG and MathML and that `content-type` attributes starting with "`namespace-prefix:`" are completely ignored.

- Headers and footers

  `static-content` elements associated with the `before` and `after` regions are converted to page headers and footers respectively.

- Page references

  Page references (`page-number-citation` elements) are supported.

- Hypertext links

  Both internal and external links are supported.

For a complete list of supported objects/properties, see the conformance statement [16].

In addition, **XFC** supports an number of proprietary and yet very useful, extensions to the XSL-FO standard:

- The aforementioned `xfc:label-format` extension attribute [29].
- Extensions attributes [30] allowing to control the rendering of `fo:leader`.
- The `xfc:outline-level` extension attribute [31].
- An XSL-FO extension for generating Structured Document Tags [59] (**SDT**) in Office Open XML (`.docx`) documents. This extension makes it possible producing simple forms which can be loaded and filled in MS-Word 2007+.

- Last but not least, an XSL-FO extension for generating named styles [44]. Using the `xfc:user-style` extension attribute, it becomes possible to generate RTF, WordprocessingML, Office Open XML (`.docx`) and OpenOffice (`.odt`) files where most of the text formatting is achieved using *named paragraph styles* ("`Normal`", "`Heading 1`", "`Heading 2`", etc) and *named character styles* ("`Strong`", "`Emphasis`", etc).

2. Limitations

Though **XFC** implements the greater part of the W3C recommendation, it does not support all XSL-FO features. Below is a list of the current major limitations of **XFC**.

- The `leader` element is only partly supported.
- The `float` and `marker` elements are not supported.
- The `writing-mode` property is not supported (value `lr-tb` is assumed).

The conformance level of **XFC** [16] may be improved in future versions, however it must be stressed that a full conformance cannot be achieved due to the own limitations of its output formats.

3. Conformance statement

The W3C Extensible Stylesheet Language (XSL) v1.0 Recommendation defines three levels of conformance for an XSL-FO processor: basic, extended and complete. Since XMLmind XSL-FO Converter currently does not conform to any of these levels, this document provides a complete list of supported objects/properties, along with additional information for objects/properties that are not fully supported.

In the following tables, the background color (white, light green or green) of each entry in the tables below indicates the level of conformance (basic, extended or complete) of that particular object/property, as specified by the Recommendation.

Table 5.1. XSL-FO objects

Object	Supported	Comments
Declarations and Pagination and Layout Formatting Objects		
root	yes	
declarations	no	
color-profile	no	
page-sequence	yes	
layout-master-set	yes	
page-sequence-master	yes	
single-page-master-reference	yes	
repeatable-page-master-reference	yes	
repeatable-page-master-alternatives	yes	
conditional-page-master-reference	yes	Limited support. See Section 4.11, “Multiple page layouts” [41] for further information.

Object	Supported	Comments
simple-page-master	yes	
region-body	yes	
region-before	yes	
region-after	yes	
region-start	no	Output format limitation.
region-end	no	Output format limitation.
flow	yes	
static-content	yes	Supported regions: body, before and after.
title	no	
Block-level Formatting Objects		
block	yes	Not supported inside inline-level objects (output format limitation).
block-container	limited	May be used with attribute `reference-orientation` to temporarily switch the page orientation from portrait to landscape [32] or to rotate the content of a `table-cell` [33]. Otherwise, ignored.
Inline-level Formatting Objects		
bidi-override	no	
character	no	
initial-property-set	no	
external-graphic	yes	Supported image formats: WMF, EMF, BMP (.NET version and Java™ 1.5+), TIFF (.NET version and Java™ with `jai_imageio.jar` in the `CLASSPATH`) GIF, JPEG and PNG. Optionally the Java™ (v1.5+) version also supports SVG and MathML.
instream-foreign-object	yes	The XML content of a `fo:instream-foreign-object` element is passed to the proper `GraphicFactory`. For this to work, the `fo:instream-foreign-object` element must have a `content-type` attribute containing a media type supported by a registered `GraphicFactory`. Note that `content-type` ``sniffing'' is implemented only for SVG and MathML and that `content-type` attributes starting with "`namespace-prefix:`" are completely ignored.
inline	yes	Cannot contain block-level objects (output format limitation).

Object	Supported	Comments
inline-container	no	
leader	yes	Limited support (most properties ignored). See Section 4.3, “Leaders” [30] for further information.
page-number	yes	
page-number-citation	yes	
Formatting Objects for Tables		
table-and-caption	yes	Not supported inside inline-level objects (output format limitation).
table	yes	
table-column	yes	
table-caption	yes	
table-header	yes	
table-footer	yes	
table-body	yes	
table-row	yes	
table-cell	yes	
Formatting Objects for Lists		
list-block	yes	Not supported inside inline-level objects (output format limitation).
list-item	yes	
list-item-body	yes	
list-item-label	yes	Multiple block-level descendants not supported.
Link and Multi Formatting Objects		
basic-link	yes	Can only contain text and inline-level objects.
multi-switch	no	
multi-case	no	
multi-toggle	no	
multi-properties	no	
multi-property-set	no	
Out-of-line Formatting Objects		
float	no	
footnote	yes	
footnote-body	yes	
Other Formatting Objects		
wrapper	yes	

Object	Supported	Comments
marker	no	
retrieve-marker	no	

Table 5.2. XSL-FO properties

Property	Supported	Comments
Common Accessibility Properties		
source-document	no	
role	no	Supported on `fo:external-graphic` and `fo:instream-foreign-object`.
Common Absolute Position Properties		
absolute-position	no	
top	no	
right	no	
bottom	no	
top	no	
Common Aural Properties		
azimuth	n/a	
cue-after	n/a	
cue-before	n/a	
elevation	n/a	
pause-after	n/a	
pause-before	n/a	
pitch	n/a	
pitch-range	n/a	
play-during	n/a	
richness	n/a	
speak	n/a	
speak-header	n/a	
speak-numeral	n/a	
speak-punctuation	n/a	
speech-rate	n/a	
stress	n/a	
voice-family	n/a	
volume	n/a	
Common Border, Padding and Background Properties		
background-attachment	no	

Property	Supported	Comments
background-color	yes	
background-image	no	May be used to add a watermark to the generated document. See Section 4.12, “Adding a watermark to the generated document” [42].
background-repeat	no	
background-position-horizontal	no	May be used to add a watermark to the generated document. See Section 4.12, “Adding a watermark to the generated document” [42].
background-position-vertical	no	
border-before-color	yes	• Not supported on block-level objects that contain other block-level objects (output format limitation). • Not supported on `inline` objects that contain other objects (output format limitation).
border-before-style	yes	
border-before-width	yes	
border-after-color	yes	
border-after-style	yes	
border-after-width	yes	
border-start-color	yes	
border-start-style	yes	
border-start-width	yes	
border-end-color	yes	
border-end-style	yes	
border-end-width	yes	
border-top-color	yes	• Not supported on block-level objects that contain other block-level objects (output format limitation). • Not supported on `inline` objects that contain other objects (output format limitation). **ODT** output format: borders and padding around a text span are *not* supported by OpenOffice and by old versions (< v5) of LibreOffice.
border-top-style	yes	
border-top-width	yes	
border-bottom-color	yes	
border-bottom-style	yes	
border-bottom-width	yes	
border-left-color	yes	
border-left-style	yes	
border-left-width	yes	
border-right-color	yes	
border-right-style	yes	
border-right-width	yes	
padding-before	yes	• Not supported on block-level objects that contain other block-level objects (output format limitation).
padding-after	yes	
padding-start	yes	

Property	Supported	Comments
padding-end	yes	• Not supported together with `border-*-style="none"` or `border-*-style="hidden"` (output format limitation).
padding-top	yes	• Not supported on block-level objects that contain other block-level objects (output format limitation). • Not supported together with `border-*-style="none"` or `border-*-style="hidden"` (output format limitation).
padding-bottom	yes	
padding-left	yes	
padding-right	yes	
Common Font Properties		
font-family	yes	
font-selection-strategy	no	
font-size	yes	
font-stretch	no	
font-size-adjust	no	
font-style	yes	Value `backslant` not supported (output format limitation).
font-variant	yes	
font-weight	yes	
Common Hyphenation Properties		
country	yes	See language [21] below.
language	yes	For attribute `language` and, optionally, attribute `country` (or equivalently, `xml:lang` [28]) to be considered to generate information for use by the word processor, attribute `language` (or equivalently, `xml:lang` [28]) must be specified at least on the `fo:root` element. More information in Section 4.6, “Adding language information to the documents created by **XFC**” [34].
script	no	
hyphenate	no	
hyphenation-character	no	
hyphenation-push-character-count	no	
hyphenation-remain-character-count	no	
Common Margin Properties - Block		
margin-top	yes	Percentages and value `auto` not supported.
margin-bottom	yes	

Property	Supported	Comments
margin-left	yes	
margin-right	yes	
space-before	yes	Conditionality not supported.
space-after	yes	
start-indent	yes	Percentages not supported.
end-indent	yes	
Common Margin Properties - Inline		
space-end	no	
space-start	no	
Common Relative Position Properties		
relative-position	no	
Area Alignment Properties		
alignment-adjust	no	
alignment-baseline	no	Values `middle`, `before-edge` and `after-edge` supported on `fo:external-graphic` and `fo:instream-foreign-object`.
baseline-shift	yes	
display-align	no	Supported on `fo:table-cell`, `fo:external-graphic` and `fo:instream-foreign-object`.
dominant-baseline	no	
relative-align	no	
Area Dimension Properties		
block-progression-dimension	no	
content-height	yes	The following XSL-FO 1.1 property values: `scale-down-to-fit`, `scale-up-to-fit` are also supported.
content-width	yes	The following XSL-FO 1.1 property values: `scale-down-to-fit`, `scale-up-to-fit` are also supported.
height	no	Supported on `fo:table-row`, `fo:external-graphic` and `fo:instream-foreign-object`.
inline-progression-dimension	no	
max-height	no	
max-width	no	
min-height	no	
min-width	no	
scaling	yes	
scaling-method	no	

Property	Supported	Comments
width	no	Supported on `fo:table`, `fo:external-graphic` and `fo:instream-foreign-object`.
Block and Line-related Properties		
hyphenation-keep	no	
hyphenation-ladder-count	no	
last-line-end-indent	no	Output format limitation.
line-height	yes	Value type `space` not supported.
line-height-shift-adjustment	no	
line-stacking-strategy	no	
linefeed-treatment	yes	
text-align	yes	Values `inside` and `outside` and value type `string` not supported.
text-align-last	no	Output format limitation.
text-indent	yes	Percentages not supported.
white-space-collapse	yes	
white-space-treatment	yes	
wrap-option	no	
Character Properties		
character	no	
letter-spacing	no	
suppress-at-line-break	no	
text-decoration	yes	In addition to the decoration type (`underline`, `overline`, `line-through`, etc), it's possible to specify the color, style (`solid`, `double`, `dotted`, `dashed`, `wavy`) and thickness of the text decoration. See Section 4.14, “Non-standard extension of XSL-FO property `text-decoration`” [42].
text-shadow	no	
text-transform	no	
treat-as-word-space	no	
word-spacing	no	
Color-related Properties		
color	yes	
color-profile-name	no	
rendering-intent	no	
Float-related Properties		
clear	no	

Property	Supported	Comments
float	no	
intrusion-displace	no	
Keeps and Breaks Properties		
break-after	yes	
break-before	yes	
keep-together	yes	Not supported on block-level objects that contain other block-level objects.
keep-with-next	yes	Not supported on block-level objects that contain other block-level objects.
keep-with-previous	no	
orphans	yes	Remember that Window/Orphan control is turned on by default as the initial value of the orphans and widows properties is 2. Also note that for MS-Word, Window/Orphan control is an all or nothing option. Therefore if you set attribute orphans *or* attribute widows to 1, Window *and* Orphan control will be turned off. If, on the contrary, you set attribute orphans *or* attribute widows to any value greater or equal than 2, Window *and* Orphan control will be turned on. Unlike MS-Word, OpenOffice/LibreOffice fully supports the orphans and widows properties.
widows	yes	
Layout-related Properties		
clip	no	
overflow	no	
reference-orientation	limited	May be used on `fo:block-container` to temporarily switch the page orientation from portrait to landscape [32] or to rotate the content of a `table-cell` [33]. Otherwise, ignored.
span	no	
Leader and Rule Properties		
leader-alignment	no	
leader-pattern	yes	Value `use-content` not supported.
leader-pattern-width	no	
leader-length	no	
rule-style	yes	Supported values: `none`, `dotted` and `solid`.

Property	Supported	Comments
rule-thickness	no	
Properties for Dynamic Effects Formatting Objects		
active-state	no	
auto-restore	no	
case-name	no	
case-title	no	
destination-placement-offset	no	
external-destination	yes	
indicate-destination	no	
internal-destination	yes	
show-destination	no	
starting-state	no	
switch-to	no	
target-presentation-context	no	
target-processing-context	no	
target-stylesheet	no	
Properties for Markers		
marker-class-name	no	
retrieve-class-name	no	
retrieve-position	no	
retrieve-boundary	no	
Properties for Number to String Conversion		
format	yes	
grouping-separator	no	
grouping-size	no	
letter-value	no	
Pagination and Layout Properties		
blank-or-not-blank	no	
column-count	yes	
column-gap	yes	
extent	no	
flow-name	yes	Values `xsl-before-float-separator` and `xsl-footnote-separator` not supported.
force-page-count	no	
initial-page-number	yes	
master-name	yes	

Property	Supported	Comments
master-reference	yes	
maximum-repeats	no	
media-usage	no	
odd-or-even	yes	
page-height	yes	
page-position	yes	Value `last` not supported.
page-width	yes	
precedence	no	
region-name	yes	
Table Properties		
border-after-precedence	no	
border-before-precedence	no	
border-collapse	yes	Value `collapse-with-precedence` not supported.
border-end-precedence	no	
border-separation	yes	
border-start-precedence	no	
caption-side	yes	Values `start`, `end`, `left` and `right` not supported (output format limitation).
column-number	yes	
column-width	yes	
empty-cells	no	
ends-row	yes	
number-columns-repeated	yes	
number-columns-spanned	yes	
number-rows-spanned	yes	
starts-row	yes	
table-layout	yes	
table-omit-footer-at-break	no	
table-omit-header-at-break	no	
Writing-mode-related Properties		
direction	no	Value `ltr` assumed.
glyph-orientation-horizontal	no	
glyph-orientation-vertical	no	
text-altitude	no	
text-depth	no	

Property	Supported	Comments
unicode-bidi	no	
writing-mode	no	Value `lr-tb` assumed.
Miscellaneous Properties		
content-type	yes	
id	yes	
provisional-label-separation	yes	
provisional-distance-between-starts	yes	
ref-id	yes	
score-spaces	no	
src	yes	
visibility	no	
z-index	no	
Shorthand Properties		
background	no	Background color specification supported.
background-position	no	
border	yes	See restrictions on individual properties.
border-bottom	yes	
border-left	yes	
border-right	yes	
border-top	yes	
border-color	yes	
border-style	yes	
border-width	yes	
border-spacing	yes	
cue	n/a	
font	yes	
margin	yes	See restrictions on individual properties.
padding	yes	See restrictions on individual properties.
page-break-after	yes	See restrictions on individual properties.
page-break-before	yes	
page-break-inside	yes	
pause	n/a	
position	no	
size	no	Value type `length` supported.
vertical-align	no	

Property	Supported	Comments
white-space	yes	
xml:lang	yes	Shorthand for language [21] and country [21].

4. Implementation specificities

4.1. Page references

4.1.1. RTF/WML/OOXML

Page references - i.e. `page-number-citation` objects - are converted to `PageRef` fields. The values of these fields are *not* automatically updated when loading an RTF/WML/OOXML document in MS-Word. The easiest way to update all field values is to force a repagination of the document, for instance by switching to the "**Page Layou**t" view (sometimes called "**Print Layout**").

If after doing that, some fields have not been updated, for example, those found in the Table of Contents and in the Index, please proceed as follows:

1. Switch to the "**Page Layout**" view (sometimes called "**Print Layout**").
2. Type **Ctrl**+**A** (**Select all**)
3. Press **F9** (**Update fields**).

Note that there is no way to automate this. Unlike the XSL-FO processors which generate PDF (e.g. Apache FOP), a paginated format, **XFC** merely translates XSL-FO to the format internally used by a word processor (e.g. RTF, DOCX, ODT). Therefore **XFC** has no control whatsoever on page numbering. It's the word processor which, after loading the file generated by **XFC**, numbers the pages of the document.

4.1.2. OpenDocument

Page references - i.e. `page-number-citation` objects - are converted to reference fields. The values of these fields are not automatically updated when loading an OpenDocument file in OpenOffice. Select `Update->Fields` in the `Tools` menu to update the field values.

4.2. Lists

XFC automatically tries to infer the numbering style from the label of the first list item. Both bulleted and numbered lists are supported. Nested lists are supported.

When the heuristics used by XFC are insufficient to infer the type of a list, it's still possible to explicitly specify this type by adding an `xfc:label-format` extension attribute to the `fo:list-block`.

When the heuristics used by XFC are insufficient to infer the type of a list and the `xfc:label-format` attribute is absent from the `fo:list-block`, then the list items are output as plain paragraphs. That is, the list items look as expected, but will not behave as proper list items when edited in MS-Word or OpenOffice.org.

4.2.1. The `xfc:label-format` extension attribute

The `xfc:label-format` attribute must be specified on a `fo:list-block`.

The namespace of this attribute is "`http://www.xmlmind.com/foconverter/xsl/extensions`". A prefix, typically xfc, must be declared for this namespace.

The syntax of the value of this attribute is:

```
label-format -> [ bullet | number ]?

bullet -> String

number -> [String]? '%{' format '}' [String]

format -> 'decimal'|'lower-alpha'|'upper-alpha'|
          'lower-roman'|'upper-roman' [inherit]? [start]?

inherit -> ';inherit'

start -> ';start=' Positive_Integer
```

Description:

- An *empty* `xfc:label-format` attribute (e.g. `xfc:label-format=""`) is allowed. It instructs **XFC** not to use any heuristic and to convert the `fo:list-block` to plain paragraphs.
- The '`%`' character must be escaped by doubling it. Example: `%%%{decimal}`, which corresponds to `%1`, `%2`, `%3`, etc.
- The format values `decimal`, `lower-alpha`, etc, correspond to the values of the CSS `list-style-type` property.
- The `inherit` optional parameter specifies that a numbered `fo:list-block` "inherits" the numbering of its ancestor numbered `fo:list-blocks`. In other words, this parameter may be used to implement what is often called *multi-level numbering* (e.g. `1.A.a.`)

  For example, let's suppose topmost `fo:list-block` is numbered `1-`, `2-`, `3-`, etc. Let's suppose its second list item contains a nested `fo:list-block` having attribute `xfc:label-format="%{upper-alpha;inherit})"`. Then this nested list will be automatically numbered `2-A)`, `2-B)`, `2-C)`, etc.
- The `start=` optional parameter specifies the starting number of the first item in an ordered list. Its default value is 1.

Limitations

- Specifying both `inherit` and `start=N` is currently not really supported and generally gives unexpected results.
- Something like `start=continue` is currently not supported.

Example:

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format"
         xmlns:xfc="http://www.xmlmind.com/foconverter/xsl/extensions">
  ...
  <fo:list-block font-family="monospace" margin-left="10pt"
                 provisional-distance-between-starts="1cm"
                 provisional-label-separation="5pt"
                 space-before="2pt"
                 xfc:label-format="&#x2022;%{lower-roman;start=10}">
    <fo:list-item>
      <fo:list-item-label end-indent="label-end()">
        <fo:block>&#x2022;x</fo:block>
      </fo:list-item-label>

      <fo:list-item-body start-indent="body-start()">
        <fo:block>This is the first item
        of the list.</fo:block>
      </fo:list-item-body>
    </fo:list-item>

    <fo:list-item>
      <fo:list-item-label end-indent="label-end()">
        <fo:block>&#x2022;xi</fo:block>
      </fo:list-item-label>

      <fo:list-item-body start-indent="body-start()">
        <fo:block>This is the second item
        of the list.</fo:block>
      </fo:list-item-body>
    </fo:list-item>
  </fo:list-block>
  ...
```

The heuristics used by XFC corresponds to the following values of `xfc:label-format`:

- `-`, `+`, `*`, `&#x2022;` (bullet), `&#x2013;` (endash).
- `%{decimal;start=0}`, `%{decimal}`, `%{lower-alpha}`, `%{upper-alpha}`, `%{lower-roman}`, `%{upper-roman}`.
- `%{decimal;start=0}.`, `%{decimal}.`, `%{lower-alpha}.`, `%{upper-alpha}.`, `%{lower-roman}.`, `%{upper-roman}.`.
- `%{decimal;start=0})`, `%{decimal})`, `%{lower-alpha})`, `%{upper-alpha})`, `%{lower-roman})`, `%{upper-roman})`.
- `(%{decimal;start=0})`, `(%{decimal})`, `(%{lower-alpha})`, `(%{upper-alpha})`, `(%{lower-roman})`, `(%{upper-roman})`.
- `[%{decimal;start=0}]`, `[%{decimal}]`, `[%{lower-alpha}]`, `[%{upper-alpha}]`, `[%{lower-roman}]`, `[%{upper-roman}]`.
- `<%{decimal;start=0}>`, `<%{decimal}>`, `<%{lower-alpha}>`, `<%{upper-alpha}>`, `<%{lower-roman}>`, `<%{upper-roman}>`.

4.3. Leaders

For lack of a corresponding element in the output formats, `leader` objects are implemented by means of tab stops. This is not very convenient given the `leader` object specification, since there is no way for

XFC to derive the tab position from the property values. Though XFC will usually set the tab position to a reasonable value by default, this arbitrary position is unlikely to result in the intended layout.

However, the actual tab position may be specified to XFC by setting an additional property on the `leader` object. This property is named `tab-position` and must be defined in the XFC namespace (`http://www.xmlmind.com/foconverter/xsl/extensions`). The property value is a `<length>` as defined in section 5.11 of the Recommendation. A positive value specifies the tab position relative to the left margin, whereas a negative value specifies the position relative to the right margin.

An additional property named `tab-align` specifies how the content following a tab is horizontally aligned. The possible values for this property are: `left`, `center`, `right` and `decimal`. Using the `tab-align` property is optional. By default, the content following a tab is left aligned.

The code samples below are excerpts from file *`xslutil_install_dir`*`/addon/config/docbook/xsl/fo/autotoc.xsl`. They illustrate a typical use of the `tab-position` and `tab-align` properties in an XSL stylesheet.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:fo="http://www.w3.org/1999/XSL/Format"
                xmlns:xfc="http://www.xmlmind.com/foconverter/xsl/extensions"
                version='1.0'>
```

```
<fo:leader leader-pattern="dots"
           leader-pattern-width="3pt"
           leader-alignment="reference-area"
           xfc:tab-position="-30pt"
           xfc:tab-align="right"
           keep-with-next.within-line="always"/>
```

4.4. Other extension attributes

4.4.1. The `xfc:outline-level` extension attribute

Extension attribute `xfc:outline-level` may be used to mark a `fo:block` as a heading having the outline level specified by the value of the attribute. The value of this attribute is an integer between 1 and 9 inclusive. Any other value will cause attribute `xfc:outline-level` to be ignored.

Specifying outline levels allows to:

- Use the Document Map and the Outline View in MS-Word. Use the Navigator Window in OpenOffice/LibreOffice.
- Insert a Table of Contents in a document edited in MS-Word or OpenOffice/LibreOffice.

Example:

```
<fo:block font-size="22pt" space-before="22pt"
          xfc:outline-level="4" color="#406080">Heading 4</fo:block>
```

4.4.2. The `xfc:render-as-table` extension attribute

Extension attribute `xfc:outline-level` may be used to specify that a `fo:block` is to be automatically converted to an equivalent `fo:table`. The value of this attribute is `true` or `false`.

This extension attribute is a quick and easy workaround for one of the most annoying limitations of XMLmind XSL-FO Converter: a `fo:block` having a border and/or background color and containing several other blocks, lists or tables was very poorly rendered in RTF, WML, DOCX and ODT. (Such container `fo:blocks` are quite commonly used, for example, to represent a complex note, admonition or sidebar.)

The reason of this limitation is due to the fact that the RTF, WML, DOCX and ODT output formats can —to make it simple— only contain a “flat” sequence of styled paragraphs and tables.

Example:

```
<fo:block margin="0.5em 2em" padding="1em 4em"
  border="1px solid #800000" background="#FFF0F0"
  xfc:render-as-table="true">
  <fo:block space-before="0.5em" space-after="0.5em">First paragraph.</fo:block>
  <fo:block space-before="0.5em" space-after="0.5em">Second paragraph.</fo:block>
  <fo:block space-before="0.5em" space-after="0.5em">Third paragraph.</fo:block>
</fo:block>
```

Note that `xfc:render-as-table="true"` is ignored when a named style (i.e. `xfc:user-style` [44]) is used to style the `fo:block`.

When converting to RTF XSL-FO files making use of extension attribute `xfc:outline-level`, you'll almost certainly want to pass option `/w` [6] to **fo2rtf**.

4.5. Special uses of `fo:block-container`

4.5.1. Using `fo:block-container` to temporarily switch the page orientation from portrait to landscape

Element `fo:block-container` with a `reference-orientation` attribute equal to `90`, `270`, `-90` or `-270` may be used to temporarily switch the page orientation from portrait to landscape. This feature is typically used to help MS-Word or OpenOffice/LibreOffice display a wide table or a wide figure.

Example:

```
<fo:block-container reference-orientation="90">
  <fo:block>...</fo:block>
  <fo:table>...</fo:table>
</fo:block-container>
```

For this feature to work:

- The `fo:block-container` must be directly contained in the `fo:flow`[1]. Outside a `fo:flow` and a `fo:table-cell` (see below [33]), `fo:block-container` is treated like a `fo:block`.
- The value of attribute `reference-orientation` must be `90`, `270`, `-90` or `-270`. **XFC** does not make any difference between these four values to implement this feature.
- The width of current page layout must be smaller than its height. That is, the current page orientation must not be already landscape.

4.5.2. Using `fo:block-container` to rotate the content of a table cell

Element `fo:block-container` also is supported inside a `fo:table-cell`, where it may be used to rotate the content of this table cell. Outside a `fo:flow` (see above [32]) and a `fo:table-cell`, `fo:block-container` is treated like a `fo:block`.

In order to rotate the content of a table cell, the `fo:table-cell` must contain a *single* `fo:block-container` with a `reference-orientation` attribute equal to `90`, `270`, `-90` or `-270`.

Example 1: simplest, most common, case:

```
<fo:table-cell>
  <fo:block-container reference-orientation="90">
    <fo:block>Short Header</fo:block>
  </fo:block-container>
</fo:table-cell>
```

In the above case, there is generally no need to specify attribute `inline-progression-dimension` (or equivalently attribute `width`) and/or attribute `block-progression-dimension` (or equivalently attribute `height`) for the `fo:block-container` element:

- Attribute `inline-progression-dimension` is automatically given by **XFC** a value equals to the maximum width[2] of the content of the `fo:block-container`.
- Attribute `block-progression-dimension` is automatically given by **XFC** a value equals to *`N` `* 1.2 * `FS`*, when *`N`* is the number of blocks, lists or tables contained the `fo:block-container` and *`FS`* is the font size[3] of the `fo:block-container`.

Example 2: simple case:

```
<fo:table-cell>
  <fo:block-container reference-orientation="-90">
    <fo:block>Short Header</fo:block>
    <fo:block>One more line!</fo:block>
  </fo:block-container>
</fo:table-cell>
```

Given the default values assigned by **XFC** to attributes `inline-progression-dimension` and `block-progression-dimension`, the above example should be also rendered correctly.

Example 3: may require specifying attribute `block-progression-dimension` (or equivalently attribute `height`):

[1] The `fo:block-container` may also be contained in a `fo:block` itself directly contained in the `fo:flow`.
[2] That is, with no word wrap.
[3] This font is generally inherited from the ancestors of the `fo:block-container` element.

```
<fo:table-cell>
  <fo:block-container reference-orientation="90"
                      block-progression-dimension="96px">
    <fo:block><fo:external-graphic src="logo96x96.png"/>ACME Corp</fo:block>
  </fo:block-container>
</fo:table-cell>
```

Example 4: requires specifying both attribute `inline-progression-dimension` (or equivalently attribute `width`) and attribute `block-progression-dimension` (or equivalently attribute `height`):

```
<fo:table-cell>
  <fo:block-container reference-orientation="270"
                      inline-progression-dimension="15em"
                      block-progression-dimension="5cm">
    <fo:block>Quite long header possibly containing
    several lines of text. (Note that a fo:block-container
    is not limited to a single fo:block or even to
    fo:blocks.)</fo:block>
  </fo:block-container>
</fo:table-cell>
```

Word processor bugs related to rotating the content of a table cell

- OpenOffice/LibreOffice only supports the simplest case, like in above example 1.
- Microsoft Word 2007/2010/2013, `.docx` format: if the content of `fo:block-container` contains an image, then the position of this image is incorrect for a `reference-orientation` attribute equal to `90` or `-270`. There is no such issue with the RTF and WordprocessingML file formats and with Microsoft Word 2003+Microsoft Office Compatibility Pack, whatever the file format.

4.6. Adding language information to the documents created by XFC

Without this information, the word processor thinks that the document is entirely written in its default language; which may be very annoying when this is not the case (false errors reported by the spell checker).

For attribute `language` and, optionally, attribute `country` (or equivalently, `xml:lang`) to be considered to generate information for use by the word processor, attribute `language` (or equivalently, `xml:lang`) **must be specified at least on the `fo:root` element**.

Other limitations:

- Will not work for right-to-left languages (e.g. `ar`, `he`).
- Attribute `script` is ignored, as well as `xml:lang` values including script information (e.g. `sr-Latn-RS`).
- Use the two-letter ISO 639-1 code of a language if this code exists (e.g. `en`, `fr`, `de`, `es`), otherwise use the 3-letter ISO 639-2 code (e.g. `fil`, `tzm`, `sah`).

- Always use the two-letter ISO 3166 code of a country (e.g. `GB`, `BE`, `AT`, `AR`).

For East Asian language (e.g. `zh`, `ja`, `ko`) detection by MS-Word to work on a Windows computer having a Western locale,

- you must select "**Region and Language Options**" from Windows Control Panel and check "**Install files for East Asian languages**";
- you may have to use a font having East Asian glyphs (e.g. "`MS Gothic`") for the text runs containing East Asian characters.

4.7. Adding metadata to the documents created by XFC

Element `xfc:document-information` may be used to to add *metadata*[4] to the documents created by **XFC**[5]. This element is expected to be a child element of standard XSL-FO element `fo:declarations`.

```
<xfc:document-information>
  Content: [ xfc:meta ]*
</xfc:document-information>

<xfc:meta
  name = non empty string
  content = string
/>
```

Example:

```
<xfc:document-information>
  <xfc:meta name="xfc:creator" content="Fox Mulder" />
  <xfc:meta name="xfc:created" content="1993-09-10" />
  <xfc:meta name="xfc:keywords"
            content="extraterrestrial life, abduction, supernatural" />
  <xfc:meta name="is_classified" content="true" />
</xfc:document-information>
```

It's also possible to restrict editing in the documents created by **XFC** using command-line arguments `/meta` *`name value`*.

When both element `xfc:document-information` and the aforementioned command-line argument are specified, it's the command-line argument which is used. In the case of the above `xfc:document-information` example, `/meta is_classified false` may be used to replace the `is_classified` custom metadata.

The attributes of element `xfc:meta` are:

`name`
: Required. The name of the metadata. This may be the name of a standard metadata (e.g. `xfc:creator`) or a custom metadata (e.g. `is_classified`).

`content`
: Required. The value of the metadata.

[4] Data stored in a document about the document, e.g. the usual author, title, date, etc, but also custom metadata.
[5] This is the **XFC** equivalent of MS-Word "**File** → **Info** → **Properties**, **Advanced Properties**".

4.7.1. Standard metadata

A standard metadata has a generic name (always starting with "`xfc:`") which, *when supported by the output format*, is translated to a “native”, case-sensitive, metadata name. For example, "`xfc:creator`" is translated to **DOCX** "`dc:creator`", **WML** "`Author`", **RTF** "`author`" and **ODT** "`meta:initial-creator`".

Table 5.3. Standard metadata

Generic name	Type	Description
`xfc:category`	String	A categorization of the content of the document.
`xfc:contentStatus`	String	The status (e.g. "**Draft**", "**Final**") of the document.
`xfc:created`	Date	The date of creation of the document.
`xfc:creator`	String	The initial author of the document.
`xfc:description`	String	An explanation of the content of the document.
`xfc:identifier`	String	An unambiguous reference to the document within a given context (e.g. **ISBN**, **URN**).
`xfc:keywords`	String	Comma-separated set of keywords to support searching and indexing.
`xfc:language`	String	The code (e.g ISO 639-1) of main language of the document.
`xfc:lastModifiedBy`	String	The user who performed the last modification.
`xfc:lastPrinted`	Date	The date of the last printing.
`xfc:modified`	Date	The date on which the document was changed.
`xfc:revision`	Positive integer	The revision number (e.g the number of saves).
`xfc:subject`	String	The topic of the content of the document.
`xfc:title`	String	The title of the document.
`xfc:version`	String	The version number of the document.
`xfc:manager`	String	The manager of the author of the document.
`xfc:company`	String	The company that employs the author of the document.
`xfc:final`	Boolean: `true` or `false`	If `true`, the author lets anyone who opens the document know that there aren't going to be any more changes made to it. This also makes the document read-only.

Supported date formats are documented in "*W3C Note on Date and Time Formats [W3CDTF]*". Examples: `2020`, `2020-09`, `2020-09-16`, `2020-09-16T19:20`, `2020-09-16T17:20:30Z`, `2020-09-16T19:20:30.45+02:00`.

It is of course possible to use the “native” name of a standard metadata rather than its generic name. When both names are specified (e.g. `xfc:creator="John Doe"` and **DOCX** `dc:creator="Jane Doe"`), it is the value specified using the native name which is stored in the generated document (e.g. `dc:creator="Jane Doe"`).

*Table 5.4. Standard metadata supported by the **DOCX** output format*

Generic name	Native name *(case sensitive)*
xfc:category	category
xfc:contentStatus	contentStatus
xfc:created	dcterms:created
xfc:creator	dc:creator
xfc:description	dc:description
xfc:identifier	dc:identifier
xfc:keywords	keywords
xfc:language	dc:language
xfc:lastModifiedBy	lastModifiedBy
xfc:lastPrinted	lastPrinted
xfc:modified	dcterms:modified
xfc:revision	revision
xfc:subject	dc:subject
xfc:title	dc:title
xfc:version	version
xfc:manager	Manager
xfc:company	Company
xfc:final	_MarkAsFinal

*Table 5.5. Standard metadata supported by the **WML** output format*

Generic name	Native name *(case sensitive)*
xfc:category	Category
xfc:contentStatus	Not a standard metadata.
xfc:created	Created
xfc:creator	Author
xfc:description	Description
xfc:identifier	Guid
xfc:keywords	Keywords
xfc:language	Not a standard metadata.
xfc:lastModifiedBy	LastAuthor
xfc:lastPrinted	LastPrinted
xfc:modified	LastSaved
xfc:revision	Revision
xfc:subject	Subject
xfc:title	Title

Generic name	Native name *(case sensitive)*
`xfc:version`	`Version` (must match regular expression `"([0-9]?[0-9].[0-9]{4})|([0-9]?[0-9])"`)
`xfc:manager`	`Manager`
`xfc:company`	`Company`
`xfc:final`	`_MarkAsFinal` (not supported by MS-Word 2003)
No generic name.	`AppName`, the name of the application that created the document.

*Table 5.6. Standard metadata supported by the **RTF** output format*

Generic name	Native name *(case sensitive)*
`xfc:category`	`category`
`xfc:contentStatus`	Not a standard metadata.
`xfc:created`	`creatim`
`xfc:creator`	`author`
`xfc:description`	`doccomm`
`xfc:identifier`	Not a standard metadata.
`xfc:keywords`	`keywords`
`xfc:language`	Not a standard metadata.
`xfc:lastModifiedBy`	`operator`
`xfc:lastPrinted`	`printim`
`xfc:modified`	`revtim`
`xfc:revision`	Not a standard metadata.
`xfc:subject`	`subject`
`xfc:title`	`title`
`xfc:version`	Not a standard metadata.
`xfc:manager`	`manager`
`xfc:company`	`company`
`xfc:final`	`_MarkAsFinal` (not supported by MS-Word 2003)
No generic name.	`comment`, comments; text is ignored.
No generic name.	`buptim`, the date/time [36] of last backup.

*Table 5.7. Standard metadata supported by the **ODT** output format*

Generic name	Native name *(case sensitive)*
`xfc:category`	Not a standard metadata.
`xfc:contentStatus`	Not a standard metadata.
`xfc:created`	`meta:creation-date`
`xfc:creator`	`meta:initial-creator`

Generic name	Native name *(case sensitive)*
`xfc:description`	`dc:description`
`xfc:identifier`	Not a standard metadata.
`xfc:keywords`	`meta:keywords`
`xfc:language`	`dc:language`
`xfc:lastModifiedBy`	`dc:creator`
`xfc:lastPrinted`	`meta:print-date`
`xfc:modified`	`dc:date`
`xfc:revision`	`meta:editing-cycles`
`xfc:subject`	`dc:subject`
`xfc:title`	`dc:title`
`xfc:version`	Not a standard metadata.
`xfc:manager`	Not a standard metadata.
`xfc:company`	Not a standard metadata.
`xfc:final`	Emulated using the `read-only` restriction. See below [39].
No generic name.	`meta:generator`, a string that identifies the application or tool that was used to create or last modify the document.
No generic name.	`meta:printed-by`, the name of the last person who printed the document.
No generic name.	`meta:editing-duration`, the total time spent editing the document. Duration format is: "`PnYnMnDTnHnMnS`".

4.7.2. Custom metadata

A metadata having a non standard native name (e.g. "`is_classified`") or a standard generic name not supported by the output format (e.g. "`xfc:manager`" not supported by **ODT**) is considered to be a custom metadata.

A custom metadata is generally stored as a typed value. Supported types are generally: boolean, number, date and string. So when you want to specify a boolean, make sure to specify `true` or `false` and when you want to specify a date, make sure to use one of the formats [36] documented in "*W3C Note on Date and Time Formats [W3CDTF]*".

4.8. Restricting editing in the documents created by XFC

Element `xfc:document-protection` may be used to control the type of changes which can be made to the documents created by **XFC**[6]. This element is expected to be a child element of standard XSL-FO element `fo:declarations`.

```
<xfc:document-protection
  restrictions = Restrictions
  password = string
```

[6]This is the **XFC** equivalent of MS-Word "**File** → **Info** → **Protect Document**, **Restrict Editing**".

```
/>

Restrictions = [ limited-formatting ]?
  [ read-only|comments-only|tracked-changes-only|fill-forms-only ]?
```

Example:

```
<xfc:document-protection password="changeit"
 restrictions="tracked-changes-only limited-formatting" />
```

It's also possible to restrict editing in the documents created by **XFC** using command-line arguments `/prot` *`restrictions`* and `/password` *`password`*.

When both element `xfc:document-protection` and any of the aforementioned command-line arguments are specified, it's the command-line argument which is used. In the case of the above `xfc:document-protection` example, `/password ""` may be used to discard the password.

The attributes of element `xfc:document-protection` are:

`restrictions`

Specifies how the generated document is to be restricted in terms of editing and/or formatting.

Restriction	Description
`read-only`	No changes are permitted; the document is read-only.
`comments-only`	No changes are permitted, but comments can be inserted. **ODT** output format: `comments-only` restriction not supported.
`fill-forms-only`	No changes are permitted, but data can be entered into forms. **ODT** output format: `fill-forms-only` restriction not supported.
`tracked-changes-only`	All changes are permitted, but they're automatically tracked.
`limited-formatting`	No direct formatting (e.g. **Bold**, **Italic**) and limit formatting to a selection of styles. **RTF**, **ODT** output formats: `limited-formatting` restriction not supported.

`password`

This clear text password lets the user of the word processor remove the restrictions specified by attribute `restrictions`.

RTF, **WML** output formats: password not supported. **ODT** output format: password supported only for restriction `tracked-changes-only`.

4.9. Special characters

XFC uses an instance of the `System.Text.Encoding` class to determine if a given character can be represented in the output encoding. Characters that cannot be encoded are then represented using a Unicode control word (RTF output) or an XML character reference (WML, Open XML and OpenDocument output).

4.10. Special support for East Asian fonts

This feature is supported by the ODT, WML and DOCX output formats, but not by the RTF output format.

When using East Asian fonts in a XSL-FO file[7] to render **CJK** (Chinese Japanese Korean) text, these fonts must be declared to **XFC**.

This is done using the eastAsiaFontFamilies property. This property is specified using command line option `/eaf` *`map`* [5]. The value of this property is a font family map having the following syntax:

```
map -> entry [',' entry]*

entry -> east_asian_family '=' western_family
```

Note that *`western_family`* must be an actual font family (e.g. `Arial`). Generic font families (e.g. `sans-serif`) are not supported here.

Example ("`MS UI Gothic`" is a Japanese font):

```
<fo:inline font-family="MS UI Gothic">ねこ  romaji neko</fo:inline>
```

Let's suppose the font family map used for the XSL-FO file containing the above example is:

```
MS UI Gothic=Times New Roman,Meiryo=Calibri
```

The above font family map has two effects on **XFC**:

1. Font families "`MS UI Gothic`" and "`Meiryo`" are declared as being East Asian fonts and will be used to render the **CJK** text segments. In the above example, "`ねこ`" is rendered using the "`MS UI Gothic`" font.
2. When a text run contains a mix of **CJK** text and Western text, the "`Times New Roman`" and "`Calibri`" fonts will be used to render the Western text segments. In the above example, "`romaji neko`" is rendered using the "`Times New Roman`" font, even if the `fo:inline` containing this segment requests "`MS UI Gothic`".

4.11. Multiple page layouts

XFC supports all `conditional-page-master-reference` element combinations that can be accommodated by a single RTF section. This means the following page sequence layouts are supported:

- Single-sided layout.
- Header page + single-sided layout.
- Double-sided layout.
- Header page + double-sided layout.

[7] Either directly in the XSL-FO file or indirectly through the use of named styles [44].

This applies to all output formats. Also, note that a single RTF section can handle different headers/footers on left/right/first pages, but does not allow page geometry changes, except for switching left and right margins on facing pages. This restriction does not apply to OpenDocument output.

Note: By default RTF, WML and Open XML output documents are given a double-sided page layout regardless of the input document properties. This results in all sections having separate headers/footers for odd and even pages, even though the content of both headers/footers may be identical. It may also result in blank pages being inserted in the document in order for every section to start on an odd page.

4.12. Adding a watermark to the generated document

Adding a watermark to the generated document is done the way which is supported by all the other XSL-FO processors, that is, by setting the `background-image` property of `fo:region-body`. Example:

```
<fo:simple-page-master master-name="center"
                       margin-bottom="1.5cm" margin-left="1.5cm"
                       margin-right="1.5cm" margin-top="1.5cm"
                       page-height="29.7cm" page-width="21cm">
  <fo:region-body border-style="solid" border-width="1pt"
                  margin-bottom="0.5cm" margin-top="0.5cm" padding="7.5pt"
                  background-image="url(images/draft.png)"
                  background-position="center"/>
  <fo:region-before display-align="before" extent="0.5cm" />
  <fo:region-after display-align="after" extent="0.5cm" />
</fo:simple-page-master>
```

Note that only the `background-image`, `background-position-horizontal` and `background-position-vertical` properties (and the corresponding shorthand properties) are supported. Other background image properties such as `background-repeat` are ignored. Moreover the only supported values for `background-position-horizontal` are: `left`, `0%`, `center`, `50%`, `right`, `100%` and the supported values for `background-position-vertical` are: `top`, `0%`, `center`, `50%`, `bottom`, `100%`.

4.13. Expressions

Use of expressions for property values specification is supported, subject to the following restrictions:

- The `proportional-column-width` function may not be part of an arithmetic expression, i.e. it must be used as a single primary expression.
- The `system-color`, `system-font` and `merge-property-values` are not supported.

4.14. Non-standard extension of XSL-FO property `text-decoration`

In addition to the decoration type (`underline`, `overline`, `line-through`, etc) supported by XSL-FO property `text-decoration`, it's possible to specify the color, style (`solid`, `double`, `dotted`, `dashed`, `wavy`) and thickness of the text decoration. The syntax used for extended simple properties[8] is identical to the syntax of CSS3 property text-decoration.

```
text-decoration = 'inherit' | [ line || style || color || thickness ]
```

[8] Notation "||" means: at least one of these items must be present, and they may appear in any order.

```
line = 'none' |
       [ [ 'underline' | 'no-underline' ] || [ 'overline' | 'no-overline' ] ||
         [ 'line-through' | 'no-line-through' ] || [ 'blink' | 'no-blink' ] ]

style = 'solid' | 'double' | 'dotted' | 'dashed' | 'wavy'

color = 'currentcolor' | Hexadecimal_color | RGB_color

thickness = 'auto' | 'from-font' | Length | Percentage
```

Actual support of the “simple properties” comprising `text-decoration` by the RTF, WML, DOCX, ODT output formats varies:

Output format	Support of “simple properties”
RTF, WML, DOCX	• `overline` not supported. • `line-through` only `solid` or `double` and always `currentcolor`. • Thickness not supported: any value larger or equal to `3pt` (e.g. `4px`) is translated to DOCX thickness "`heavy`" (which means thicker that normal thickness). • DOCX thickness "`heavy`" not supported for style "`double`".
ODT	• `line-through` only `solid` or `double` and always `currentcolor`. • Thickness not supported: any value larger or equal to `3pt` (e.g. `4px`) is translated to ODT thickness "`bold`" (which means thicker that normal thickness).

Remember that *OpenOffice/LibreOffice automatically underlines hyperlinks*

By default, OpenOffice/LibreOffice automatically underlines and gives a blue color to hyperlinks. In some cases, this automatic feature may give you the impression that there is something wrong with the `text-decoration` property you have specified, except that you probably did not specify any `text-decoration` property there at all!

Chapter 6. XSL-FO extension for generating named styles

1. Why generate named styles?

As of XMLmind XSL-FO Converter (**XFC** for short) v5[1], it becomes possible to generate RTF, WordprocessingML, Office Open XML (`.docx`) and OpenOffice (`.odt`) files where most of the text formatting is achieved using *named paragraph styles* ("`Normal`", "`Heading 1`", "`Heading 2`", etc) and *named character styles* ("`Strong`", "`Emphasis`", etc).

Moreover, a named paragraph style may reference a *named numbering scheme* (also known as a “list style”). This allows to implement numbered headings and advanced —multilevel— lists purely by using named paragraph styles.

The main benefits of generating named styles are for the end-user of the word processor files:

- Thanks to the names of the styles, the document, when opened in MS-Word or OpenOffice/LibreOffice, looks familiar and its organization is easier to understand.
- After a change, the numbering of headings and list items is automatically updated by the word processor.
- The formatting of the document is a snap to modify using the various style editors included in the word processor.

2. How it works

2.1. Putting named styles to work

Named styles are specified in an XML file conforming to the `styles.xsd` schema. The recommended extension for this kind of file is "`.xfc`". Simple example, `sample0.xfc`:

```
<styles xmlns="http://www.xmlmind.com/foconverter/xsl/extensions"
        xmlns:xfc="http://www.xmlmind.com/foconverter/xsl/extensions">

  <text-style name="Warning" font-weight="bold" color="red" />

</styles>
```

The location of the `.xfc` file containing the style definitions must be passed as the value of the *`styles`* parameter to **XFC**, for example by the means of the `/sty` command-line option [4].

The named styled is referenced by the means of the `xfc:user-style` extension attribute. Simple example, `sample0.fo`:

```
<fo:block>During take-off and landing,
  <fo:inline xfc:user-style="Warning">always keep your seat belt
  fastened</fo:inline>.</fo:block>
```

Command-line example:

[1] Prior releases of **XFC** only supported direct formatting.

```
fo2docx /sty sample0.xfc sample0.fo sample0.docx
```

2.2. The effect of the `xfc:user-style` extension attribute on an XSL-FO element

If set on a `fo:inline` element, attribute `xfc:user-style` must reference the name of an existing `xfc:text-style` element. If set on a `fo:block` element, attribute `xfc:user-style` must reference the name of an existing `xfc:paragraph-style` element.

The following `fo:inline` element

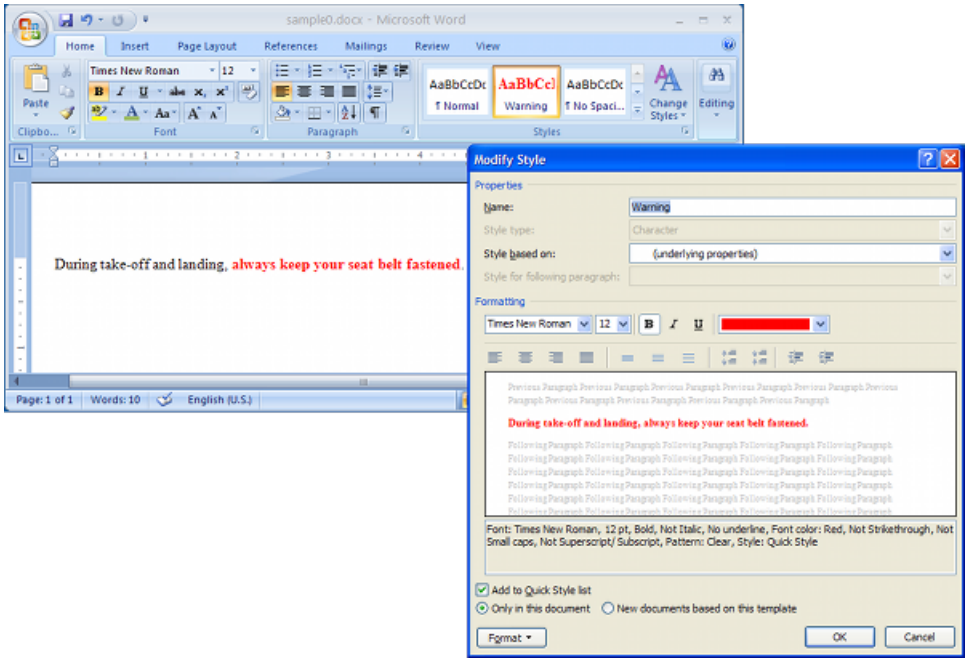
```
<fo:inline xfc:user-style="Warning">always keep your seat belt
fastened</fo:inline>
```

is rendered by the target word processor exactly as if it was specified as[2]:

```
<fo:inline font-weight="bold" color="red">always keep your seat belt
fastened</fo:inline>
```

The main difference between the two specifications is that, with the first specification, the user of the word processor may use the style editor to specify, for example, that all warning text runs are to be rendered in orange rather than in red.

Figure 6.1. The style editor of MS-Word 2007



The second specification is said to generate *direct style properties* on the resulting text run. When this is the case, there is no way for the user of the word processor to use the style editor to specify that all warning text runs are to be rendered in orange rather than in red.

[2] **XFC** named styles are similar to XSLT `xsl:attribute-sets`. However `xsl:attribute-set` elements are processed by the XSLT engine, while `text-style` and `paragraph-style` elements are processed by **XFC** (which is an XSL-FO processor, and not an XSLT engine).

It's of course possible, and often useful, to mix `xfc:user-style` with standard XSL-FO attributes:

- In the following example, *redundant attributes* such as `font-weight="bold"` an `color="red"` (already contained in the "Warning" `text-style`) are simply ignored by **XFC**:

```
<fo:inline xfc:user-style="Warning"
           font-weight="bold" color="red">always keep your seat belt
fastened</fo:inline>
```

  This is an important feature as we'll see it in Section 5, “Adding named styles support to an existing XSLT stylesheet” [56].

- With the following snippet, the resulting warning text run will be rendered using a bold, italic, font and a red color:

```
<fo:inline xfc:user-style="Warning"
           font-style="italic">always keep your seat belt
fastened</fo:inline>
```

- With the following snippet, the resulting warning text run will be rendered using a bold font and a blue color:

```
<fo:inline xfc:user-style="Warning"
           color="blue">always keep your seat belt
fastened</fo:inline>
```

  Directly specified attribute `color="blue"` overrides the `color="red"` attribute found in the "Warning" `text-style`.

- With the following snippet, the resulting warning text run will be rendered using a bold, italic, larger font and a red color:

```
<fo:block font-weight="normal"
          font-style="italic" font-size="larger">During take-off and landing,
  <fo:inline xfc:user-style="Warning">always keep your seat belt
  fastened</fo:inline>.</fo:block>
```

  Attributes `font-weight="normal"`, `font-style="italic"` and `font-size="larger"` are inherited by the `fo:inline` from its parent `fo:block`. However, inherited attribute `font-weight="normal"` has no effect on the resulting warning text run as the "Warning" `text-style` contains attribute `font-weight="bold"`.

3. Style reference

About namespaces in the following sections

In the following sections, all the element names have a http://www.xmlmind.com/foconverter/xsl/extensions namespace and all attribute names have no namespace.

3.1. The `styles` element

The stylesheet passed as a parameter to **XFC** (`/sty` command-line option) is specified in an XML file conforming to the `styles.xsd` schema. The recommended extension for these XML files are "`.xfc`".

```
<styles>
Content: [ text-style | paragraph-style | numbering ]*
</styles>
```

Example:

```
<styles xmlns="http://www.xmlmind.com/foconverter/xsl/extensions"
        xmlns:xfc="http://www.xmlmind.com/foconverter/xsl/extensions">
  ...
</styles>
```

3.2. The `text-style` element

```
<text-style
  name = non empty token
  abstract = boolean : false
  base-style = name of another text-style
  Some standard XSL-FO text attributes
  Some standard XSL-FO background attributes
/>
```

Specifies a text style (also known as a “character style”) which can be applied to a `fo:inline` element by the means of the `xfc:user-style` extension attribute. Ignored if applied to any element other than `fo:inline`.

`name`
: Required. Unique name of this text style.

`abstract`
: If true, this text style is not intended to be directly applied to any `fo:inline` element. Instead, it is intended to be inherited by other `text-style` elements by the means of their `base-style` attributes.

`base-style`
: Specifies another `text-style` element. This causes this `text-style` element to inherit all the XSL-FO attributes found in the base `text-style` element

The standard XSL-FO attributes allowed in a `text-style` element are:

- `font-family`
- `font-size`
- `font-style`
- `font-weight`
- `font-variant`
- `font`
- `text-decoration`
- `baseline-shift`
- `color`
- `background-color`

- `background`

Note that specifying any other XSL-FO attribute (e.g. `text-transform`) is reported as a fatal error.

Examples:

```
<text-style name="Basic" abstract="true" font="10pt sans-serif" />

<text-style name="Red" base-style="Basic" color="red" />
```

3.3. The `paragraph-style` element

```
<paragraph-style
  name = non empty token
  abstract = boolean : false
  base-style = name of another paragraph-style
  next-style = name of another paragraph-style
  numbering = name of a numbering
  numbering-level = integer between 1 and 10 inclusive
  outline-level = non empty string
  Some standard XSL-FO text attributes
  Some standard XSL-FO background attributes
  Some standard XSL-FO paragraph attributes
/>
```

Specifies a paragraph style which can be applied to a `fo:block` element by the means of the `xfc:user-style` extension attribute. Ignored if applied to any element other than `fo:block`.

`name`
: Required. Unique name of this paragraph style.

`abstract`
: If true, this paragraph style is not intended to be directly applied to any `fo:block` element. Instead, it is intended to be inherited by other `paragraph-style` elements by the means of their `base-style` attributes.

`base-style`
: Specifies another `paragraph-style` element. This causes this `paragraph-style` element to inherit all the XSL-FO attributes and also the `next-style`, `numbering`, `numbering-level` and `outline-level` attributes found in the base `paragraph-style` element

`next-style`
: Specifies the name of a `paragraph-style` element, this one or another one. A paragraph having `next-style` style will be automatically created by the word processor if the user presses key Enter inside a paragraph having this style.

`numbering`
: Specifies that paragraphs having this style are to be automatically numbered by the word processor, the numbering scheme to be used being specified by the value of this attribute. See Section 3.4, “The `numbering` element” [50].

`numbering-level`

Required if `numbering` attribute has also been specified, but not required if this paragraph style is abstract. Specifies the list level of paragraphs having this style. See Section 3.4, “The `numbering` element” [50].

`outline-level`

Same extension attribute, except for the empty namespace, as Section 4.4.1, “The `xfc:outline-level` extension attribute” [31].

The standard XSL-FO attributes allowed in a `paragraph-style` element are:

- `break-after`
- `break-before`
- `keep-together`
- `keep-with-next`
- `keep-with-previous`
- `orphans`
- `widows`
- `space-before`
- `space-after`
- `start-indent`
- `end-indent`
- `text-align`
- `text-align-last`
- `text-indent`
- `line-height` (Number, percentage or length only. Not space.)
- `padding-top`
- `padding-bottom`
- `padding-left`
- `padding-right`
- `padding`
- `border-top-style`
- `border-top-width`
- `border-top-color`
- `border-top`
- `border-bottom-style`
- `border-bottom-width`
- `border-bottom-color`
- `border-bottom`
- `border-left-style`
- `border-left-width`
- `border-left-color`
- `border-left`
- `border-right-style`
- `border-right-width`
- `border-right-color`
- `border-right`
- `border`
- `font-family`
- `font-size`
- `font-style`
- `font-weight`

- font-variant
- font
- text-decoration
- baseline-shift
- color
- background-color
- background

Note that specifying any other XSL-FO attribute (e.g. padding-after, margin-left, keep-together.within-column, space-before.mininum) is reported as a fatal error.

Examples:

```
<paragraph-style name="Caption" base-style="Centered"
                 keep-with-previous="always"
                 font-style="oblique" font-size="smaller"
                 start-indent="4em" end-indent="4em" />

<paragraph-style name="Bullet 3" numbering="Bullets" numbering-level="3"
                 start-indent="2*24pt" />
```

3.4. The numbering element

```
<numbering
  name = non empty token
  show-all-levels = boolean : false
>
Content: [ level ]{1,10}
</numbering>

<level
  format = non empty string
  text-align = non empty string : start
  provisional-distance-between-starts = non empty string : 24pt
  provisional-label-separation = non empty string : 6pt
  Some standard XSL-FO text attributes
  Some standard XSL-FO background attributes
/>
```

Element numbering specifies a numbering scheme (also known as a “list style”) for use by a paragraph style. For this, the name of the numbering scheme must be referenced in the numbering attribute of element paragraph-style [48].

Attributes of element numbering:

name

Required. Unique name of this numbering scheme.

show-all-levels

If true, prepend to the number of a list item the numbers of all its “parent” list items. Ignored if this numbering scheme specifies bullets and not numbers.

For example, if list item "`d.`" is “nested” inside list item "`3.`", itself “nested” inside list item "`IV.`", then the label found at the beginning of list item "`d.`" will be in fact "`IV.3.d.`".

A `numbering` element may contain up to 10 `level` elements. A `level` element specifies a number or bullet format for a list item “nested” at the corresponding level. That is, top-level (“non-nested”) list items have a numbering level equal to 1 and their number/bullet formats are specified by the first `level` child of element `numbering`; list items “nested” inside top-level list items have a numbering level equal to 2 and their number/bullet formats are specified by the second `level` child of element `numbering`; and so on up to 10 “nesting” levels.

Attributes of element `level`:

`format`

Required. Number or bullet format specified using the syntax documented in Section 4.2.1, “The `xfc:label-format` extension attribute” [29].

`text-align`

Standard XSL-FO attribute `text-align`. Specifies the horizontal alignment of the number or bullet within the space specified using `provisional-distance-between-starts`.

`provisional-distance-between-starts`

Standard XSL-FO attribute `provisional-distance-between-starts`. If specified as a positive length, this gives a hanging indent to the list item.

`provisional-label-separation`

Standard XSL-FO attribute `provisional-label-separation`. Useful when `provisional-distance-between-starts` is 0 because it allows to separate the number or bullet from the body of the list item.

The other standard XSL-FO attributes allowed in a `level` element are:

- `font-family`
- `font-size`
- `font-style`
- `font-weight`
- `font-variant`
- `font`
- `text-decoration`
- `baseline-shift`
- `color`
- `background-color`
- `background`

Note that specifying any other XSL-FO attribute is reported as a fatal error.

Examples:

```
<numbering name="Bullets">
  <level format="&#x2022;"
         provisional-label-separation="0" />
  <level format="-"
         provisional-label-separation="0" />
  <level format="&#x25CF;"
```

```
         text-align="right"
         provisional-distance-between-starts="48pt"
         provisional-label-separation="0" />
</numbering>

<numbering name="Numbers" show-all-levels="true">
  <level format="%{decimal}."
         font-family="sans-serif" font-weight="bold" font-size="10pt"
         color="#800000" />
  <level format="%{lower-alpha}."
         font-family="sans-serif" font-weight="bold" font-size="10pt"
         color="#008000" />
  <level format="-%{lower-roman}-"
         text-align="center"
         font-family="sans-serif" font-weight="bold" font-size="10pt"
         color="#000080" />
</numbering>
```

3.5. The `xfc:user-style` extension attribute

This extension attribute specifies which named style to use for a `fo:inline` or `fo:block` element. Example:

```
<fo:inline xfc:user-style="Warning">always keep your seat belt
fastened</fo:inline>
```

When an "`.xfc`" file has been passed as a parameter to **XFC**, for example by the means of the `/sty` command-line option [4]:

- If set on a `fo:inline` element, attribute `xfc:user-style` must reference the name of an existing `xfc:text-style` element, otherwise a fatal error is reported.
- If set on a `fo:block` element, attribute `xfc:user-style` must reference the name of an existing `xfc:paragraph-style` element, otherwise a fatal error is reported.
- It's a fatal error to specify `xfc:user-style` on any XSL-FO element other than `fo:inline` and `fo:block`.

Attribute `xfc:user-style` is ignored, whatever its value, if no "`.xfc`" file has been passed as a parameter to **XFC**.

Attribute `xfc:user-style=""` (empty string value) is ignored in all cases.

3.6. The `xfc:restart-numbering` extension attribute

Using this boolean extension attribute is required to reuse the same numbered paragraph styles to create several logical lists.

Attribute `xfc:restart-numbering` is best explained using a simple example. The `numbering` element is:

```
<numbering name="Item Numbers" show-all-levels="true">
  <level format="%{decimal}."
         provisional-distance-between-starts="20pt"
         provisional-label-separation="0"
         font-family="serif" font-size="10pt" color="#004080" />
  <level format="%{upper-alpha}."
         provisional-distance-between-starts="30pt"
         provisional-label-separation="0"
         font-family="serif" font-size="10pt" color="#004080" />
</numbering>
```

The numbered paragraph styles are:

```
<paragraph-style name="Numbered Item 1" base-style="Numbered Item"
                 numbering-level="1" start-indent="2em" />

<paragraph-style name="Numbered Item 2" base-style="Numbered Item"
                 numbering-level="2" start-indent="2em + 20pt" />
```

What follows is meant to specify *two* “logical lists” separated by a paragraph.

```
<fo:block xfc:user-style="Numbered Item 1">First item.</fo:block>
  <fo:block xfc:user-style="Numbered Item 2">First sub-item
  of first item.</fo:block>
  <fo:block xfc:user-style="Numbered Item 2">Second sub-item
  of first item.</fo:block>
<fo:block xfc:user-style="Numbered Item 1">Second item.</fo:block>
  <fo:block xfc:user-style="Numbered Item 2">First sub-item
  of second item.</fo:block>
  <fo:block xfc:user-style="Numbered Item 2">Second sub-item
  of second item.</fo:block>

<fo:block>A paragraph.</fo:block>

<fo:block xfc:user-style="Numbered Item 1">First item.</fo:block>
  <fo:block xfc:user-style="Numbered Item 2">First sub-item
  of first item.</fo:block>
  <fo:block xfc:user-style="Numbered Item 2">Second sub-item
  of first item.</fo:block>
<fo:block xfc:user-style="Numbered Item 1">Second item.</fo:block>
  <fo:block xfc:user-style="Numbered Item 2">First sub-item
  of second item.</fo:block>
  <fo:block xfc:user-style="Numbered Item 2">Second sub-item
  of second item.</fo:block>
```

However the above XSL-FO snippet is converted to:

1.
  1.A
  1.B
2.
  2.A

```
   2.B
A paragraph.
 3.
   3.A.
   3.B.
 4.
   4.A.
   4.B
```

by **XFC**.

After adding attribute `xfc:restart-numbering="true"` to the first item of each logical list:

```
<fo:block xfc:user-style="Numbered Item 1"
  xfc:restart-numbering="true">First item.</fo:block>
  <fo:block xfc:user-style="Numbered Item 2">First sub-item
  of first item.</fo:block>
  <fo:block xfc:user-style="Numbered Item 2">Second sub-item
  of first item.</fo:block>
<fo:block xfc:user-style="Numbered Item 1">Second item.</fo:block>
  <fo:block xfc:user-style="Numbered Item 2">First sub-item
  of second item.</fo:block>
  <fo:block xfc:user-style="Numbered Item 2">Second sub-item
  of second item.</fo:block>

<fo:block>A paragraph.</fo:block>

<fo:block xfc:user-style="Numbered Item 1"
  xfc:restart-numbering="true">First item.</fo:block>
  <fo:block xfc:user-style="Numbered Item 2">First sub-item
  of first item.</fo:block>
  <fo:block xfc:user-style="Numbered Item 2">Second sub-item
  of first item.</fo:block>
<fo:block xfc:user-style="Numbered Item 1">Second item.</fo:block>
  <fo:block xfc:user-style="Numbered Item 2">First sub-item
  of second item.</fo:block>
  <fo:block xfc:user-style="Numbered Item 2">Second sub-item
  of second item.</fo:block>
```

This gives the expected result:

```
 1.
   1.A
   1.B
 2.
   2.A
   2.B
A paragraph.
 1.
   1.A
   1.B
 2.
```

```
2.A
2.B
```

It is not required to add attribute `xfc:restart-numbering="true"` to the first item of the very first “logical list” of the XSL-FO file, however doing so is simpler and is harmless.

4. A comprehensive example

A comprehensive example demonstrating almost everything you can do with named styles is found in *`XFC_install_dir`*`/samples/styles.fo`:

```
...
<fo:block xfc:user-style="Heading 1">This is a block
having xfc:user-style="Heading 1".</fo:block>
...
```

The associated style definition file is *`XFC_install_dir`*`/samples/styles.xfc`:

```
...
<numbering name="Heading Numbering" show-all-levels="true">❶
  <level format="%{decimal}."
         provisional-distance-between-starts="0"
         provisional-label-separation="8pt"/>
  <level format="%{decimal}."
         provisional-distance-between-starts="0"
         provisional-label-separation="7pt" />
  <level format="%{decimal}."
         provisional-distance-between-starts="0"
         provisional-label-separation="6pt" />
</numbering>

<paragraph-style name="Heading" abstract="true" next-style="Paragraph"❷
                 numbering="Heading Numbering"❸
                 keep-with-next="always"
                 font-family="sans-serif" font-weight="bold"
                 color="#004080" />

<paragraph-style name="Heading 1" base-style="Heading"❹
                 outline-level="1"❺ numbering-level="1"❻
                 font-size="16pt" line-height="0.82em"
                 space-before="0.82em" space-after="0.82em" />
...
```

❶ Specifies the numbering, up to 3 levels, of the headings found in the generated word processor file.

❷ This is an *abstract* `paragraph-style` which is inherited by the "`Heading 1`", "`Heading 2`" and "`Heading 3`" actual `paragraph-styles`.

❸ This specifies how headings are to be automatically numbered by the word processor.

❹ A "`Heading 1`" `paragraph-style` which is applied to all first level headings.

❺ This specifies the *outline level* of a "`Heading 1`".

❻ This specifies the *list level*, that is, which `level` child element of the `numbering` element, applies to a "`Heading 1`".

You can generate `styles.odt`, `styles.rtf`, `styles.word.xml`, `styles.docx` by running `make_samples` inside the *`XFC_install_dir`*`/samples/` folder.

5. Adding named styles support to an existing XSLT stylesheet

Retrofitting named styles support in an existing XSLT stylesheet which has been designed to generate XSL-FO for use by Apache FOP, RenderX XEP or Antenna House XSL Formatter (or **XFC**, but without named styles) is tedious and error prone. We strongly recommend to avoid doing this.

However, it's not difficult to design from scratch an XSLT stylesheet which generates XSL-FO making using of named styles and which works equally well when used in conjunction with XSL-FO processors other than **XFC**.

The key ideas allowing to do this are:

1. An extension attribute such as `xfc:user-style` should be ignored by XSL-FO processors other than **XFC**.
2. Specifying the same XSL-FO attributes *twice* —one time inside the named style for use by **XFC** and a second time directly on the XSL-FO element for use by the other XSL-FO processors— will not predate the possibility for the user of the word processor to later modify the aspect of the generated document by editing the named styles.

   This works fine because as explained in Section 2.2, “The effect of the `xfc:user-style` extension attribute on an XSL-FO element” [45], ***XFC** ignores redundant attributes*, that is, XSL-FO attributes specified at the same time inside the named style and also directly on the XSL-FO element.

A sample XSLT stylesheet is found in `sample1.xsl`:

```
...
<xsl:attribute-set name="plain">
  <xsl:attribute name="font-family">serif</xsl:attribute>
  <xsl:attribute name="font-size">10pt</xsl:attribute>
  <xsl:attribute name="line-height">1.3em</xsl:attribute>
</xsl:attribute-set>
...

<xsl:attribute-set name="p" use-attribute-sets="plain">
  <xsl:attribute name="text-align">justify</xsl:attribute>
  <xsl:attribute name="space-before">1.3em</xsl:attribute>
  <xsl:attribute name="space-after">1.3em</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="h:p">
  <fo:block xsl:use-attribute-sets="p">❶
    <xsl:if test="$use-styles = 'yes'">
      <xsl:attribute name="xfc:user-style">Paragraph</xsl:attribute>❷
```

```
    </xsl:if>

    <xsl:apply-templates />
  </fo:block>
</xsl:template>
```

❶ This `fo:block` element has a number of XSL-FO attributes directly set on it by the means of `xsl:attribute-set` "p".

❷ The very same XSL-FO attributes are found in the "`Paragraph`" `paragraph-style`. Excerpts from `sample1.xfc`:

```
<paragraph-style name="Paragraph" text-align="justify"
                 font-size="10pt" line-height="1.3em"
                 space-before="1.3em" space-after="1.3em" />
```

Run for example Saxon 6, to generate an XSL-FO file, `sample1.fo`, for use by XSL-FO processors other than **XFC**:

```
java -jar saxon.jar -o sample1.fo sample1.xhtml sample1.xsl
```

After doing that, convert `sample1.fo` to PDF for example using Apache FOP:

```
fop -r -q -fo sample1.fo -pdf sample1.pdf
```

Run for example Saxon 6, to generate an XSL-FO file, `sample1_sty.fo`, for use by **XFC**:

```
java -jar saxon.jar -o sample1_sty.fo sample1.xhtml sample1.xsl use-styles=yes
```

After doing that, convert `sample1.fo` to `sample1.docx` for example:

```
fo2docx /sty sample1.xfc sample1_sty.fo sample1.docx
```

6. Troubleshooting

6.1. Is it possible to use the standard styles names of MS-Word —"`Normal`", "`Heading 1`", "`Heading 2`", "`Strong`", "`Emphasis`", etc— in my `.xfc` style definition file?

Yes, however it's recommended to avoid the name "`Normal`" for a `paragraph-style` as this has strange side-effects in MS-Word.

Note that using "`Normal`" as the name of a `text-style` works fine, except that MS-Word automatically renames this text style to "`Normal1`".

6.2. When I attempt to modify the generated paragraph style in MS-Word or OpenOffice/Libre, the space after the paragraph is always set to `0pt`.

More precisely, I've defined paragraph-style "`Foo`" as follows:

```
<paragraph-style name="Foo"
                 space-before="10pt" space-after="20pt" />
```

and the `fo:block` referencing paragraph-style "`Foo`" has no attribute `space-after` or `margin-bottom` directly set on it.

The generated word processor file looks as expected. However, when I used the style editor of MS-Word or OpenOffice/Libre Office to modify the "`Foo`" paragraph style, I've found that, while the space before the paragraph was indeed set to `10pt`, the space after the paragraph was set to `0pt`. Please fix this bug.

This is not a bug. This is a limitation which, due to the internal design of **XFC**, cannot be removed.

```
<paragraph-style name="Foo"
                 space-before="10pt" space-after="20pt" />
...
<fo:block xfc:user-style="Foo">...</fo:block>
```

is processed by **XFC** as if it was:

```
<paragraph-style name="Foo"
                 space-before="10pt" space-after="0pt" />
...
<fo:block xfc:user-style="Foo"
          space-after="20pt">...</fo:block>
```

6.3. I use a set of numbered paragraph styles (i.e. `<paragraph-style numbering="`*xxx*`"/>`) to create several lists. However all the list items are continuously numbered across the generated DOCX file as if it were a single, giant list. How to use a set of numbered paragraph styles to create several, *distinct* lists in the DOCX file?

See Section 3.6, “The `xfc:restart-numbering` extension attribute” [52].

Chapter 7. XSL-FO extension for Office Open XML

1. Introductory example

XMLmind XSL-FO Converter supports an XSL-FO extension to generate structured document tags (**SDT**s) in an Office Open XML document. Structured document tags are WordprocessingML elements that may be used to include form fields - such as text fields and drop-down lists - in an **OOXML** document and store form data in a dedicated part - called a Custom XML Data part - of the document. In other words, the **SDT** technology makes it possible to produce simple forms that can be loaded and filled in MS-Word 2007+[1]. As Custom XML Data parts are simple XML files the form data can then be easily extracted and processed. For further information regarding structured document tags refer to section 2.5.2 of part 4 (Markup Language Reference) of the Office Open XML specification, available from Ecma International.

The implementation and application area of this extension are better understood with a concrete example. Consider the simple XML instance below:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<organization>
  <name>Pixware</name>
  <category></category>
  <creation-date></creation-date>
  <logo></logo>
</organization>
```

Now imagine we would like a simple form to collect and retrieve the missing information. We will illustrate how to use the XSL-FO extension for Office Open XML to create a form that can be loaded and filled in MS-Word 2007.

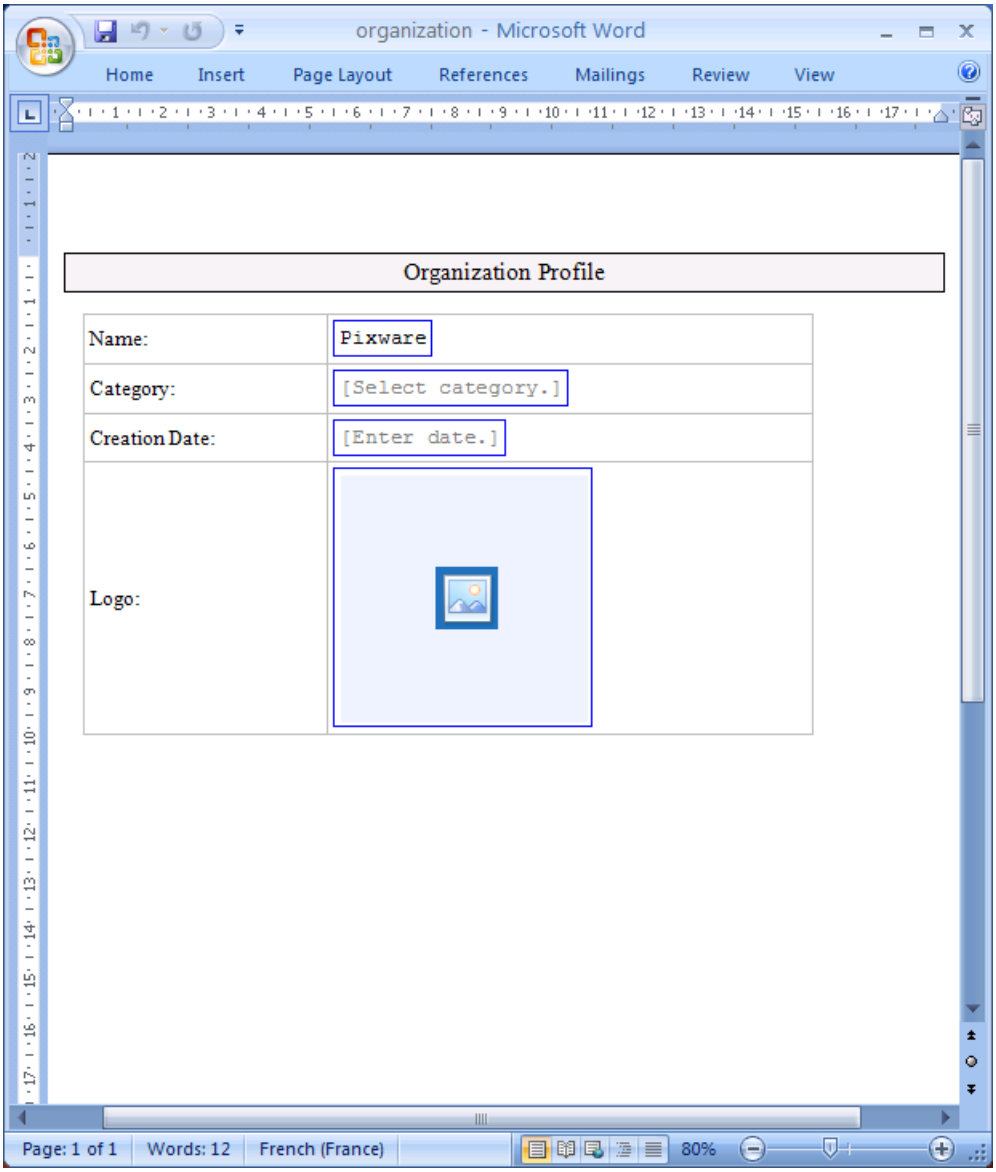
1. Starting from our XML instance we first create an XSL-FO document, by applying an XSLT stylesheet or any other means. The XSL-FO tree will include custom elements that translate to form fields in the **OOXML** document. For instance the block below will provide a drop-down list with 3 entries for input of the organization category.

   ```
   <fo:block><fo:inline border="solid 1pt blue" font-family="Courier"
       padding="1mm"><sdt:drop-down-list
       binding="category" prompt="[Select category.]"
       title="Category">
       <sdt:list-entry value="business" />

       <sdt:list-entry value="non-profit" />

       <sdt:list-entry value="other" />
     </sdt:drop-down-list></fo:inline></fo:block>
   ```

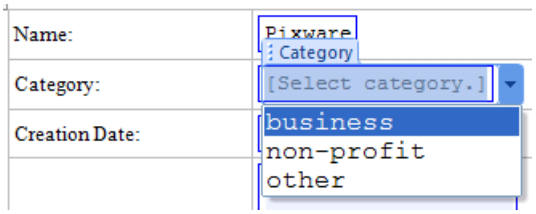
   The `binding` attribute of the `sdt:drop-down-list` element establishes the mapping between the field and an XML element in the Custom XML Data part. In the simplest case the value of this attribute is an XML element name, and the Custom XML Data part is automatically generated by **XFC**. In the above example the field value will be stored as the content of element `category` in the Custom XML Data part when the **OOXML** document is saved.

[1] This also works in MS-Word 2010 and 2013.

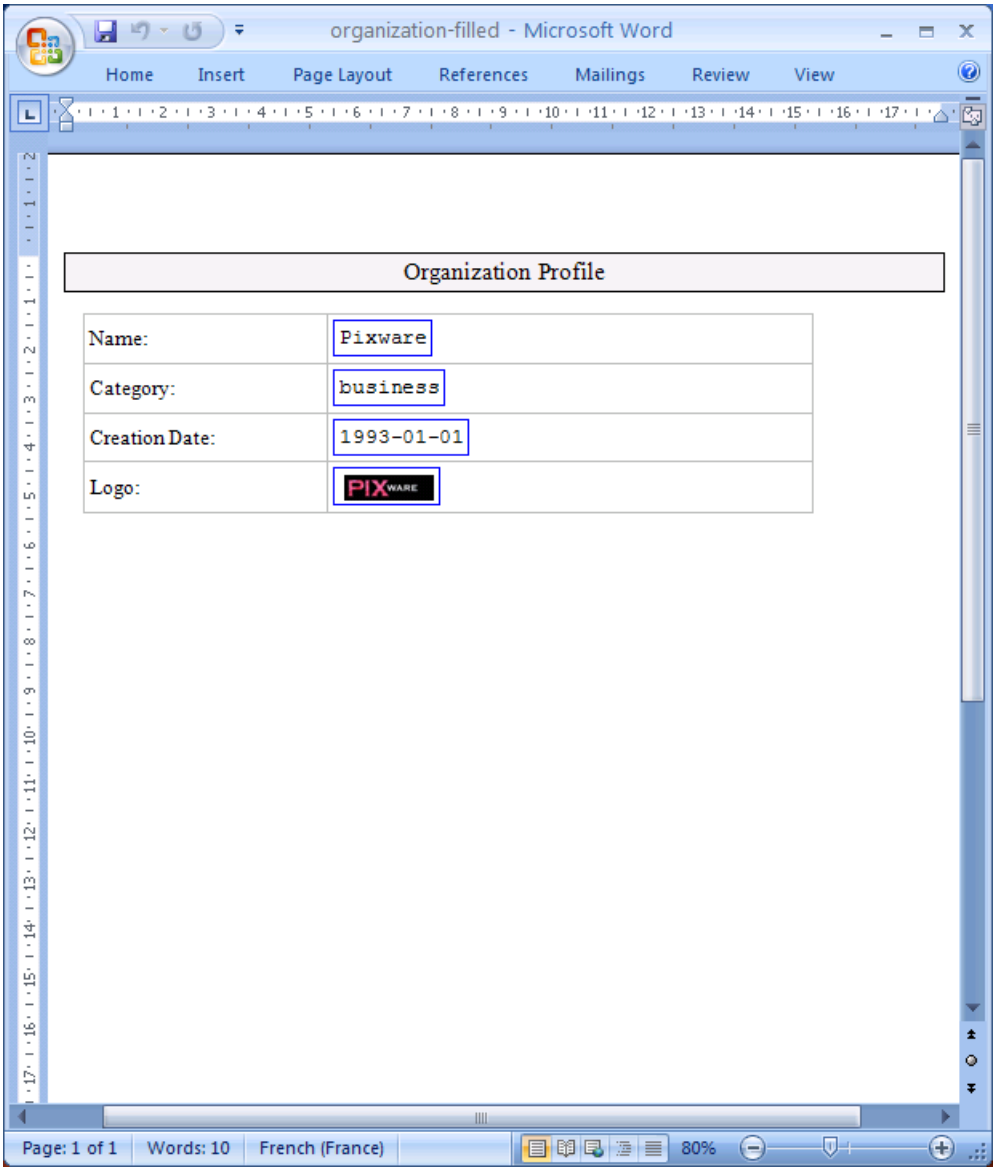
2. Using **XFC** we then convert the XSL-FO document to Office Open XML. The initial display of our sample document in MS-Word 2007 is shown below.



This simple form includes a drop-down list for input of the organization category, a date field - a specialized text field which provides a date picker - for input of the creation date, and an image chooser for input of the logo. The figure below shows the appearance of the drop-down list when selected.



3. This form may be used as a convenient means of collecting the missing information. The image below shows our sample document after it has been completed in MS-Word 2007.



4. After the form has been filled the form data can be easily extracted and processed. (Office Open documents are basically ZIP archives, and the Custom XML part is stored in file `customXml/item1.xml`.) The Custom XML part of our sample document after it has been completed is shown below. (The content of the logo element is the base64-encoded image data. Part of the content has been deleted for the sake of clarity.) Typical processing of the form data includes updating the original XML document or data in an XML repository.

```
<?xml version="1.0" encoding="UTF-8"?><root>
  <name>Pixware</name>
  <category>business</category>
  <creation-date>1993-01-01</creation-date>
  <logo>R0lGODdhjgAoAOcAAAICAuM7YdIiNLYkLZkfKfH+/XkfK+jn6bbFw2oaItTS0OKWsl8VIba+
kmiCMJoncmYbtGAcTaCmm5PVBBvUmafKkoSe9dwEPZf5z4AOtKAHTehCG/rQiE60olUSEAA7</logo>
</root>
```

This is just a simple example to introduce the basics of the XSL-FO form field extension for Office Open XML. For further information and reference material, see below. You can also download the sample **OOXML** document to experiment with the form fields.

2. How it works

To include form fields in an **OOXML** document one must embed custom elements in the XSL-FO tree. These elements must be in a separate namespace specified by XMLmind. This namespace - referred to by prefix `sdt` in this document - must be declared in the opening tag of the root element of the XSL-FO tree, as shown below.

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format"
         xmlns:sdt="http://www.xmlmind.com/foconverter/xsl/extensions/docx/sdt">
```

2.1. Text field example

Consider the XSL-FO snippet below:

```
<fo:block margin-left="1cm" margin-right="1cm">Name: <fo:inline
border="solid 1pt blue" padding="1mm"><sdt:text-field binding="name"
prompt="[Enter your name here.]" title="Name" /></fo:inline></fo:block>
```

The `sdt:text-field` element will be converted by **XFC** to a plain text **SDT**, which provides the functionality of a basic text field. The `prompt` attribute specifies placeholder text to be initially displayed in the field. The `sdt:text-field` element is wrapped in an `fo:inline` object that carries presentation properties. The initial display of the whole block in MS-Word 2007 is shown below. The next image shows the appearance of the field when selected, and the last one shows the field once filled.

Figure 7.1. Text field (initial display)

Figure 7.2. Text field (selected)

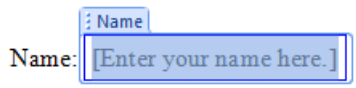
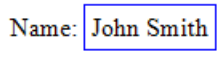


Figure 7.3. Text field (filled)



The `binding` attribute of the `sdt:text-field` element establishes the mapping between the field and an XML element in the Custom XML Data part. In the simplest case the value of this attribute is an XML element name. The Custom XML Data part will be automatically generated by **XFC**, in the form of a simple XML instance where all elements associated with form fields are children of the root element. Assuming the document contains no other field, **XFC** will therefore generate the XML instance below:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
```

```
  <name></name>
</root>
```

When saving the document after an editing session MS-Word will store the current value of the field as the content of the `name` element in the Custom XML Data part, as shown below.

```
<?xml version="1.0" encoding="UTF-8"?><root>
  <name>John Smith</name>
</root>
```

2.2. Drop-down list example

Consider the XSL-FO snippet below:

```
<fo:block margin-left="1cm" margin-right="1cm">Favorite Animal:
<fo:inline border="solid 1pt blue" padding="1mm"><sdt:drop-down-list
    binding="favorite-animal" initial-value="cat"
    title="Favorite Animal">
    <sdt:list-entry value="cat" />

    <sdt:list-entry value="dog" />

    <sdt:list-entry value="hamster" />
  </sdt:drop-down-list></fo:inline></fo:block>
```

The `sdt:drop-down-list` element will be converted by **XFC** to a drop-down list **SDT**, which provides the ability to select a single value from a predefined list. The list entries are specified by the `sdt:list-entry` children. The `initial-value` attribute of the `sdt:drop-down-list` element specifies the initial value of the field. The initial display of the whole block in MS-Word 2007 is shown below. The next image shows the appearance of the field while selecting an entry in the list.

Figure 7.4. Drop-down list (initial display)

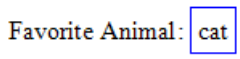
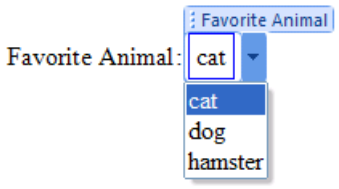


Figure 7.5. Drop-down list (selecting an entry)



The `initial-value` attribute differs from the `prompt` attribute in that the specified value is initially stored in the Custom XML Data part. Assuming the document contains no other field, **XFC** will therefore generate the Custom XML Data part below:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
```

```
  <favorite-animal>cat</favorite-animal>
</root>
```

2.3. Specifying a Custom XML Data template

Sometimes it may be desirable to have form data stored in an XML instance more complex than the default instance generated by **XFC**. In this case a Custom XML Data template may be specified by inserting an `sdt:configuration` element before the first `fo:page-sequence` object in the XSL-FO tree, e.g.:

```
<sdt:configuration custom-xml-template="custom.xml" />
```

The `custom-xml-template` attribute specifies the URL of an XML template to be used as the initial content of the Custom XML Data part. This XML template must be encoded in UTF-8 or UTF-16.

When a Custom XML Data template is specified, the `binding` attribute of a form field associated with an XML element in the Custom XML Data part references that particular element by means of an XPath 1.0 expression. For instance, consider the XML template below:

```
<?xml version="1.0" encoding="UTF-8"?>
<order>
  <product>
    <reference />
    <quantity />
  </product>
  <product>
    <reference />
    <quantity />
  </product>
</order>
```

To associate the `reference` child of the first `product` element with a form field one would set the `binding` attribute value of that field to `/order/product[1]/reference`. Moreover, when a Custom XML Data template is specified the `initial-value` attribute of form fields is ignored. If a field is to be initialized the initial value must be stored in the Custom XML Data template as the content of the XML element associated with that field.

2.4. Extracting the Custom XML Data part

Office Open XML documents are basically ZIP archives, so the Custom XML Data part can be easily extracted. In accordance with MS-Word's naming scheme **XFC** stores the Custom XML Data part in ZIP entry `customXml/item1.xml`.

3. Reference Material

This section provides a comprehensive description of the custom elements that make up the XSL-FO extension for Office Open XML. These elements must be in a separate namespace specified by XMLmind. This namespace - referred to by prefix `sdt` in this document - must be declared in the opening tag of the root element of the XSL-FO tree, as shown below.

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format"
         xmlns:sdt="http://www.xmlmind.com/foconverter/xsl/extensions/docx/sdt">
```

There are five elements that translate into a form field:

- sdt:text-field
- sdt:drop-down-list
- sdt:combo-box
- sdt:date
- sdt:picture

These are inline-level elements that may appear anywhere inline-level Formatting Objects are allowed.

3.1. Generic attributes

The attributes described below apply to all form fields, except for the `initial-value` and `prompt` attributes that do not apply to the `sdt:picture` element.

- `binding`

  This attribute establishes the mapping between a field and an XML element in the Custom XML Data part. In the simplest case the value of this attribute is an XML element name. The Custom XML Data part will be automatically generated by **XFC**, in the form of a simple XML instance where all elements associated with form fields are children of the root element. When a Custom XML Data template is specified the attribute value is an XPath 1.0 expression that identifies the XML element associated with the field. If this attribute is omitted no mapping is established.

- `editable`

  This attribute specifies whether or not the field content is editable. Possible values are `true` (default) and `false`.

- `initial-value`

  This attribute specifies the initial value of the field. The specified value will be stored in the Custom XML Data part, unless a Custom XML Data template is in use. (This attribute has no effect if a Custom XML Data template has been specified. In this case the initial value must be stored in the Custom XML Data template as the content of the XML element associated with the field.)

- `locked`

  This attribute specifies whether or not the field is locked. Possible values are `true` (default) and `false`. (The feature of a locked field is that it cannot be deleted from the document.)

- `prompt`

  This attribute specifies placeholder text to be initially displayed in the field if no initial value is provided. (If both the `prompt` and `initial-value` attributes are specified the latter will take precedence.)

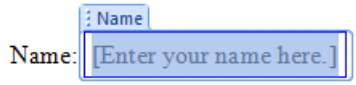
- `title`

  This attribute specifies the field title. This title is displayed as part of the field outline when the field is selected. The default value is specific to each field type.

3.2. sdt:text-field

This element is converted to a plain text **SDT**, which provides the functionality of a basic text field.

Figure 7.6. Text field



Attributes:

- `binding`

  See generic attributes.

- `editable`

  See generic attributes.

- `initial-value`

  See generic attributes.

- `locked`

  See generic attributes.

- `multi-line`

  This attribute specifies whether or not line breaks are allowed in the field value. Possible values are `true` and `false` (default).

- `prompt`

  See generic attributes.

- `title`

  See generic attributes. (The default value is `Text Field`).

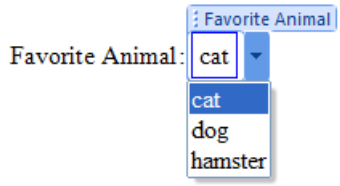
Content model:

```
EMPTY
```

3.3. sdt:drop-down-list

This element is converted to a drop-down list **SDT**, which provides the ability to select a single value from a predefined list.

Figure 7.7. Drop-down list



Attributes:

- `binding`

  See generic attributes.

- `editable`

  See generic attributes.

- `initial-value`

  See generic attributes.

- `locked`

  See generic attributes.

- `prompt`

  See generic attributes.

- `title`

  See generic attributes. (The default value is `Drop-Down List`).

Content model:

```
(sdt:list-entry)+
```

3.4. sdt:list-entry

This element specifies an entry in the list of possible values of a drop-down list or combo box **SDT**.

Attributes:

- `display-text`

  This attribute specifies alternative text to be displayed when this entry is selected. (By default the actual entry value is displayed.)

- `value`

  This attribute specifies the actual entry value. This is the value that will be stored in the Custom XML Data part when this entry is selected. This attribute is required. (The `sdt:list-entry` element is ignored if this attribute is omitted.)

Content model:

```
EMPTY
```

3.5. sdt:combo-box

This element is converted to a combo box **SDT**, which combines a text field and a drop-down list.

Attributes:

- `binding`

  See generic attributes.

- `editable`

  See generic attributes.

- `initial-value`

  See generic attributes.

- `locked`

  See generic attributes.

- `prompt`

  See generic attributes.

- `title`

  See generic attributes. (The default value is `Combo Box`).

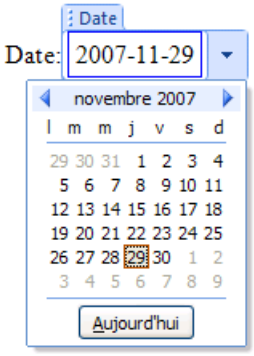
Content model:

```
(sdt:list-entry)+
```

3.6. sdt:date

This element is converted to a date **SDT**, which is a text field with date semantics. This **SDT** provides a date picker for fast and secure input, though a date value may be typed in as well.

Figure 7.8. Date



Attributes:

- `binding`

  See generic attributes.

- `editable`

  See generic attributes.

- `format`

  This attribute specifies the date format. (This format is used by the date picker but is not enforced when a value is typed in directly.) The attribute value is a character string in which the following variables are recognized:

Variable	Expanded Value
%D	day of month (01-31)
%M	month (01-12)
%Y	year (4 digits)
%y	year (last 2 digits)

  The default value is `%Y-%M-%D`.

- `initial-value`

  See generic attributes.

- `locked`

  See generic attributes.

- `prompt`

  See generic attributes.

- `title`

  See generic attributes. (The default value is `Date`).

Content model:

```
EMPTY
```

3.7. sdt:picture

This element is converted to a picture **SDT**, which provides the ability to select, display and edit images. The value of this field - stored as the content of the associated XML element in the Custom XML Data part - is the Base64-encoded image data.

Figure 7.9. Picture

Attributes:

- `binding`

  See generic attributes.

- `editable`

  See generic attributes.

- `locked`

  See generic attributes.

- `title`

  See generic attributes. (The default value is `Picture`).

Content model:

```
(sdt:image-data)?
```

3.8. sdt:image-data

This element specifies the initial value of an `sdt:picture` element. It contains the Base64-encoded image data to be initially displayed in the picture **SDT**. If this element is omitted an image placeholder will be displayed. This placeholder includes a button to open an image selection dialog.

Attributes:

- `format`

  This attribute specifies the image data format, in the form of a MIME type. Supported formats are GIF (`image/gif`), JPEG (`image/jpeg`) and PNG (`image/png`). This attribute is required. (The `sdt:image-data` element is ignored if this attribute is omitted.)

Content model:

```
#PCDATA
```

3.9. sdt:configuration

This element specifies optional parameters related to the Custom XML Data part. If this element is present in the XSL-FO tree it must occur before the first `fo:page-sequence` object.

Attributes:

- `custom-xml-template`

  This attribute specifies the URL of an XML template to be used as the initial content of the Custom XML Data part. This XML template must be encoded in UTF-8 or UTF-16. The URL is resolved by **XFC** using its current URI resolver.

- `prefix-mappings`

  This attribute specifies the mapping of namespace prefixes used in XPath expressions that identify an element in a Custom XML Data template. The attribute value is a list of namespace declarations separated by white space. This attribute is required if the Custom XML Data template makes use of namespaces. For instance, consider the XML template below:

```
<?xml version="1.0" encoding="UTF-8"?>
<order xmlns="http://www.xmlmind.com/ns/order">
  <product>
    <reference />
    <quantity />
  </product>
</order>
```

  As this template contains a namespace declaration, names in XPath expressions that identify an element in the template should be qualified. For this purpose one would set the `prefix-mappings` attribute and use the so declared namespace prefix to qualify element names in XPath expressions, as shown below.

```
<sdt:configuration
  custom-xml-template="custom.xml"
  prefix-mappings="xmlns:ns=&amp;quot;http://www.xmlmind.com/ns&amp;quot;/order" />
```

```
<sdt:text-field binding="/ns:order/ns:product/ns:reference"
  prompt="[Enter product reference.]" title="Reference" />
```

Content model:

```
EMPTY
```