TECHNICAL REPORT

ISO/IEC TR 29166

First edition
2011-12-15

Information technology — Document description and processing languages — Guidelines for translation between ISO/IEC 26300 and ISO/IEC 29500 document formats

Technologies de l'information — Description des documents et langages de traitement — Lignes directrices pour la traduction des formats de document ISO/CEI 26300 et ISO/CEI 29500

Reference number
ISO/IEC TR 29166:2011(E)

Contents

Page

Foreword ... vi
Introduction ... vii
1 Scope ... 1
2 Normative references ... 1
3 Terms and definitions ... 2
4 Basic principles ... 4
4.1 Structure of the report ... 4
4.1.1 Enterprise view ... 4
4.1.2 Computational view ... 5
4.1.3 Information view ... 5
4.1.4 Engineering view ... 5
4.1.5 Technical view ... 5
4.2 Approach ... 6
5 Use cases ... 8
5.1 Introduction ... 8
5.2 Word processing documents ... 8
5.2.1 Empty document ... 8
5.2.2 Simple text and paragraph formatting ... 10
5.2.3 Asian language support ... 11
5.2.4 Line breaks in East Asian text ... 14
5.2.5 Text direction ... 16
5.2.6 Phonetic guide functions ... 18
5.2.7 Tables and field functions ... 20
5.2.8 Footnotes and endnotes ... 22
5.2.9 Itemization and numeration ... 24
5.2.10 Indices and tables of contents ... 25
5.2.11 Metadata and settings ... 26
5.2.12 Change tracking and collaboration support ... 28
5.2.13 Bibliographies and optional document parts ... 31
5.2.14 Sub documents and books ... 32
5.2.15 Forms ... 35
5.2.16 Vector graphics ... 36
5.2.17 Font embedding and paper size ... 38
5.2.18 Font metrics and font substitution ... 40
5.2.19 Document fields ... 41
5.2.20 Inclusion of user defined XML ... 43
5.2.21 Mathematical formulas ... 45
5.3 Spreadsheet documents ... 46
5.3.1 Empty spreadsheet document ... 46
5.3.2 Listing and structural features ... 48
5.3.3 Formulas and calculation ... 49
5.4 Presentation documents ... 51
5.4.1 Empty presentation document ... 51
5.4.2 Simple text formatting ... 52
5.4.3 Itemization and numeration ... 54
5.4.4 Positioning and layout ... 55
5.4.5 Slide blending and animation effects ... 57
5.4.6 Animations ... 58
5.4.7 Comments ... 60
5.4.8 Multimedia content ... 62

5.4.9 Master layout 64
5.5 Common properties and mutual inclusion of documents 65
5.5.1 Hyperlinks between documents 65
5.5.2 Colours 67
5.5.3 Embedded spreadsheet documents 69
5.5.4 Simple text formatting and embedded documents 71
5.5.5 Embedded charts 73

6 Features and functionality 74
6.1 Introduction 74
6.2 Word processing documents 75
6.2.1 Text formatting 75
6.2.2 Paragraph formatting 77
6.2.3 Header and footer 82
6.2.4 Tables 82
6.2.5 Itemization and numeration 84
6.2.6 Metadata language entries 86
6.2.7 Indices 86
6.2.8 Change tracking and collaborative functions 87
6.2.9 Bibliographies and optional document parts 88
6.3 Spreadsheet documents 89
6.3.1 Introduction 89
6.3.2 Formatting 89
6.3.3 Calculation 90
6.3.4 Additional properties 94
6.4 Presentation documents 95
6.4.1 Introduction 95
6.4.2 Slides 95
6.4.3 Text formatting 96
6.4.4 Master layout 97
6.5 Common aspects 98
6.5.1 Alternative presentations 98
6.5.2 Colour models 99
6.5.3 Custom XML parts 100
6.5.4 Packages 100

7 Representation and XML structure 102
7.1 Introduction 102
7.2 Word processing documents 103
7.2.1 Logical structure 103
7.2.2 Paragraphs 105
7.2.3 Styles 106
7.2.4 Tables 107
7.2.5 Lists - Itemization and numeration 111
7.2.6 Indices 112
7.2.7 Change tracking and collaboration support 115
7.2.8 Section and page layout 118
7.3 Spreadsheet documents 120
7.3.1 Logical structure 120
7.3.2 Table contents 121
7.3.3 Table style 123
7.3.4 Formulas and calculation 124
7.3.5 Charts 126
7.4 Presentation documents 128
7.4.1 Logical structure 128
7.4.2 Text formatting 129
7.4.3 Master layout 131
7.4.4 Animations 132
7.5 Summary 137

8 Translation 137

8.1 Introduction 137
8.2 Translation complexity 137
8.3 Sample translations 139
8.3.1 Easy translation 139
8.3.2 Moderate translation 143
8.3.3 Difficult translations 149
8.4 Guidelines for evaluating translatability 150
8.4.1 Translation fidelity 151
8.4.2 Document interoperability 152

9 Examples and tools 153

10 Conclusion 154
10.1 Resume 155

Bibliography 156

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

In exceptional circumstances, when the joint technical committee has collected data of a different kind from that which is normally published as an International Standard (“state of the art”, for example), it may decide to publish a Technical Report. A Technical Report is entirely informative in nature and shall be subject to review every five years in the same manner as an International Standard.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC TR 29166 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 34, *Document description and processing languages*.

Introduction

OASIS Open Document Format ODF 1.0 (ISO/IEC 26300) and Office Open XML (ISO/IEC 29500) are both open document formats for saving and exchanging word processing documents, spreadsheets and presentations. Both formats are XML based but differ in design and scope.

OASIS ODF 1.0 was published by OASIS as an *OASIS standard* in May 2005. The second edition of ODF 1.0 has been published by OASIS as a *committee specification* in July 2006 and accepted as an International Standard by ISO (ISO/IEC 26300) in December 2006. Office Open XML was first approved in December 2006 by the ECMA International General Assembly as ECMA-376. An updated version was published in November 2008 by ISO (ISO/IEC 29500). The corresponding version, ECMA-376 2nd edition, was published in December 2008.

This Technical Report addresses both developers seeking to implement either the OpenDocument or the Office Open XML International Standard and template designers and other power users whose competences cut across the spectrum of XML and XML-related technologies which directly or remotely deal with one or both of the two International Standards. This Technical Report will be of great assistance to those seeking to exchange documents between formats, to extract data from or import data into documents, or to write applications supporting the two formats.

This Technical Report aims at analysing the two International Standards and their underlying concepts in terms of interoperability issues for a selected set of features. It analyses the way these features are implemented in both International Standards and estimates the degree of translatability between them using a table-based comparison. This Technical Report serves as a preliminary technical translation guideline for evaluating translatability between certain parts of the two International Standards. It does not compare different implementations which can cause additional kinds of interoperability problems.

Both Office Open XML and OpenDocument formats are basically descriptions of schemas used for word processing documents, spreadsheets and presentations created by office application suites. Both are open formats. A key design objective is to guarantee long-term access to data without the legal or technical barriers associated with proprietary binary formats. XML schema definitions are normative parts of both International Standards.

Manipulating documents is fundamentally facilitated by separating a document’s layout from its content. Editing the layout and data components independently of one another affords considerable flexibility in creating and editing office documents. Defining the structure and content of documents has been the focus of both International Standards. A document’s layout is ultimately governed by the implementation of the office suite, in particular by the rendering engine. Thus, as depicted in Figure 1, using exactly the same standard to describe a document does not guarantee that different office suites will produce identical layouts. Consequently, this Technical Report focuses more on the definition of guidelines for the translation of document structure, content and presentation instructions than on the preservation of document layout.

In this Technical Report the two International Standards will be examined in their universality and not by comparing specific implementations such as Microsoft Office or OpenOffice.org/LibreOffice. For this reason, various examples have been developed using a simple XML editor which supports both standards. The names of specific implementations may be used in the use cases to illustrate the real world scenario behind the use case. The figures in this Technical Report are created for illustration purposes, using available tools such as OpenOffice 3.* and Microsoft Office 2010. It should not be assumed that the current versions of these implementations support all the features needed to implement the use case, especially the document standards and the translation between them.

Several use cases do not mention existing tools, but rather use abstract names such as document format A (DF-A) and document format B (DF-B).

This Technical Report begins with a presentation of typical use cases characterizing scenarios where specific features supported by both document formats are used. It then analyses the most important features of one document format and shows how those features can best be represented in the other format. It then reviews the concepts and various features of the two document formats in order to provide a good understanding of the formats' common features and especially their differences. Most features can be translated to the other format with varying degrees of fidelity. For the most important features, this Technical Report provides detailed information on the implementation of the feature and the extents to which that feature can be translated, including typical translation rules. Finally, an overview summing up the most important results and deriving guidelines for the translation between both formats concludes this Technical Report.

The following abbreviations are used throughout this Technical Report:

- *ODF*, which stands for OpenDocument Format (ISO/IEC 26300:2006);
- *OOXML*, which stands for Office Open XML (ISO/IEC 29500:2008).

It is hoped that this Technical Report will be useful in understanding how the ODF and OOXML International Standards compare and how their functionality can be mapped between the two formats. It is a necessary step to the goal of helping achieve interoperability and harmonization between the two formats.

History of ODF and OOXML

ODF was originally developed by Sun Microsystems between 2000 and 2002 with the following objective:

"To create as a community, the leading international office suite that will run on all major platforms and provide access to all functionality and data through open-component based APIs and an XML-based file format." [1]

In 2002, the standardization process was initiated at OASIS, and in 2005 the standard was published as *OASIS Open Document Format for Office Applications*, abbreviated as *OpenDocument* or *ODF.* In 2006, *Open Document Format for Office Applications v.1.0* became an ISO International Standard (ISO/IEC 26300). *Open Document Format for Office Applications v.1.1* has been published by OASIS as an *OASIS standard* in February 2007. At the time of writing (June 2011) Version 1.2 has been released as a Committee Specification 1.0. While version 1.0 of the ODF standard only consists of one part, the current version is structured into three parts: core, formulas, and packages.

Microsoft followed suit in 2006 via the *Open Specification Promise* (OSP[2]) by opening the format of its 2007 version of the Microsoft office suite (version 12) for which it also uses XML as an exchange and storage format. OOXML is a file format originally developed by Microsoft as a successor to its earlier Office 2003 file formats. It is used for representing spreadsheet, presentation and word processing documents. In 2006 Office Open XML became an ECMA standard (ECMA-376, 1st edition). In 2008, a revised version of ECMA-376 became an ISO International Standard (ISO/IEC 29500:2008), which has its equivalent in the ECMA-376, 2nd edition.

ISO/IEC 29500 is structured into four parts, each of which contains normative as well as informative material: Fundamentals and Markup Language Reference, Open Packaging Conventions, Markup Compatibility and Extensibility, and Transitional Migration Features.

At the time of writing (June 2011) the following corrigenda and amendments have been published:

- ISO/IEC 29500-1:2008/Cor.1:2010, ISO/IEC 29500-2:2008/Cor.1:2010, ISO/IEC 29500-3:2008/Cor.1:2010 and ISO/IEC 29500-4:2008/Cor.1:2010, containing minor technical corrections and editorial modifications;
- ISO/IEC 29500-1:2008/Amd.1:2010 and ISO/IEC 29500-4:2008/Amd.1:2010, containing namespace changes and modifications concerning the usage of percentage (%) values;

1 **http://www.openoffice.org/about_us/ooo_release.html**

2 **http://www.microsoft.com/interop/osp/default.mspx**

- ISO/IEC 29500:2011 (ECMA 376 3rd edition) as a consolidated version of OOXML containing the above-mentioned corrigenda and amendments;
- ISO/IEC 26300:2006/Cor.1:2010, containing editorial modifications;
- ISO/IEC 26300:2006/Cor.2:2011, fixing editorial errors.

In addition, the following Amendments are under preparation:

- Amendment 1 to ISO/IEC 29500-1:2011 and Amendment 1 to ISO/IEC 29500-4:2011 about ISO 8601 dates;
- Amendment 1 to ISO/IEC 26300:2006 introducing ODF 1.1.

Information technology — Document description and processing languages — Guidelines for translation between ISO/IEC 26300 and ISO/IEC 29500 document formats

1 Scope

This Technical Report provides guidelines for translation between ISO/IEC 26300 and ISO/IEC 29500 document formats. It starts by studying common use cases to identify how the most important functionalities of one document format can be represented in the other format. This is followed by a thorough review of the concepts, architectures and various features of the two document formats in order to provide a good understanding of the commonalities and differences. It is expected that functionalities will be able to be translated with different degrees of fidelity to the other format. As an illustrative sample of this functionality, detailed information is provided on the extent to which those functionalities can be translated. This Technical Report is a necessary step to the goal of helping achieve interoperability and harmonization between the two formats.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 26300:2006, *Information technology — Open Document Format for Office Applications (OpenDocument) v1.0*

ISO/IEC 29500-1:2008, *Information technology — Document description and processing languages — Office Open XML File Formats — Part 1: Fundamentals and Markup Language Reference*

ISO/IEC 29500-2:2008, *Information technology — Document description and processing languages — Office Open XML File Formats — Part 2: Open Packaging Conventions*

ISO/IEC 29500-3:2008, *Information technology — Document description and processing languages — Office Open XML File Formats — Part 3: Markup Compatibility and Extensibility*

ISO/IEC 29500-4:2008, *Information technology — Document description and processing languages — Office Open XML File Formats — Part 4: Transitional Migration Features*

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1
translation type
methods used when translating between ODF and OOXML documents

NOTE This Technical Report distinguishes four *translation types*:

- one way ODF to OOXML translation;
- one way OOXML to ODF translation;
- round trip ODF to OOXML to ODF translation;
- round trip OOXML to ODF to OOXML translation.

3.2
translation fidelity
quality of a translation process between the ODF and OOXML *document formats*

NOTE 1 *Translation fidelity* depends on *document properties*.

NOTE 2 *Translation fidelity* cannot be measured in an absolute manner; it depends on the intentions of the document's authors.

3.3
document type
characterization of the specific purpose and content of a document

NOTE 1 This Technical Report distinguishes three major *document types*: word processing, spreadsheet and presentation documents.

NOTE 2 Some *document features* only exist in one *document type*; other *features* have been defined for more than one *document type*.

NOTE 3 The association between *document type* and *document feature* can be different for ODF and OOXML.

3.4
document property
description of different yet independent dimensions within the specification of a document

NOTE 1 As defined in 4.2 this Technical Report distinguishes the following *document properties*:

- presentation instructions;
- content;
- dynamic content;
- meta data;
- annotations and security;
- document parts.

NOTE 2 *Document properties* are implemented using *document features*.

3.5
document feature
characterization of a document used to implement specific aspects of a *document property*

NOTE 1 *Document features* are visible to a user.

NOTE 2 *Document features* are illustrated by associated use cases in this Technical Report.

NOTE 3 The terms *feature*, *functionality* and *sub functionality* are used to structure the comparison of both *document formats* in Clause 6.

NOTE 4 *Document features* implement *document properties*.

3.6
document format
synonym for *document standard* within this Technical Report

3.7
functionality
refinement of *document features*

NOTE 1 For example, the format of a paragraph is a *feature* and the height of a line is a *functionality*.

NOTE 2 In many cases *functionality* is implemented using XML types or elements.

3.8
sub functionality
itemization of *functionality*

NOTE 1 For example, the height of a line can be defined as fixed, font-independent, automatic, etc.

NOTE 2 In many cases *sub functionality* is implemented using XML attributes.

3.9
translatability level
rough scale for translation fidelity

NOTE 1 Translatability *levels* are used in Clause 6.

NOTE 2 *Translatability levels* have a three tier range (low, medium, high).

3.10
translation complexity
description of the complexity of the translation process for *document features*, considering their structures and associated translation rules

NOTE 1 *Translation complexity* is a three value metric system (easy, moderate, difficult*).*

NOTE 2 *Translation complexity* is used in Clause 8 to classify the translation of *document features* or *functionalities* from one format to the other.

4 Basic principles

4.1 Structure of the report

The report is structured according to the viewpoints introduced in the reference model for Open Distributed Processing ODP (ISO/IEC 10746). Refer to ISO/IEC 10746-1:1998 and ISO/IEC 10746-3:1996.

4.1.1 Enterprise view

The enterprise viewpoint is concerned with the purpose, scope and policies governing the activities of the specified system within the organization of which it is a part. All requirements that are relevant to defining the architecture and properties of the system are gathered in this viewpoint.

In clause 5 the TR describes the translation process from the enterprise viewpoint. It focuses on use cases that describe how a document is used in a specific scenario. Features and functionalities of documents like presentation instructions, structural information, application context, and the content itself as well as certain conformance classes based on translation types and properties have been taken into consideration. Users of document standards and decision makers are the intended readers of this section.

4.1.1.1 Use case template

To facilitate comparisons and a quick overview, use cases are described using the following template:

Textual description:

- *Describe the scenario/story the use case is going to tell;*
- *Include one or more figures demonstrating the use case (optional);*
- *Define the translation type and fidelity to be demonstrated.*

Implementation:

- *Describe the features necessary to implement the use case.*

Use case[3] name: Translation type and properties:	
One-trip translation	ODF → OOXML or/and OOXML → ODF
Round trip translation	ODF→OOXML→ODF or/and OOXML→ODF→OOXML
Presentation instructions	✓
Document content	✓
Dynamic content	✓
Metadata	✓
Annotations and security	✓
Document parts	✓

3 For details about the use case table please refer to section 4.2.

Additional properties	...
Required features: • *Feature a including references to standards* • *Feature b including references to standards*	

Requirements:

- *Describe the expected behaviour of a feature's translation between both International Standards;*
- *Describe how the document(s) used in the use case should be defined to achieve the intended fidelity.*

Conclusion:

- *Compare the applicable features in both International Standards and the translation rules and fidelity as elaborated on in clauses* 6 *and* 8.

4.1.2 Computational view

The computational viewpoint is concerned with the functional decomposition of a system into a set of objects that interact at interfaces: thus enabling system distribution.

In clause 6 the TR describes the translation process from the computational viewpoint. It focuses on the features and functionalities of a document. The *what* is described, independent of *how* the feature is implemented in the particular standard. Power users and developers are the intended readers of this section.

4.1.3 Information view

The information viewpoint is concerned with the kinds of information handled by a system and constraints on the use and interpretation of that information. An information specification of a system defines the semantics of information and the semantics of information processing.

In clause 7 the TR describes the translation process from the information viewpoint. It is focusing on *how* the functionality and features of a document are implemented in the standards. The document structure and its XML markup are described. Power users, developers and persons responsible for the maintenance of the standards are the intended readers of this section.

4.1.4 Engineering view

The engineering viewpoint is concerned with the infrastructure required to support system distribution. An engineering specification defines the mechanisms and functions required to support distributed interaction between objects.

In clause 8 the TR describes the translation process from the engineering viewpoint. It focuses on how the features and structures are translated and preserved in the translation process. Developers and persons responsible for the maintenance of the International Standards are the intended readers of this section.

4.1.5 Technical view

The technology viewpoint is concerned with the choice of technology used to support system distribution.

In clause 9 the TR describes the translation process from the technical viewpoint. It focuses on available resources and tools for creating, editing and translating documents. All groups mentioned above are the intended readers of this section.

4.2 Approach

This TR takes a use case based approach to identify the requirements needed for translating between ODF and OOXML. As depicted in Figure 2, use cases are selected and categorized along two lines: *type of translation* and *document properties* defining the fidelity of a translation. This approach covers all aspects of translations between the two document formats. Both International Standards define a storage and exchange format for documents, including information about both a document's presentation and its content. The process of document rendering or laying out is beyond the scope of the actual standards, and thus beyond the scope of both the translation process and this report.

Graphic fidelity between different rendering engines (i.e. layout implementations), another important category of uses cases, is also beyond the scope of this report. In such use cases, different rendering engines are provided with the same information, but may produce visually different results. Since the actual layout process is not described by either the ODF or the OOXML International Standard, this report does not deal with such use cases. However, it does cover preservation of layout information and presentation instructions around format translations so that the *selfsame* rendering engine can produce the same visual result from the same information encoded in different formats as depicted in Figure 1. Nevertheless, from a user's point of view in many use cases the graphical appearance of a document will be the major criterion for the evaluation of the fidelity of a transformation process. Therefore the graphical appearance of the documents depicted in Figure 1 should be independent of the chosen path.

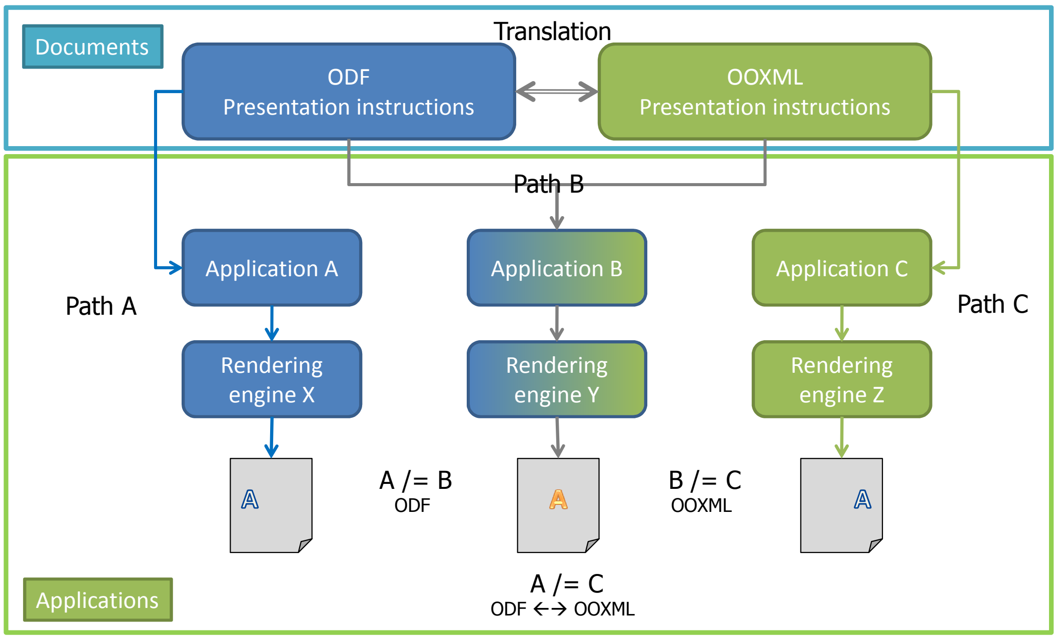


Figure 1 — Translation of presentation instructions

Use case descriptions reference subclause 4.2 for the description of demonstrated translation types and fidelities and clause 6 for a comparison of required features and functionalities.

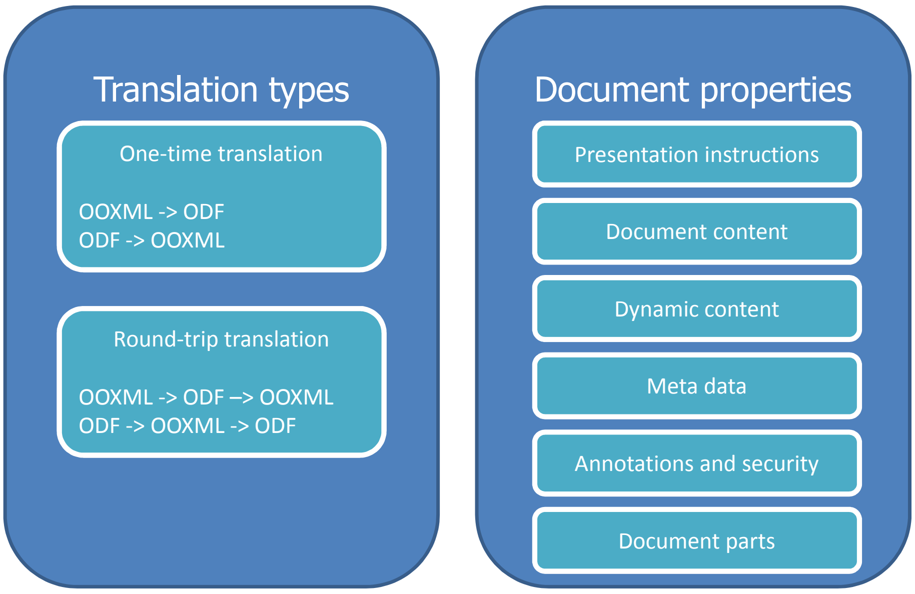


Figure 2 — Use case category overview

Translation fidelity of a document depends on the following document properties:

- *Presentation instructions* include all layout and presentation related information such as fonts, spacing, margins, colours, paper layout and settings, and animation in office documents.
- *Document content* covers all properties of content (such as text, graphics and formulas) defined directly by the author of a document.
- *Dynamic content* covers all aspects of automatically generated content including calculations or form functionalities such as fields, generated tables or dynamic references.
- *Metadata* cover all information apart from the core document content. Metadata are used to describe meta information about the document such as the generator, version, authors and to ensure the accessibility of documents, for instance by using certificates.
- *Annotation and security* covers all aspects of annotations used in a document including comments, change tracking, collaborative functions and security features such as encryption and access control.
- *Document parts* cover all aspects (editing semantics) of structural document properties such as paragraphs, headings, headers, footers, tables, lists, footnotes, indices and captions.

Some use cases focus on a one way translation process from standard A to standard B; a typical scenario is where the author of a document uses a tool that supports a different standard than the tool used by the reader. In the case that different authors of a document are using tools supporting different standards a round trip translation process has to be supported. The requirements for such processes are much higher than for one-way translations because mutual mappings between the International Standards have to be possible.

The report introduces the concept of *use cases* to demonstrate document *features*. As depicted in Figure 3, these features are used to implement the document *properties* described above. Each feature is refined by *functionalities* and *sub functionalities*. The use cases, together with their required document features, are introduced in clause 5. Features, functionalities and sub functionalities are described and compared in clause 6.

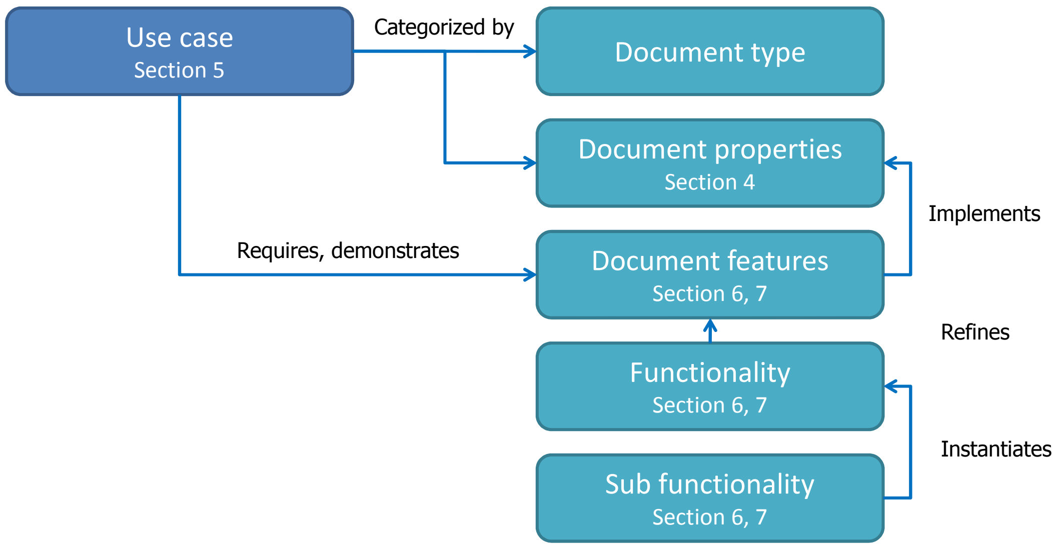


Figure 3 — Relation between functionality and document properties and features

A detailed description of the XML representations of selected document features used in the use cases is given in clause 7. These descriptions show how similar features are implemented in specific ways in both document formats. In some cases a feature is implemented using comparable and easy translatable structures, in some cases a feature has to be implemented by a combination of corresponding features in the other document format and in some cases a feature is only available in one of the two formats. Clause 8 concludes the analysis of specific XML representations and introduces examples of characteristic translation algorithms covering the whole translation spectrum. It gives examples for high visual fidelity as well as for high structural fidelity. Guidelines for end users telling what can be done and what should be avoided during the joint preparation of a document will be derived from this variety of translation fidelities.

5 Use cases

5.1 Introduction

This TR takes a use case based approach in order to identify the necessary requirements for translating between ODF and OOXML documents. It describes the translation process from the ODP enterprise viewpoint when utilizing use cases and states how a document should be used in a specific scenario. The expected and observable behaviour of a translation process is described here, as based on the translation types and documents properties explained above. The comparison of both behavioural types is then used to measure the fidelity of this translation process.

The TR defines use cases for each of the three major document types and for documents consisting of mutual included document types.

5.2 Word processing documents

5.2.1 Empty document

Textual description:

When a new document is created either in ODF format or in OOXML format, the user initially sees an empty document. When the document is saved without any further editing, it is generated without user content but it contains some initial metadata and presentation instructions. This initial content should be preserved as much as possible during the translation process.

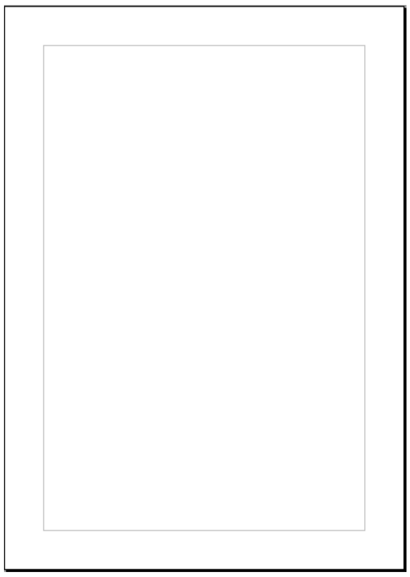


Figure 4 — Empty word processing document

Implementation:

The term *empty document* is not defined in both International Standards. Each application has its own initial metadata and presentation instructions. For example, an empty document has a default section, a default paragraph and a default definition of a page layout without user defined content.

Use case name: Empty wordprocessing document Translation type and properties:	
One-trip translation	✓
Round trip translation	✓
Presentation instructions	✓
Document content	
Dynamic content	
Metadata	✓
Annotations and security	
Document parts	
Required features: • Metadata o OOXML: Subclause 8.3; 17.* o ODF: Subclause 3.1	

Requirements:

The following behaviour should be maintained no matter which standard is used:

- Presentation and style instructions remain unchanged;
- Metadata remain unchanged.

It is not necessarily expected that both International Standards will use similar defaults for metadata.

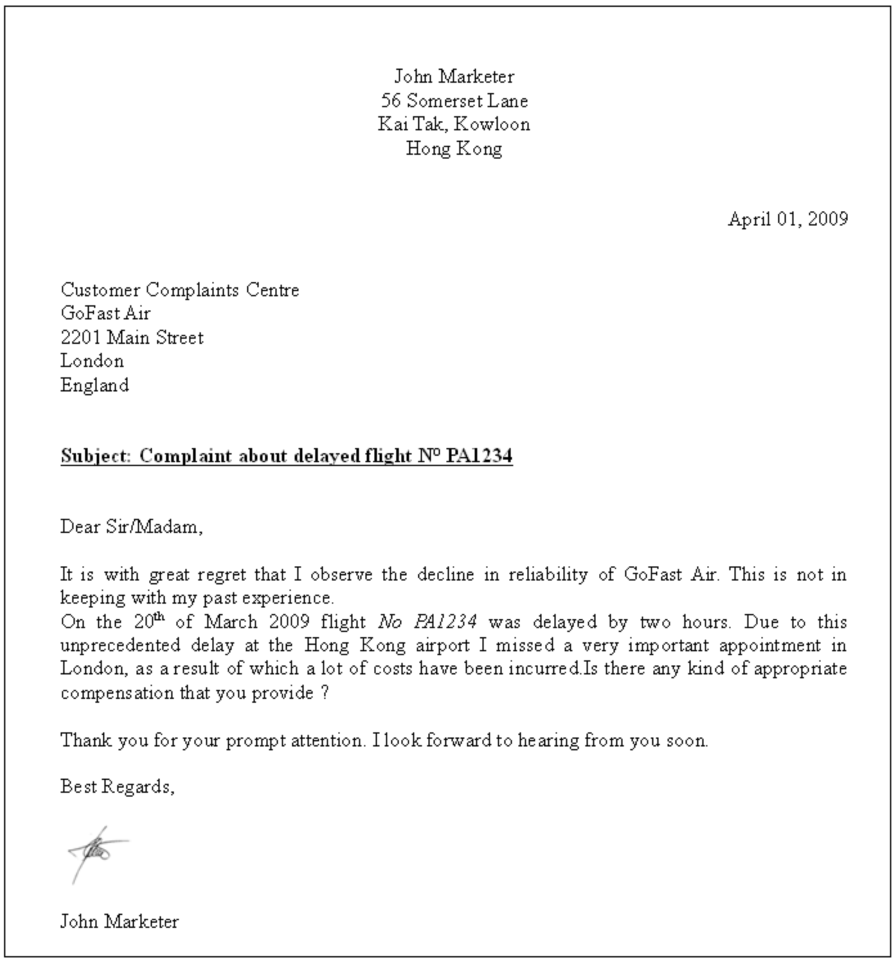
Conclusion:

Neither ODF nor OOXML can precisely define the term *empty document*. Thus the content of an empty document depends more on the application used to create it than on the standard. When an empty document defined in format A is opened in format B, presentation instructions can be preserved. However, the initial view of the empty document may be slightly different, depending on the rendering engine. Metadata can be translated accordingly, even though some information like the application creating the document may be modified.

5.2.2 Simple text and paragraph formatting

Textual description:

This use case describes the issue of a business letter and its translation between the International Standards ODF and OOXML with a special focus on formal aspects. The scenario starts with user John who has decided to file a complaint to his preferred airline about a delayed flight which caused some trouble to his agenda. John works on his private laptop using format A and starts to write the letter which looks like the one depicted in Figure 5. After finishing the letter, John emails it to his secretary Mary. Mary imports the document to a tool using format B to check visual appearance and spelling. Then she emails it to the customer complaints centre (CCC) of GoFast Air in London. The receiving agent in the CCC in London works again with format A.



John Marketer
56 Somerset Lane
Kai Tak, Kowloon
Hong Kong

April 01, 2009

Customer Complaints Centre
GoFast Air
2201 Main Street
London
England

<u>Subject: Complaint about delayed flight N° PA1234</u>

Dear Sir/Madam,

It is with great regret that I observe the decline in reliability of GoFast Air. This is not in keeping with my past experience.
On the 20th of March 2009 flight *No PA1234* was delayed by two hours. Due to this unprecedented delay at the Hong Kong airport I missed a very important appointment in London, as a result of which a lot of costs have been incurred.Is there any kind of appropriate compensation that you provide ?

Thank you for your prompt attention. I look forward to hearing from you soon.

Best Regards,

[illegible]

John Marketer

Figure 5 — Sample letter

Implementation:

This sample letter makes use of all typical text formatting features. There is centred text on the top and the date information is positioned on the right. The receiver's address is aligned on the left. The letter's body paragraph is justified. The letter contains a bold paragraph as the subject line as an example of *paragraph formatting*. Embedded italic characters in the body text are used as an example of the *text formatting* feature that allows specific attributes for parts of a paragraph to be defined. An image representing the signature of the author is embedded at the end of this document. Presentation instructions, content and parts of the document must be preserved during translation.

Use case name: Simple text formatting **Translation type and properties:**	
One-trip translation	✓
Round trip translation	✓
Presentation instructions	✓
Document content	✓
Dynamic content	
Metadata	
Annotations and security	
Document parts	✓

Required features:
• Text formatting
  o OOXML: Sublause 8.3, 17.*
  o ODF: Subclause 2.3, 4, 9.5, 14.*, 15.4
• Paragraph formatting
  o OOXML: Subclause15.2, 17.*
  o ODF: Subclause 2.8, 4.2, 9.3, 7.12, 14.*, 15.*	

Requirements:

This scenario requires the preservation of presentation instructions during multiple translations of a formal business document. A formal business letter is a common example of the application of basic text processing functionality. A formal letter should strictly conform to a set of guidelines which can be divided into aspects of presentation and content. Regardless of the text processing applications used to create it, a business letter's appearance and structure should remain identical throughout the translation process.

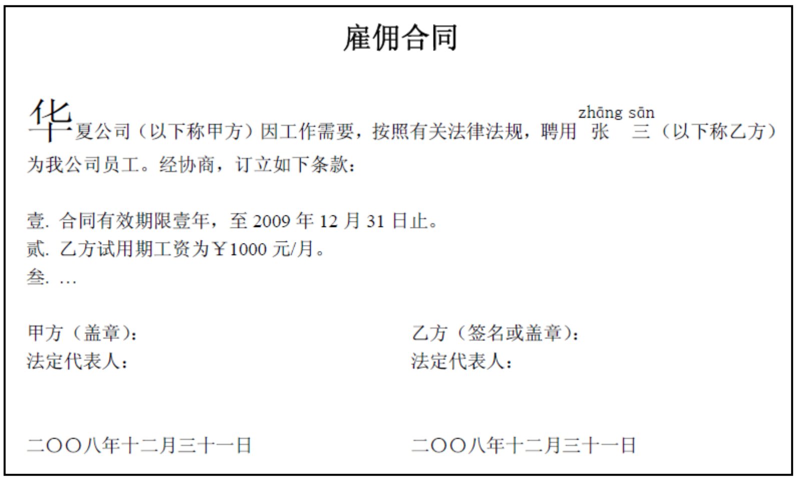
Conclusion:

Table 1 and Table 2 in Subclauses 6.2.1 and 6.2.2 show how good the required features can be translated between the two International Standards. Simple text formatting such as bold or italic characters and paragraph formatting such as text alignments can easily be converted between the two formats, with the exception of *theme fonts*, which are not supported in ODF.

5.2.3 Asian language support

Textual description:

Mr Zhang San is going to sign an employment contract with Huaxia, Inc. The human resources (HR) department of Huaxia, Inc. prepares the draft in format A. The draft is then sent to Mr Zhang San's email inbox to ask for comments. Mr Zhang San opens the draft using software supporting format B. After some discussion Mr Zhang San finally agrees with the content. He signs the contract and sends it back to Huaxia, Inc. The HR department also signs the contract and prints paper copies.



雇佣合同

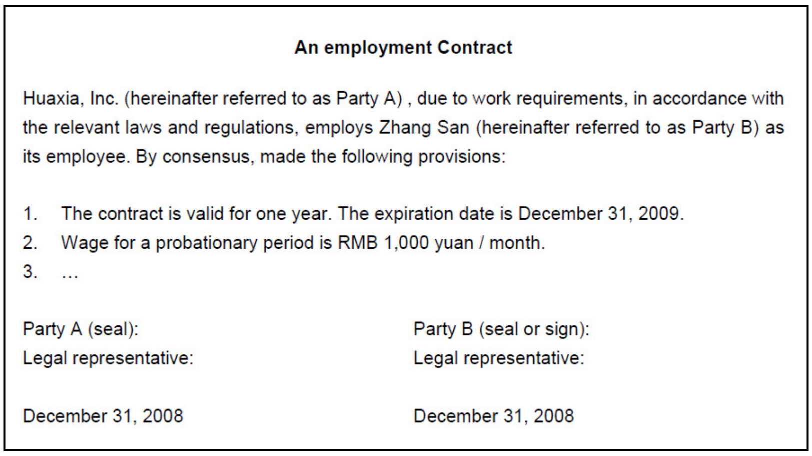
华夏公司（以下称甲方）因工作需要，按照有关法律法规，聘用 $\overset{\text{zhāng}}{\text{张}}$ $\overset{\text{sān}}{\text{三}}$（以下称乙方）为我公司员工。经协商，订立如下条款：

壹. 合同有效期限壹年，至 2009 年 12 月 31 日止。
贰. 乙方试用期工资为￥1000 元/月。
叁. …

甲方（盖章）:	乙方（签名或盖章）:
法定代表人:	法定代表人:
二〇〇八年十二月三十一日	二〇〇八年十二月三十一日

Figure 6 — Chinese employment contract

The screenshot above is a short employment contract in Chinese. The translated version is shown in Figure 7:



An employment Contract

Huaxia, Inc. (hereinafter referred to as Party A) , due to work requirements, in accordance with the relevant laws and regulations, employs Zhang San (hereinafter referred to as Party B) as its employee. By consensus, made the following provisions:

1. The contract is valid for one year. The expiration date is December 31, 2009.
2. Wage for a probationary period is RMB 1,000 yuan / month.
3. …

Party A (seal):	Party B (seal or sign):
Legal representative:	Legal representative:
December 31, 2008	December 31, 2008

Figure 7 — Translated contract

In the first line of the body of text above, the first character[4] 华 is dropped into two lines. The employee name is expressed in Chinese characters with Pin-Yin (phonetic transcription). The contract item list is numbered in Chinese sequence characters 壹、贰、叁. The first item in the list shows the date which is a mixture of Arabic numbers and Chinese characters. The second item in the list uses the currency symbol ￥. Another date is used in the bottom line. It consists of Chinese characters only and is different from the previous date.

4 The first character in the sample text is a SimSun character.

Implementation:

Use case name: Asian language support **Translation type and properties:**	
One-trip translation	✓
Round trip translation	✓
Presentation instructions	✓
Document content	✓
Dynamic content	
Metadata	
Annotations and security	
Document parts	✓
Required features: • Text formatting o OOXML: Subclause 17.3.1*, 17.3.2.10, 17.3.2.20, 17.3.2.26, 17.3.3.25-28, 17.15.1, 17.16.4.3.1, 17.16.5.*, 17.18.59, 18.3.1.92, 18.18.6.*, 18.18.27, 18.18.73, 20.1.10.61, M.1.9.2 o ODF: Subclause 5.4, 6.2.*, 12.2.2, 15.2.21, 15.4.*, 15.5.32-33, 15.6	

Requirements:

This scenario requires the preservation of presentation instructions during multiple translations of a formal business document. Chinese document editing usually has to consider the following properties:

a) *Font family,* which depends on the encoded character set being used. Furthermore, a word processing document should be able to specify up to four fonts which can be used in the contents of a run, for example, ASCII font, High ANSI font, East Asian font and Complex Script font.

b) *Font size,* which may use different measurements from *pt* (point), e.g. hao (号);

c) *Chinese special characters*, such as currency symbol ￥, which can be found in certain fonts only. It requires *number formats*, especially in spreadsheets that support such currency symbols.

d) *Special numbers* such as full-width decimal １, ２, ３; traditional ideograph 甲,乙,丙; zodiac ideograph 子,丑,寅; Chinese counting 一,二, 三; and Chinese legal simplified, which are used in the numbered list of the above use case: 壹,贰,叁.

e) *Special date and time format* such as “August 9, 2008” can be expressed as 二〇〇二年八月九日 or 2008 年 8 月 9 日; “5:36” am can be expressed as 上午五时三十六分 or 上午 5 时 36 分.

f) *Digital formats* such as 12345 can be expressed as 一万二千三百四十五 or 壹万贰仟叁佰肆拾伍 (popularly used in accounting).

g) *Writing direction* such as lines from right to left and text flow from top to bottom;

h) *East Asian typography rules* for first and last character per line (kinsoku) such as the characters which are not allowed to appear at the beginning and end of the lines, or the rule defining whether the space between Chinese and Western characters should be adjusted or not.

i) *Sorting method* to be used when sorting data in either word processing tables or spreadsheet tables, defining whether data should be sorted by Pin-Yin or stroke;

j) *Chinese typography settings* such as characters above other characters, phonetic transcription Pin-Yin used in the above use case, and similar;

k) *Automatically adjusting* the spacing of Latin and Chinese text as well as Chinese text and numbers;

l) *Settings for the document grid*, which enables precise layout of full-width Chinese characters within a document by specifying the desired number of characters per line and lines per page for all Chinese text content in the section.

A formal Chinese document should strictly adhere to these typesetting conventions. A formal Chinese document's appearance and structure should remain identical throughout the translation process, regardless of the word processing applications used to create it.

Conclusion:

Both formats offer support for Asian languages, such as Chinese fonts, sorting methods, kinsoku, text direction and document grids mentioned in the requirements. But due to differences in descriptive power, the appearance and structure created by either format may have some discrepancies when translated into another format. For example, as fonts for text runs (span) both formats support Western fonts, the East Asian font and the Complex Script font. However, ODF can specify different sizes and weights for each of them while OOXML can't. When displaying special numbers in numbering, OOXML provides a number construction method (counting mode) as well as a number sequence where each sequence can have different numbering formats, e.g., *chineseCountingThousand* and *chineseCounting*. In ODF only the *number sequence* is supported, and it can use one numbering format only. Both formats provide date and time formats, but they are not identical. Therefore it is not known if every format can be translated without difficulty. For example, 二〇〇八年十二月三十一日星期三 in OOXML cannot be translated into ODF. Differences also occur in typesetting taboos and rules for Asian languages. OOXML provides a superset of the typesetting taboos and rules in ODF, therefore some of this information will be lost in the translation. Regarding the phonetic guide, OOXML specifies the appearance in more detail than ODF does. For example, OOXML allows the distance between phonetic guide text and phonetic guide base text as well as the phonetic guide text font size to be appointed; this is not supported by ODF although the font size of the ruby text can be adjusted by the associated character style.

5.2.4 Line breaks in East Asian text

Textual description:

Chulsoo uses a format A word processing application to write down Korea's national anthem lyrics for his cousin who uses a format B word processing application. Since Chulsoo's cousin is a second generation American Korean who does not speak Korean well, Chulsoo wants to make sure that his cousin learns correct Korean including word spacing and the correct grammar rules through the lyrics.

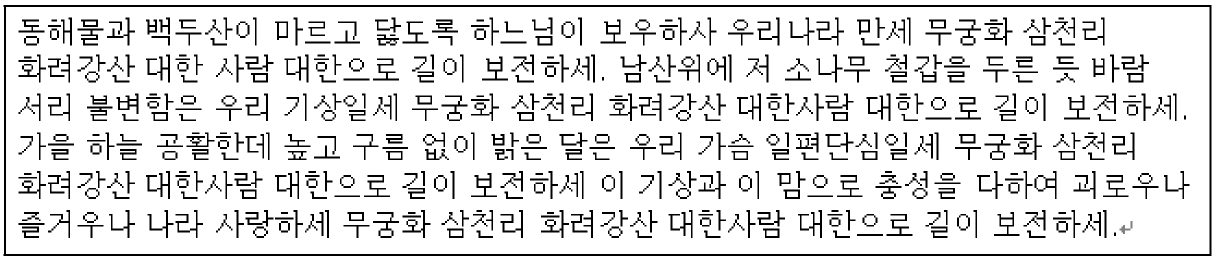

동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라 만세 무궁화 삼천리
화려강산 대한 사람 대한으로 길이 보전하세. 남산위에 저 소나무 철갑을 두른 듯 바람
서리 불변함은 우리 기상일세 무궁화 삼천리 화려강산 대한사람 대한으로 길이 보전하세.
가을 하늘 공활한데 높고 구름 없이 밝은 달은 우리 가슴 일편단심일세 무궁화 삼천리
화려강산 대한사람 대한으로 길이 보전하세 이 기상과 이 맘으로 충성을 다하여 괴로우나
즐거우나 나라 사랑하세 무궁화 삼천리 화려강산 대한사람 대한으로 길이 보전하세.↵

Figure 8 — Korean text with character unit support

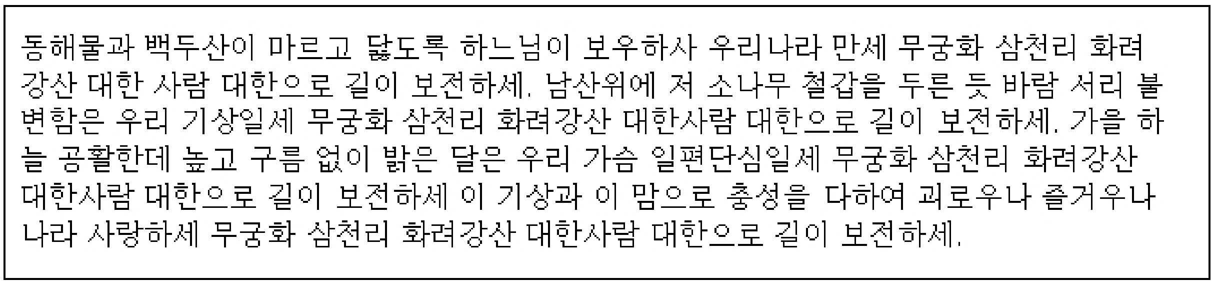

동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라 만세 무궁화 삼천리 화려
강산 대한 사람 대한으로 길이 보전하세. 남산위에 저 소나무 철갑을 두른 듯 바람 서리 불
변함은 우리 기상일세 무궁화 삼천리 화려강산 대한사람 대한으로 길이 보전하세. 가을 하
늘 공활한데 높고 구름 없이 밝은 달은 우리 가슴 일편단심일세 무궁화 삼천리 화려강산
대한사람 대한으로 길이 보전하세 이 기상과 이 맘으로 충성을 다하여 괴로우나 즐거우나
나라 사랑하세 무궁화 삼천리 화려강산 대한사람 대한으로 길이 보전하세.

Figure 9 — Korean text without character unit support

Implementation:

To ensure the Korean word spacing grammar rules are correctly observed, character units for East Asian text should be implemented in both word processing formats. At the end of first lines in Figure 8 and Figure 9, the word 대한 is supposed to be one unit after the 화려강산 unit. Format A in Figure 8 changes the line to correctly display the Korean words while format B in Figure 9 breaks down one word into two lines.

Use case name: Line breaks in East Asian languages Translation type and properties:	
One-trip translation	✓
Round trip translation	
Presentation instructions	✓
Document content	✓
Dynamic content	
Metadata	
Annotations and security	
Document parts	
Required features: • Indexes o OOXML: Subclause 21.1.2.2.2, 17.3.1.16 o ODF: Subclause 15.5.34	

Requirements:

If a format A user choose not to allow Korean texts to wrap in the middle of a word, then at the end of a line, a Korean word would stick together without breaking into two parts in two different lines regardless of the line break. After the translation to format B is complete, a user of format B should able to see the Korean word as one unit. The main requirement in this scenario is that the Korean words at the end of the lines should stay as one word when the document translates from format A to format B.

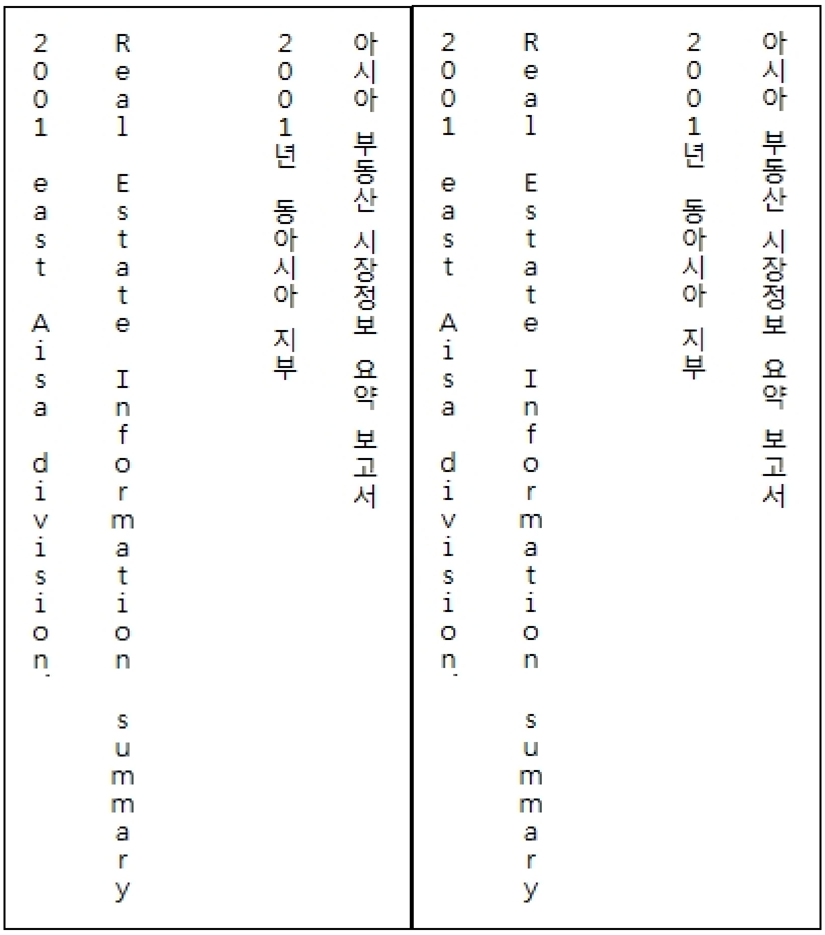
Conclusion:

OOXML supports the function that East Asian words should not be broken in case of a line break. DrawingML also introduces an attribute called @*enLnBrk* which specifies whether an East Asian word can be broken in the middle of a word and wrapped onto the next line without a hyphen being added. ODF does not support line break rules for East Asian words. Furthermore, when users use East Asian in their documentation, both International Standards do not support hyphenation use in line breaks. Detailed requirements on Japanese text layout including line breaks are given in the W3C Working Group Note http://www.w3.org/TR/2009/NOTE-jlreq-20090604/.

5.2.5 Text direction

Textual description:

John needs to figure out how his company's East Asia Division EAD has been working on their quest to balance the worldwide real estate business. He asks his EAD partner for a sales report. John uses his format A word processor to open the EAD report provided by his EAD partner who in turn uses a format B word processor.



아시아 부동산 시장정보 요약 보고서

2001년 동아시아 지부

Real Estate Information summary

2001 east Aisa division

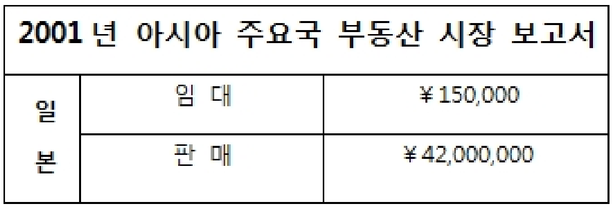
아시아 부동산 시장정보 요약 보고서

2001년 동아시아 지부

Real Estate Information summary

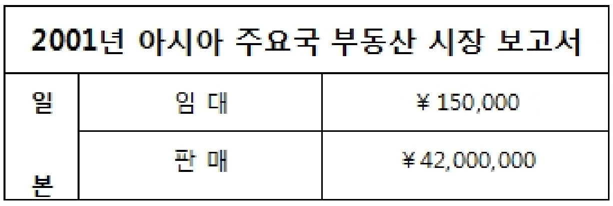
2001 east Aisa division

Figure 10 — Top to bottom vertically and right to left text flow in paragraphs in OOXML and ODF



2001 년 아시아 주요국 부동산 시장 보고서		
일본	임 대	¥150,000
	판 매	¥42,000,000

Figure 11 — Top to bottom vertically and left to right text flow in an OOXML table



2001년 아시아 주요국 부동산 시장 보고서		
일본	임 대	¥150,000
	판 매	¥42,000,000

Figure 12 — Top to bottom vertically and left to right text flow in an ODF table

Implementation:

Various text flows can occur in a document. Text can flow from top to bottom vertically, left to right horizontally and vice versa respectively in the entire document or a document part such as a paragraph, a table, and a section.

Use case name: Text direction **Translation type and properties:**	
One-trip translation	✓
Round trip translation	
Presentation instructions	✓
Document content	✓
Dynamic content	
Metadata	
Annotations and security	
Document parts	✓

Required features:
• Text direction
  o OOXML: Subclause 17.3.1.41, 17.4.7.3, 17.6.20, 17.18.93
  o ODF: Subclause 15.2.19, 15.4.42~43, 15.5.39~37, 15.7.8, 15.8.13, 15.11.1, 15.11.3, 15.27.9~12, 15.27.27	

Requirements:

In Figure 10 the text flow in the paragraph is top to bottom vertically. In Figure 11 and Figure 12 the text flow in the tables is top to bottom vertically for country names and the other texts flow left to right horizontally. The main requirement in this scenario is that the text flow in a paragraph and a table in a format B document can be correctly translated into a corresponding paragraph and table in a format A document.

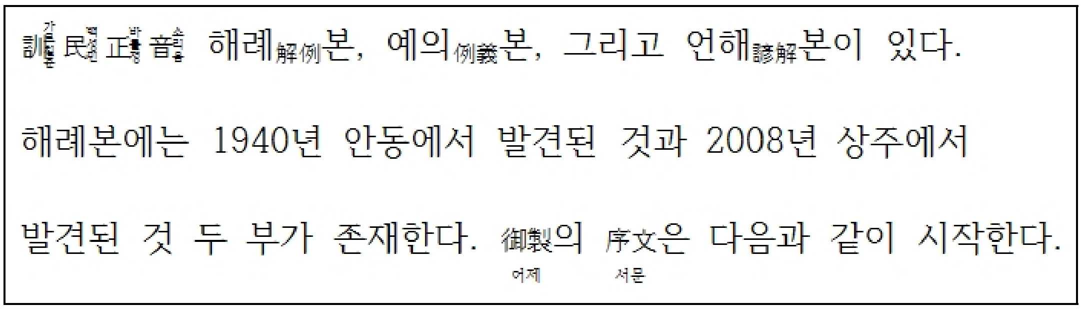
Conclusion:

Text flow in paragraphs and tables can be well translated between both formats. Both International Standards support top to bottom vertically, right to left horizontally, and vice versa respectively. In OOXML text directions apply in paragraphs, tables, and sections. In ODF text directions apply in paragraphs, tables, and frames. However, note that when dealing with text direction within a table, ODF inherits the text direction of the associated paragraph while OOXML uses separate text directions within the table.

5.2.6 Phonetic guide functions

Textual description:

Yunjung opens a report about the history of the Korean language from her colleague Prof. Kim. Yunjung uses a word processing application which uses format A to open the file. Prof. Kim had created it with a different word processing application which uses format B. Figure 13 shows a sample paragraph of the report created with an OOXML tool.



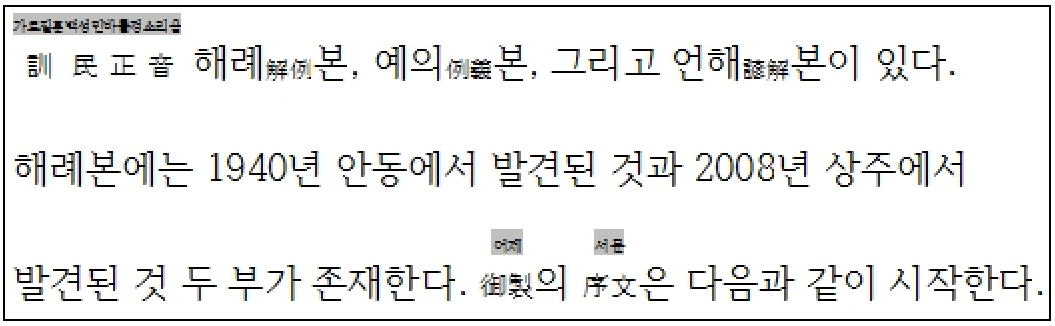
訓民正音 해례解例본, 예의例義본, 그리고 언해諺解본이 있다.

해례본에는 1940년 안동에서 발견된 것과 2008년 상주에서

발견된 것 두 부가 존재한다. 御製의 序文은 다음과 같이 시작한다.

어제 서문

Figure 13 — Sample ruby text[5] in Korean in OOXML



가르칠훈 백성민 바를정 소리음

訓 民 正 音 해례解例본, 예의例義본, 그리고 언해諺解본이 있다.

해례본에는 1940년 안동에서 발견된 것과 2008년 상주에서

어제 서문

발견된 것 두 부가 존재한다. 御製의 序文은 다음과 같이 시작한다.

Figure 14 — Possible result of a translation to ODF

Implementation:

One of the more advanced features of word processing is the usage of the ruby text especially in Asian languages such as Chinese, Japanese, and Korean. The Korean language is based on Hanja (Chinese characters). These characters are sometimes insufficient to determine meanings if the intended readers of the text are not familiar with Hanja. To avoid this, Korean text uses ruby text to provide additional information. This excerpt shows a sample paragraph using ruby texts in a Korean text document. There are right vertical, right, and bottom ruby texts associated with the base characters in Figure 13.

5 Refer to http://www.w3.org/TR/ruby/ for information about "ruby text"

Use case name: Phonetic guide functions **Translation type and properties:**	
One-trip translation	✓
Round trip translation	
Presentation instructions	✓
Document content	✓
Dynamic content	
Metadata	
Annotations and security	
Document parts	✓
Required features: • Text direction o OOXML: Subclause 17.3.3.25~28; 17.18.75 o ODF: Subclause 5.4; 14.8.4; 15.2.21~28; 15.6.*	

Requirements:

During the translation between the two formats, presentation instructions such as the vertical, above and below ruby text position with its associated base text, as shown in Figure 13, should be preserved. Document parts should be consistently translated, thus enabling readers to edit the ruby text and super/sub scripts within their position.

Conclusion:

ODF and OOXML both support ruby text. OOXML supports the @*rightVertical* attribute, which specifies that phonetic guide text must be right aligned with respect to the base text, and displayed vertically and to the right of the base text, regardless of the alignment of the base text. ODF does not support this right vertical alignment which is used quite often in Asian languages including Korean with Hanja characters.

5.2.7 Tables and field functions

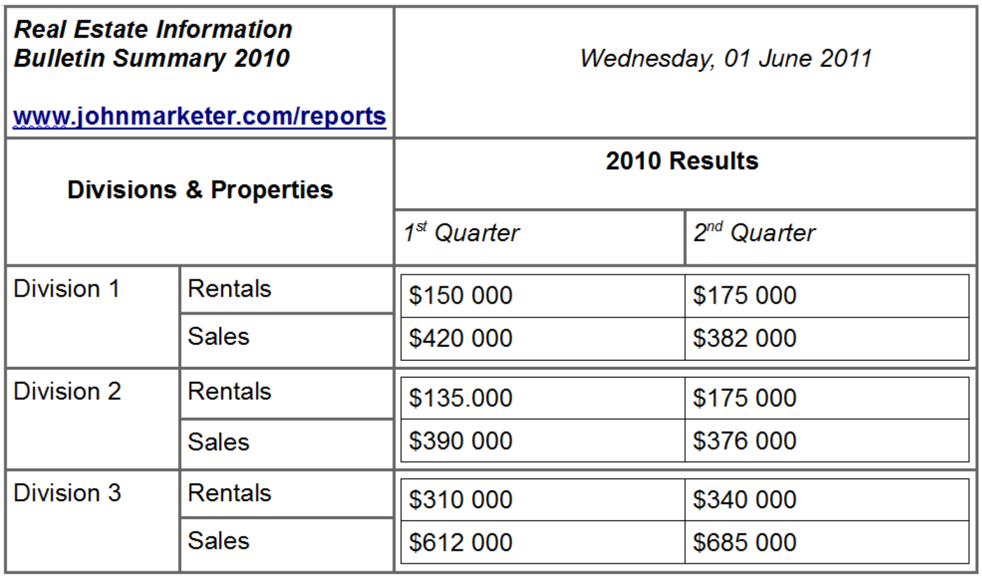
Textual description:

Using format A John plans to provide an invoice summary draft for his marketers to inform them about their monthly trading results. After filling out the sales report John emails it to his marketers. Using different word processing applications supporting format B, the marketers wait for the figures so they can spread the news to their own division staff.

Figure 15 shows a brief excerpt from the sales report.

Implementation:

One of the more advanced features of text processing is the usage of tables and predefined field functions as seen in Figure 15. This excerpt shows a table with joined cells and common text formatting. Cells are joined up to span multiple rows and columns. Different cell alignments appear as left, centre and right aligned text. A hyperlink is inserted into the header row of the table. The figures are presented as nested tables.



Real Estate Information Bulletin Summary 2010 www.johnmarketer.com/reports		*Wednesday, 01 June 2011*	
Divisions & Properties		**2010 Results**	
		1st Quarter	*2nd Quarter*
Division 1	Rentals	$150 000	$175 000
	Sales	$420 000	$382 000
Division 2	Rentals	$135.000	$175 000
	Sales	$390 000	$376 000
Division 3	Rentals	$310 000	$340 000
	Sales	$612 000	$685 000

Figure 15 — Sample table

Use case name: Tables and field functions **Translation type and properties:**	
One-trip translation	✓
Round trip translation	
Presentation instructions	✓
Document content	✓
Dynamic content	
Metadata	
Annotations and security	
Document parts	✓
Required features: • Tables ○ OOXML: Subclause 17.* ○ ODF: Subclause 8.1, 15.*	

Requirements:

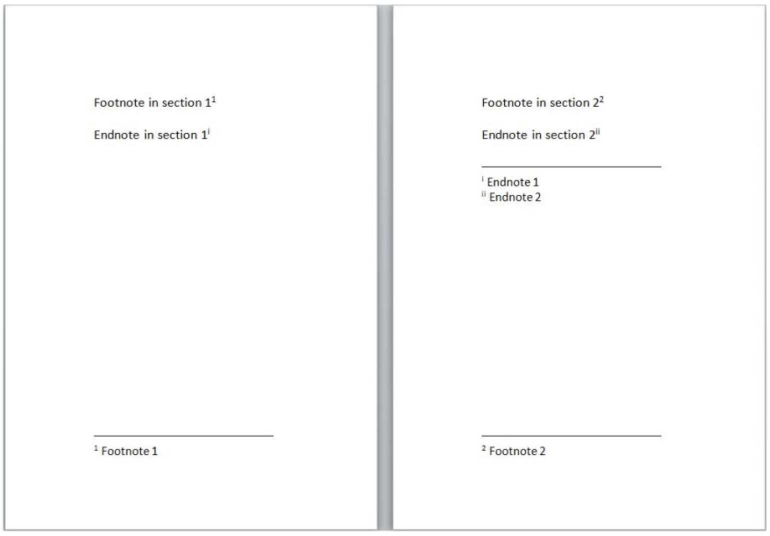
When translating such a document between the ODF and OOXML International Standards, the result must meet structure related requirements in addition to preserving the visual appearance of simple text and paragraphs as shown in Figure 5. Document parts must be consistently translated to enable users to edit hyperlinks, table cells and even complex nested tables after converting the document's format.

Conclusion:

The table *tables* in subclause 6.2.3 shows that the translation of table structures between ODF and OOXML is supported in most cases. Problems appear when using table background patterns (not supported by ODF) as well as sub tables (not supported by OOXML). Another problem is ODF's lack of support for certain layouts, such as the *right to left* layout. Such layout options could be emulated within the options available to ODF, but even so they would still require a complex translation. ODF's lack of support for document themes that are frequently used in OOXML could cause information loss during translation. These differences restrict the translatability of tables between the two International Standards.

5.2.8 Footnotes and endnotes

Textual description:



Footnote in section 1[1]

Endnote in section 1[i]

[1] Footnote 1

Footnote in section 2[2]

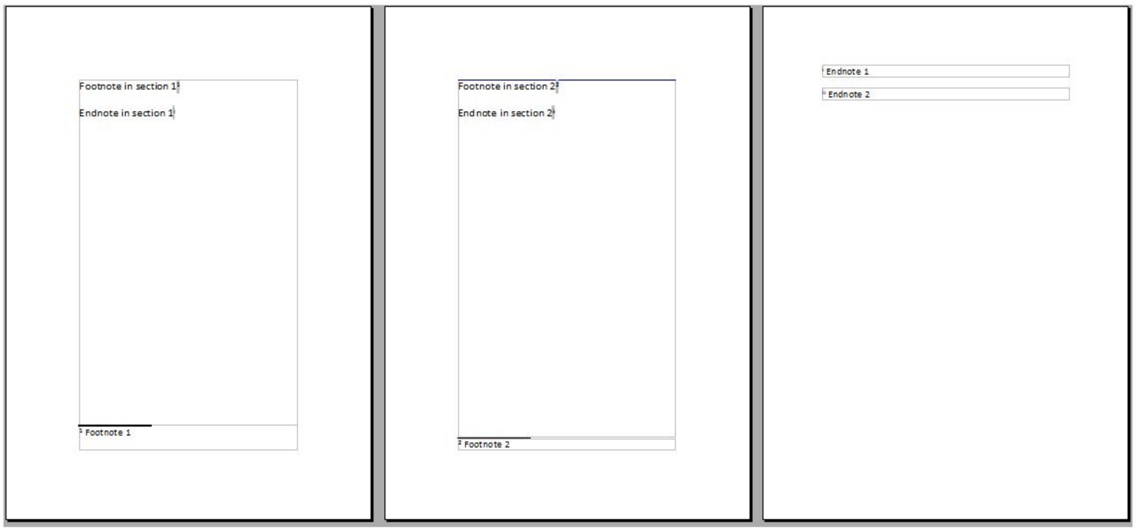
Endnote in section 2[ii]

[i] Endnote 1

[ii] Endnote 2

[2] Footnote 2

Figure 16 — Footnotes and endnotes in OOXML



Footnote in section 1[1]

Endnote in section 1[i]

[1] Footnote 1

Footnote in section 2[2]

Endnote in section 2[ii]

[2] Footnote 2

[i] Endnote 1

[ii] Endnote 2

Figure 17 — Footnotes and endnotes in ODF

John is working on scholarly papers for his standardization forum. He sends two draft papers with footnotes and endnotes to his colleague Smith to check conformance of contents and references. John uses a format A word processor. He saves the two drafts as *paper1_draft* and *paper2_draft*. Smith makes a few corrections on John's papers with his format B word processor and saves them as *paper1_final* and *paper2_final*. Afterwards Smith sends the final versions to the other members of the forum. Some open the files with a format A word processor, others open them with a format B word processor.

Implementation:

Authors use footnotes and endnotes to present references in a document. A *footnote reference mark* in the body of the text is used to indicate that additional information is in a footnote and an *endnote reference mark* indicates that additional information is in a endnote. Both footnotes and endnotes use a numbering system to show readers if they need to look for footnotes at the bottom of the pages or endnotes at the end of the section or at the end of the document.

Use case name: Footnote and endnote Translation type and properties:	
One-trip translation	✓
Round trip translation	
Presentation instructions	✓
Document content	✓
Dynamic content	
Metadata	
Annotations and security	
Document parts	✓
Required features: • Footnote & Endnote o OOXML: Subclause 11.3.4, 17.11.* o ODF: Subclause 5.3.1, 6.3.1, 6.6.5, 14.9.2, 15.2.18, 15.2.20, 15.7.9	

Requirements:

In both sections footnotes should be located at the bottom of the page under the separation lines. Endnotes should be located at the end of the document. The main requirement in Figure 16 and Figure 17 is that endnotes have to be placed at the end of the document.

Conclusion:

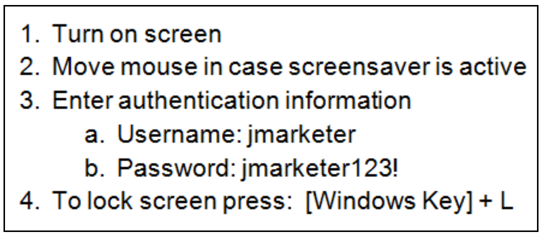
The numbering system used for footnotes and endnotes (Arabic and Roman numerals) translates well between OOXML and ODF. The location of the footnotes is similar in both formats. The thickness and length of the separation lines are a little bit different but also quite similar.

The major difference is the location of endnotes. The author intends to place the endnote at the end of the document. OOXML places the endnotes at the end of the document within the same page. ODF places all endnotes in a new page after the last page of the text. OOXML and ODF also support the location of endnotes at the end of each section.

5.2.9 Itemization and numeration

Textual description:

Besides common table functionality, other important features commonly used in office documents are numerations and lists, which are often used in non-fictional texts like technical documentations as a common means of presenting structured information. John gets a documentation paper from his technical department about how to log on to his new workstation. The document was created using format B. It describes the related tasks in a few steps. John opens the document using format A to reveal the following:



1. Turn on screen
2. Move mouse in case screensaver is active
3. Enter authentication information
   a. Username: jmarketer
   b. Password: jmarketer123!
4. To lock screen press: [Windows Key] + L

Figure 18 — Numbered items

Implementation:

The example shown in Figure 18 contains a simple list of instructions typed in plain text. The instructions are listed using simple numerals and can contain special characters.

Use case name: Itemization and numeration Translation type and properties:	
One-trip translation	✓
Round trip translation	
Presentation instructions	✓
Document content	✓
Dynamic content	✓
Metadata	
Annotations and security	
Document parts	
Required features: • Itemization and Numeration ○ OOXML: Subclause 8.3, 17.9 ○ ODF: Subclause 4.2, 14.9	

Requirements:

During a translation between both International Standards it should be possible to retain the numeration and the structural order of list items.

Conclusion:

Due to the ambiguous wording of the ODF International Standard for numeration, multiple interpretations of certain itemization and numeration properties are possible. Both formats have multiple ways of applying numbering to text segments. Maintaining visual fidelity would probably call for relatively complicated transformation methods between the two International Standards, even if the logical hierarchy of different layers of indices was preserved.

The translation of itemization and numeration properties between the International Standards ODF and OOXML is described in more detail in subclause 6.2.5.

5.2.10 Indices and tables of contents

Textual description:

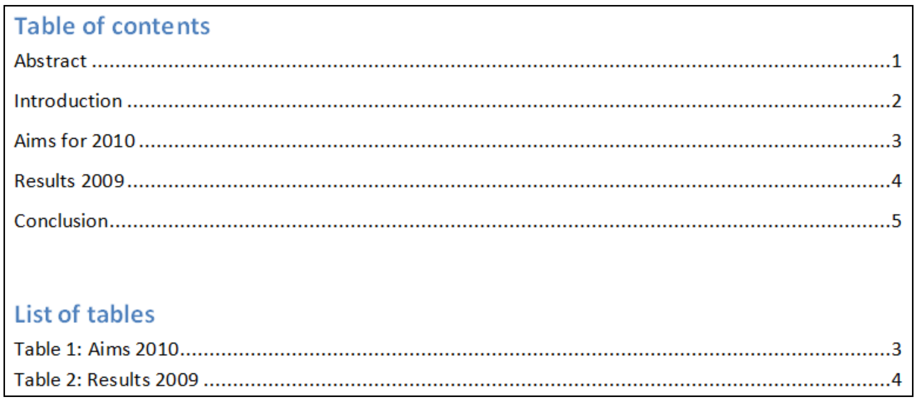


Table of contents

Abstract 1
Introduction 2
Aims for 2010 3
Results 2009 4
Conclusion 5

List of tables

Table 1: Aims 2010 3
Table 2: Results 2009 4

Figure 19 — Auto-generated indices

After opening a document with his format A application, John Marketer likes to cut out different chapters to generate a condensed version of the document to email it to his colleagues. The screenshot in Figure 19 shows an example index consisting of a *Table of contents* and a *List of tables* of a market report John downloaded from the internet. It was originally created using format B.

Implementation:

In addition to continuous text and structural and presentation features, large documents can contain indices and tables of contents, to enhance readability and to make them more human searchable. Indices like the one shown in Figure 19 should display the document's structure based on headings and page numbers as well as figures and tables based on their captions. Indices should be generated automatically by the available word processing application and kept updated as necessary.

Use case name: Indices **Translation type and properties:**	
One-trip translation	✓
Round trip translation	
Presentation instructions	✓
Document content	
Dynamic content	✓
Metadata	
Annotations and security	
Document parts	✓
Required features: • Indices o OOXML: Subclause 17.16 o ODF: Subclause 7.*	

Requirements:

The indices should be adapted automatically; deleted chapters or tables should no longer appear in the index and the page numbers of the remaining parts of the document should be updated. The main requirement in this scenario is that the table of contents and an index from a format A document can be correctly translated into a corresponding table of contents and index in a format B document.

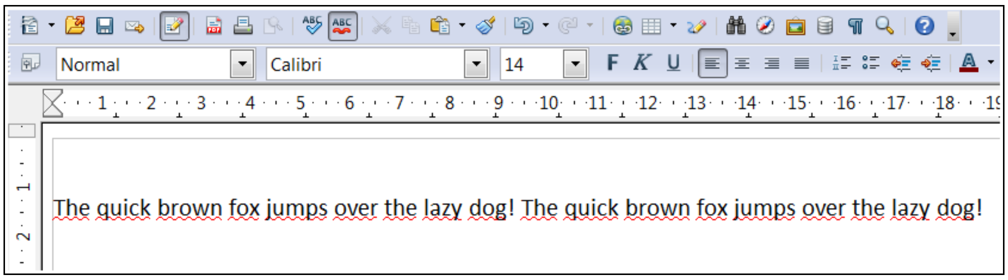
Conclusion:

Although the two document formats differ in their approaches to the generation of tables of contents and indices, they do indeed offer comparable levels of support for this feature. Implementations must take into account the different models, which makes the translation much more complex, especially when documents combine the available models. A more detailed view of index handling is given in subclause 6.2.7. While a table of contents may retain its property to be generable after translation from OOXML to ODF this property may be lost for a list of tables/figures. Additionally a user defined appearance of the table of contents will be lost during a translation process from OOXML to ODF although ODF allows user adjusted appearance of indices.

5.2.11 Metadata and settings

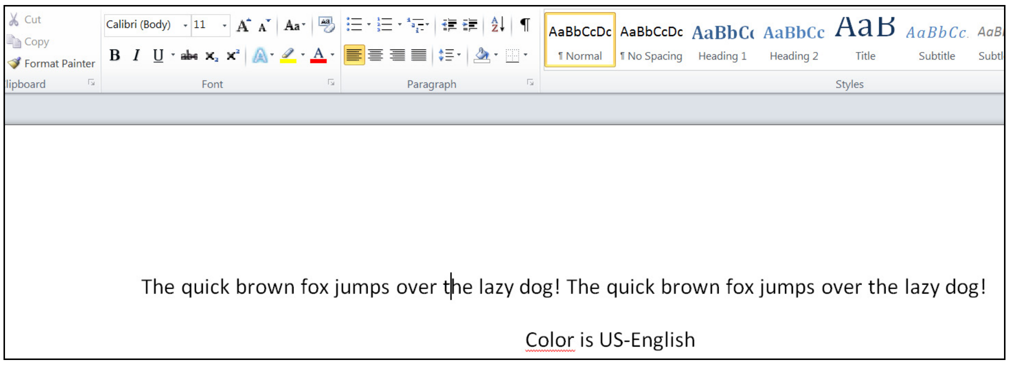
Textual description:

John authors a text document using format A, which is to be edited by his secretary Mary who uses an application supporting format B. The original letter is written using an English vocabulary, punctuation and spell check. When Mary receives the document, her application should immediately recognize which language was used when creating the document.


Normal Calibri 14

The quick brown fox jumps over the lazy dog! The quick brown fox jumps over the lazy dog!

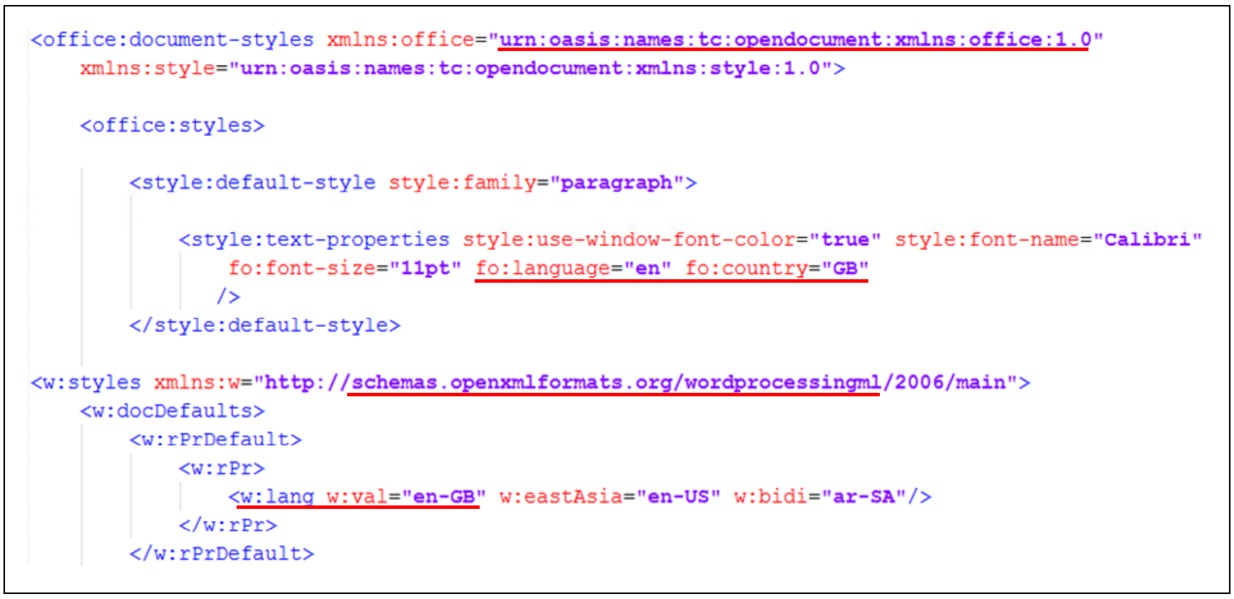
Figure 20 — English text with German settings



The quick brown fox jumps over the lazy dog! The quick brown fox jumps over the lazy dog!

Color is US-English

Figure 21 — English text with English (en-GB) settings


```
<office:document-styles xmlns:office="urn:oasis:names:tc:opendocument:xmlns:office:1.0"
    xmlns:style="urn:oasis:names:tc:opendocument:xmlns:style:1.0">

    <office:styles>

        <style:default-style style:family="paragraph">

            <style:text-properties style:use-window-font-color="true" style:font-name="Calibri"
                fo:font-size="11pt"  fo:language="en"  fo:country="GB"
              />
        </style:default-style>

<w:styles xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
    <w:docDefaults>
        <w:rPrDefault>
            <w:rPr>
                <w:lang w:val="en-GB" w:eastAsia="en-US" w:bidi="ar-SA"/>
            </w:rPr>
        </w:rPrDefault>
```

Figure 22 — Language metadata info in ODF and OOXML

Implementation:

To ensure a correct interpretation and description of word processing documents, certain additional information must be stored as metadata. One example of such metadata is the paragraph style indicating the language used in authoring a document. Grammar and spelling checkers will need this information when working with the translated document.

The document shown in Figure 20 was written in English with an application normally using German as its default language. Thus, the written words are not recognized by the German spelling checker, as shown by the squiggly red lines displayed under each word. In Figure 21, the language settings have been modified, as evidenced by the absence of the red lines denoting misspelling. Excerpts from the documents' metadata files are given in Figure 22, where the position indicating the default language is underlined in red.

Use case name: Metadata and settings **Translation type and properties:**	
One-trip translation	✓
Round trip translation	
Presentation instructions	✓
Document content	
Dynamic content	
Metadata	✓
Annotations and security	
Document parts	
Required features: • Metadata o OOXML: Subclause 17.3 o ODF: Subclause 2.*, 3.1	

Requirements:

A target word processing application must be able to correctly interpret a document's metadata if errors are to be avoided. Translation tools should ensure adequate mapping or meaningful default mapping of the metatags when translating between standards.

Conclusion:

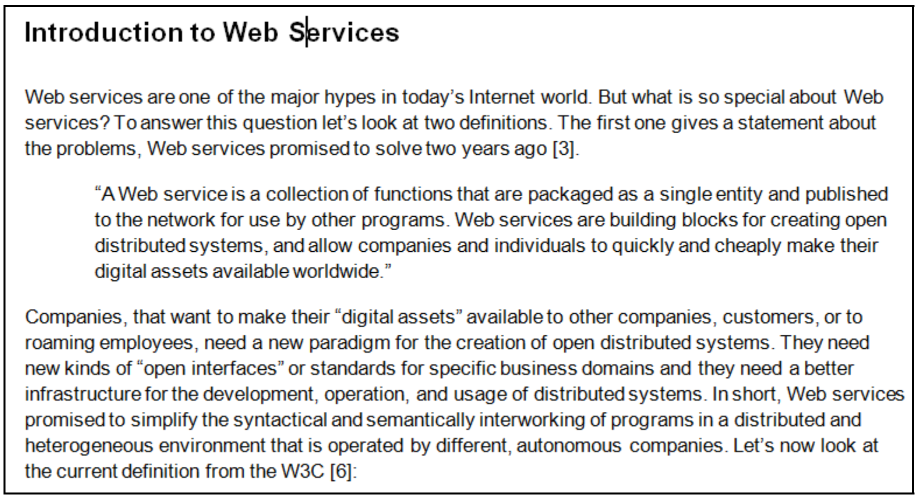
Both International Standards support different types of metadata. In OOXML metadata are stored in the app.xml and core.xml files. ODF stores metadata in meta.xml. Language information is stored in the style.xml files. It can be adequately translated. Information about document settings contains presentation instructions. They are stored in setting.xml files. Several metadata such as *creating application* and *creation date* have to be modified during a translation process.

5.2.12 Change tracking and collaboration support

Textual description:

The following scenario illustrates how collaboration between different authors using different text processors should proceed.

The format A user John Marketer and some of his partners are planning to launch an article in the online magazine *OpenBusinessMag*. John Marketer and some other persons will all contribute to it, authoring in different formats. The first draft of the document will be provided by John himself. Figure 23 illustrates the initial version of the article.



Introduction to Web Services

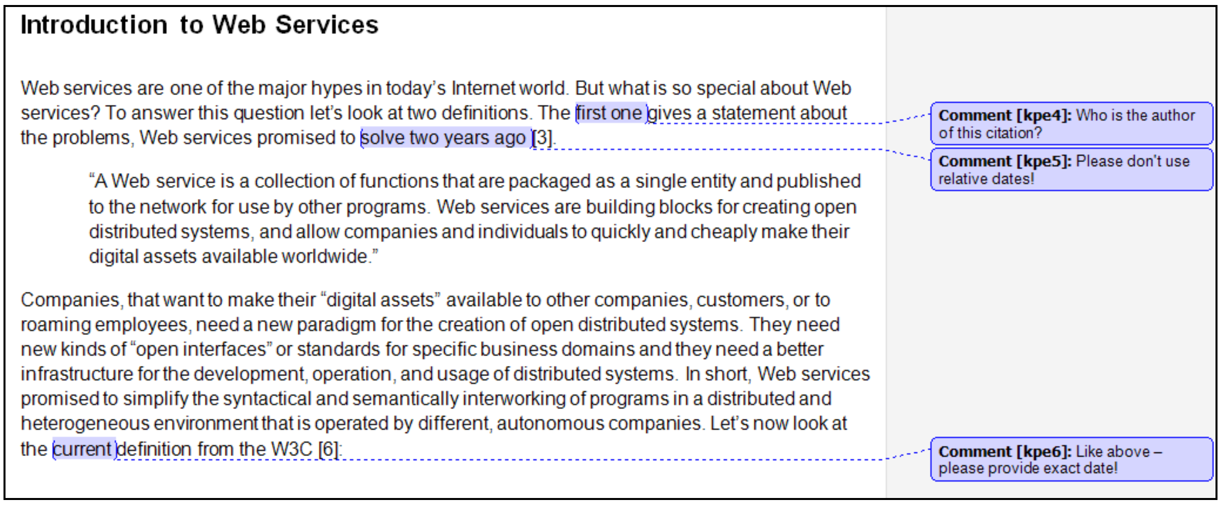
Web services are one of the major hypes in today's Internet world. But what is so special about Web services? To answer this question let's look at two definitions. The first one gives a statement about the problems, Web services promised to solve two years ago [3].

"A Web service is a collection of functions that are packaged as a single entity and published to the network for use by other programs. Web services are building blocks for creating open distributed systems, and allow companies and individuals to quickly and cheaply make their digital assets available worldwide."

Companies, that want to make their "digital assets" available to other companies, customers, or to roaming employees, need a new paradigm for the creation of open distributed systems. They need new kinds of "open interfaces" or standards for specific business domains and they need a better infrastructure for the development, operation, and usage of distributed systems. In short, Web services promised to simplify the syntactical and semantically interworking of programs in a distributed and heterogeneous environment that is operated by different, autonomous companies. Let's now look at the current definition from the W3C [6]:

Figure 23 — Continuous text

John sends this document to his co-author with request for comments. Using the format B commenting and change tracking feature the co-authors reviews the document. Comments are marked with the initials of the user entering the comment, with different colours marking comments made by different users. The change tracking function highlights added, edited, moved or deleted text parts. It shows the obsolete text parts in coloured comment boxes as shown in Figure 24 and Figure 25.



Introduction to Web Services

Web services are one of the major hypes in today's Internet world. But what is so special about Web services? To answer this question let's look at two definitions. The first one gives a statement about the problems, Web services promised to solve two years ago [3].

"A Web service is a collection of functions that are packaged as a single entity and published to the network for use by other programs. Web services are building blocks for creating open distributed systems, and allow companies and individuals to quickly and cheaply make their digital assets available worldwide."

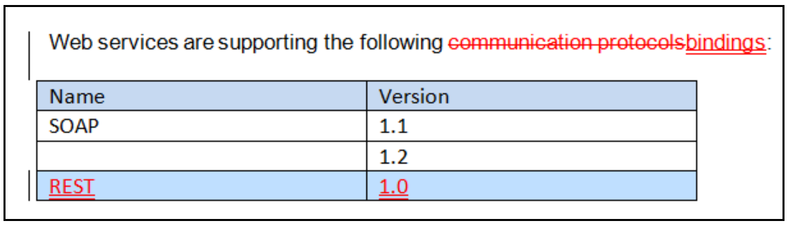
Companies, that want to make their "digital assets" available to other companies, customers, or to roaming employees, need a new paradigm for the creation of open distributed systems. They need new kinds of "open interfaces" or standards for specific business domains and they need a better infrastructure for the development, operation, and usage of distributed systems. In short, Web services promised to simplify the syntactical and semantically interworking of programs in a distributed and heterogeneous environment that is operated by different, autonomous companies. Let's now look at the current definition from the W3C [6]:

Comment [kpe4]: Who is the author of this citation?

Comment [kpe5]: Please don't use relative dates!

Comment [kpe6]: Like above – please provide exact date!

Figure 24 — Continuous text with annotations



Web services are supporting the following ~~communication protocols~~bindings:

Name	Version
SOAP	1.1
	1.2
REST	1.0

Figure 25 — Continuous text and table with tracked changes

After the co-author has sent back the revised version of the article, John can revise the text by accepting or rejecting the comments and proposed changes.

Implementation:

One of the most important features for editing large documents with multiple authors is called *collaborative functions* which include user-specific comments and tracking of changes. These functions enable collaborative workflows, allowing document editing and reviewing by multiple participants. The information required for such workflows, including user data, notes or tracked changes, is embedded within the document itself. Proper adoption of such meta-information plays an important role in the collaborative authoring processes.

This type of application, with its workflow support, substantially alleviates the difficulties of revising documents with multiple authors. The foundation for this document lifecycle is the proper conversion of meta-information from one standard to the other, to correctly retain comments and proposed revision information.

Use case name: Change tracking and collaboration functions **Translation type and properties:**	
One-trip translation	✓
Round trip translation	✓
Presentation instructions	
Document content	✓
Dynamic content	
Metadata	✓
Annotations and security	✓
Document parts	
Required features: • Change tracking and document revision o OOXML: Subclause 17.* o ODF: Subclause 3.1, 4.6, 8.3, 12.3	

Requirements:

The references to the paragraphs, words and characters made by John's co-authors using OOXML should be accurately translated in Johns ODF supporting application. The information used for change tracking should also reflect the exact editing (such as highlighted changes) in such a way that it can be accurately reproduced, since it is vital that all proposed changes be rendered properly.

Conclusion:

Both document formats offer support for revision handling, although there are significant differences in the scope of their revision-handling functionality and their approach to the underlying technical details. For example, ODF does not allow for tracking changes made within tables, while OOXML tracks changes to the content of tables as well as changes to the structure of tables themselves. While ODF only records the fact that a text attribute, such as the used text font, has changed, OOXML records the full history of changes made, ensuring the ability to reconstruct the previous text version. Another difference is in the understanding of text comments. While OOXML allows adding comments to arbitrary text ranges, this feature is not supported by ODF. However similar functionality may be provided by inserting notes associated with a point within the text rather than a range. The table *change tracking and document revision* in subclause 6.2.8 details how collaborative functions could be used when translating between the different document formats.

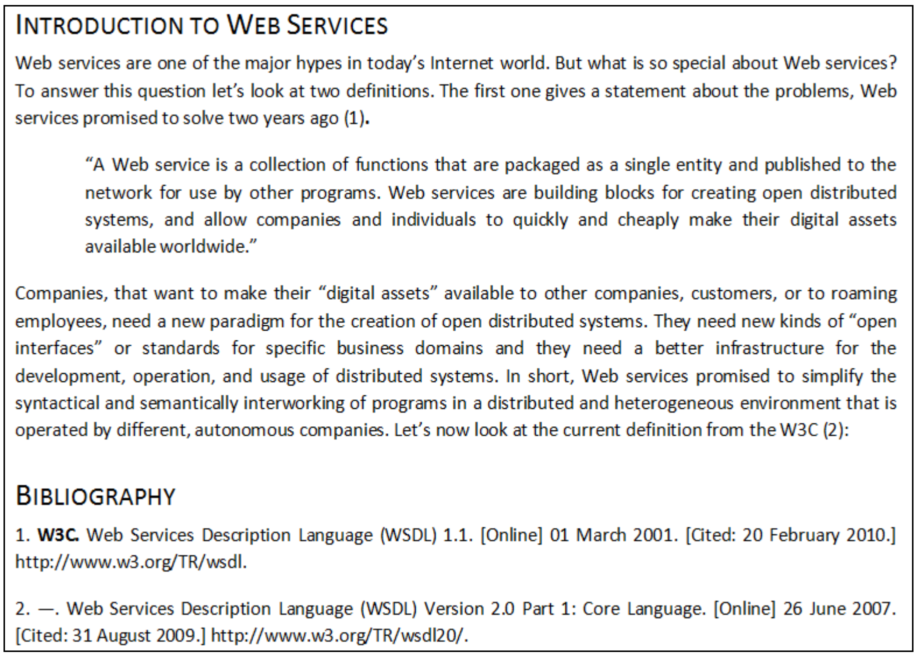
ODF (ISO/IEC 26300) does not provide enough information for a meaningful analysis of the support for revision-handling and change tracking. For this reason OASIS has introduced the *Advanced Document Collaboration Subcommittee* late 2010 that will define corresponding features in upcoming versions of ODF.

5.2.13 Bibliographies and optional document parts

Textual description:

The following scenario illustrates the management of bibliographies and optional document parts.

Format A user John Marketer and his partners working on an article decide to add citations and a bibliography to the article. John Marketer copies citations from his private bibliography database to the document's internal bibliography sources and sends it to his co-authors who are using format B. The co-authors add their references to the document's bibliography sources and update the bibliography. One author adds an optional section to the paper that should only be used in the paper's long version.



INTRODUCTION TO WEB SERVICES

Web services are one of the major hypes in today's Internet world. But what is so special about Web services? To answer this question let's look at two definitions. The first one gives a statement about the problems, Web services promised to solve two years ago (1).

> "A Web service is a collection of functions that are packaged as a single entity and published to the network for use by other programs. Web services are building blocks for creating open distributed systems, and allow companies and individuals to quickly and cheaply make their digital assets available worldwide."

Companies, that want to make their "digital assets" available to other companies, customers, or to roaming employees, need a new paradigm for the creation of open distributed systems. They need new kinds of "open interfaces" or standards for specific business domains and they need a better infrastructure for the development, operation, and usage of distributed systems. In short, Web services promised to simplify the syntactical and semantically interworking of programs in a distributed and heterogeneous environment that is operated by different, autonomous companies. Let's now look at the current definition from the W3C (2):

BIBLIOGRAPHY

1. **W3C.** Web Services Description Language (WSDL) 1.1. [Online] 01 March 2001. [Cited: 20 February 2010.] http://www.w3.org/TR/wsdl.

2. —. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. [Online] 26 June 2007. [Cited: 31 August 2009.] http://www.w3.org/TR/wsdl20/.

Figure 26 — Citations and bibliographies

Implementation:

In most scientific papers the authors have to provide a bibliography with references to cited articles, books, or Web sites. In case an author writes several papers it is desirable to have a bibliography database storing the information about all important papers and to be able to copy the information about the cited papers to the document. Additionally it is advantageous to store the same information attributes common to all cited papers.

Word processing applications should be able to generate a bibliography from the cited papers and update this bibliography if some new citations have been added.

With optional text document parts it is important that different applications treat these parts in the same way and that text remains optional after a mapping between the International Standards.

Use case name: Bibliographies **Translation type and properties:**	
One-trip translation	✓
Round trip translation	✓
Presentation instructions	✓
Document content	✓
Dynamic content	
Metadata	
Annotations and security	
Document parts	✓
Required features: • Bibliography o OOXML: Subclause 11.3.8 o ODF: Subclause 7.9, 7.1.4 • Glossary document/hidden sections o OOXML: Subclause 17.12 o ODF: Subclause 4.4.1	

Requirements:

A minimum requirement is that citations remain citations and bibliographies remain bibliographies. Thus it should be possible to add a new and to change or remove an existing citation and regenerate the bibliography. It is desirable that the type of information stored in the document preserves its semantics, thus an author should remain an author and a title should remain a title. The document internal “databases” should be mapped as well as possible.

It is not necessary to translate external bibliography databases between both formats because these databases depend on the word processing application and not on the document standard.

Conclusion:

Both the ODF and OOXML formats support bibliographies and optional document parts. Unfortunately both International Standards use different concepts which make it impossible to define generic translations. However some word processing applications may be able to provide such mappings in a restricted context.

5.2.14 Sub documents and books

Textual description:

Format A user John Marketer decides to split the article on Web services into separate sections to be assigned to the responsible authors. John wants to compose the article from the sub documents created by his co-authors. To guarantee a common "layout" of the final document he distributes document templates to all of them. Each co-worker provides a single chapter that is written in its own document in file format A. All sub documents are stored independently in the file system.

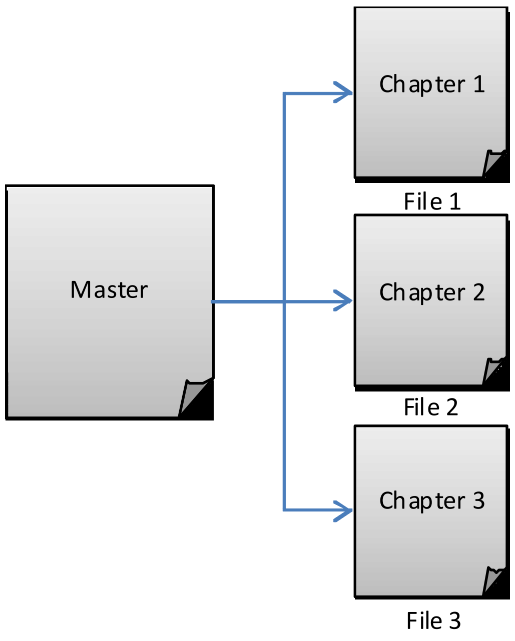


Figure 27 — Master document referencing its sub documents

The full document is combined using a master document, which allows the sub documents to be merged into a single document. The styles are automatically adjusted to the master document's layout properties and settings.

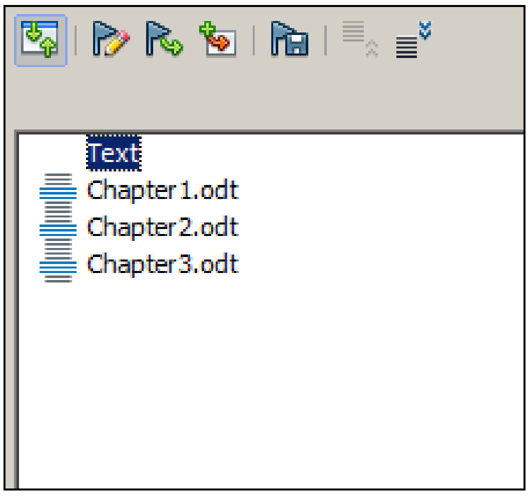


Figure 28 — Linked text sections in an ODF application

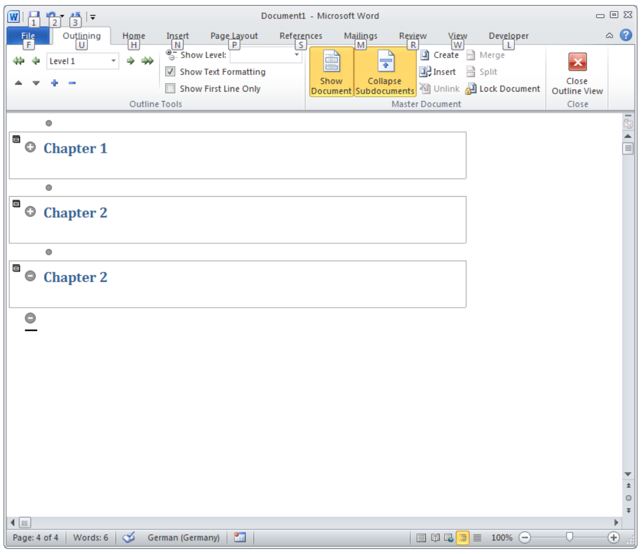


Figure 29 — Sub documents in an OOXML application

Subsequently the complete document will be send to an external proofreader who uses file format B. The revised text will then be returned to John Marketers company via email.

Implementation:

Both ODF and OOXML allow large documents to be broken into a number of smaller ones that can be distributed and edited independently. Both International Standards introduce different terms for the central document referencing sub documents. OOXML uses the name *master document* whereas ODF defines the term *global document* in its specification. The linked entities are called *sub documents* in OOXML and *linked text sections* in ODF.

Since all chapters in the example above are stored in separate documents, they can be modified independently by all co-workers. A master document has to be created and references to the appropriate sub documents. The style properties of the main documents can automatically be applied to all chapters that were included.

Use case name: Sub documents and books **Translation type and properties:**	
One-trip translation	✓
Round trip translation	
Presentation instructions	✓
Document content	✓
Dynamic content	✓
Metadata	
Annotations and security	
Document parts	✓
Required features: • Linked text s ODF: Subclause 2.3.1, 4.4 • Sub documents and books OOXML: Subclause 11.6	

Requirements:

The full document created by John and his co-workers has to be organized in equivalent document parts (master and sub documents) when it is opened on ODF or OOXML platforms. After switching from one format to another, sub documents still have to be readable and writeable as standalone documents. The representation of the compound document has to be unified automatically. The tables of contents, bibliographies and other indexes have to be updated without manually editing the master document.

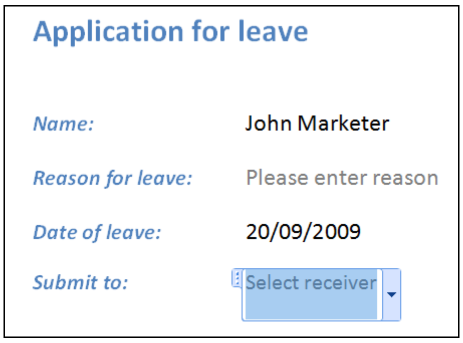
Conclusion:

Both International Standards provide similar features allowing the creation of master or global documents that combine independent document entities using references. In both cases the formatting can automatically be unified by the presentation instructions of the master. For this reason the translatability between both International Standards initially seems to be quite high. The translatability in a particular case is tightly coupled to the translatability of the features that were used. Since generic mapping for the commonly used citations and bibliographies seems to be impossible, many translations of compound documents will fail as well.

5.2.15 Forms

Textual description:

John Marketer's company has been optimizing its internal workflow processes for years. Paper based workflows have become so rare that almost all corporate forms are digitalized. At John's company, these different formats are gathered on an internal website accessed by most people at the company. The aforementioned workflows sometimes involve the transfer of forms between computers running different word processing applications. This use case illustrates some simple features commonly used in forms:



Application for leave

Name:	John Marketer
Reason for leave:	Please enter reason
Date of leave:	20/09/2009
Submit to:	Select receiver

Figure 30 — OOXML form

Implementation:

Modern word processing documents are tightly integrated into electronic workflows. They serve as static output formats for reports or certificates and, with their extended form functionalities, they can also be integrated as dynamic, data driven front-ends.

The form in Figure 30 shows different textboxes. The form is filled out by an applicant and submitted to a workflow system which integrates the form's data directly into applications which further process the data.

Use case name: Forms **Translation type and properties:**	
One-trip translation	✓
Round trip translation	
Presentation instructions	✓
Document content	✓
Dynamic content	✓
Metadata	
Annotations and security	
Document parts	✓
Required features: • Forms o OOXML: Subclause 17.16 o ODF: Clause 11	

Requirements:

To pass this form between applications based on ODF and applications based on OOXML, behaviour and content type of the form fields needs to be preserved. Translating the form from one format to the other for processing or viewing should not result in data corruption.

Conclusion:

Both International Standards define forms based on text boxes, check boxes, and drop-down lists. Nevertheless translation of forms between ODF and OOXML is likely to prove problematic, since the two technologies diverge strongly in many aspects of form handling. While ODF is directed to the open standard XForms (Version 1.0 from 2004), OOXML uses simply *form fields* that support insertion of data through *form controls*. Although both concepts work with user-defined XML structures, which help to export structured data from text processing documents, the translatability potential of forms between the two International Standards is merely low to average.

5.2.16 Vector graphics

Textual description:

In the following scenario, a logo stored in the common WMF (Windows Metafile) format is embedded into a format A document by Mary and emailed to John who is using format B. John evaluates and approves the logo. Ideally, the logo should be displayed by the format B application in a similar way as it has been displayed by the format A application.

Figure 31 — Embedded vector graphic

Implementation:

Vector graphics are essential elements of modern document content and presentation, especially for printing purposes. They are flexible in use, editable to a certain extent, and scalable to nearly any size without any need for special expertise in graphics.

Use case name: Vector graphics **Translation type and properties:**	
One-trip translation	✓
Round trip translation	
Presentation instructions	✓
Document content	✓
Dynamic content	
Metadata	
Annotations and security	
Document parts	✓
Required features: • Vector graphics o OOXML: Subclause 8.6 o ODF: Subclause 9.3	

Requirements:

Vector graphics embedded in documents should maintain their appearance, scaling, and quality when translating documents between the two formats. There should be no discernible difference between graphics presentation under ODF and OOXML. This applies equally to graphics properties such as pixel size, colour encoding etc.

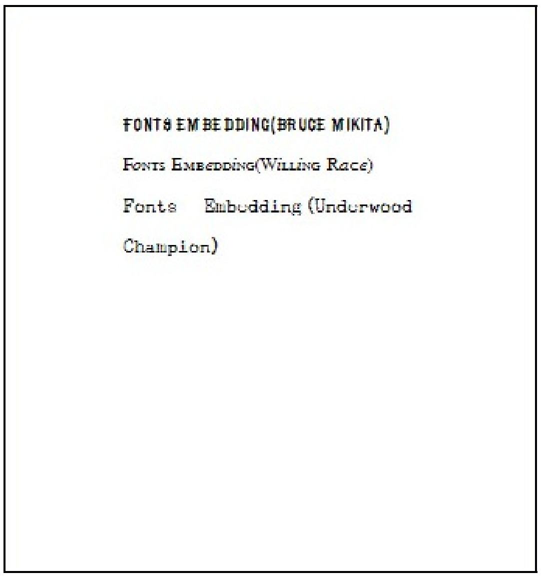
Conclusion:

Unlike bitmap graphics which are represented simply through a MIME type and are virtually platform-independent, vector graphics pose bigger translation challenges. OOXML essentially defines its own DrawingML format to which the now obsolete VML (Vector Markup Language) was a precursor. ODF recommends the use of SVG (Scalable Vector Graphics) which is not as rich in features and functionality as DrawingML. The ODF International Standard merges the SVG namespace with ODF's namespaces, so the SVG objects in ODF documents can't be handled by generic SVG tools and technologies. These types of disparities could pose potential interoperability problems between the two International Standards in the area of vector graphics.

5.2.17 Font embedding and paper size

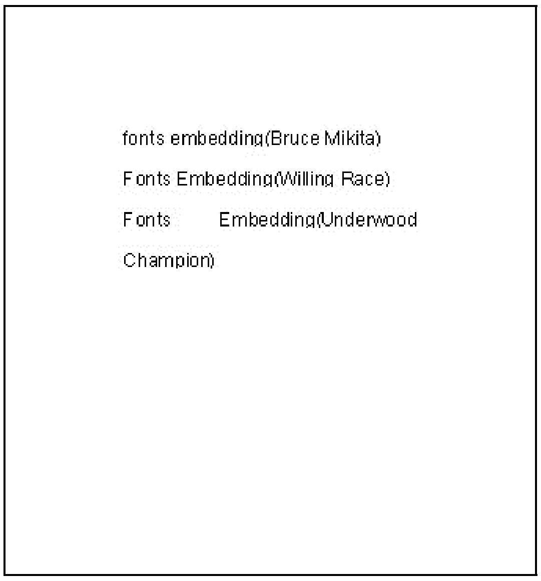
Textual description:

Mary is a graphic designer. Her computer is equipped with a large font library. One day she sends a design draft to John using format A. However John’s computer can only process documents in format B and has limited fonts installed. Mary thus embeds all the fonts used in the document, and sets the paper size for printing to 14cm width and 15cm height. John opens the document and noticed that the paper size is almost correct, however, some text cannot be displayed using the expected fonts although he could still read the text.



FONTS EMBEDDING(BRUCE MIKITA)
Fonts Embedding(Willing Race)
Fonts Embudding (Underwood Champion)

Figure 32 — OOXML document with embedded fonts and 14 x 15cm paper size



fonts embedding(Bruce Mikita)
Fonts Embedding(Willing Race)
Fonts Embedding(Underwood Champion)

Figure 33 — ODF document using similar fonts and 14 x 15cm paper size

Implementation:

When a computer does not have enough fonts installed, font embedding will help the user to display the document in the same way as the original. It is implemented by adding the font data into the document, thus the font goes with the document to ensure the proper font is available where required. Document size will increase to some extent after font embedding. Some protected fonts are unable to embed without authorization.

For paper size for printing, the software usually has a set of predefined names for frequently used paper sizes, e.g. A4, B5, etc. Additionally, it allows the user to define special paper sizes which are not listed in the predefined list. The paper size setting affects the layout of the document typesetting.

Use case name: Font embedding and paper size **Translation type and properties:**	
One-trip translation	✓
Round trip translation	
Presentation instructions	✓
Document content	✓
Dynamic content	
Metadata	
Annotations and security	
Document parts	✓
Required features: • Font embedding o OOXML: Subclause 17.8 o ODF: - • Paper size o OOXML: Subclause 17.6 o ODF: Subclause 15.2	

Requirements:

Ideally, both document formats would support font embedding. However, in case font embedding is not supported in a target format, the translator should capture enough information from the source format to allow the software find similar fonts to display the document using certain algorithms to ensure the text is still readable.

Ideally, it is required that both document formats share the same definition for frequently used paper sizes. If a type of common paper size, e.g. *letter*, is not defined in the source format, the right width and height of the paper corresponding to *letter* should be able to be specified in the source, and when it is translated into the target format, the size will still be mapped to *letter*. In any case, the paper size should be consistent in the source and target documents.

Conclusion:

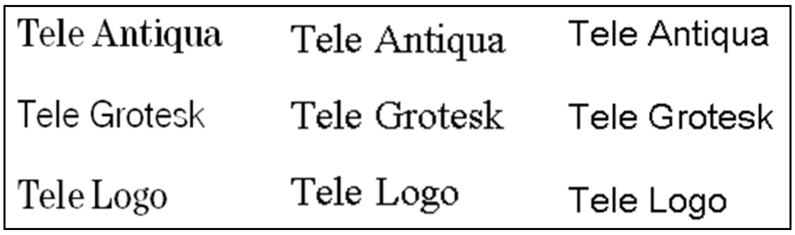
OOXML supports embedded fonts whereas ODF does not support this feature. Therefore there is no way to embed a font in an ODF document. When an OOXML document with embedded fonts is translated into the ODF format, we can expect that the text will still be readable, however, the fonts may not be exactly matching.

On the other hand, paper size information can be kept consistent in the source and target documents, thus the documents will be printed out using the same paper size.

5.2.18 Font metrics and font substitution

Textual description:

Mary writes a letter using format A and emails it to John using format B. The letter is typed using proprietary fonts. John opens the letter, reviews it, prints it out, signs it and sends it to the mail centre for postage.



Tele Antiqua	Tele Antiqua	Tele Antiqua
Tele Grotesk	Tele Grotesk	Tele Grotesk
Tele Logo	Tele Logo	Tele Logo

Figure 34 — Proprietary font and substituted fonts in OOXML and ODF tools

Implementation:

New fonts such as Microsoft's C-fonts (Calibri, Cambria, Candara, etc.) or company specific fonts like the tele-* fonts of German Telekom are not available in every environment. To guarantee high visual fidelity the metrics of the available fonts are used to identify an alternative font in case the primary font is not available.

Use case name: Font metrics **Translation type and properties:**	
One-trip translation	✓
Round trip translation	
Presentation instructions	✓
Document content	
Dynamic content	
Metadata	
Annotations and security	
Document parts	
Required features: • Font metrics o OOXML: Subclause 17.81, 7.8.2, 17.8.3.10 o ODF: Subclause 2.6, 14.6	

Requirements:

The document John opens should look similar in terms of line and page breaks as in Mary's environment.

Conclusion:

Font substitution is a process by which an application, when it cannot locate a specific font, attempts to locate the closest possible match as a suitable alternative to the intended appearance of the font. However, based on the availability of a font an application might not be able to locate the specified font. The exact algorithm which is used for font substitution is highly dependent on the characteristics which are most desirable when performing the substitution. Typical criteria are similar appearance of each glyph to maximize visual familiarity or similar physical characteristics to minimize changes in line height and breaking. Both document formats consider font substitution as being implementation dependent.

OOXML (WML) uses *font properties* such as name, family, metrics, the Panose-1 typeface classification number, and code pages and Unicode sub-ranges referring to ISO/IEC 14496-22:2007 for each font used in the document. OOXML recommends that applications looking for the closest match considering specific properties, for example the Panose-1 classification number for the current font using the mechanism defined in §4.2.7.17 of ISO/IEC 14496-22:2007. However, applications are free to apply different strategies.

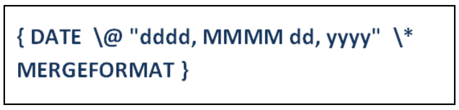
A document in ODF may contain *font face declarations*. A font face declaration provides information (font descriptors) about the fonts used in the document, so that these fonts or fonts that are very close to these fonts may be located on other systems. Font face declarations directly correspond to the @font-face font description of the Cascading Style Sheet specification[6] and the <font-face> element of Scalable Vector Graphics specification[7]. Conforming applications should implement the CSS2 font matching algorithm but they may also implement variants of it. They are especially allowed to implement a font matching based only on the font face declarations, that is, a font matching that is not applied to every character independently but only once for each font face declaration.

Because both formats refer to different standards for the definition of font properties and matching algorithms, translatability between the International Standards is limited. The visual appearance of documents may differ because of the implementation dependency of font substitution.

5.2.19 Document fields

Textual description:

In the first week of every month John's company invites its partners to send over representatives for a business luncheon and business update meeting. For this purpose Johns office sends out hand signed letters every month. To automate this recurring task, Mary, using format A has created a letter template with document fields that automate tasks such as including addressee addresses for mass mailings, portions of text or the current date. When John using format B reopens one of these template-generated letters for review, amendment and printing, certain fields are auto-completed, which saves him both time and trouble.



```
{ DATE  \@ "dddd, MMMM dd, yyyy"  \*
MERGEFORMAT }
```

Figure 35 — Field function displaying the current date

6 CSS2: Cascading Style Sheets, level 2; http://www.w3.org/TR/1998/REC-CSS2-19980512, W3C, 1998

7 SVG: Scalable Vector Graphics 1.1, http://www.w3.org/TR/2003/REC-SVG11-20030114/, W3C, 2003

Implementation:

The concept of document fields was introduced to provide text documents with dynamic content. Fields have become one of the most basic tools in preparing document templates. Fields automatically update to include changing data in the document. Combining fields with *auto text* creates a powerful documentation toolbox.

Use case name: Generic fields **Translation type and properties:**	
One-trip translation	✓
Round trip translation	
Presentation instructions	✓
Document content	
Dynamic content	✓
Metadata	
Annotations and security	
Document parts	
Required features: • Generic fields o OOXML: Subclause 17.16 o ODF: Subclause 6, 11.3	

Requirements:

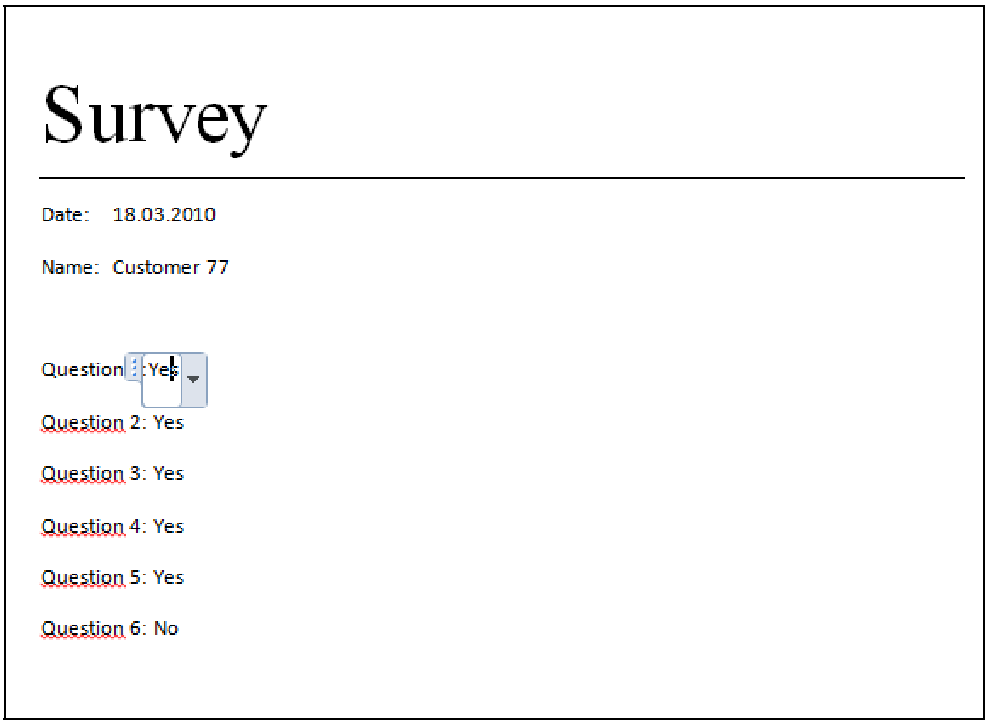
The document created by Mary should contain the same document fields when it is opened by John, and function the same way on both ODF and OOXML platforms. The current system date, for example, should be recognized by the application which opens the document and should be displayed correctly in the automatically updated date fields. It is important that applications correctly interpret all formats and conventions.

Conclusion:

Document fields such as date, time, or page numbers can be translated between both International Standards. Unlike ODF, OOXML allows text fields to contain content that can be associated to user-generated XML schema. This functionality is used by third-party applications to extend the document's functionality, i.e. by dynamically inserting (meta-)data into a document, or by extracting data in order to perform calculations, please refer to subclause 5.2.20 about the inclusion of user defined XML.

5.2.20 Inclusion of user defined XML

Textual description:


Survey

Date: 18.03.2010

Name: Customer 77

Question 1: Yes

Question 2: Yes

Question 3: Yes

Question 4: Yes

Question 5: Yes

Question 6: No

Figure 36 — User defined XML

John Marketer’s company wants to undertake a customer satisfaction survey. John is therefore going to create a digital form using his word processing application. This document has to be exchanged via email. The data contained in the completed surveys has to be exported to XML. Since John’s company has a large number of customers, he wants to reduce the effort of manual post-processing or transcription as far as possible. Hence he is using a format A feature that allows him to bind a control of his form to an XML attribute or element.

This following XML document is an example of a given XML schema. It looks like:

```
<survey>
  <date>2010-03-18T00:00:00</date>
  <name>Customer 77</name>
  <question1>Yes</question1>
  <question2>Yes</question2>
  <question3>Yes</question3>
  <question4>Yes</question4>
  <question5>Yes</question5>
  <question6>No</question6>
</survey>
```

Some participants of the survey use format A, as John Marketer does, others use format B.

Implementation:

OOXML offers the possibility to bind so called *content controls* to *custom XML parts*. This can be done using third party tools (e.g. Content Control Toolkit), scripting or manually editing the OOXML package. Whenever an OOXML form with a binding to a custom XML part is changed, the relating XML data within the package will be changed as well. In the other direction modified XML data will lead to modified content of the form.

ODF uses *XForms*[8] to map the content of document parts such as *text fields* or *check boxes* to XML elements and attributes. A filled out form can be submitted – similar to an html form – to a host or can be directly written to a file.

Use case name: User defined XML Translation type and properties:	
One-trip translation	✓
Round trip translation	✓
Presentation instructions	
Document content	✓
Dynamic content	
Metadata	(✓)[9]
Annotations and security	
Document parts	✓
Required features: • Form and control attributes o ODF: Subclause 11.2, 11.4.3 • Custom XML o OOXML: Subclause 18.16, 22.5, 23, Annex F	

Requirements:

The document created by John has to contain the equivalent controls when it is opened by a format B user, and has to be bound to exactly the same XML elements and attributes using ODF and OOXML applications. The bindings have to persist after converting in both directions from OOXML to ODF and from ODF to OOXML.

Conclusion:

Both International Standards define similar controls or components to create digital forms. Nonetheless, the technologies differ in many aspects of form handling. ODF 1.0 (XForms 1.0) and OOXML (custom XML) enable the user or developer to bind the values of a form to user defined XML elements or XML attributes. But both variants differ especially in the way the XML data is stored. In the case of custom XML, the data is part of the document package itself. XForms bind the XML data to a file that can be submitted or written to the file system.

8 ISO/IEC 26300 refers to the 2004 version of XForms 1.0.

9 The ODF 1.2 Committee Specification provides a solution utilizing metadata concepts.

Currently the concepts of XForms (ODF) and the combination of content controls and Custom XML (OOXML) vary strongly in their underlying concepts. Therefore the translatability potential of both solutions can be classified as low.

The upcoming version ODF 1.2 introduces an RDF metadata feature which offers a solution for metadata annotation whose intent and purpose are comparable to the usage of custom XML in OOXML.

After losing a patent suit in 2009, Microsoft has been banned from selling copies of Microsoft Word containing the custom XML technology. Therefore Word 2007 added features allowing content controls to be mapped to XML data stored in a DOCX file. Hence content controls and XML data stored within DOCX or DOCM files should not be affected by this modification.

5.2.21 Mathematical formulas

Textual description:

John uses his format A application to write an article he intends to publish in a journal. The article contains several embedded formulas, which require special formatting to appear in a certain way so as to properly resemble formulas. He emails the article to Mary, his secretary, to look over and correct. Mary views and modifies the document using her format B application.

$$\boxed{f(x) = \int_0^x g(t\cancel{x})\,dt\cancel{x}}$$

Figure 37 — Embedded formula

Implementation:

In ODF formulas are described using the W3C recommendation MathML and anchored as part of drawing elements within or between text paragraphs. With the additional semantic content definition (in the form of semantic tags and annotations) provided by MathML, equations could also be communicated in different ways. MathML encodes the notational structure of an expression in a sufficiently abstract way to facilitate rendering to various media. Thus, the same presentation markup can be rendered with relative ease on screen in either wide and narrow windows, in ASCII or graphics, in print, or it can be enunciated in a sensible way when spoken.

Formulas in OOXML are described in the shared Office Math Markup Language (OMML) language. These formulas are embedded in OOXML documents. They support features such as revision markings, images and regular styles and formatting found in regular WordprocessingML. OMML can be transformed into MathML via XSLT.

Use case name: Formulas **Translation type and properties:**	
One-trip translation	✓
Round trip translation	
Presentation instructions	✓
Document content	✓
Dynamic content	
Metadata	
Annotations and security	✓
Document parts	
Required features: • Formulas o OOXML: Subclause 8.6; 17.5 o ODF: Subclause 9.3	

Requirements:

The formula John inserts in the document should be displayed on Mary's system in a form equivalent to that created by John thus all elements of the formula such as operators and operands must be identifiable and modifiable. Formula elements may not be omitted, swapped or placed in the wrong position.

Conclusion:

Mathematical content such as formulas is represented via MathML in ODF, even though ODF does not import or reference the MathML schema definition. For this reason it cannot be guaranteed that ODF documents containing equations are always schema compliant. OOXML implements the shared markup language OMML for handling mathematical formulas. In OOXML shared part types can refer to both MathML and OMML even though OOXML uses only OMML as its native format for formulas. Because OOXML is able to understand MathML the translatability between both International Standards utilizing XSLT transformations is quite high. Nevertheless the translatability depends very much on the implementation of the translator. In many situations the result of a translation process looks like an equation but it is simply a character string or a graphic and cannot be further edited like an equation.

Change tracking is not possible in MathML.

5.3 Spreadsheet documents

5.3.1 Empty spreadsheet document

Textual description:

When a new document is created either in ODF format or in OOXML format, the user initially receives an empty document, even though it may look differently in the application user interface. When the document is saved without any further editing, a document is generated without user content. However, it does contain some metadata as well as some initial settings, e.g. default sheets, style information and presentation instructions. Some extra information which could be specified depends on the applications, for example, the initial number of sheets, the number of rows and columns in each sheet, the settings of sheets and views, number styles and currency styles, etc. This initial content should be preserved as far as possible during the translation process.

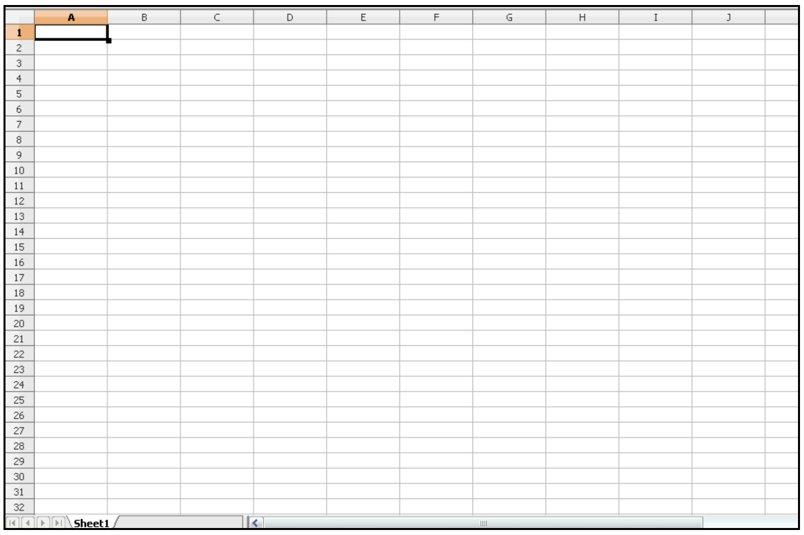


Figure 38 — Empty spreadsheet document

Implementation:

The term *empty document* is not defined in both International Standards. Some default information like settings, styles, presentation instructions and metadata are defined by the application. For example, an empty document has one or more default worksheets and a default definition of sheet or page layout without user-defined content. Generally, the application supports a presentation view and one or more sheets without user-defined content.

Use case name: Empty presentation document Translation type and properties:	
One-trip translation	✓
Round trip translation	✓
Presentation instructions	✓
Document content	
Dynamic content	
Metadata	✓
Annotations and security	
Document parts	
Required features: • OOXML: Subclause 8.4; 12, 18 • ODF: Subclause 2.2, 2.3.4; 2.4; 3; 8.5; 14; 15	

Requirements:

When a document is translated from one format into another, all the default styles of cell, row, column, and table, as well as page layout and master page styles should remain unchanged in the target format in order to facilitate further editing. Furthermore, default settings, presentation instructions and metadata should be preserved if possible: for example, the initial number of sheets, the number of rows and columns in each sheet, settings of sheets and views, number styles and currency styles. It is not expected that both International Standards will necessary use similar defaults for metadata.

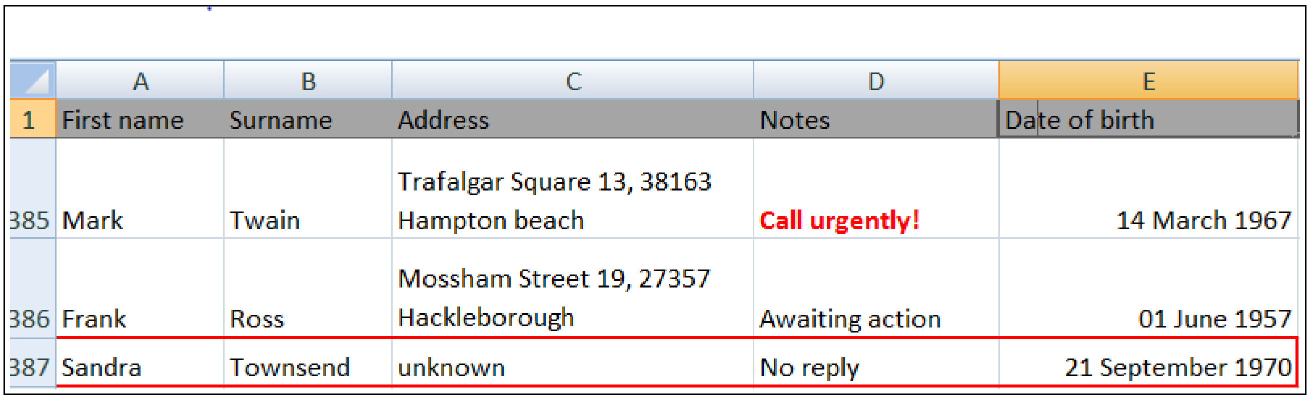
Conclusion:

Neither ODF nor OOXML precisely define the term *empty document*. Thus the content of an empty document depends more on the creating application than on the International Standard. When an empty document defined in format A is opened in format B, default settings, styles, presentation instructions as well as metadata can be preserved. However, the initial view of the empty document may be slightly different, depending on the rendering engine. Metadata can be translated accordingly, even though some information like the application creating the document may be modified.

5.3.2 Listing and structural features

Textual description:

John Marketer makes use of a spreadsheet document to store contact information of his personal clients using a format A application on his private laptop. The table has 5 columns and about 400 entries with names, addresses and birthdays. The top row contains the title of the columns containing *first name, surname, address*, *notes* and *date of birth*. To facilitate navigation, the top row is fixed, and will not move while scrolling down the rows. The screenshot shows an excerpt from the spreadsheet. John emails this document to his secretary Mary to update the customer database manually. Mary is using a format B application on her workstation. After she has finished the work she returns the document to John.



	A	B	C	D	E
1	First name	Surname	Address	Notes	Date of birth
385	Mark	Twain	Trafalgar Square 13, 38163 Hampton beach	Call urgently!	14 March 1967
386	Frank	Ross	Mossham Street 19, 27357 Hackleborough	Awaiting action	01 June 1957
387	Sandra	Townsend	unknown	No reply	21 September 1970

Figure 39 — Address list in a sheet

Implementation:

One of the main applications for spreadsheets is the listing and structuring of large amounts of data in sortable tables. Presentation instructions can define the frames, shading and colours used for highlighting and structuring certain parts of the spreadsheet. This use case illustrates the most important functionalities used in spreadsheets. The graphical characteristics of this sheet include its fixed top row, the grey shading of the top row, the coloured text in a single cell and the highlighting coloured frame on a complete row. The last column uses date formatting which formats any entry as a date.

Use case name: Listing and structural features Translation type and properties:	
One-trip translation	✓
Round trip translation	✓
Presentation instructions	✓
Document content	✓
Dynamic content	
Metadata	
Annotations and security	
Document parts	
Required features: • Formatting o OOXML: Subclause 8.4, 12, 17.4, 18.*, 21.2 o ODF: Subclause 2.3.4, 6.7, 9.3, 14.7, 15.*	

Requirements:

When employee Mary opens John's list, it should remain obvious that certain elements, such as the row marked red or the red text, are more relevant than others. All the applied presentation characteristics created in format A must be reproduced accurately in Mary's format B application. Mapping of colours is covered in subclause 5.5.2.

Conclusion:

Though certain non-vital features such as shared formulas are not supported by both International Standards, and features like *cell protection* are implemented with different granularity, more important features such as highlighted cell borders, background images and the assignment of formulas and functions to particular cells are well-translatable. For this use case, the level of translatability with respect to preservation of content and presentation is high. See subclause 6.3.2 for more details.

5.3.3 Formulas and calculation

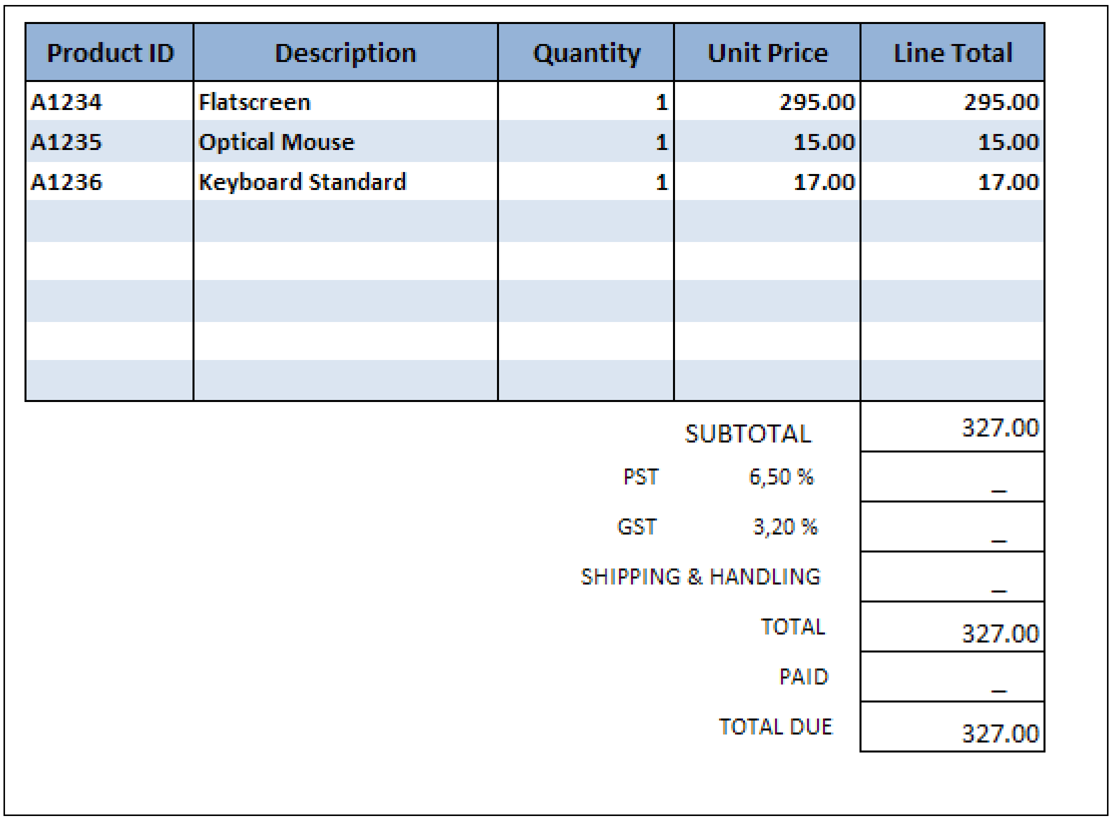
Textual description:

John is working for a big marketing services company. The IT department of John’s company provides format A spreadsheet templates like the one shown in Figure 40 to the employees enabling them to place orders for their demand on new computer equipment. John is using the template in a format B application and returns it to the company's format A environment.

Implementation:

In addition to storing and organizing data, spreadsheets are a powerful tool for managing complex and dynamic calculations. Within a spreadsheet, any cell can contain a formula which references the values of other cells using row and column numbers.

Use case name: Formulas and calculation Translation type and properties:	
One-trip translation	✓
Round trip translation	✓
Presentation instructions	✓
Document content	✓
Dynamic content	✓
Metadata	
Annotations and security	
Document parts	
Required features: • Calculation ○ OOXML: Subclause 18.* ○ ODF: Subclause 8.1	



Product ID	Description	Quantity	Unit Price	Line Total
A1234	Flatscreen	1	295.00	295.00
A1235	Optical Mouse	1	15.00	15.00
A1236	Keyboard Standard	1	17.00	17.00

SUBTOTAL		327.00
PST	6,50 %	_
GST	3,20 %	_
SHIPPING & HANDLING		_
TOTAL		327.00
PAID		_
TOTAL DUE		327.00

Figure 40 — Spreadsheet based invoice template

Requirements:

The essential part of this spreadsheet consists of a table for the invoice line items and *Total Due* cell for automatically calculating the total cost of the items ordered. Each time a new line item is added, the *Total Due* field is updated automatically. Translation of calculation spreadsheets should preserve formula logic as well as presentation and layout information.

Conclusion:

One problem likely to arise when translating spreadsheets is that formula evaluation is generally application dependent; calculations may work differently if used in different applications. Possible workarounds for such difficulties could be:

- The use of self-written formulas as against the native out of the box ones provided by the application which might pose problems during adaptation to non-native platforms;

- The mapping of formulas to specific programming/script languages.

The general underlying problem is the lack of a uniform implementation standard for formulas; such a standard would go a long way towards alleviating formula incompatibilities. The OOXML International Standard includes a documented formula syntax, but ODF does not include a standardized syntax for formulas. The ODF 1.2 Committee Specification includes a standardized Open Formula syntax, which may enable implementers to more reliably map formulas between ODF and OOXML spreadsheets. This is less of a conversion/mapping problem than an end user inconvenience. Further details are given in subclause 6.3.3.

5.4 Presentation documents

5.4.1 Empty presentation document

Textual description:

When a new document is created either in ODF format or in OOXML format, the user initially receives an empty document, even though it may look different in the software user interface. For example, in Microsoft Office, the user sees an empty title/subtitle slide. When the document is saved without any further editing, a document is generated without user content but with some metadata as well as some initial settings, e.g. various styles, masters and slide layout information, page layout and presentation instructions. This initial content should be preserved as far as possible during the translation process.

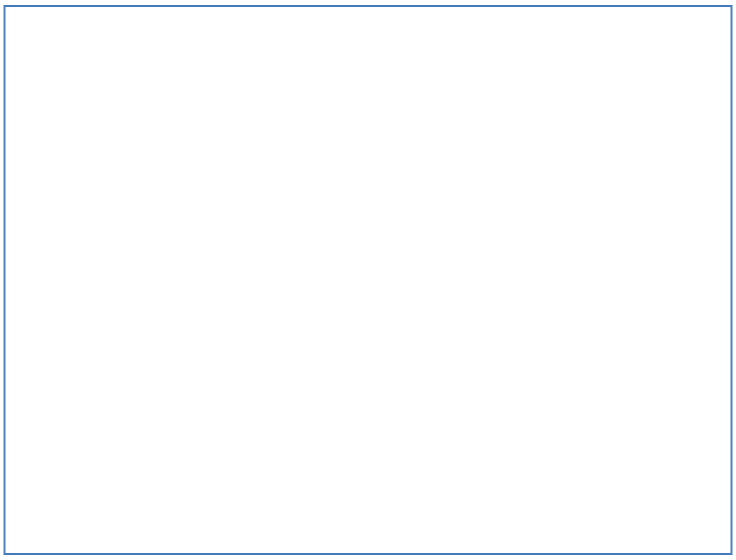


Figure 41 — Empty presentation document

Implementation:

The term *empty document* is not defined in both International Standards. Generally, an application defines some default information like settings, master layout, presentation instructions and metadata. An application may define a default slide without any user-defined content.

Use case name: Empty presentation document Translation type and properties:	
One-trip translation	✓
Round trip translation	✓
Presentation instructions	✓
Document content	
Dynamic content	
Metadata	✓
Annotations and security	
Document parts	
Required features: • OOXML: Subclause 8.5; 13; 19.* • ODF: Subclause 2.2; 2.3.3; 2.4; 3; 9.10; 9.11; 14;15	

Requirements:

OOXML and ODF maintain their own default sets of slide layouts, slide masters, handout masters, notes masters and colour schemes. If a document is translated from one format into another these entities should remain unchanged during the translation in order to facilitate further editing. Furthermore, default presentation and style instructions as well as metadata should be preserved if possible. It is not expected that both International Standards will necessary use similar defaults for metadata.

Conclusion:

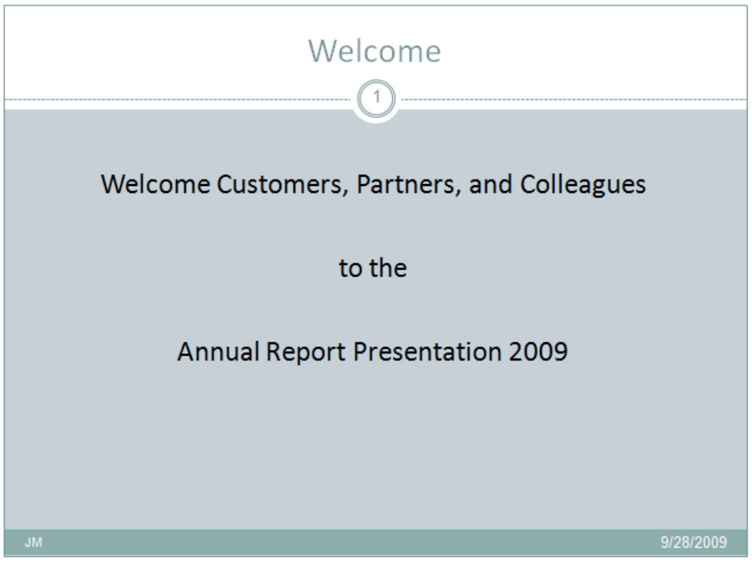
Neither ODF nor OOXML precisely define the term *empty document*. Thus the content of an empty document depends more on the creating application than on the International Standard. When an empty document defined in format A is opened in format B, default slide layouts, slide/handout/notes masters, colour schemes and presentation instructions can be preserved. However, the initial view of the empty document may be slightly different, depending on the rendering engine. Metadata can be translated accordingly, even though some information like the application creating the document may be modified.

5.4.2 Simple text formatting

The basic features of presentation documents are quite similar to those of text processing documents. The following scenario describes the common features of presentation documents exemplified by a simple presentation of an *Annual Report*. The annual report was created by John Marketer using a format A application and should be reviewed by his secretary Mary using a format B application. John uses a customized design to layout the presentation.

Textual description:

Mary opens the annual report for review. She checks to make sure the formatting is correct and there are no grammatical errors.



Welcome

1

Welcome Customers, Partners, and Colleagues

to the

Annual Report Presentation 2009

JM

9/28/2009

Figure 42 — Simple text formatting in presentation documents

Implementation:

The introductory slide makes use of common text formatting features such as centred and bold text. The slide consists of text and a footline consisting of the author's initials, date and slide number. The design defines text fonts, background colours and the position of the footline's elements.

Use case name: Simple text formatting Translation type and properties:	
One-trip translation	✓
Round trip translation	✓
Presentation instructions	✓
Document content	✓
Dynamic content	
Metadata	
Annotations and security	
Document parts	✓
Required features: • Formatting ○ OOXML: Subclause 2.3.3, 13.3, 19.*, 21.1 ○ ODF: Subclause 4.4, 14.6, 15.*	

Requirements:

Any corrections Mary makes in format B, (such as changes to fonts, indentation or layout) should be reproduced without significant discrepancies when John reopens the presentation using his format A application.

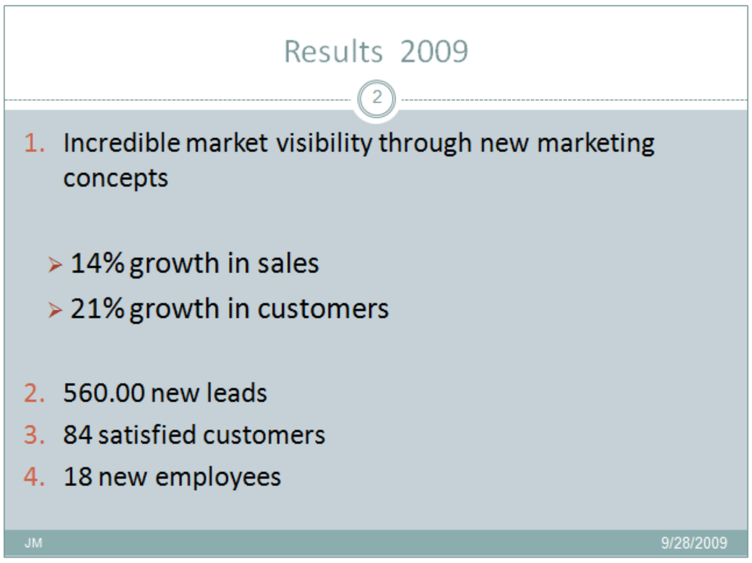
Conclusion:

The requirements of this use case concerning text properties are relatively easy to translate between the two International Standards. Details can be found in subclause 6.4.2. Because ODF does not support the concept of *themes*[10] it is solely possible to map theme's properties such as colours, fonts, and effects to the ODF master layout but not vice versa. Thus round trip translation is not supported for themes.

5.4.3 Itemization and numeration

Textual description:

John Marketer shows the following slide to the management board during their annual executive board meeting where he aims to present the company's achievements for the past year in short concise points. The slide has been cross-checked by Mary.



Results 2009

2

1. Incredible market visibility through new marketing concepts
   - 14% growth in sales
   - 21% growth in customers
2. 560.00 new leads
3. 84 satisfied customers
4. 18 new employees

JM 9/28/2009

Figure 43 — Itemization and numeration in presentation documents

Implementation:

This slide contains a text list similar to that used in word processing applications. The list is comprised of a combination of both numbered bullet point and list items. The bullet points are demarcated by symbols, while the main points are demarcated by numerals.

10 Please consider that themes are not used in this use case. The example can be implemented utilizing styles.

Use case name: Itemization and numeration Translation type and properties:	
One-trip translation	✓
Round trip translation	✓
Presentation instructions	✓
Document content	✓
Dynamic content	✓
Metadata	
Annotations and security	
Document parts	✓
Required features: • Itemization and numeration o OOXML: Subclause 13.3.* o ODF: Subclause 4.4	

Requirements:

The combination of bullet points and numbered list items should be displayed identically by both applications, since any change in indentation, formatting or symbols used could cause confusion or distortion of facts.

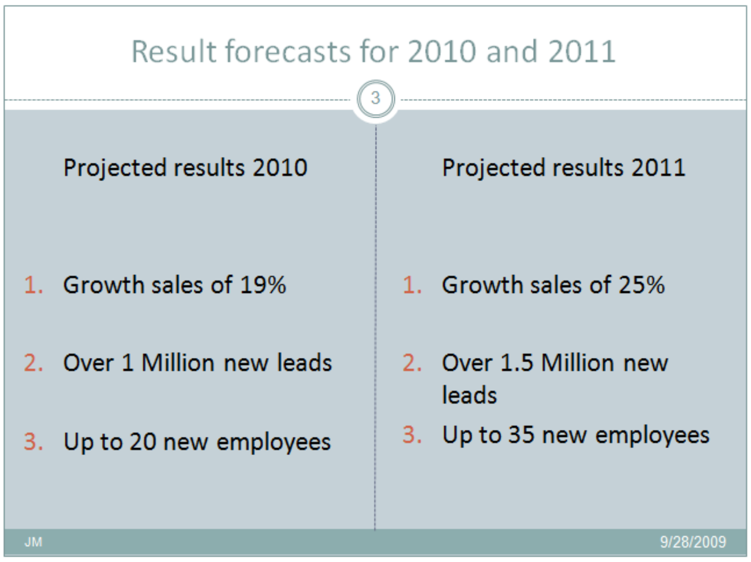
Conclusion:

The minor problems evident in the translatability of itemization and numeration in word processing documents also apply to presentations because ODF implements these features identical for all document types. In this use case, however, translatability between the two International Standards is on a high level.

5.4.4 Positioning and layout

Textual description:

John Marketer has created a slide, which portrays projected results for two different years. These two years will be compared with three short bullet points, and the difference between the statistics for the two years should be recognized easily at the first glance. The slide has been crosschecked by Mary.



Result forecasts for 2010 and 2011

3

Projected results 2010

1. Growth sales of 19%
2. Over 1 Million new leads
3. Up to 20 new employees

Projected results 2011

1. Growth sales of 25%
2. Over 1.5 Million new leads
3. Up to 35 new employees

JM

9/28/2009

Figure 44 — Positioning and layout in presentation documents

Implementation:

The slide contains two sections. Each contains distinctive text. The text in each section is a combination of headers, regular text portions and numbered list items. The two sections differ in content but not, however, in format.

Use case name: Positioning and layout Translation type and properties:	
One-trip translation	✓
Round trip translation	✓
Presentation instructions	✓
Document content	✓
Dynamic content	
Metadata	
Annotations and security	
Document parts	✓
Required features: • Positioning and layout ○ OOXML: Subclause 13.3.9 ○ ODF: Subclause 14.15	

Requirements:

In this use case, the fragmentation of the text into two separate windows is significant. When Mary opens the slide in her format B application, it should display precisely as it did in John's format A application. All changes made by Mary should be visible to John when he reopens the document and display as they did in the format B application.

Conclusion:

In this use case, translatability is high for both content and presentation instructions.

5.4.5 Slide blending and animation effects

Textual description:

To enhance the presentation she has prepared, Mary applies visual animation effects between the slide transitions. John reviews the presentation shortly before a board meeting using his format A application.

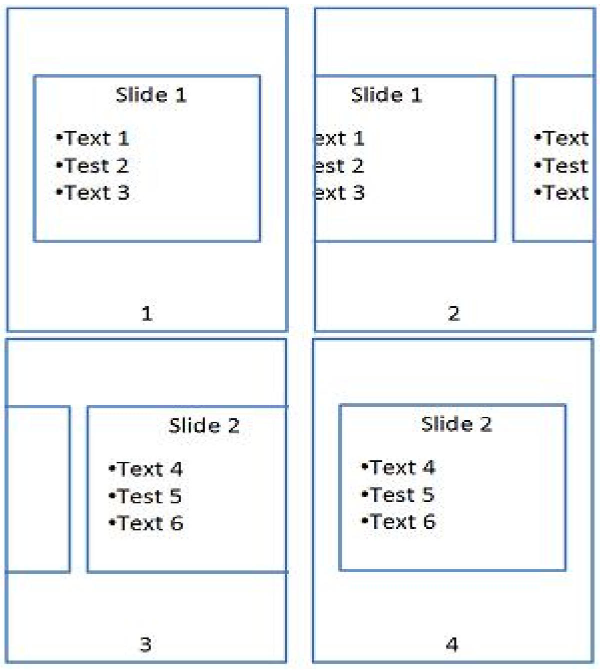


Figure 45 — Slide blending in presentation documents

Implementation:

Instead of simple transfers from slide to slide, Mary uses blending effects where one slide blends over into another, as in the *fades* or *push* transitions illustrated in Figure 45. Animation transitions make the slide changes appear more fluid and give the presentation a smoother overall look.

Use case name: Slide blending and effects **Translation type and properties:**	
One-trip translation	✓
Round trip translation	✓
Presentation instructions	✓
Document content	
Dynamic content	
Metadata	
Annotations and security	
Document parts	
Required features: • Presentation o OOXML: Subclause 19.5 o ODF: Subclause 13.1, 15.36	

Requirements:

The same visual effects should be visible when John opens the presentation using format A. If Mary later alters or adds to the effects already applied using a format B application, such changes should be reflected the next time John reopens the document using format A. A round trip translation should be possible.

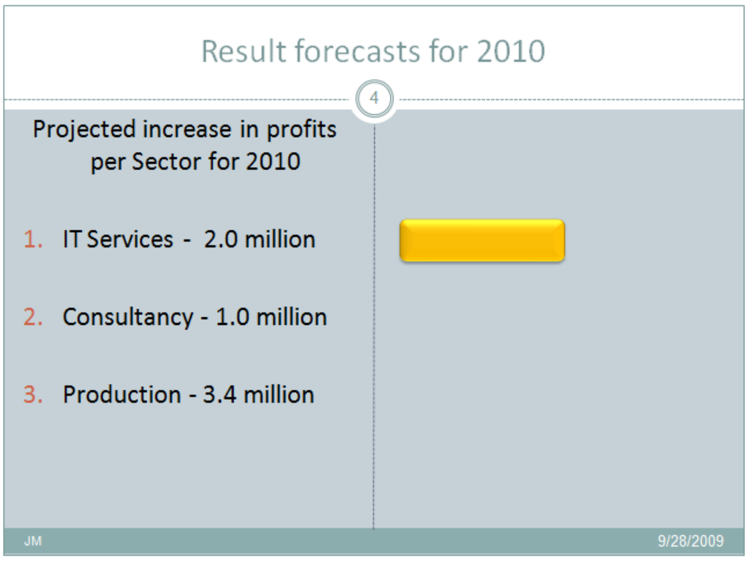
Conclusion:

Certain features such as time line functionality or transitioning slides along Bezier curves or polylines are not supported by ODF. OOXML provides a far richer set of features which are only marginally translatable, or indeed impossible to transform into ODF. This makes for restricted translatability between the two International Standards with regard to animated slide transition features.

5.4.6 Animations

Textual description:

To be able to visualize the quoted statistics better, John Marketer adds some animations to be displayed to the right of the text box. He wants Mary to review and make sure the statistics displayed are correct before he presents them at a meeting.



Result forecasts for 2010

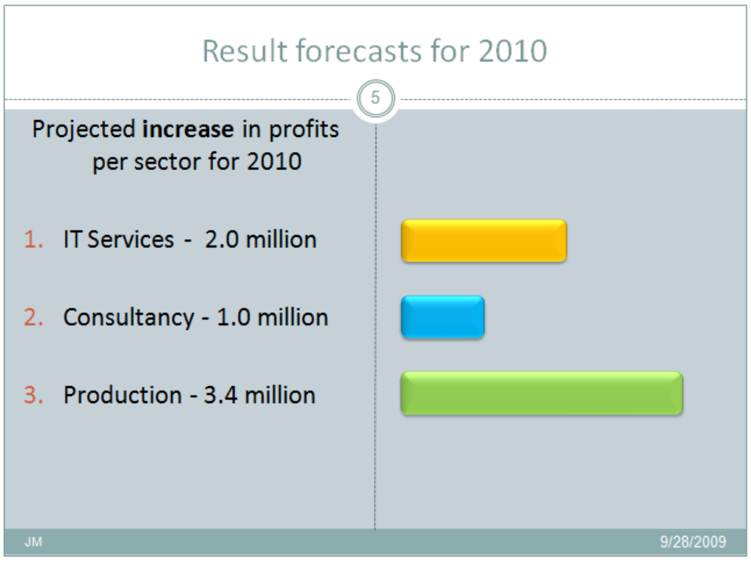
4

Projected increase in profits per Sector for 2010

1. IT Services - 2.0 million
2. Consultancy - 1.0 million
3. Production - 3.4 million

JM 9/28/2009

Figure 46 — Slide before animation



Result forecasts for 2010

5

Projected increase in profits per sector for 2010

1. IT Services - 2.0 million
2. Consultancy - 1.0 million
3. Production - 3.4 million

JM 9/28/2009

Figure 47 — Slide after animation

Implementation:

The bars shown in Figure 47 seem animated as they appear one by one with the help of graphic effects which are triggered by a mouse click or shown at timed intervals. The embedded animation is visible for as long as the slide is active.

Use case name: Animations **Translation type and properties:**	
One-trip translation	✓
Round trip translation	✓
Presentation instructions	✓
Document content	
Dynamic content	
Metadata	
Annotations and security	
Document parts	
Required features: • Presentation o OOXML: Subclause 19.5 o ODF: Subclause 9.7	

Requirements:

Mary should be able to replay these animations in her format B environment without noticing any difference; any changes she makes to the animations should also be reproducible in John's format A environment.

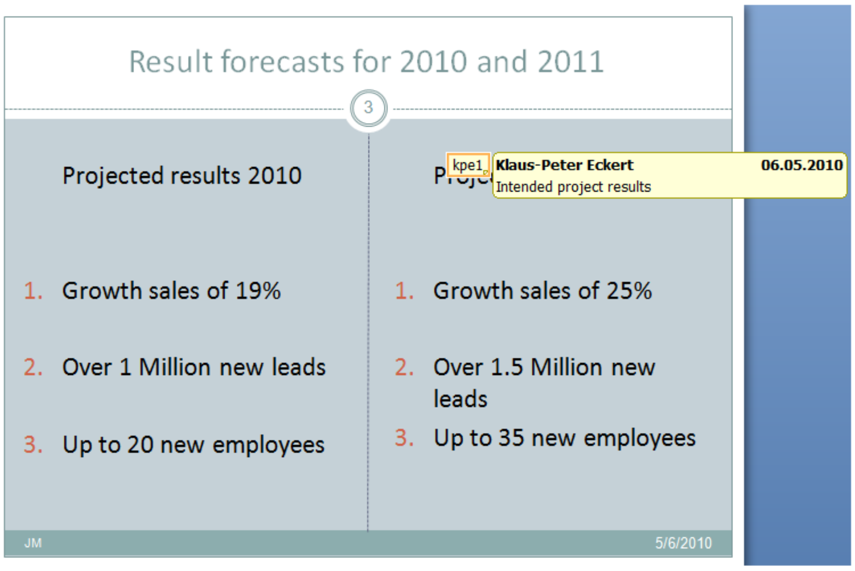
Conclusion:

Both International Standards have a well-developed set of tools to animate graphic elements. There could be slight difficulties in translatability between applications since animations based on OOXML can be manipulated with finer granularity than those based on ODF. This imposes more constraints on the translation of ODF based applications. One possible way of circumventing some of these setbacks is through the use of SMIL (Synchronized Multimedia Integration Language), which offers a common animation platform for the two International Standards. While SMIL animations can be embedded in ODF presentations, the presentation markup used in OOXML uses similar concepts as SMIL.

5.4.7 Comments

Textual description:

After most parts of the annual report have been created by John, he wants to hear feedback on the layout of presentation as well as contents of it. He asks Mary to give some comments on the presentation document. John will check Mary’s comments on his format A application after she creates a few comments using her format B application.



Result forecasts for 2010 and 2011

3

Projected results 2010

1. Growth sales of 19%
2. Over 1 Million new leads
3. Up to 20 new employees

Proje

kpe1 Klaus-Peter Eckert 06.05.2010
Intended project results

1. Growth sales of 25%
2. Over 1.5 Million new leads
3. Up to 35 new employees

JM 5/6/2010

Figure 48 — Presentation document with comment

Implementation:

The comments on the slide should not be visible during a slide show. They should be visible when editing the presentation. Small comment author's initial text box is appeared where the author has placed it on the slide. When a user clicks on this small initial comment box, it should open and show content of the comment, date and time it was created and the author of the comment.

Use case name: Comment **Translation type and properties:**	
One-trip translation	✓
Round trip translation	✓
Presentation instructions	
Document content	✓
Dynamic content	
Metadata	✓
Annotations and security	✓
Document parts	
Required features: • Comment ○ OOXML: Subclause 19.4 ○ ODF: Subclause 14.4.2	

Requirement:

John should be able to see Mary's comments on his slides. Any comment created by Mary's format B application should be shown in John's format B application.

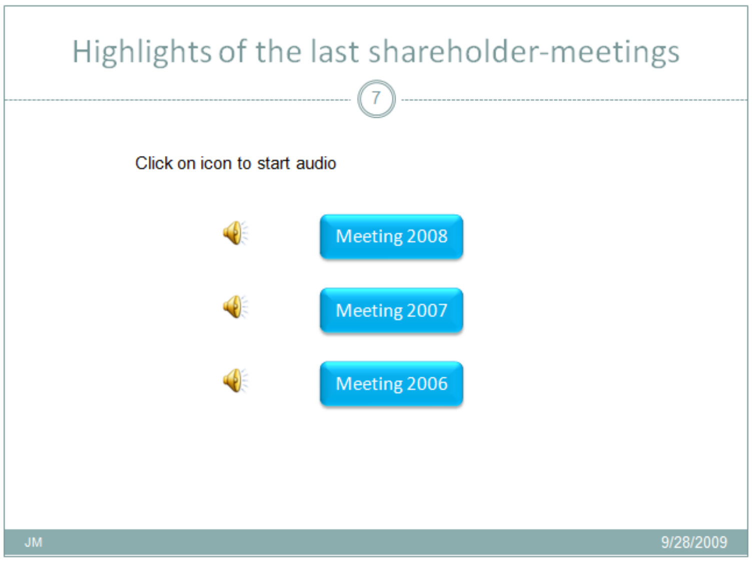
Conclusion:

The translatability of comment between the two formats is very low. In this use case, comments in format A documents are not shown in format B and vice versa. Even though the two International Standards provide visually similar comment function, the structures used by the two formats are quite different. ODF stores comments on the slides together with other information as <*note*> element, while OOXML stores the comment part in a separate XML-document as <*cm*>.

5.4.8 Multimedia content

Textual description:

To make a more lively presentation, John has decided to incorporate multimedia / audio content into his slides. He instructs Mary on where and how to place the multimedia elements. Subsequently he crosschecks the new slides to ensure everything is working smoothly.


Highlights of the last shareholder-meetings

7

Click on icon to start audio

Meeting 2008

Meeting 2007

Meeting 2006

JM 9/28/2009

Figure 49 — Multimedia content in presentation documents

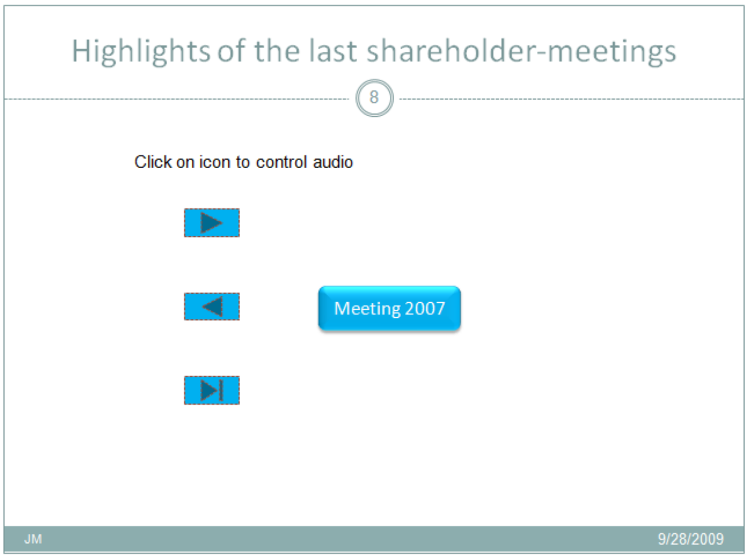

Highlights of the last shareholder-meetings
8
Click on icon to control audio
Meeting 2007
JM
9/28/2009

Figure 50 — Control icons for multimedia content

Implementation:

John has embedded three multimedia elements (audio) each associated with additional graphic elements, serving as clickable icons. When an audio is played, animated *forward*, *backward* and *end* icons appear.

Use case name: : Multimedia content **Translation type and properties:**	
One-trip translation	✓
Round trip translation	✓
Presentation instructions	✓
Document content	✓
Dynamic content	
Metadata	
Annotations and security	
Document parts	✓
Required features: • Multimedia content and vector graphics ○ OOXML: Subclause 15.2.2, 15.2.10 ○ ODF: Subclause 9.8	

Requirements:

When John reopens the slide, all media assets should be properly referenced, and the animated icons should work in the same way they did in Mary's application.

Conclusion:

The only means provided by ODF to implement these functionalities is SMIL which is a good alternative to the usual <*animations*> element when mixtures of multiple animations are running at the same time. ODF's use of SMIL for certain animation effects is not likely to give rise to any major translatability issues since the schema and syntax of OOXML's PresentationML is loosely based on SMIL.

5.4.9 Master layout

Textual description:

Mary creates slide templates for layouts she tends to use very often such as recuring topics, weekly jour fixes or periodical board meetings. John decides to introduce some general changes to the layout. He opens one of the layout tempates emailed to him by Mary and edits it.

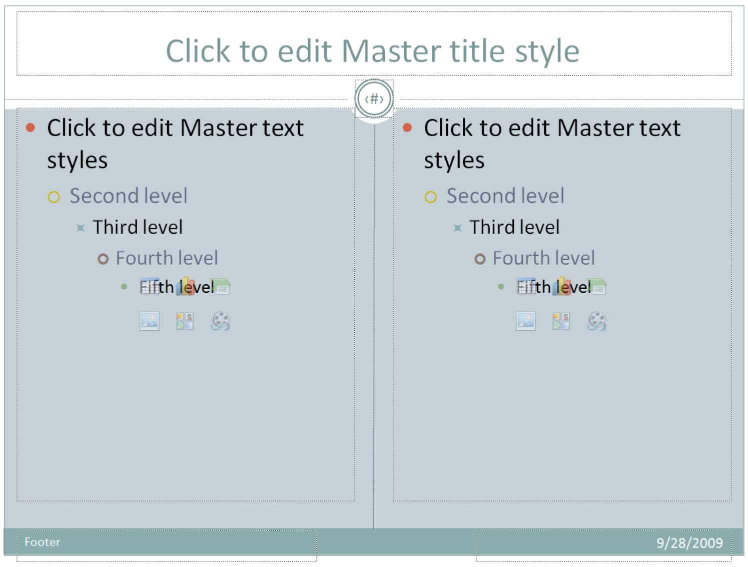


Figure 51 — OOXML master slide in presentation documents

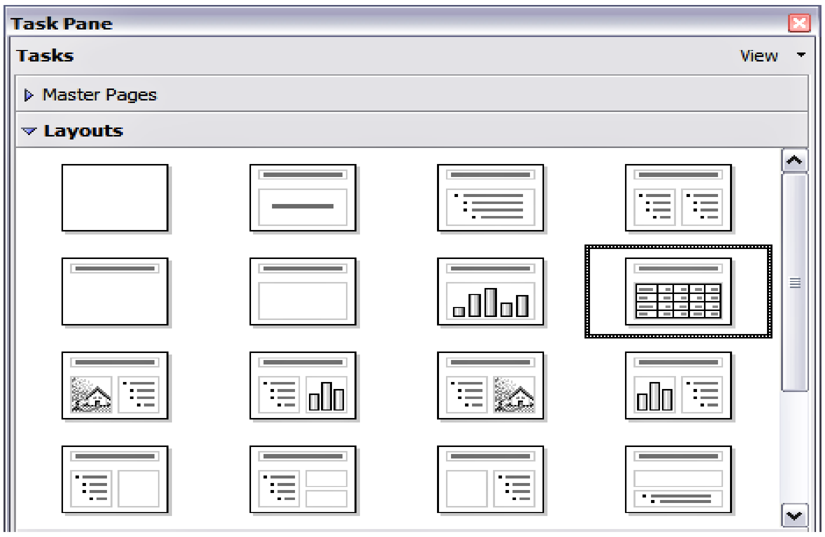


Figure 52 — ODF master layout in presentation documents

Implementation:

Mary uses the master slide to simultaneously edit layout on multiple slides (see Figure 51). John then manipulates the master slide to further adjust the slide layout (see Figure 52) and returns the improved template to Mary.

Use case name: Master layout Translation type and properties:	
One-trip translation	✓
Round trip translation	✓
Presentation instructions	
Document content	✓
Dynamic content	
Metadata	✓
Annotations and security	
Document parts	✓
Required features: • Presentation Masters o OOXML: Subclause 8.5, 19.*, 20.1 o ODF: Subclause 9.*, 13.5, 14.*, 15.36	

Requirements:

The changes made by John should be reflected in the slide master when Mary reopens the presentation in format B. John should also be able to automatically see the master changes reflected on each individual slide without having to open the master slide settings.

Conclusion:

Translatability between the master slides in OOXML and master layouts in ODF is very high and satisfies most requirements. For more details see subclause 6.4.4.

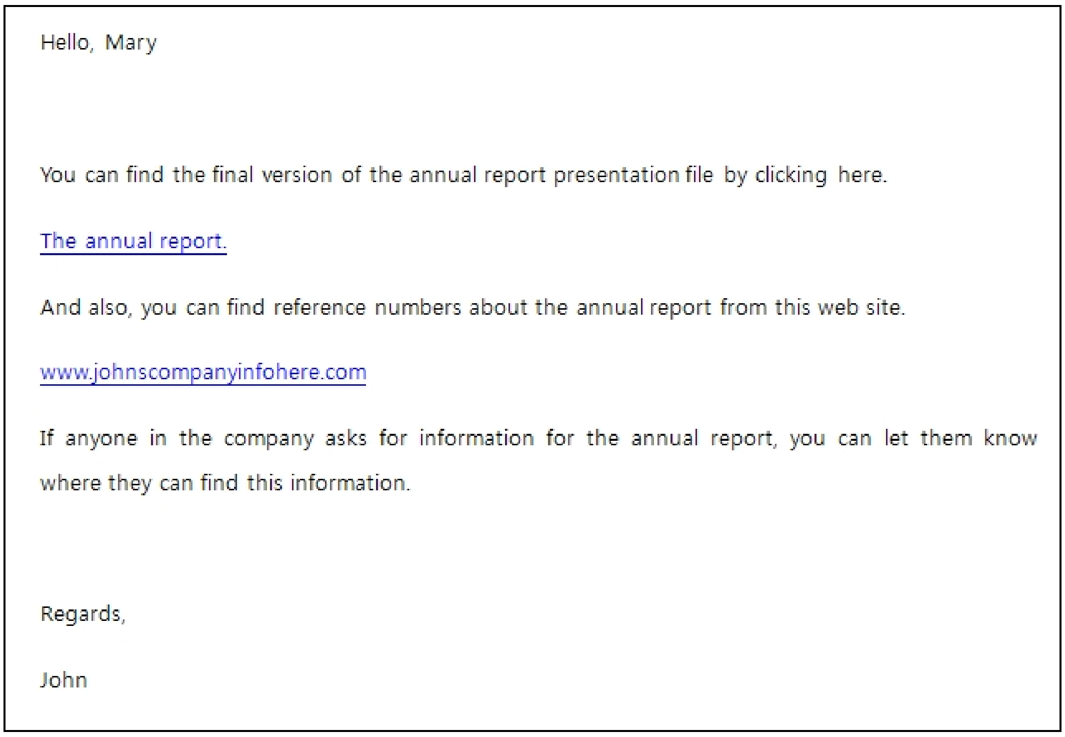
5.5 Common properties and mutual inclusion of documents

This section describes document properties that are independent of a document type. Two use cases for basic properties such as hyperlinks and colours are described in this section. In addition, use cases for mutual included documents are given. Both ODF and OOXML allow including documents of type A into a document of type B. The resulting documents cannot be categorized by a specific type even though the top level document has a specific type.

5.5.1 Hyperlinks between documents

Textual description:

John sends a format A text processing document to Mary to inform her where she can find the information indicated by hyperlinks in the document. Mary is using a format B word processing application. The annual report file is a format B presentation document located on the shared computer of John and Mary.



Hello, Mary

You can find the final version of the annual report presentation file by clicking here.

The annual report.

And also, you can find reference numbers about the annual report from this web site.

www.johnscompanyinfohere.com

If anyone in the company asks for information for the annual report, you can let them know where they can find this information.

Regards,

John

Figure 53 — Hyperlinks in a text document

Implementation:

Hyperlinks should be able to lead users to target destinations. Destinations could be a file, web address or a particular location of a file. Hyperlinks are shown in blue and underlined. In this use case, the first hyperlink is either an absolute path or a relative path to a file that is located on the same computer system. The second hyperlink refers to a certain website.

Use case name: Hyperlinks between documents **Translation type and properties:**	
One-trip translation	✓
Round trip translation	
Presentation instructions	✓
Document content	
Dynamic content	✓
Metadata	
Annotations and security	
Document parts	✓
Required features: • Formatting ○ OOXML: Subclause 15.3, 17.16.2.2, 17.16.5.25, 18.3.1.47~48, 20.1.4.1.15, 20.1.4.1.19, 21.1.2.3.5~6, 22.2.2.11~14 ○ ODF: Subclause 3.1.14, 5.1.4, 7.12.7, 9.3.10	

Requirement:

Hyperlink parts should be distinguished from other text parts and should lead to the destinations. When Mary clicks the blue underlined hyperlinks, she should be able to open a format B presentation file, as well as the web site. After she has visited the hyperlinked destinations, the colour should change to another colour to show that the links have been visited.

Conclusion:

Hyperlinks from ODF that follow the URL and the file in a certain location work in OOXML. A hyperlink from OOXML that follows the URL works in ODF. However, a hyperlink from OOXML that follows the destination to a certain location of the file may not work in ODF because ODF supports only relative paths in hyperlinks while OOXML supports both relative and absolute paths.

5.5.2 Colours

Textual description:

When Mary creates templates for word processing and presentation documents in her format B application she defines a set of colours that should be used in all documents for texts, lines and shapes. John uses these templates in his format A application.

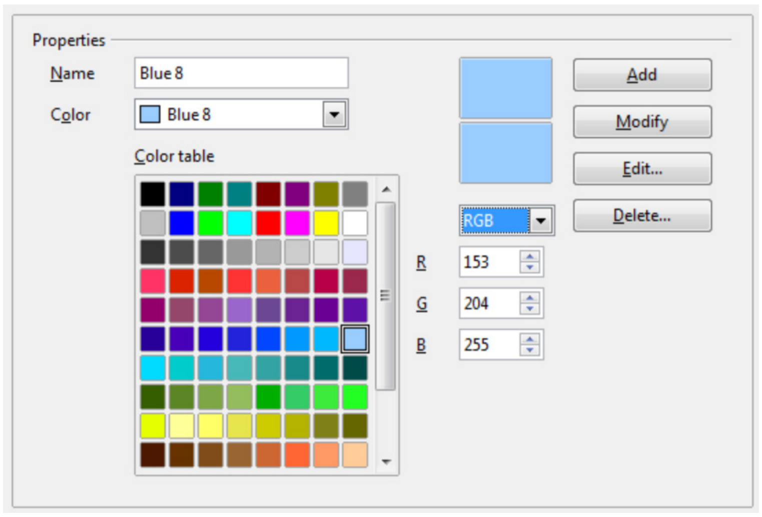

Properties
Name: Blue 8
Color: Blue 8
RGB
R 153
G 204
B 255

Figure 54 — Colour definition in an ODF application

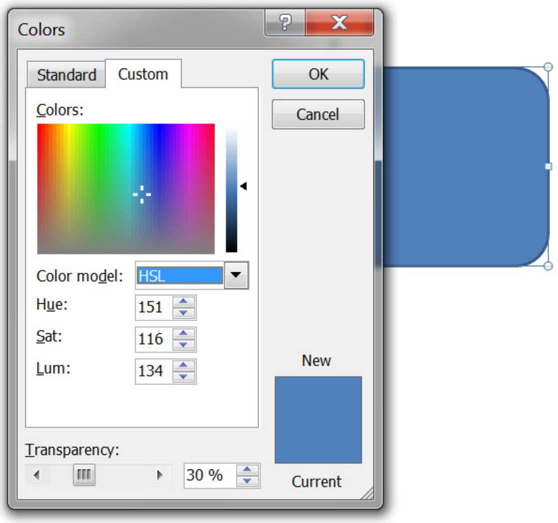


Figure 55 — Colour definition in an OOXML application

Implementation:

Use case name: Colours Translation type and properties:	
One-trip translation	✓
Round trip translation	✓
Presentation instructions	✓
Document content	
Dynamic content	
Metadata	
Annotations and security	
Document parts	
Required features: • Colour ○ OOXML: Subclause 20.1.2.3, 20.1.4, 14.2.7 ○ ODF: Subclause 15.17	

OOXML defines colours in different models. A single colour can be defined either in the *red, green, blue* (RGB) colour model or in the *hue, saturation, lightness* (HSL) colour model. ODF solely uses the RGB model. OOXML uses *themes* and *accents* to define groups of colours and to reference to a single colour. ODF uses direct references to the specification of a colour. More sophisticated support to end users is provided by ODF applications, not by the standard itself.

Requirements:

When John uses the templates in his format A environment the colours should be defined in a similar way as in Mary's format B environment.

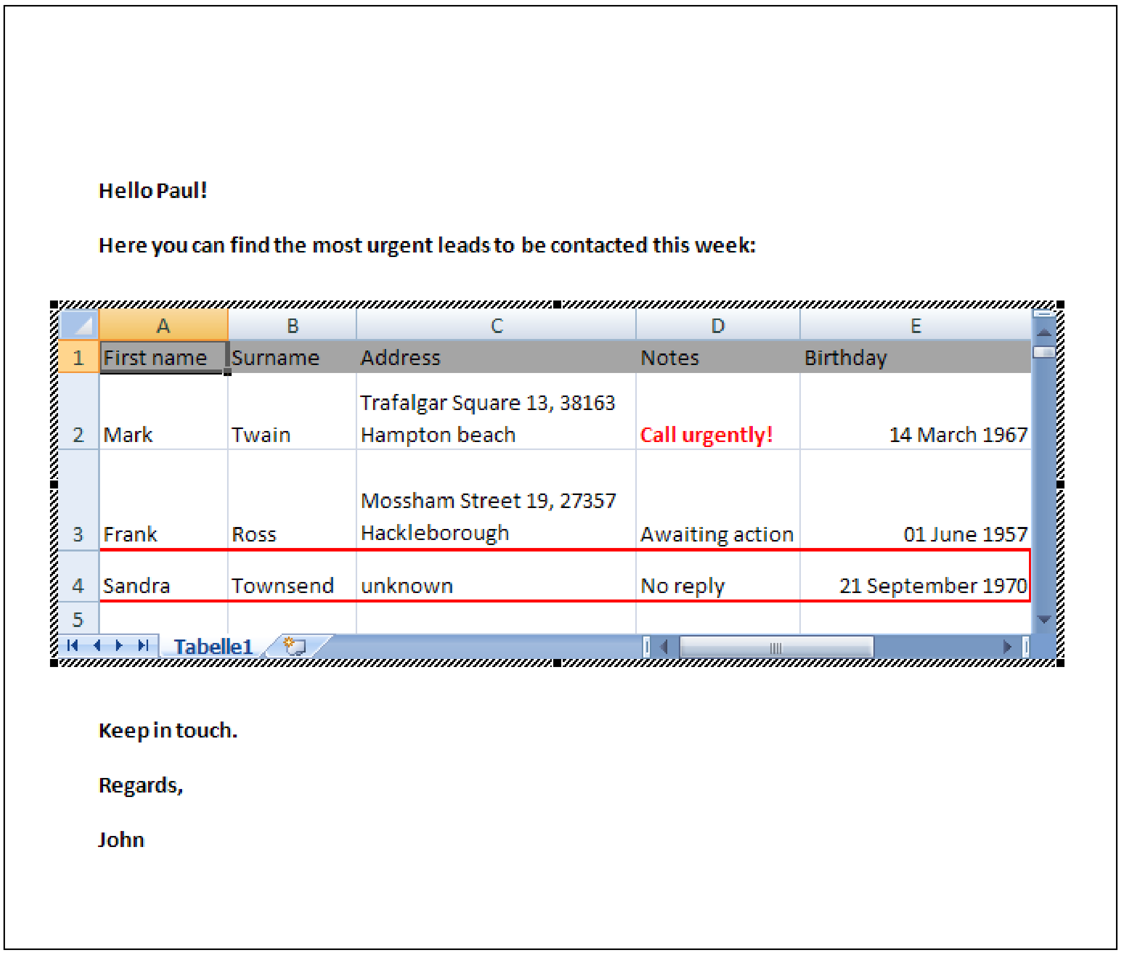
Conclusion:

While OOXML supports different ways to represent, group and reference colours, ODF uses solely one colour model. It is possible to translate colour specifications defined in the RGB-model between both formats, and additionally from HSL to RGB within OOXML. Thus the basic translatability of colours is high although round trip translation of higher level concepts and HSL defined colours is not possible or must be performed on application level.

5.5.3 Embedded spreadsheet documents

Textual description:

John wants to pass on information, contained in a format A spreadsheet, to Paul, who is using format B. Instead of recreating the portion of the spreadsheet he wants to send, he simply embeds the pertinent spreadsheet information in a text document containing a note and instructions as shown in Figure 56.



Hello Paul!

Here you can find the most urgent leads to be contacted this week:

	A	B	C	D	E
1	First name	Surname	Address	Notes	Birthday
2	Mark	Twain	Trafalgar Square 13, 38163 Hampton beach	Call urgently!	14 March 1967
3	Frank	Ross	Mossham Street 19, 27357 Hackleborough	Awaiting action	01 June 1957
4	Sandra	Townsend	unknown	No reply	21 September 1970
5					

Tabelle1

Keep in touch.

Regards,

John

Figure 56 — Spreadsheet embedded in a word processing document

Implementation:

An obvious advantage of this approach is that the data in the embedded spreadsheet can be edited and manipulated directly as a dynamic source by the spreadsheet engine rather than being handled statically.

ODF accomplishes this by making use of the <*insertion*> element which contains the information required to identify any insertion of content. Placing a frame within the text area, such as a drawing shape in which a spreadsheet has been embedded, can also be used to create the same effect.

OOXML proposes two options for embedding a spreadsheet within a text document:

- Embedded Packages - Two documents (in this case: a SpreadsheetML document embedded in a WordprocessingML document) are stored together in a format defined by OOXML as an *embedded package*.
- Embedded Objects – The data stored in the object is identified by a unique string (ProgID) which identifies the kind of object data to be embedded.

Use case name: Embedded spreadsheet documents Translation type and properties:	
One-trip translation	✓
Round trip translation	
Presentation instructions	✓
Document content	✓
Dynamic content	
Metadata	
Annotations and security	
Document parts	✓
Required features: • Embedded spreadsheets o OOXML: Subclause 15.2 o ODF: Subclause 8, 9.3	

Requirements:

When Paul opens the document containing the embedded spreadsheet, he expects all edited features of the spreadsheet such as colour boundaries and highlighted text to be presented exactly as they were when John saved the original spreadsheet. For example, the date format needs to be maintained exactly, since an incorrect representation of the original date data could lead to confusion or errors.

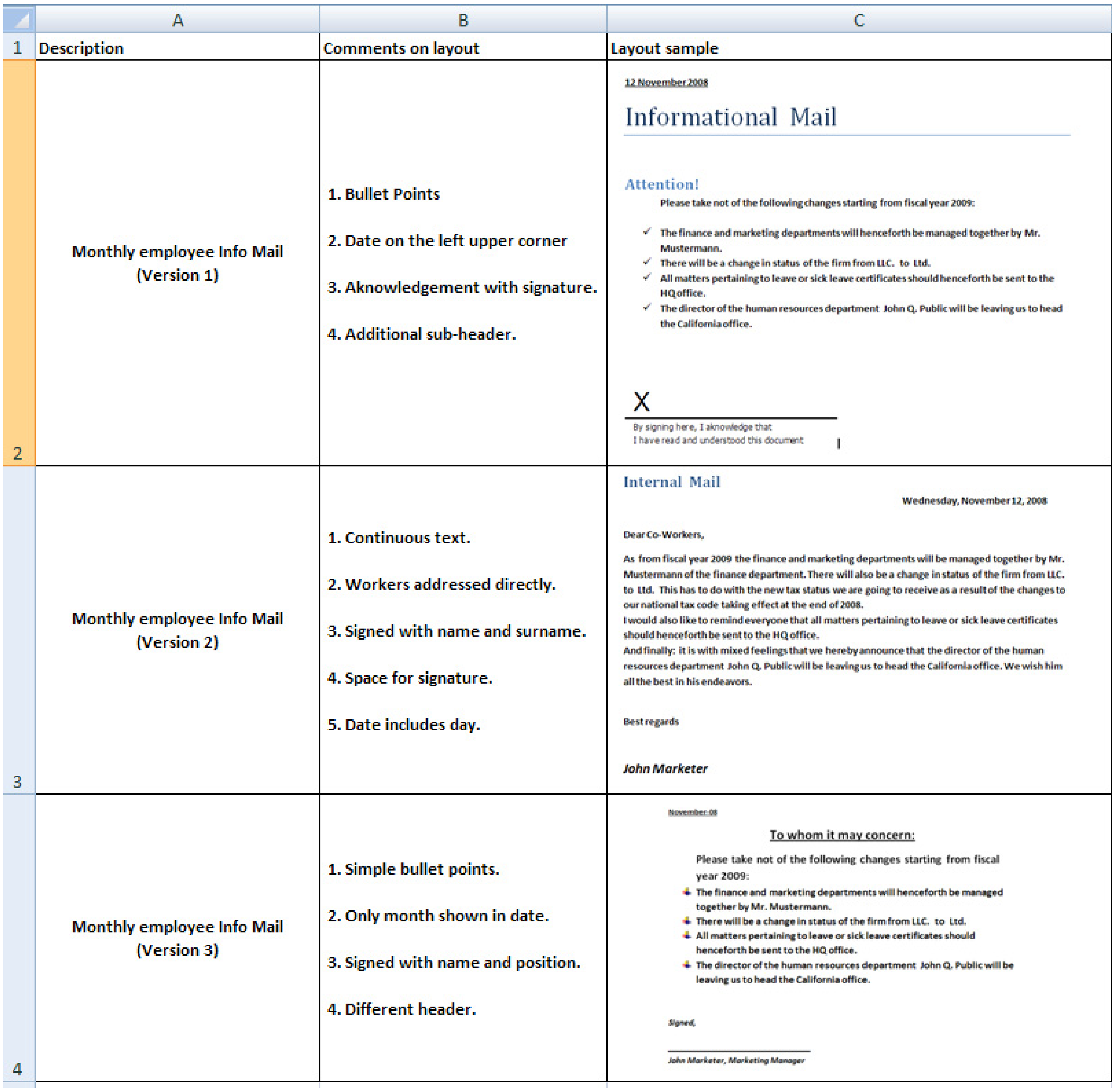
Conclusion:

Translation of embedded objects between ODF and OOXML does not present any major barriers; both International Standards support Object Linking and Embedding (OLE) as well as alternative image representations of linked objects. Slight translation difficulties may occur in the latter case, since when representing alternative images OOXML may refer to elements of the deprecated VML format which is not an open standard.

5.5.4 Simple text formatting and embedded documents

Textual description:

John Marketer's secretary creates a spreadsheet containing several sample newsletter layouts, and saves it in format B before sending it to John who opens it with his format A supporting application.



	A	B	C
1	Description	Comments on layout	Layout sample
2	Monthly employee Info Mail (Version 1)	1. Bullet Points 2. Date on the left upper corner 3. Aknowledgement with signature. 4. Additional sub-header.	12 November 2008 Informational Mail Attention! Please take not of the following changes starting from fiscal year 2009: ✓ The finance and marketing departments will henceforth be managed together by Mr. Mustermann. ✓ There will be a change in status of the firm from LLC. to Ltd. ✓ All matters pertaining to leave or sick leave certificates should henceforth be sent to the HQ office. ✓ The director of the human resources department John Q. Public will be leaving us to head the California office. X By signing here, I aknowledge that I have read and understood this document
3	Monthly employee Info Mail (Version 2)	1. Continuous text. 2. Workers addressed directly. 3. Signed with name and surname. 4. Space for signature. 5. Date includes day.	Internal Mail Wednesday, November 12, 2008 Dear Co-Workers, As from fiscal year 2009 the finance and marketing departments will be managed together by Mr. Mustermann of the finance department. There will also be a change in status of the firm from LLC. to Ltd. This has to do with the new tax status we are going to receive as a result of the changes to our national tax code taking effect at the end of 2008. I would also like to remind everyone that all matters pertaining to leave or sick leave certificates should henceforth be sent to the HQ office. And finally: it is with mixed feelings that we hereby announce that the director of the human resources department John Q. Public will be leaving us to head the California office. We wish him all the best in his endeavors. Best regards *John Marketer*
4	Monthly employee Info Mail (Version 3)	1. Simple bullet points. 2. Only month shown in date. 3. Signed with name and position. 4. Different header.	November 08 To whom it may concern: Please take not of the following changes starting from fiscal year 2009: ✦ The finance and marketing departments will henceforth be managed together by Mr. Mustermann. ✦ There will be a change in status of the firm from LLC. to Ltd. ✦ All matters pertaining to leave or sick leave certificates should henceforth be sent to the HQ office. ✦ The director of the human resources department John Q. Public will be leaving us to head the California office. *Signed,* *John Marketer, Marketing Manager*

Figure 57 — Spreadsheet with simple text and embedded documents

Implementation:

In spreadsheet documents, portions of text are often included as cell content. The use case illustrates one such scenario which is also associated with the formatting and inclusion of graphics.

Spreadsheets often contain formatted text as cell content. This use case illustrates one such scenario which is also associated with formatting and the inclusion of graphics.

The example given in Figure 57 contains three rows and three columns. Column A contains a short text description. Column B contains comments describing the newsletter layout. Column C contains a short text sample formatted using the proposed layout. In addition to paragraph and word formatting, the sample layout in column C also contains embedded graphic elements. Each layout sample fits into the last cell on the row which bears the scaled down proportions of a letter-format page, and is displayed as a page in miniature. The layout samples included in the sheet can either be linked to or embedded within the document.

Use case name: Simple text formatting and embedded documents **Translation type and properties:**	
One-trip translation	✓
Round trip translation	
Presentation instructions	✓
Document content	✓
Dynamic content	
Metadata	
Annotations and security	
Document parts	✓
Required features: • Formatting o OOXML: Subclause 12.3, 15.2 o ODF: Subclause 9.3	

Requirements:

In translating the information needed to present this spreadsheet using a format A application, all presentation instructions settings should be preserved. The graphic elements and images should likewise maintain their original graphical appearance.

Conclusion:

Translation between ODF and OOXML does not present any major barriers as both International Standards support Object Linking and Embedding (OLE) as well as alternative image representations of linked objects. Translating vector graphics could pose slight problems as mentioned in subclause 5.5.1.

5.5.5 Embedded charts

Textual description:

John creates a slide which serves to compare the result forecasts between two years at a board meeting. Mary has been asked to add the chart to the annual report, which will include many of the points John discussed at the board meeting for additional visualisation.

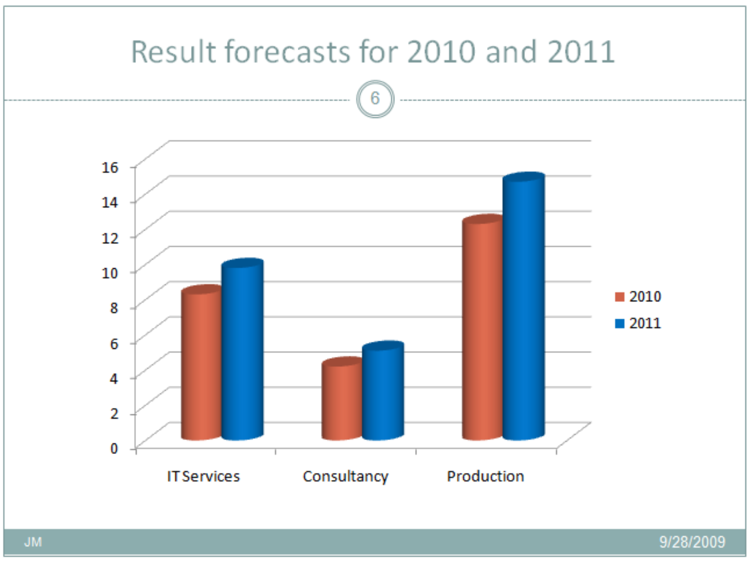


Figure 58 — Diagram in presentation documents

Implementation:

Presentation documents can contain simple embedded graphics, called shapes in ODF. Diagrams used in presentation documents in the case of ODF are basically drawing shapes which differ only in their attribute/style-family elements. Presentation shapes are assigned presentation attributes with a style from the *presentation* family, while drawing shapes are assigned drawing attributes with a style from the *graphic* family. In addition, presentation shapes are further classified based on usage. Examples of such classifications include *text*, *graphic* or, as shown in Figure 58, *chart*. The chart is created from a spreadsheet document and embedded into the presentation.

Use case name: Embedded charts Translation type and properties:	
One-trip translation	✓
Round trip translation	✓
Presentation instructions	✓
Document content	✓
Dynamic content	
Metadata	
Annotations and security	
Document parts	
Required features: • Diagrams o OOXML: Subclause 14; 12.3 o ODF: Subclause 9.2; 10	

Requirements:

When Mary reviews the document, it is displayed in exactly the same way it looks in John format A application: The lines, colours and proportions should be the same in both applications. When John opens the improved slide set the diagram should display like in Mary's format B application.

Conclusion:

The original view would, to a great extent, be retained during a translation between the International Standards as the translatability between graphic components is high.

6 Features and functionality

6.1 Introduction

This section explains the features needed to implement the use cases described in clause 5. The tables in the following subsections summarize the availability of various features for each of the two document formats as well as offering an estimate of the *translatability level* of the various features, which is defined as follows:

- *Low translatability*; either one of the International Standards does not support this feature at all, or the way the feature is implemented differs so significantly that feature translation is impossible without information loss.

- *Medium translatability*; these features are supported in both formats, although some aspects may differ and workarounds may be required. Features marked as *medium* may support a one way translation, but will result in information loss during round trip translations. The "Notes" column provides further details on each relevant feature.

- *High translatability*; these features are supported by both International Standards, round trip translation should pose no problems.

The characterization of translatability by the above mentioned metric indicates whether it is possible or in general impossible to translate a feature between the International Standards. It cannot be assumed that a given tool actually has an implementation for all translations, indicated as *high*. On the other hand it cannot be excluded that a given tool has a specific implementation for a translation, indicated as *low*. Translation rules will always be tool specific.

It is important to note that the focus of this section is to describe the translatability of various document features between formats and not to engender discussion about the relevance of certain features or to make recommendations for the addition or removal of features from one of the International Standards. All characterizations are focused on strictly conformant OOXML documents. Transitional conformance as described in ISO/IEC 29500-4 is not considered. All statements about ODF refer to ISO/IEC 26300.

Technical remark

XML elements and attributes in this Technical Report are shown in an unqualified style like <*element*> and *@attribute* instead of <*ns:element*> and *@ns:attribute*. The namespaces can be derived easily from the application context if necessary.

6.2 Word processing documents

6.2.1 Text formatting

This subclause describes the attributes that define the functionality and sub functionality of text formatting in word processing documents. Both formats support formatting text at the paragraph level as well as finer granularity. OOXML calls this capability a *run*, ODF calls it a *span*. The following table summarizes the features which appear in the use cases described in clause 5. Examples of XML representations in both formats are given in subclause 7.2.2.

Table 1 — Text formatting

Functionality	Sub functionality	OOXML	ODF	Transla-tability	Notes
Bold text (font weight)		Yes 17.3.2.1	Yes 14.6.3	Medium	In addition to bold, ODF allows font weight to be specified numerically (100-900).
Text borders		Yes 17.3.2.4	No	Low	ODF only supports borders on whole paragraphs.
Whitespaces		Yes 17.15.1.18 17.18.7 ISO/IEC 29500-3 10.	Yes 1.6	Medium	Because certain OOXML elements (such as the *@preserve* attribute defined separately in ISO/IEC 29500-3), are not supported by ODF, translatability of this feature could be problematic.
Capitalization					
	All upper case	Yes 17.3.2.5	Yes 15.4.2	High	
	Small caps	Yes 17.3.2.33	Yes 15.4.1	High	
	All lower case	No	Yes 15.4.2	Low	
Text colour					

Functionality	Sub functionality	OOXML	ODF	Transla-tability	Notes
	RGB	Yes 17.3.2.6	Yes 14.7.8	High	
	Background colour	Yes 17.3.2.6	Yes 15.4.37	High	
	Based on theme	Yes 17.15.1.20 17.18.97	No	Medium	ODF has no concept of a *document theme*.
	Blinking text	No	Yes 15.4.36	Low	OOXML supports only blinking backgrounds, but no blinking text.
	Text highlighting	Yes 17.3.2.15	No	Medium	Only a limited range of colours is available for text highlighting.
Complex script support		Yes 17.3.2.7	Yes 15.4.13 15.4.14	Medium	The formats differ in how complex scripts (east-Asian, right-to-left scripts) are supported.
East-Asian text					
	Packing two lines into one	Yes 17.3.2.10	No	Low	
	Brackets around two-lined text	Yes 17.3.2.10 17.18.8	No	Low	In ODF, left and right brackets can be specified independently.
	Vertical text	Yes 17.3.2.10	Yes 15.4.42	Medium	ODF supports rotating text by 0, 90 and 270 deg.; OOXML supports only 0 and 90 deg. rotation.
	Emphasis marks	Yes 17.3.2.12	Yes 15.4.40	Medium	ODF offers more fine-grained support. Marks can be placed above or below text.
Font selection					
	By font name	Yes 17.8	Yes 15.4.13	High	
	By font family	Yes 17.8.3.9	Yes 15.4.14	High	
	Theme fonts	Yes 17.18.96	No	Low	ODF does not support the concept of document themes.
Font effects					
	Emboss	Yes 17.3.2.13	Yes 15.4.26	High	
	Imprint / engrave	Yes 17.3.2.18	Yes 15.4.26	Medium	OOXML has an effect termed *imprint* while ODF offers *engrave*.
	Outline	Yes 17.3.2.23	Yes 15.4.5	High	

Functionality	Sub functionality	OOXML	ODF	Transla-tability	Notes
	Shadow	Yes 17.3.2.31	Yes 9.5.1	Medium	ODF allows for fine-grained control of text-shadow parameters, OOXML only allows turning the shadow on or off.
Manual specification of run/ span width		Yes 17.3.2.14 17.3.2.43	Yes 15.4.41	Medium/ low	OOXML uses absolute values, ODF uses percentages. This may lead to translation problems.
Italic text		Yes 17.3.2.16	Yes 15.4.25	Medium	ODF supports italic and oblique text, OOXML makes no such distinction.
Kerning		Yes 17.3.2.19	Yes 15.4.35	High	
Text language		Yes 17.3.2.20	Yes 15.4.23	High	
Enable/ disable spell checking for run/ span		Yes 17.3.2.21 17.15.1.52	No	Low	ODF does not support this feature.
Raised/ lowered text		Yes 17.3.2.24	Yes 15.4.12	Medium	OOXML uses absolute values, ODF uses percentages. This may lead to translation problems.
Strikethrough		Yes 17.3.2.37 17.3.2.9	Yes 15.4.34	Medium	OOXML allows single and double strikethrough. ODF offers more fine-grained control of strikethrough options and styles.
Underline		Yes 17.3.2.40	Yes 15.4.28	Medium	The note on strikethrough applies equally to text underlining.

6.2.2 Paragraph formatting

In the context of word processing documents, a paragraph is the smallest unit of text upon which layout is performed. Both document formats support applying the text formatting properties given above on a per-paragraph basis. In fact OOXML simply embeds a run-properties element within the paragraph format whereas ODF paragraph styles may contain paragraph and text properties. Examples of XML representations in both formats are given in subclause 7.2.2.

Table 2 — Paragraph formatting

Functionality	Sub functionality	OOXML	ODF	Transla-tability	Notes
Line height					
	Fixed	Yes 17.3.1.33	Yes 15.5.1	High	
	Minimum	Yes 17.3.1.33	Yes 15.5.2	High	
	Line spacing	No	Yes 15.5.3	Low	
	Font-independent line spacing	No	Yes 15.5.4	Low	
	Automatic	Yes 17.3.1.33	No	Low	OOXML provides a (Boolean) option that specifies HTML-like line spacing.
Text alignment (left/ right/ centered/ justified)		Yes 17.3.1.13	Yes 15.5.5	Medium	OOXML supports a range of additional values for Arabic and Thai text.
	For last line in paragraph	No	Yes 15.5.6	Low	
	Justify single word	No	Yes 15.5.7	Low	
Keep paragraph on same page as following paragraph		Yes 17.3.1.15	Yes 15.5.8	High	
Do not split paragraph into multiple pages		Yes 17.3.1.14	Yes 15.5.10 15.5.9 15.5.8	Medium	OOXML only supports keeping a paragraph on a page without specifying the minimum number of lines and the position of the paragraph.
Tab stops		Yes	Yes	High	
	Position	Yes 17.3.1.37	Yes 7.12.6	High	
	Type (left, centre, right, decimal)	Yes 17.3.1.37	Yes 7.12.6	Medium	OOXML does not support specifying the decimal character.
	Type (bar, clear, list)	Yes 17.18.84	No	Low	These tab stop styles are supported in OOXML but their use is discouraged.

Functionality	Sub functionality	OOXML	ODF	Transla-tability	Notes
	Leader properties	Yes 17.18.72	Yes 7.12.6	Medium	The formats support different kinds of leader styles. ODF reuses the same styles which allows for underline and strikethrough. OOXML supports a fixed list of styles.
	Default tab stop	Yes 17.15.1.25	Yes 15.5.12 14.2	High	
Hyphenation					OOXML only allows suppressing automatic hyphenation on a per-paragraph basis.
	Last word on page	Yes 17.15.1.10	Yes 15.4.44	High	
	max. consecutive hyphenated lines	Yes 17.15.1.22	No	Low	
Drop Caps		Yes 17.3.1.11	Yes 15.5.15	Medium	OOXML handles drop caps via specialized text frames. ODF's approach is more straight-forward.
Register truth (same text line distance across multiple pages / columns)		No	Yes 15.2.12	Medium	ODF supports a paragraph style attribute which can specify the reference line distance for all paragraphs. This functionality is not supported directly by OOXML. Fixed width tables in OOXML may be able to compensate for this drawback, however there may be difficulties in translatability.
Margins					
	Absolute, relative	No	Yes	Medium	OOXML only supports absolute values for paragraph margins.
	Left/right/ top/bottom	Yes	Yes	Medium	OOXML supports contextual spacing where top/bottom spacing is ignored for identically formatted paragraphs.
First line indent		Yes	Yes	High	
	Absolute, relative	Yes 17.3.1.12	Yes 15.5.18	Medium	OOXML only supports absolute values for first-line indentation.
	Based on font size	No	Yes 15.5.19	Low	ODF supports an auto-text-indent property specifying that the first line of a paragraph is indented by a value that is based on the current font size.
Page/ column break					
	Before paragraph	Yes 17.3.1.23	Yes 2.8	Medium	OOXML does not support column breaks as paragraph properties.

Functionality	Sub functionality	OOXML	ODF	Transla-tability	Notes
Background colour		Yes 17.3.1.31	Yes 15.5.23	Medium	OOXML allows using theme colour attributes. ODF does not support the concept of a *document theme*.
Background pattern		Yes 17.3.1.31	No	Low	
Background image		No	Yes 15.5.24	No	
	Filter	No	Yes 15.5.24	No	
	Opacity (percent)	No	Yes 15.5.24	No	ODF manipulates the opacity of the background image in the form of a percentage, while in OOXML the background colour (or filled vector graphics) can be influenced indirectly via alpha colour transformations which can be used to modify opacity. Alpha colour transformations are expressed as percentages.
Embedded Images		Yes 15.2.14	Yes 9.3.2	Medium	Bitmaps can be easily translated. However, due to discrepancies between SVG (used by ODF) and DrawingML (used by OOXML), there is a high probability that compatibility issues will arise when vector graphics are to be translated.
Borders		Yes	Yes	High	
	Top/bottom/ left/ right	Yes 17.3.1.24	Yes 15.5.25	High	
	Between/ bar	Yes 17.3.1.24	No	Low	In OOXML a paragraph may have a *bar* (a border on the *inner* side of the paragraph when a book-like layout is used). Additionally a *between* border can be specified for paragraphs with identical border formatting. ODF allows for merging the borders of consecutive, identically formatted paragraphs.
	Colour	Yes 17.3.4	No	Medium	OOXML allows for using theme colour attributes. ODF does not support the concept of a *document theme*.
	Frame effect	Yes 17.3.4	No	Low	
	Shadow effect	Yes 17.3.4	Yes 15.5.28	Medium	ODF offers more fine-grained control of shadow parameters.
	Spacing	Yes 17.3.4	Yes 15.5.27	High	

Functionality	Sub functionality	OOXML	ODF	Transla-tability	Notes
	Width	Yes 17.3.4	Yes 15.5.26	High	
	Type	Yes 17.18.2	Yes 15.5.26	Medium	OOXML documents can specify *art borders*, a concept not supported by ODF. While both document formats support a wide range of border styles, the sets differ. However, common styles (single/ double/ dotted lines) are supported by both formats.
Padding		Yes 17.3.1.11	Yes 15.5.27	High	
Shadow		Yes 17.3.2.31 17.3.1.29	Yes 15.5.28	High	
Line numbering		No	Yes 14.9.1	Low	OOXML only supports line numbering on a per-section level, not as a paragraph setting. Individual paragraphs can be exempted from line numbering.
	(Re-)set start value	No	Yes 15.5.31	Low	
Vertical alignment (top, middle, bottom, baseline)		Yes 17.3.1.39 17.18.91	Yes 15.27.11		
Asian / complex text layout properties					
	Add space between Asian, ctl and Western text	Yes 17.3.1.2	Yes 15.5.32	Medium	OOXML allows for specifying extra spacing between Asian and Roman text as well as Asian Text and Numbers. ODF allows for spacing between Asian, ctl (complex text layout) and Western text (but not numbers).
	Allow punctuation to hang into margin	Yes 17.3.1.21	Yes 15.5.33	High	
	Snap to layout grid	Yes 17.3.2.34	Yes 15.2.21 15.5.38	High	
	Line breaking behaviour (strict / auto)	Yes 17.3.1.16	Yes 15.5.34	Medium	OOXML allows more specific settings (kinsoku).

Functionality	Sub functionality	OOXML	ODF	Transla-tability	Notes
Writing mode (lr/rl/tb)		Yes 17.3.1.6	Yes 15.2.19	Medium	OOXML only supports setting paragraph properties to right-to-left or left-to-right.
Text frames		Yes 17.3.1.11	Yes 9.3	High	
	Suppress overlap	Yes 17.3.1.36	Yes 15.30.5	Medium	In ODF chart text label overlaps may be suppressed. In OOXML this feature is supported with reference to drawing objects. If a text is treated like a drawing object (for example by being grouped with a text) this feature can be used.
Lists		Yes 17.9	Yes 4.3	High	

6.2.3 Header and footer

OOXML and ODF both support the definition of header and footer. While OOXML assigns them to the whole document or to single sections, ODF aligns them with the concept of a master page. OOXML supports multiple content types; ODF supports textual headers and footers. Both International Standards use the terms *header* and *footer* in a slightly different way. To display additional content types than text on the top or bottom of a page, in ODF this content has to be associated with the page instead with the header and footer.

Table 3 — Header and footer

Functionality	Sub functionality	OOXML	ODF	Translata-bility	Notes
Content type		Yes 11.3.6/9	Yes 14.4	Medium	ODF supports text only but other content can be added as part of the master page.
Properties	Separate definitions for right, left, first page	Yes 17.10	Yes 14.4	Medium	ODF allows separate definitions for right and left pages.
Formatting		Yes 17.6.11	Yes 14.3 15.3	Medium	ODF allows formatting headers and footers while OOXML allows formatting pages including headers and footers.

6.2.4 Tables

Both OOXML and ODF support the insertion of tables inside a document. Both formats allow table cells to span across multiple rows and / or columns and provide detailed control over the display of table elements. The table below covers the table features from the use case in subclause 5.2.3 and highlights further areas where functionality varies between the document formats. While OOXML uses the concept of tables only in WordprocessingML, ODF use the same concept for word processing and spreadsheet documents. Therefore the following comparison is also valid concerning the description of ODF spreadsheets. Examples of XML representations in both formats are given in subclause 7.2.4.

Table 4 — Tables

Functionality	Sub functionality	OOXML	ODF	Translata-bility	Notes
Table properties					
	Right-to-left layout	Yes 17.7.6.1	No	Medium	ODF does not support rtl layout for tables. However the functionality can be emulated by reversing the cell order appropriately.
	Alignment of whole table (left, center, right, auto, indented)	Yes 17.4.29	Yes 15.8.2	Medium	ODF has no support for floating tables. However, this functionality may be emulated by placing a table inside a frame.
	Background colour	Yes 17.4.32	Yes 15.8.8	Medium	ODF does not support document themes, so information may be lost in translation.
	Background pattern	Yes 17.4.32	No	Low	
	Background image	Yes 17.2.1	Yes 15.8.8	High	
Data alignment	Horizontal / vertical	Yes 17.3.1.13	Yes 15.11.1	High	OOXML aligns cell data in tables embedded in word processing documents at paragraph level.
Column settings					
	Adjust column width	Yes 17.4.16	Yes 15.9.1	High	
Row settings					
	Adjust row height	Yes 17.4.81	Yes 15.10.1	High	
Cell settings					
	Span multiple columns	Yes 17.4.17	Yes 8.1.3	High	
	Span multiple rows	Yes 17.4.85	Yes 8.1.3	High	OOXML does this via the <*vMerge*> element.

Functionality	Sub functionality	OOXML	ODF	Translata-bility	Notes
Sub tables		No	Yes 8.1.3 8.2.6	Low	ODF supports the concept of sub tables, e.g. tables embedded seamlessly within a table cell. While the same effect may be reproduced by splitting and rejoining cells in the containing table, this would require a translator who could render the complete table internally.
Borders					
	Colour / width / style	Yes 17.4.67	Yes 8.3.3 15.8.12	Medium	Both formats allow the same values as for paragraph borders.
Table headings		No	Yes 8.2.2 8.2.4	Low	OOXML has no way of identifying certain table cells as being part of a table header. It does contain a <*tblHeader*> element; however this specifies that the affected row should be repeated on every page the table spans.

6.2.5 Itemization and numeration

Since ODF and OOXML differ in the way they handle numbering (e.g. of lists or headings), the following two subsections contain a short discussion of each document format's approach. Numbering in this context includes the handling of bulleted (itemized) lists as both document formats handle them the same way as numbered lists. Examples of XML representations in both formats are given in subclause 7.2.5.

6.2.5.1 Numbering in ODF

ODF contains two ways of expressing lists: an approach based on the nesting of the individual XML tags used to define the list (structural approach) and another one in which regular paragraphs are marked as belonging to a list (attribute approach). The numbering and list formatting applied to a list item or heading is determined by a list style associated to the list (or numbered paragraph).

The structural approach is reminiscent of the way lists are constructed in XHTML[11] with specialized tags denoting lists and list items and the list level being determined by the nesting of list tags in the XML representation of the document content. The attribute approach, on the other hand, simply annotates regular paragraphs with attributes identifying them as items of a specific list style at a certain list level. Both approaches are functionally equivalent, however only the attribute approach can be used to apply numbering information to headings.

Unfortunately, the ODF International Standard is worded ambiguously and thus allows for different interpretations of the attribute approach described above. It is unspecified whether the numbering logically resides with the list style or if there is a global counter for each list level which needs to be restarted manually. For example, the XML code in Figure 59 may be rendered as in Figure 60 when the numbering resides with the list style. However, when a global counter is used, the list would show up as in Figure 61.

11 http://www.w3.org/TR/xhtml1/

```
<text:numbered-paragraph text:style-name="L2">
  <text:p>List Item</text:p>
</text:numbered-paragraph>
<text:numbered-paragraph text:style-name="L3">
  <text:p>List Item</text:p>
</text:numbered-paragraph>
<text:numbered-paragraph text:style-name="L2">
  <text:p>List Item</text:p>
</text:numbered-paragraph>
```

Figure 59 — Numeration in ODF - XML

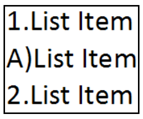

1.List Item
A)List Item
2.List Item

Figure 60 — Numeration in ODF - counter associated with list style

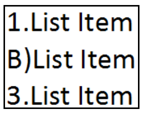

1.List Item
B)List Item
3.List Item

Figure 61 — Numeration in ODF - global counter

6.2.5.2 Numbering in OOXML

OOXML has no distinct concept of lists. Instead, it uses an approach similar to the ODF *attribute* approach explained above. List items (and headings) are simply regular paragraphs to which special properties are attached which contain information about list structure (an identifier for the list the paragraph belongs to and its list level) and a reference to the formatting information for the list. Headings are treated in the same way, except that they contain additional information about the heading's outline level within the document.

A detailed explanation of the concepts used for numbering information in OOXML is contained in Part 1, subclause 17.9 of the OOXML International Standard. Numbering information may be applied to a paragraph in three different ways.

- In the simplest case, the paragraph is annotated with a reference to a *numbering definition* which in turn inherits the actual numbering settings from an abstract numbering definition.
- Alternatively, a numbering style may be applied to the paragraph via one of two distinct yet equivalent approaches. In both cases, the numbering style is not referenced directly; rather, a numbering definition which references the style via its associated abstract numbering definition is applied as shown above.
- The numbering style may also reference a separate numbering definition.

6.2.5.3 Comparison of numbering and enumeration

Both document formats offer a comparable level of support for numbered and/or bulleted lists. OOXML allows for more flexibility when specifying the formatting of nested numbering. To give an example: using individual suffixes, prefixes and separators on each level, in OOXML the third-level heading - 1.2.3 *heading* - looks like:

Section I,2.b) *heading*

ODF allows the specification of one common prefix, suffix, and separator for the whole numbering. Thus using the prefix: "Section ", and the suffix: ")" the example will look like:

Section I.2.b) *heading*

Since both formats offer multiple ways of applying numbering information to text segments, a translation implementation will most likely require fairly complex processing in order to retain the best possible fidelity.

6.2.6 Metadata language entries

Under both platforms, the code is defined by a two or three letter language code taken from the ISO 639 International Standard optionally followed by a hyphen (-) and a two-letter country code taken from the ISO 3166 International Standard.

This is how the default language for a run would be specified under OOXML:

```
<w:lang w:val="fr-CA"/>
```

The language definition is quite similar for ODF. Generally it could be determined that the metadata for language information can be adequately translated from one format to the other.

6.2.7 Indices

Office documents may contain various types of indices, including the table of contents, but also indices of figures, tables, etc. Since the two document formats follow different approaches in the way indices are represented, this section offers an overview of both approaches in subclauses 6.2.7.1 and 6.2.7.2. Examples of XML representations in both formats are given in subclause 7.2.6.

6.2.7.1 Indices in ODF

ODF supports three different types of indices: tables of content, alphabetical indices and user-defined indices. Each index in turn is composed of two parts: an *index template* specifying all the information needed to generate the index and an *index body* containing a rendition of the index, using standard text processing markup.

The information contained within the index template varies according to the index type. The index template specifies the source material for the index, along with an optional title and a template specifying how the title and each index entry should be rendered.

For example, the table of contents described in the use case 5.2.10 is built from the document's headings. Since the index has no title, the template would not specify one. Each entry is built from:

- The entry's title;
- A tab stop;
- The page number of the heading.

ODF has three ways to specify the source material for the table of contents:

- **Text outline**: the document structure, i.e. the headings and their associated outline level are used to generate the table of contents.
- **Index marks**: this approach only indexes paragraphs and headings which are explicitly marked with an index mark.
- **Styles**: the index is built from paragraphs to which certain text formatting styles are applied.

6.2.7.2 Indices in OOXML

In OOXML, the concepts of tables of content and indices are implemented as dynamic content fields. Thus, a table of content is represented by a TOC field and its presentation and source material are specified by the field switches.

The source material may be based on the following:

- Paragraph-outline level; this approach corresponds to ODF's approach to using the document structure.
- Index marks (implemented via TC fields in OOXML) or bookmarks;
- Styles; this approach is similar to the third approach offered by ODF.
- A sequence; commonly used for lists of figures, tables, etc.

6.2.7.3 Summary

Although the two document formats differ in their approaches to the generation of tables of contents and indices, they do offer comparable levels of support for these features. Implementations will have to take into account the different models, which causes some complexity, especially when documents combine many of the approaches outlined above.

6.2.8 Change tracking and collaborative functions

Both document formats offer support for change tracking and textual annotations in word processing documents. In addition to the common operations, OOXML allows highlighting text regions with a limited set of colours (for more information, see subclause 6.2.1). ODF’s change tracking support is more coarse-grained than that of OOXML in that formatting changes, including those in tables, are recorded but no information about the previous state is kept so that the previous state cannot be reconstructed by rejecting the changes. Examples of XML representations in both formats are given in subclause 7.2.7.

Table 5 — Annotations

Functionality	Sub functionality	OOXML	ODF	Translata-bility	Notes
Text insertion		Yes 17.13.5	Yes 4.6.3	Medium	Change tracking in lists may cause problems in ODF.
Text deletion		Yes 17.13.5	Yes 4.6.4	Medium	Change tracking in lists may cause problems in ODF.
Formatting changes		Yes 17.13.5	Yes 4.6.5	Medium	ODF only records the fact that a change has occurred. However, no further information is recorded, so that it is impossible to reconstruct the previous state.
Comments		Yes 17.13.4	No	Medium	OOXML allows adding comments to arbitrary text ranges. This is not supported by ODF, however similar functionality may be provided by inserting notes (associated with a point in the text, not a range).

Functionality	Sub functionality	OOXML	ODF	Translata-bility	Notes
Text highlighting		Yes 17.3.2.15	No	Medium	Although ODF does not support text highlighting, the functionality may be emulated by setting the text background colour (see the section on text formatting).
Metadata					
	Name	Yes 17.13	Yes 3.1.6	High	
	Date / Time	Yes 17.13	Yes 3.1.9	High	
	Author shorthand for comments	Yes 17.13 17.13.4	Yes 12.3 8.3.3	High	

6.2.9 Bibliographies and optional document parts

Both the ODF and OOXML formats support bibliographies. ODF introduces a <*bibliography-mark*> element that contains the text and information for a bibliography index entry. This entry supports attributes for several types of bibliographical data that a bibliography index may contain. Some attributes are user defined. The entries may be stored in the document or in an external database. They are stored as *bibliography-marks* within the document and contain all attribute values that have been defined.

OOXML uses the *customXML* feature to implement bibliographies. Thus there are no hard coded attributes used to describe an entry. Instead the entry's properties can be defined outside the standard.

From these approaches it is obvious that no generic mapping between ODF and OOXML bibliographies exists. On the other hand an ODF implementation could map its entries to OOXML by introducing a corresponding XML schema as custom XML. An OOXML application can map its entries to ODF in case a semantic mapping exists which will be obvious in most cases.

Both International Standards use different approaches when it comes to optional document parts. ODF supports the concept of *hidden* and *conditional sections*. This property of a section is defined by corresponding text section attributes.

OOXML introduces the term *glossary document*. Within a WordprocessingML file, the glossary document is a supplemental storage location for additional document content which should travel with the document, but which should not be displayed or printed as part of the main document until it is explicitly added to that document by a deliberate action. Glossary document parts can contain any block level WordprocessingML element. Title pages are typical parts of a glossary document in OOXML.

Again both document formats support optional text but use totally different concepts. Generic mapping seems to be impossible, although some word processing applications may be able to provide such mapping in a restricted context.

6.3 Spreadsheet documents

6.3.1 Introduction

This section describes the properties which may be applied to the elements of spreadsheet documents. For the purposes of this paper, the properties to be examined have been narrowed down to formatting and calculation functions and those in any way related to such. Examples of XML representations in both formats are given in subclause 7.3.

ODF spreadsheets have tables as root elements. Tables in turn contain rows. Rows are divided into cells by columns. ODF does not differentiate between tables embedded in word processing documents and those which make up spreadsheets. Essentially the same XML structures, nodes and attributes are used in both cases. The only difference is the <*spreadsheet*> element used within the <*body*> element as against the <*text*> element used in word processing documents.

In a similar vein, OOXML has <*workbook*>*s* as root elements. Workbooks contain <*worksheets*>. These sheets are further divided into a grid of <*cells*>.

6.3.2 Formatting

The cell is the most elementary unit of a spreadsheet to which properties can be applied. Rows, columns and tables (ODF) or worksheets (OOXML) can also be manipulated

The following table summarizes the features pertaining to formatting for the use cases covered. For more information, see subclause 6.2.1.

Table 6 — Spreadsheet formatting

Functionality	Sub functionality	OOXML	ODF	Translata-bility	Notes
Row fixing		Yes 18.3.1.66	Yes ---	Low	This functionality can be applied in ODF only by manipulating the horizontal/vertical @*Split Mode* and @*Split Position* attributes via the settings.xml file. This file is undefined and application specific.
Cell / Row background Shading		Yes 17.4.33	Yes 15.11.6 15.10.3	High	
Coloured text in a single cell		Yes 18.3.1.53 18.4.7	Yes 14.7.7 15.4.3	High	
Highlighted colour frame on single row		Yes 18.8.5	Yes 15.5.25	High	
Date formatting		Yes 18.17.4	Yes 6.7.7	High	
Graphic cell content					

Functionality	Sub functionality	OOXML	ODF	Translata-bility	Notes
	Linked	Yes 21.2.2.63	Yes 9.3.2	High	
	Embedded	Yes 21.2.2.63	Yes 9.3.2	Medium	When using embedded images, the use of vector graphics could prove problematic due to the different vector graphic formats used by ODF and OOXML.
Spreadsheet-Embedding in other applications		Yes 18.3.1.60	Yes 9.3.7	Medium	A few problems could arise due to the use, by OOXML, of VML- which is not supported by ODF- in certain areas.

6.3.3 Calculation

OOXML and ODF calculations are performed by equations also known as *formulas*.

In OOXML named formulas are known as *functions*. Formulas are represented by the text of the formula and the text version of the last computed value for that formula. The return value of a function is specified within the @*t*-attribute of the cell containing the formula.

The ODF spreadsheet document content model contains a spreadsheet calculation setting for formulas. The presentation of the value of a variable is set using a <*variable-set*> variable setter element in which the attribute @*formula* contains the formula to compute the value of the variable field. Settings pertaining to the calculation of formulas are set via the <*calculation-settings*> element. The @*formula* attribute generally contains the formula for a table cell.

This section describes the translation of functionality provided by the properties used in applying formulas to cells as well as their behaviour and underlying logic operations, as used in the use case example in subclause 5.3.3.

Table 7 — Spreadsheet calculation

Functionality	Sub functionality	OOXML	ODF	Translata-bility	Notes
Assigning formulas/ functions to a cell		Yes 18.3.1.40	Yes 8.1.3	High	
Manual/ automatic calc. mode		Yes 18.18.4	No	Low	In OOXML the formulas can be executed whenever a cell value changes or when a user initiates an action. Can be configured in ODF: *config:name='AutoCalculate'*
Shared formulas		Yes 18.3.1.40	No	Low	In OOXML primary/ shared formulas are used to cut down redundancy where a formula is used more than once. This functionality is not present in ODF although OOXML formulas can be translated to ODF with some effort.
Externally referenced formulas		Yes 18.14 18.14.1 18.18.11	No	Medium	In ODF cells but not formulas can be referenced. OOXML allows the direct referencing of both.
Caching of externally referenced workbook		Yes 18.10.1.95 18.14.7	Yes 8.3.2	Medium	External workbooks cannot be referenced in ODF but sheets of external workbooks can be referenced.
Defined names in place of cell references in formulas		Yes 18.17.2.5	No	Low	Names to be used in place of references or formulas do not exist in ODF.
Auto filtering		Yes 18.3.2	Yes 8.7	Medium	Both formats support filter criteria for table rows based on specific properties of table cells and rows.

Both formats, OOXML and ODF support several mathematical and statistical inline functions. Several of these functions are identical; others are only defined in one of the two International Standards. The following table shows, which functions are only defined in one of the International Standards and which of those functions can be mapped to the other International Standard using mathematical or other transformations.[12]

[12] ODF 1.0 does not define any formula language. For this reason the list of ODF functions has been generated by OpenOffice.org, the quasi reference implementation of ODF 1.0. A complete list of the supported mathematical functions including several functions that are not supported by OpenOffice.org is introduced in ODF 1.2. For this reason the translatability between math functions in ODF 1.2 and OOXML will be well defined and better than the translatability shown in the table below.

Table 8 — Math functions in OOXML and ODF

OOXML function	ODF function	Remarks
Can be converted	ACOT	Can be converted to OXML according to the following example: ACOT(B12) = ACOS(B12/SQRT(1+B12^2))
Can be converted	ACOTH	Can be converted to OXML according to the following example: ACOTH(B12)=LN((B12+1)/(B12-1))/2
AVERAGEIF	Not supported	OOXML only
AVERAGEIFS	Not supported	OOXML only
Not supported	B	ODF only
Not supported	BASE	ODF only
Not supported	BESSELI	ODF only
Can be converted	COMBINA	Can be converted to OXML according to the following example: COMBINA(A3;A4)=COMBIN(A3+A4-1;A4)
Not supported	CONVERT	ODF only
CONVERT	CONVERT_ADD	Name changed Note: unit conversion function in OOXML is named CONVERT, while in ODF it is named CONVERT_ADD. It is important not to confuse it with ODF CONVERT, which converts European currencies.
Can be converted	COT	Can be converted to OXML according to the following example: COT(A1) = COS(A1)/SIN(A1)
Can be converted	COTH	Can be converted to OXML according to the following example: COTH(A1) = COSH(A1)/SINH(A1)
COUNTIFS	Not supported	OOXML only
CUBEKPIMEMBER	Not supported	OOXML only
CUBEMEMBER	Not supported	OOXML only
CUBEMEMBERPROPERTY	Not supported	OOXML only
CUBERANKEDMEMBER	Not supported	OOXML only
CUBESET	Not supported	OOXML only
CUBESETCOUNT	Not supported	OOXML only
CUBEVALUE	Not supported	OOXML only
Not supported	CUMIPMT	ODF only
CUMIPMT	CUMIPMT_ADD	Name changed
Not supported	CUMPRINC	ODF only
CUMPRINC	CUMPRINC_ADD	Name changed
Not supported	CURRENT	ODF only

OOXML function	ODF function	Remarks
Not supported	DAYS	ODF only
Not supported	DAYSINMONTH	ODF only
Not supported	DAYSINYEAR	ODF only
Not supported	DDE	ODF only
Not supported	DECIMAL	ODF only
Not supported	DURATION	ODF only
DVAR	DVAR	Name changed
Not supported	EASTERSUNDAY	ODF only
EFFECT	EFFECT_ADD	Name changed
Not supported	EFFECTIVE	ODF only
Not supported	FORMULA	ODF only
Not supported	GAUSS	ODF only
Not supported	GCD_ADD	ODF only
GETPIVOTDATA	Not supported	OOXML only
IFERROR	Can be converted	Can be converted to ODF according to the following example: IFERROR(A2;B2)=IF(ISERROR(A2);B2;A2)
Not supported	ISEVEN	ODF only
ISEVEN	ISEVEN_ADD	Name changed
Not supported	ISFORMULA	ODF only
Not supported	ISLEAPYEAR	ODF only
Not supported	ISODD	ODF only
ISODD	ISODD_ADD	Name changed
Not supported	LCM	ODF only
LCM	LCM_ADD	Name changed
MIDB	Not supported	OOXML only
Not supported	MONTHS	ODF only
Not supported	MUNIT	ODF only
Not supported	NOMINAL	ODF only
NOMINAL	NOMINAL_ADD	Name changed
Not supported	PERMUTATIONA	ODF only
Not supported	PHI	ODF only
Not supported	ROT13	ODF only
Not supported	SHEET	ODF only
Not supported	SHEETS	ODF only
Not supported	STYLE	ODF only
SUMIFS	Not supported	OOXML only

OOXML function	ODF function	Remarks
Not supported	WEEKNUM	ODF only
WEEKNUM	WEEKNUM_ADD	Name changed
Not supported	WEEKS	ODF only
Not supported	WEEKSINYEAR	ODF only
Not supported	YEARS	ODF only
ZTEST	Can be converted	Can be converted to ODF according to the following example: ZTEST(A1;n;sigma)= IF(ZTEST(A1;n;sigma)>0.5; 2*(1ZTEST(A1;n;sigma));2*ZTEST(A1;n;sigma))

6.3.4 Additional properties

This table contains an extended list relating to the analysis of the translatability of selected functionalities for spreadsheet documents.

Table 9 — Additional spreadsheet functionality

Functionality	Sub functionality	OOXML	ODF	Translata-bility	Notes
Width adjustment		Yes 18.3.1.13	Yes 8.1.1 15.7.4	Medium	In ODF columns must have fixed width; relative width is only an option, specified as a percentage.
Alignment		Yes 18.8.1	Yes 15.11.1 8.1.3	Medium	In ODF L, R, C, margins exist. Additionally, OOXML offers header and footer margins.
Page number		Yes 18.8.1 18.18.88	Yes 15.11.1 8.1.3	High	
Table or worksheet background/ image		Yes 18.8.1 18.18.40	No	Low	
Shadow		Yes 18.8.1	Yes 15.11.12 15.11.13	High	
Vertical alignment		Yes 13.3.3	Yes 6.2.3 15.2.2	High	
Shadow		Yes 18.3.1.67	Yes 15.5.24	High	OOXML (SpreadsheetML) applications are not required to render according to the *shadow* flag.
Cell border		Yes 18.8.36	Yes 15.2.9	High	

Functionality	Sub functionality	OOXML	ODF	Translata-bility	Notes
Rotation angle/align		Yes 18.8.4	Yes 15.11.7 8.1.3	High	
Cell protect		Yes 18.8.33	Yes 15.11.14 8.1.3	Medium	In OOXML cell protection does not take effect unless the sheet has been protected.

6.4 Presentation documents

6.4.1 Introduction

ODF and OOXML use different approaches to define presentation documents. In ODF, presentation documents are composed of a set of <*page*> elements within an <*presentation*> element. A <*page*> element acts as a container for content.

OOXML presentation documents are based on PresentationML, a framework loosely based on SMIL, in which all definitions are stored as a schema (XSD) which can be one of either structural or presentation level data types.

Examples of XML representations in both formats are given in subclause 7.4.

6.4.2 Slides

6.4.2.1 OOXML slides

In OOXML, the transition from one slide to another is performed via animation effects that are displayed in between slides. Slides, layouts and notes can be defined via *masters*. These master layout components can be overridden individually by specifying local attribute values within each presentation slide.

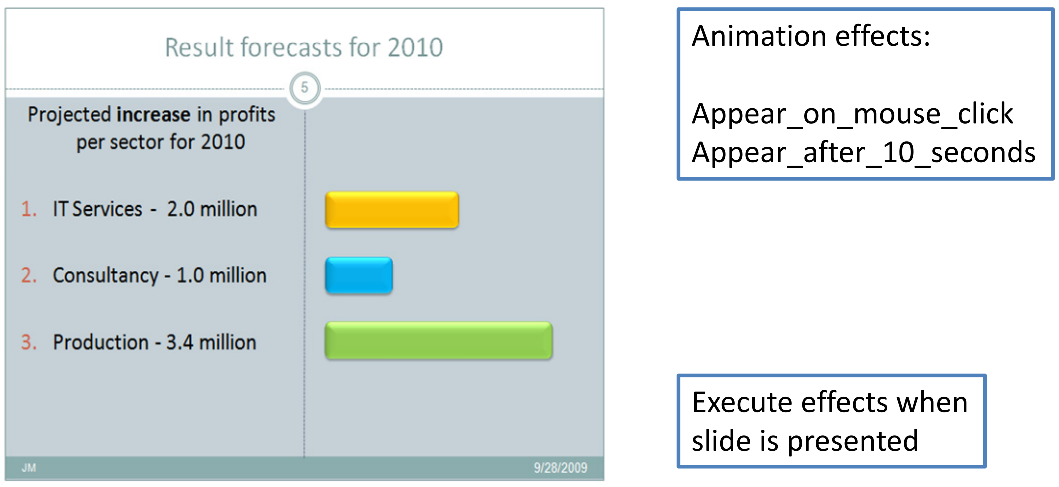
Hierarchy and inheritance are central to the concept of slides in OOXML.

6.4.2.2 ODF slides

ODF animation effects are carried out on so called presentation shapes (these are differentiated from drawing shapes by the @*class* attribute).

It is possible to specify multiple effects for each shape within a page. However this could be hampered by the application on which the presentation is running which can in some cases restrict the extent to which this feature can be utilized.

Several effects can also be initiated at the same time via animation groups:



Result forecasts for 2010

5

Projected **increase** in profits per sector for 2010

1. IT Services - 2.0 million
2. Consultancy - 1.0 million
3. Production - 3.4 million

JM 9/28/2009

Figure 62 — Animation effects

As an alternative, the animations in ODF presentation documents can be manipulated using the XML based SMIL language on which the OOXML PresentationML schema is loosely based.

6.4.3 Text formatting

This section contains properties that may be applied to text in presentation documents based on the use cases in subclause 5.4. Text formatting in presentation documents is similar to text formatting in word processing and spreadsheet documents. Examples of XML representations in both formats are given in subclause 7.4.2.

Table 10 — Text formatting

Functionality	Sub functionality	OOXML	ODF	Translata-bility	Notes
Bold type		Yes 19.2.1.1	Yes 14.6.3 15.4.32	Medium	In addition to bold, ODF allows font weight to be specified numerically (100-900).
Listing and itemization		Yes 21.1.2.4.1 (19.3.1.5, 19.3.1.35 19.3.1.52)	Yes 7.1	Medium	Since both formats offer multiple ways of applying numbering information to text segments, an implementation will most likely require fairly complex processing in order to retain the best possible graphical fidelity.
Text animation		Yes 19.5 M.3.4.7	Yes 15.15	Medium	ODF: setting attributes via <*frames*> controlling style or SMIL. OOXML: build animations can be applied.
Text language		Yes 21.1.2.3.9	Yes 15.4.23	High	

6.4.4 Master layout

ODF makes use of master pages for creating slides. A *master page* is actually a reference to a specific page layout which is used as a base template when beginning to develop a presentation. This template specifies properties common to each page, such as size, content, headers, and footers, which are displayed on every page in a presentation. ODF specifies that all documents must contain at least one master page element.

OOXML follows a similar principle. In Microsoft Office 2007/2010 these layout templates are known as slide masters. Slide layouts can override definitions that were pre-set by masters, and can be applied additionally to individual Office presentation slides. This makes for more flexibility - with regard to master layouts - while using OOXML.

Examples of XML representations in both formats are given in subclause 7.4.3.

The following table compares the functionality based on the use cases in subclause 5.4 dealing with presentation documents.

Table 11 — Master layout

Functionality	Sub functionality	OOXML	ODF	Translata-bility	Notes
Layout and positioning		Yes 19.7.15	Yes 14.15	High	
Animations		Yes 19.5.1 M3.4 (SMIL)	Yes 9.7 9.8 (SMIL)	Medium	OOXML animations can be applied in a greater number of ways than ODF specified ones. This provides for more granularities in creating slide animations.
	Specialized path descriptions	Yes 19.5.4	No	Low	OOXML allows for animation via motion descriptions over polyline or Bezier paths. ODF does not support this.
	Timeline functionality (using time nodes)	Yes 19.3.1.48 19.5.87	No	Low	In addition to inheritance from, or overriding of, master-layouts: OOXML makes use of the concept of time-lines to orchestrate its animations. ODF does not support the concept of timelines.
Slide synchroni-zation		Yes 19.6	No	Low	An update function used by OOXML for synchronizing slides being loaded from SharePoint servers. ODF documents can at most load texts stored in a SQL database if an appropriate driver has been installed.
Applying sounds to slides		Yes 19.5.69	Yes 9.7.1	High	
Diagrams		Yes 20.1.2.2.1	Yes 9.7.2	High	

Functionality	Sub functionality	OOXML	ODF	Translata-bility	Notes
Slide blending and effects		Yes 19.3.1.50	Yes 9.7 9.8.1 15.36.2	Medium	In ODF specification of multiple effects could become problematic since the application on which the presentation is being run can in some cases restrict the extent to which this feature can be utilized. The restriction varies from application to application.
Multimedia content		Yes 19.3.1.33	Yes 9.8 13. 15.36.10	Medium	In OOXML media can be orchestrated to play in sync with a slides timeline. If the media supplying the sound for instance is a CD other attributes such as track indexes or the start or end track can be specified.
Vector graphics		Yes 20.1 M.5	Yes 9.2.6 14.14.2	Low	Due to the use of different graphic engines, the vector graphics are not translatable. However both ODF and OOXML individually support the representation of vector graphics.
Master layout		Yes 19.2.1.36	Yes 14.4	High	

6.5 Common aspects

This section covers functionalities spanning multiple document types.

6.5.1 Alternative presentations

Metadata, such as alternative text representations for non-text entities within a document, play an important role not only in granting people with disabilities better access to document content, but also in improving the automated extraction and processing of information contained within a document.

The following table gives a brief comparison of alternative presentations supported by ODF and OOXML.

Table 12 — Alternative presentations

Functionality	Sub functionality	OOXML	ODF	Translata-bility	Notes
Alternative text					
	Images	Yes 18.3.1.56 17.3.3.19	Yes 9.3.9	High	
	Image maps	No	Yes 9.3.11	Low	OOXML does not support image maps.

Functionality	Sub functionality	OOXML	ODF	Translata-bility	Notes
	Lines / arrows	Yes 18.3.1.56 17.3.3.19	Yes 9.3.9	High	
	Auto shapes	Yes 18.3.1.56 17.3.3.19	Yes 9.3	High	
	Grouped objects	Yes 18.3.1.56 17.3.3.19	Yes 9.3.9	High	
	Sounds	Yes 18.3.1.56 17.3.3.19	No	Low	
	Videos	Yes 18.3.1.56 17.3.3.19	No	Low	
	Charts	Yes 18.3.1.56 17.3.3.19	Yes 9.3	High	
	Text-box, titles, captions	Yes 18.3.1.56 17.3.3.19	Yes 9.3.9	High	
	Links	Yes 18.3.1.56 17.3.3.19	Yes 9.3.10	High	

6.5.2 Colour models

OOXML refers to http://www.w3.org/Graphics/Color/sRGB as a normative standard defining sRGB as the primary colour model supported by the International Standard. Following part 1 (20.1.2.3) of the International Standard, colours in the RGB space can be defined in three different ways:

- Using byte RGB values in the interval [0..255] (sRGB);
- Using real (%) RGB values in the interval [0..1] (scRGB);
- Using HSL values in the (hue x saturation x lightness) colour space with h in [0..360*6000] and s , l in [0..1];

The RGB colour model is an additive colour model in which red, green, and blue light is added together in various ways to reproduce a broad array of colours. It is important to notice that different RGB colour models such as sRGB, Adobe RGB, or scRGB (16 bit) exist. Even if a colour has the same 8 bit RGB-representation it can look different in different colour models. Thus the RGB values are not sufficient to specify a colour unambiguously.

HSL describes colours as points in a cylinder whose central axis ranges from black at the bottom to white at the top, with neutral colours in between. The angle around the axis corresponds to *hue*, the distance from the axis corresponds to *saturation*, and the distance along the axis corresponds to *lightness*. HSL can be mapped to RGB using simple transformations.

ODF does not reference any specific colour model (15.17), thus it is up to the implementation to decide which 8 bit RGB model to use. For this reason translatability between OOXML and ODF and even between different ODF implementation may be limited.

OOXML uses the concepts of *themes* (M.4.3.2) and *accents* to specify a family of related colours to be used within one package. A *package theme* contains a *colour scheme* that itself is a set of colours. The colour scheme is responsible for defining a list of twelve colours. The twelve colours consist of six *accent colours*, two dark colours, two light colours, and a colour for a hyperlink and another for a followed hyperlink. The colours defined in an OOXML colour scheme can be mapped to RGB equivalents in ODF. Thus the colour of every OOXML entity can be mapped to an equivalent colour of the corresponding ODF entity. Due to the indirect definition of schemes, reverse mapping from ODF colours to OOXML colour schemes is not possible.

6.5.3 Custom XML parts

Custom parts of documents contain arbitrary XML markup not necessarily defined by the document's standard itself. OOXML (Subclause 22.5) allows arbitrary XML instances to be stored in a document, and the nodes of a particular XML instance may be bound to form controls (content controls). ODF does not support arbitrary custom XML parts, so these would be lost in a round trip to ODF.

6.5.4 Packages

A *package* is an aggregation of document parts or other types of content. It provides a convenient way to store and distribute documents. ODF and OOXML use specific concepts to aggregate the document parts in a package. There is no need for an explicit translation from an ODF package to an OOXML package or vice versa. Instead the target package format will be generated implicitly during the translation process.

6.5.4.1 ODF packages

ODF supports two ways of document representation:

- A single XML document;
- A collection of several *sub documents* within a *package*. Each sub document stores a part of the complete document. ODF supports text documents, drawing documents, presentation documents, spreadsheet documents, chart documents and image documents. At least the four XML-files meta.xml, setting.xml, style.xml and content.xml are combined in a package. The document body in content.xml contains an element indicating the type the document. ODF uses the ZIP file format specification from PKWARE[13].

ISO/IEC 26300:2006 does not support digital signatures. However, the ODF 1.2 Committee Specification from OASIS supports the digital signature specification from W3C.

The manifest file *manifest.xml* describing the content of an ODF package contains:

- The package relationships;
- Information about the files contained in the package list;
- The media type of each file;
- Information about encryption and decryption in the package.

[13] http://www.pkware.com/products/enterprise/white_papers/appnote.txt, PKWARE Inc., 2004.

The files in an ODF package have explicit relationships as shown in Figure 63.

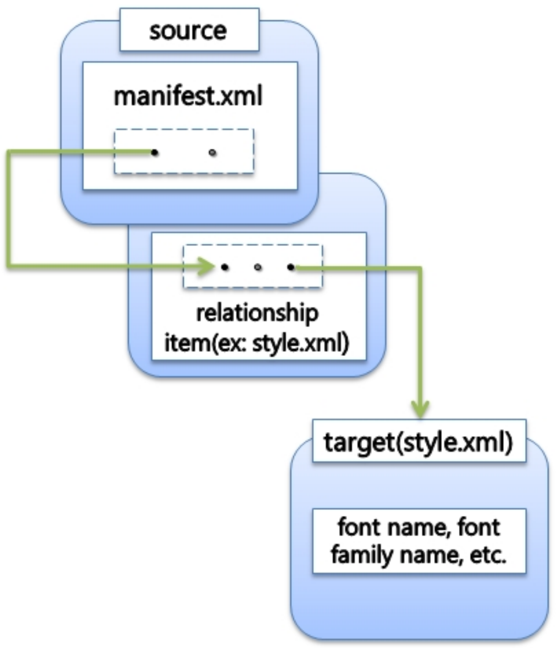

source
manifest.xml
relationship item(ex: style.xml)
target(style.xml)
font name, font family name, etc.

Figure 63 — Explicit relationships in ODF packages

6.5.4.2 OOXML packages

In OOXML, a document is represented as a set of related *parts* that are stored in a container called *package*. OOXML packages can contain word processing, spreadsheet and presentation documents together with other referenced content. Different document types are stored in different physical files/parts of the package.

OOXML uses the same ZIP format as ODF. Encryption and decryption of OOXML packages is vendors specific. OOXML uses the digital signature specification from W3C and some additional package specific digital signatures.

The files in an OOXML package have either explicit or implicit relationships stored in _*rels*-files as shown in Figure 64. In an explicit relation the relationship item contains information about (a link to) the referenced item. In an implicit relation the relationship item contains information about (a link to) a container storing the referenced item.

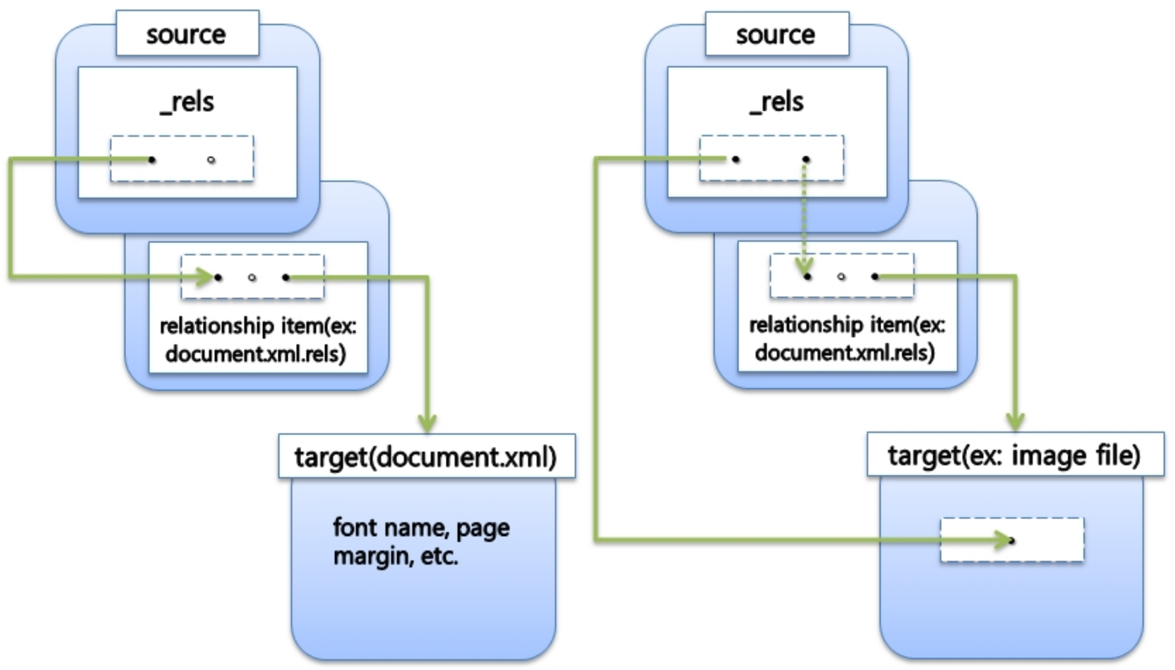

source
_rels
relationship item(ex: document.xml.rels)
target(document.xml)
font name, page margin, etc.
source
_rels
relationship item(ex: document.xml.rels)
target(ex: image file)

Figure 64 — Explicit and implicit relationships in OOXML packages

7 Representation and XML structure

7.1 Introduction

This subclause describes the implementation of selected features that have been used in the use cases in clause 5. The descriptions refer to clause 6 for a detailed elaboration of differences between the implementations of the associated functionalities in ODF and OOXML.

In this subclause the features of word processing, spreadsheet and presentation documents are discussed separately. For each type of document, the discussion focuses on:

- *Logical structures*; here we describe how a document is composed of smaller parts. The representation and XML structure of a document are explained.
- *Features*; here we explain the representation and XML structure of selected features that have been used in clauses 5 and 6.

Some figures are generated by the XML editor Oxygen utilizing the schema definitions of both document formats. ODF's RelaxNG schema definitions have been converted to XSD schema definitions before the figures have been generated. These figures are used to describe the XML structure of a document and its features. In the figures ☐ indicates elements, ▪▪ indicates groups, and @ indicates attributes. Additional figures show sample XML code of the use cases introduced in clause 5 to illustrate the implementation of specific features.

7.2 Word processing documents

7.2.1 Logical structure

7.2.1.1 Word processing documents in ODF

ODF denotes word processing documents as *text documents*. A text document contains a *prelude*, main document *content*, and an *epilogue*:

- The *prelude* contains the document's form data, change tracking information, and variable declarations.

- The document's *main content* contains zero or more text content groups *(*represented as *<text-content>)* and a single page sequence. The XML structure of the document's main content is depicted in Figure 65, where all possible elements in a text content group are listed. Text content can be a choice among XML elements such as
    - paragraph (and heading),
    - text section (and index),
    - table
    - list, and
    - graphical shape.

  As well as such normal content, text content can also include some extensional content. For example, a table of contents (<*table-of-content*>) provides the user with a guide through the content of the document. Change marks (<*change-marks*>) are used to mark the changed regions. A text *section* (<*section*>) is a named region of paragraph level text content. Sections start and end on paragraph boundaries and can contain any number of paragraphs

- The *epilogue* contains elements that implement enhanced table features.

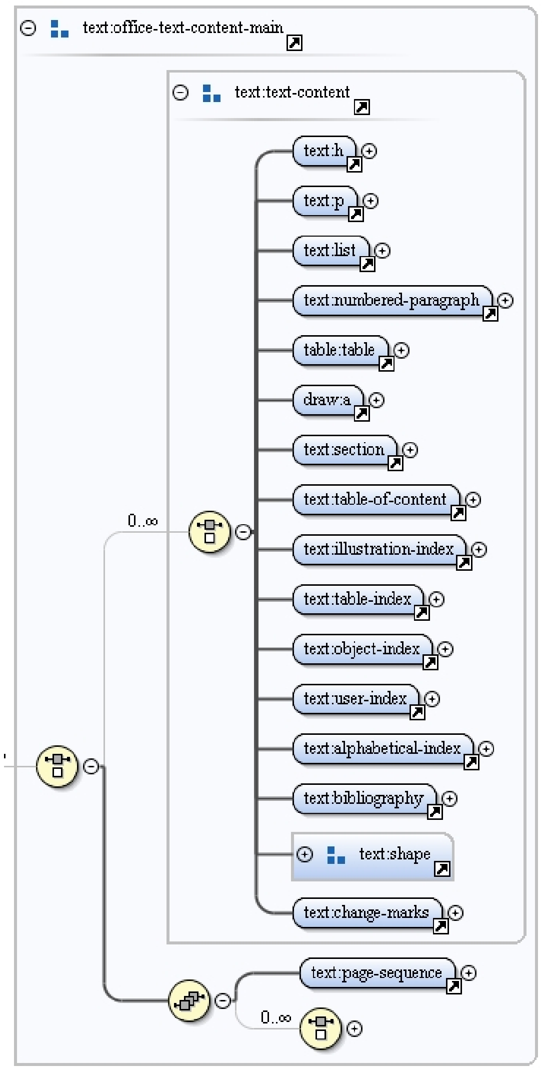


Figure 65 — XML structure of the main content in ODF

7.2.1.2 Word processing documents in OOXML

In OOXML a document describes the graphic *background* and the *attributes* of a document and the *document body*. The document body is a sequence of zero or more *sections* that are composed of *block level elements*, followed by *section properties* (<*sectPr*>). Its XML structure is depicted in Figure 66. A block level element is a choice of elements such as paragraphs, tables or run level elements. External content can be imported into the main document by one or more <*altChunk*> elements. A document can also contain *structured document tags* (<*sdt*>) and *custom markup* (<*customXML*>), which apply user-defined semantics to arbitrary document content. Detailed information about sections is given in subclause 7.2.8.2.

7.2.1.3 Summary

Many concepts in ODF and OOXML are very similar such as paragraphs, tables, and sections, etc. Other concepts are defined and implemented in different ways in the two formats.

- There are more types of block level elements in ODF than in OOXML. For example, paragraphs (<*p*>) and headings (<*h*>) are represented by different elements in ODF. In OOXML both structures are represented by the same element <*p*>. ODF has list elements (<*list*>), but OOXML does not have these types of elements.

- The concept of a *section* is introduced in both formats but the details are defined differently in ODF and OOXML. More information about sections is given in subclause 7.2.8.

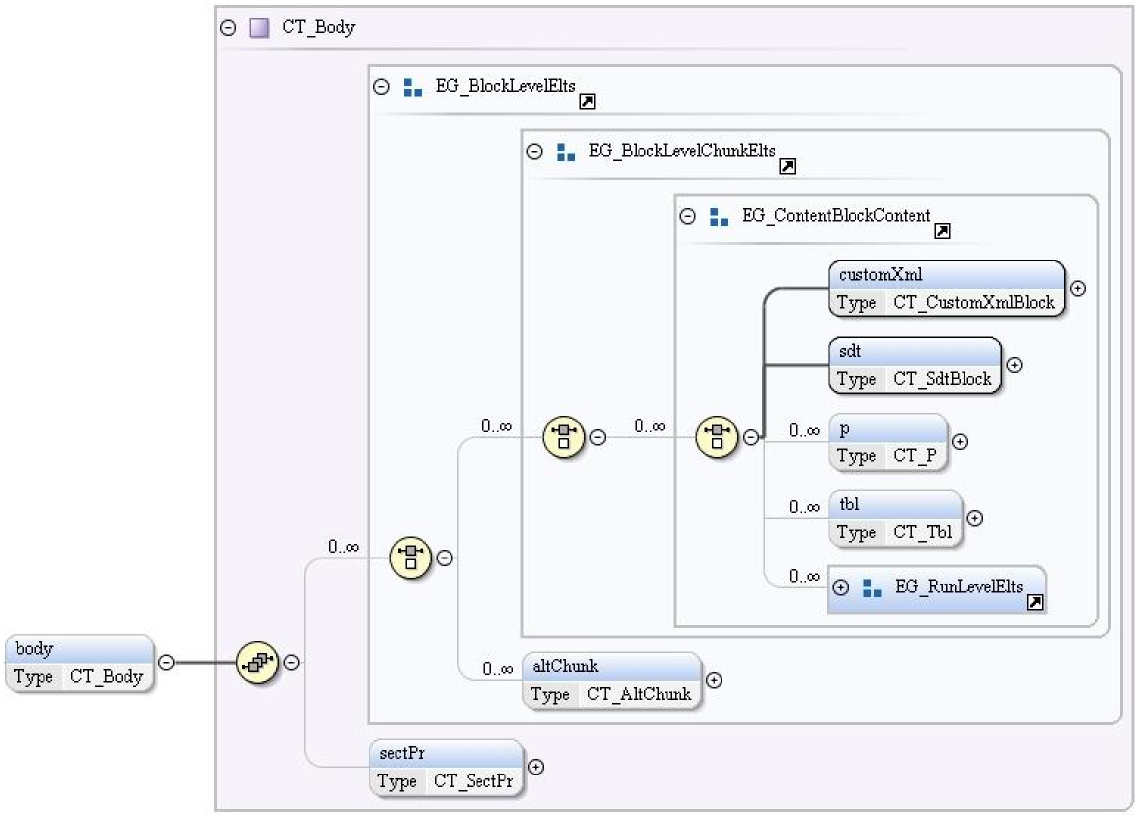


Figure 66 — XML structure of word processing documents in OOXML

7.2.2 Paragraphs

7.2.2.1 Paragraphs in ODF

The paragraph element <*p*> in ODF consists of zero or more <*paragraph-content*> elements. A typical paragraph content is text contained in span elements <*span*>, which represents portions of text that are attributed using a certain text style or class. Spans can be nested. With the exception of spans, a <*paragraph-content*> element can be a choice of nearly 100 different types of elements in a flat structure. For example:

- *Field elements* display information about the current document or about a specific part of the current document, such as the author, the current page number, or the document creation date. These fields are collectively referred to as *document fields*. The group of document fields includes: date and time fields, page number fields, sender and author fields, chapter fields, file name fields and document template fields.
- <*a*> elements represent hyperlinks in documents.
- <*ruby*> elements represent Ruby texts that are usually displayed above or below the main text.
- <*change-marks*> elements record information of changes.

7.2.2.2 Paragraphs in OOXML

In OOXML a paragraph element <*p*> defines a distinct division of content that begins on a new line. The contents of a paragraph consist of a combination of the following types of content:

- *Paragraph properties*; all rich formatting elements at the paragraph level are stored within the <*pPr*> element. Some examples of paragraph properties are alignment, border, hyphenation override, indentation, line spacing, shading, text direction, and widow/orphan control.
- *External references* such as hyperlinks and sub documents
- *Run-level content* such as runs, mathematical content, smart tags and custom markup. The contents of a run consist of run properties <*rPr*> together with a choice of run content such as text, graphics (drawing), internal references or embedded objects.

7.2.2.3 Summary

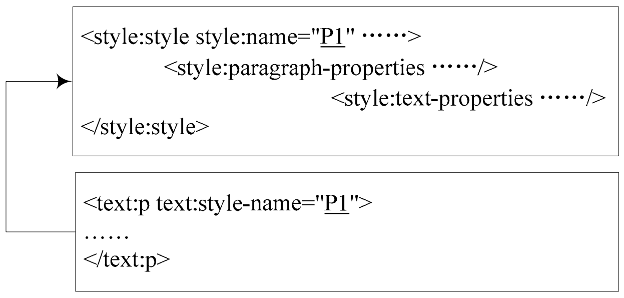
Paragraphs are represented by an element <*p*> in ODF and OOXML. But the XML structures of <*p*> in ODF and OOXML are not similar. ODF supports more types of elements in a paragraph than OOXML does. The element <*span*> in ODF corresponds to the run <*r*> element in OOXML.

7.2.3 Styles

Many objects in a document have formatting properties. Subclause 5.2.2 introduces a related use case. In the two International Standards, formatting information is used in different ways.

7.2.3.1 Styles in ODF

In ODF formatting properties are stored within *styles*. They exist as independent entities which are referenced by name. As shown in Figure 67, a paragraph style named “P1” defined in <*style*> is referenced by the @*style-name* attribute of the <*p*> element.



```
<style:style style:name="P1" ……>
        <style:paragraph-properties ……/>
                        <style:text-properties ……/>
</style:style>
```

```
<text:p text:style-name="P1">
……
</text:p>
```

Figure 67 — Styles in ODF

In ODF each style belongs to a kind of style family. The family is specified by the attribute @*family* of the element <*style*> and refers to specific elements such as paragraph, text, section, table, table-column, table-row, table-cell, table-page, chart, default, drawing-page, graphic, presentation, control and ruby.

7.2.3.2 Styles in OOXML

In OOXML formatting properties are associated with elements. For example the formatting information of a paragraph is stored within the element <*pPr*> and the formatting information of a run is stored within the element <*rPr*>. An example is shown in Figure 68.

```
<w:p>
        <w:pPr>
                <w:jc w:val="center"/>
                <w:rPr>
                        <w:rFonts w:ascii="Times New Roman" ....../>

                                        ......

                </w:rPr>
        </w:pPr>
        <w:r>
                <w:rPr>
                        <w:rFonts w:ascii="Times New Roman" ...... />
                        ……
                </w:rPr>
                <w:t>John Marketer</w:t>
        </w:r>
</w:p>
```

Figure 68 — Formatting properties in OOXML

7.2.3.3 Summary

Both International Standards use different concepts to assign styles to paragraphs. Similar concepts are used for other document parts such as table, list, and page layout. Detailed information about *text formatting* and *paragraph formatting* is given in subclauses 6.2.1 and 6.2.2.

7.2.4 Tables

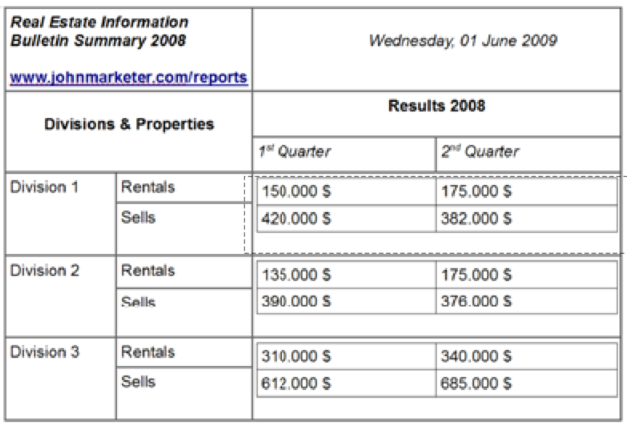
7.2.4.1 Tables in ODF

In ODF every column in a table has a column description element <*table-column*> whose primary use is to reference a table column style that specifies properties such as the column's width. A row in a table is described by the <*table-row*> element.

Sample XML code of the use case introduced in subclause 5.2.7 is shown in Figure 69. The style names used in the XML code are implementation dependent. In ODF the table consists of four columns and nine rows, the element <*table*> includes four column elements <*table-column*> and nine row elements <*table-row*>.

The <*table-cell*> and <*covered-table-cell*> elements specify the content of the table cells. They are contained in <*table row*> elements. A table cell can contain paragraphs and other text content as well as sub tables. Table cells may be empty. Cells can span more than one column or row. For example the cell surrounded by a rectangle in Figure 69 spans two columns and two rows. The number of columns or rows that a cell spans is specified by the attribute @*number-columns-spanned* or @*number-rows-spanned* of the element <*table-cell*>. When a cell covers another cell because a column or row span value is greater than one, a <*covered-table-cell*> element must appear in the table to describe the covered cell.

All parts of a table such as column, row, cell and the entire table itself use the attribute @*style-name* to specify the name of their specific style.



Real Estate Information Bulletin Summary 2008 www.johnmarketer.com/reports		*Wednesday, 01 June 2009*	
Divisions & Properties		Results 2008	
		1st Quarter	*2nd Quarter*
Division 1	Rentals	150.000 $	175.000 $
	Sells	420.000 $	382.000 $
Division 2	Rentals	135.000 $	175.000 $
	Sells	390.000 $	376.000 $
Division 3	Rentals	310.000 $	340.000 $
	Sells	612.000 $	685.000 $

A cell spans two columns and two rows.
It includes a table with two rows and two column.
The cell spans two rows and two columns.
The embedded table

```
<table:table table:name=… table:style-name=…>
      <table:table-column table:style-name=…/>
      <table:table-column table:style-name=…/>
      <table:table-column table:style-name=…/>
      <table:table-column table:style-name=…/>
      <table:table-row ……>                ……              </table:table-row>
      <table:table-row ……>                ……              </table:table-row>
      <table:table-row ……>                ……              </table:table-row>
      <table:table-row table:style-name=…>
            <table:table-cell table:style-name=… table:number-rows-spanned="2" office:value-type="string">
                        <text:p text:style-name=…>Division 1</text:p>
            </table:table-cell>
            <table:table-cell table:style-name=…  office:value-type="string">
                        <text:p text:style-name=…>Rentals</text:p>
            </table:table-cell>
            <table:table-cell  …… table:number-rows-spanned="2"  table:number-columns-spanned="2" >
              <table:table table:name="Tabelle2" table:style-name="Tabelle2">
                    <table:table-column table:style-name="Tabelle2.A"/>
                    <table:table-column table:style-name="Tabelle2.B"/>
                    <table:table-row>
                          <table:table-cell table:style-name="Tabelle2.A1" office:value-type="string">
                                <text:p text:style-name="P5">150.000 $</text:p>
                          </table:table-cell>
                          <table:table-cell table:style-name="Tabelle2.B1" office:value-type="string">
                                <text:p text:style-name="P5">175.000 $</text:p>
                          </table:table-cell>
                    </table:table-row>
                    <table:table-row>
                        <table:table-cell table:style-name="Tabelle2.A2" office:value-type="string">
                              <text:p text:style-name="P5">420.000 $</text:p>
                        </table:table-cell>
                        <table:table-cell table:style-name="Tabelle2.B2" office:value-type="string">
                              <text:p text:style-name="P5">382.000 $</text:p>
                        </table:table-cell>
                    </table:table-row>
              </table:table>
              <text:p text:style-name="P5"/>
            </table:table-cell>
            <table:covered-table-cell/>
       </table:table-row>
       <table:table-row ……>                ……              </table:table-row>
       <table:table-row ……>                ……              </table:table-row>
       <table:table-row ……>                ……              </table:table-row>
       <table:table-row ……>                ……              </table:table-row>
       <table:table-row ……>                ……              </table:table-row>
</table:table>
```

Figure 69 — Excerpt from the XML code of <*table*> in ODF

7.2.4.2 Tables in OOXML

In OOXML, a table is a set of paragraphs (and other block level content) arranged in rows and columns. A <*tbl*> element has two elements that define its properties:

- <*tblPr*> that defines the set of table-wide properties such as *style* and *width;*
- <*tblGrid*> that defines the grid layout of the table.

A <*tbl*> element can also contain an arbitrary number of rows, where each row is specified by a <*tr*> element. Each <*tr*> element can contain an arbitrary non-zero number of cells, where each cell is specified by a <*tc*> element. All elements <*tbl*>, <*tr*> and <*tc*> have specific properties <**Pr*> to describe their style name.

Sample XML code of the use case introduced in subclause 5.2.7 is shown in Figure 70. The table consists of four columns and nine rows. The table element <*tbl*> includes nine row elements <*tr*> and the <*tblGrid*> has four <*gridCol*> elements to define the grid layout.

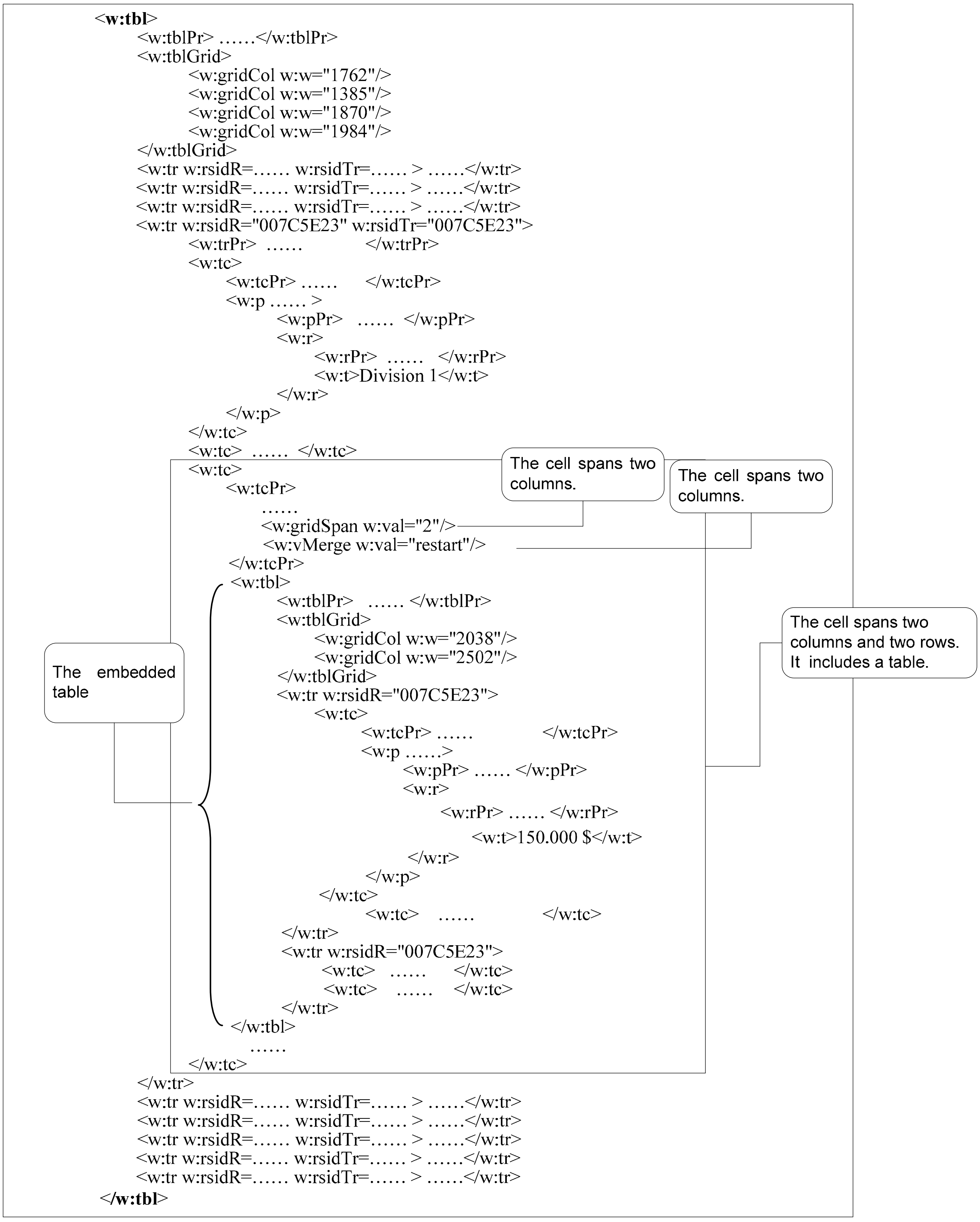
A <*hMerge*> element contained in the cell property element <*tcPr*> specifies that this cell is part of a horizontally merged set of cells in a table. The *@val* attribute of this element determines how this cell is defined with respect to the previous cell in the table i.e., whether this cell continues the horizontal merge or starts a new merged group of cells. Similarly, the <*vMerge*> element specifies that this cell is part of a vertically merged set of cells in the table. The <*gridSpan*> element specifies the number of grid columns in the parent table's table grid which shall be spanned by the current cell. This property allows cells to be merged in case they span vertical boundaries of other cells in the table.

The XML code of the cell surrounded by a rectangle in Figure 70 and in Figure 69 describes the same cell in the referenced use case.

7.2.4.3 Summary

In ODF and OOXML the structures of tables are quite similar. The representation of tables, described by <*table*> element in ODF and <*tbl*> element in OOXML, is based on a grid of rows and columns. Rows take precedence over columns. A table is divided into rows, described by the <*table-row*> element in ODF and the <*tr*> element in OOXML. Rows are divided into cells. A table consists of one or more rows and a row consists of one or more cells. Cells are allowed to span over columns and rows. They can contain another table. Table, row and cell have their own specific properties.

ODF allows the specification of column level properties. The element <*table*> has a <*table-column*> sub element to describe its column’s formatting information. OOXML only supports the specification of the width of a column using the element <*tblGrid*>.



```
<w:tbl>
      <w:tblPr> ……</w:tblPr>
      <w:tblGrid>
              <w:gridCol w:w="1762"/>
              <w:gridCol w:w="1385"/>
              <w:gridCol w:w="1870"/>
              <w:gridCol w:w="1984"/>
      </w:tblGrid>
      <w:tr w:rsidR=…… w:rsidTr=…… > ……</w:tr>
      <w:tr w:rsidR=…… w:rsidTr=…… > ……</w:tr>
      <w:tr w:rsidR=…… w:rsidTr=…… > ……</w:tr>
      <w:tr w:rsidR="007C5E23" w:rsidTr="007C5E23">
              <w:trPr>  ……            </w:trPr>
              <w:tc>
                    <w:tcPr> ……      </w:tcPr>
                    <w:p …… >
                            <w:pPr>   ……  </w:pPr>
                            <w:r>
                                  <w:rPr>  ……   </w:rPr>
                                  <w:t>Division 1</w:t>
                            </w:r>
                    </w:p>
              </w:tc>
              <w:tc>  ……  </w:tc>
              <w:tc>
                    <w:tcPr>
                          ……
                          <w:gridSpan w:val="2"/>
                          <w:vMerge w:val="restart"/>
                    </w:tcPr>
                    <w:tbl>
                            <w:tblPr>   …… </w:tblPr>
                            <w:tblGrid>
                                  <w:gridCol w:w="2038"/>
                                  <w:gridCol w:w="2502"/>
                            </w:tblGrid>
                            <w:tr w:rsidR="007C5E23">
                                  <w:tc>
                                         <w:tcPr> ……              </w:tcPr>
                                         <w:p ……>
                                                <w:pPr> …… </w:pPr>
                                                <w:r>
                                                      <w:rPr> …… </w:rPr>
                                                           <w:t>150.000 $</w:t>
                                                </w:r>
                                         </w:p>
                                  </w:tc>
                                         <w:tc>    ……             </w:tc>
                            </w:tr>
                            <w:tr w:rsidR="007C5E23">
                                   <w:tc>   ……      </w:tc>
                                   <w:tc>    ……    </w:tc>
                            </w:tr>
                    </w:tbl>
                          ……
              </w:tc>
      </w:tr>
      <w:tr w:rsidR=…… w:rsidTr=…… > ……</w:tr>
      <w:tr w:rsidR=…… w:rsidTr=…… > ……</w:tr>
      <w:tr w:rsidR=…… w:rsidTr=…… > ……</w:tr>
      <w:tr w:rsidR=…… w:rsidTr=…… > ……</w:tr>
      <w:tr w:rsidR=…… w:rsidTr=…… > ……</w:tr>
</w:tbl>
```

The cell spans two columns.
The cell spans two columns.
The cell spans two columns and two rows. It includes a table.
The embedded table

Figure 70 — Excerpt from the XML code of <*tbl*> in OOXML

7.2.5 Lists - Itemization and numeration

7.2.5.1 Lists in ODF

In ODF a list is represented by the <*list*> element whose XML structure is depicted in Figure 71. A list contains an optional list *header*, followed by any number of list *items*. Every list has a list *level*, which is determined by the nesting of the <*list*> elements. If a list is not contained within another list, the list level is 1. If the list is contained within another list, the list level is the list level of the list in which it is contained incremented by one. If a list is contained in a table cell or text box, the list level returns to 1, even though the table or text box itself may be nested within another list.

The optional attribute @*style-name* of a list specifies the style name that is applied to the list. The list styles contain relevant layout information, such as

- type of list item label, such as bullet or number,
- list item label width and distance,
- bullet character or image (if any),
- number format for the bullet numbering (if any), and
- paragraph indent for list items.

The list header element <*list-header*> or each list item element <*list-item*> contains a sequence of paragraphs, headings or list elements. Lists can be nested. A list item cannot contain tables. If a list header or a list item has numbering applied, an optional <number> element includes the text of the formatted number. This text can be used by applications that do not support numbering, but it will be ignored by applications that support numbering.

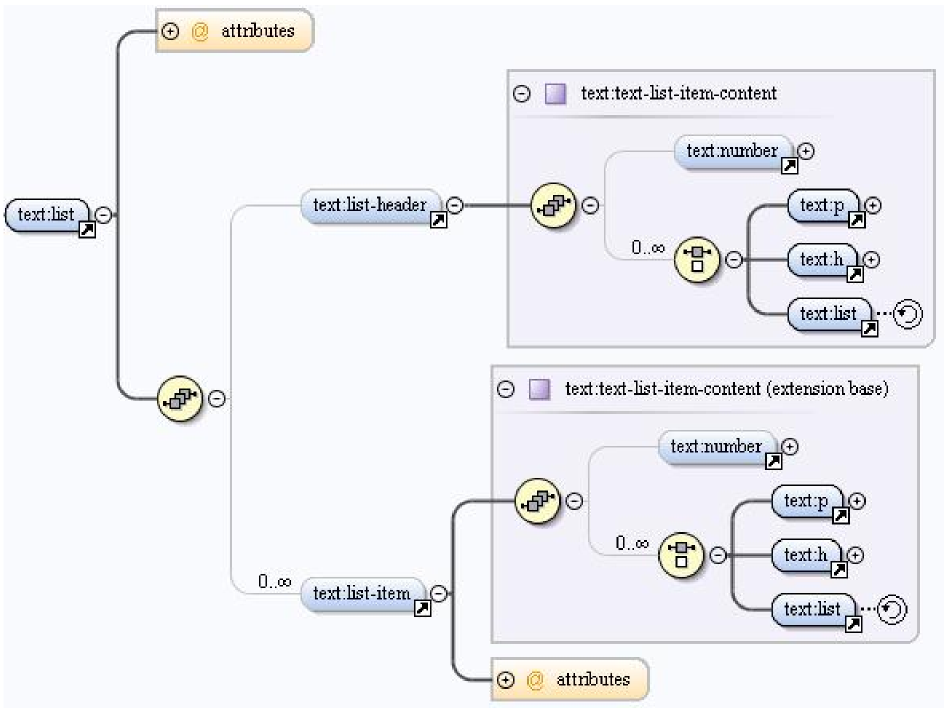


Figure 71 — XML structure of <*list*> in ODF

Numbered paragraphs represented by the element <*numbered-paragraph*> may use the same continuous numbering properties that list items use, and therefore form an equivalent, alternative way of specifying lists. A list in the <*list*> representation could be converted into a list in the <*numbered-paragraph*> representation and vice versa.

7.2.5.2 Lists in OOXML

In OOXML there is no special element to represent a list. List items are simply regular paragraphs <*p*> to which special properties are attached to specify the list structure and a reference to the associated formatting information.

Sample XML code for the first list item in the use case introduced in subclause 5.2.9 is shown in Figure 72. The <*pPr*> element stores the formatting properties at paragraph level. <*pPr*> has sub elements such as <*pStyle*> and <*numPr*>. If the value of attribute @*val* contained in <*pStyle*> is *ListParagraph*, it means that the paragraph is a list item. The <*numPr*> element is used to specify the numbering information.

```
<w:p ……>
        <w:pPr>
                <w:pStyle w:val="ListParagraph"/>
                <w:numPr>
                        <w:ilvl w:val="0"/>
                        <w:numId w:val="1"/>
                </w:numPr>
                ……
        </w:pPr>
        <w:r ……>
                <w:rPr>……  </w:rPr>
                <w:t xml:space="preserve">Turn on screen</w:t>
        </w:r>
</w:p>
```

Figure 72 — An example of a list item in OOXML

7.2.5.3 Summary

The two International Standards describe the structure of lists in different ways. In OOXML lists can be used in all places where paragraphs <*p*> are allowed. In ODF, lists (<*list*> and <*numbered-paragraph*>) can only be used in places where they are explicitly allowed. Detailed information of *itemization and numbering* is given in subclause 6.2.5.

A paragraph in OOXML has more properties than a list item in ODF. Therefore a list defined by a paragraph element in OOXML can be more complex than a list in ODF.

7.2.6 Indices

7.2.6.1 Indices in ODF

There are seven types of index entries in ODF: chapter information, entry text, page number, fixed string, bibliography information, tab stop, and hyperlink start and end. These entries are referenced by the different types of indices:

- *Table of contents* represented by the <*table-of-index*> element; a table of contents provides the user with a guide through the content of the document. It is typically found at the beginning of a document and contains the chapter headings with their respective page numbers.

- *Index of illustrations* represented by the <*illustration-index*> element; the index of illustrations lists all images and graphics in the current document or chapter. The index entries can be derived from the illustration's caption or its name.
- *Index of tables* represented by the <*table-index*> element; the index of tables lists all tables in the current document or chapter. It works in exactly the same way as the index of illustrations.
- *Index of objects* represented by the <*object-index*> element; the index of objects lists all objects in the current document or chapter. It gathers its entries from the known object types.
- *User-defined index* represented by the <*user-index*> element; a user-defined index combines the capabilities of the indexes discussed earlier in this subclause. A user-defined index can gather entries from the following sources: index marks, paragraphs formatted using particular paragraph styles, tables, images, or objects and text frames.
- *Alphabetical index* represented by the <*alphabetical-index*> element; an alphabetical index gathers its entries solely from index marks.
- *Bibliography* represented by the <*bibliography*> element; a bibliography gathers its entries from bibliography index marks.

All types of indices have the same structure. An index consists of two parts:

- *Index source*; the index source is specific for the type of index it is used for. It contains the information necessary to generate the index content.
- *Index body* represented by the <*index-body*> element; the index body is the same for all types of indices. It contains the text generated from the information in the index source. The text contained in an index body is the common text content. The content of the index body can be generated at any time from the information contained in the index source and the remainder of the document.

Both parts are used to define an index *element*. For example Figure 73 depicts the structure of the <*table-of-content*> element in ODF. In the <*table-of-content*> element the <*table-of-content-source*> element contains:

- an optional template element <*index-title-template*> for the index title,
- any number of optional template elements <*table-of-content-entry-template*> for index entries and one per level,
- any number of optional elements <*index-source-styles*> to be used for gathering index entries and some attributes.

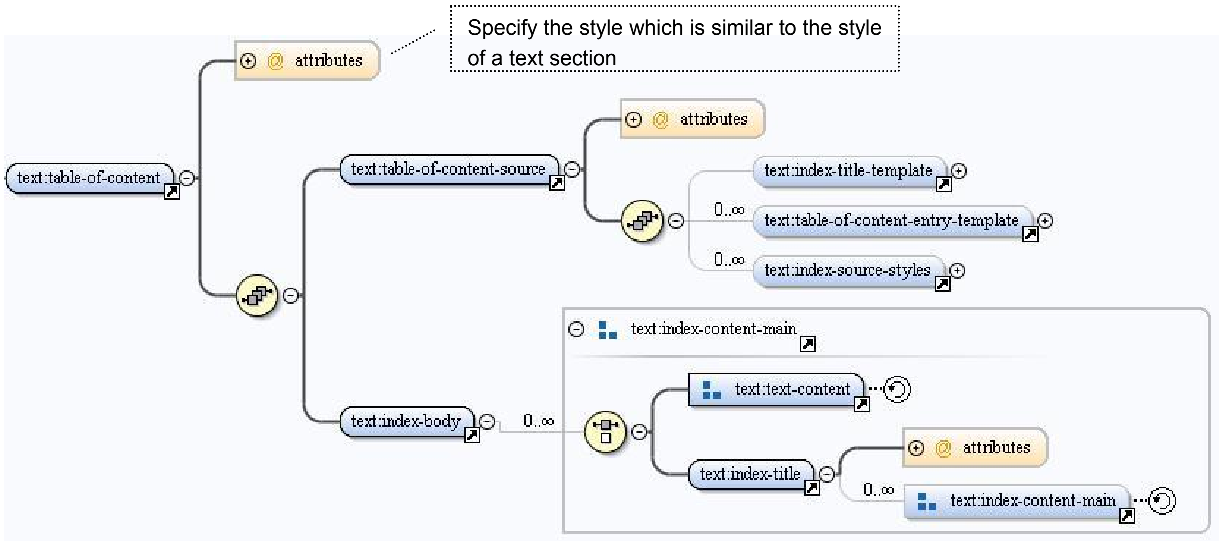
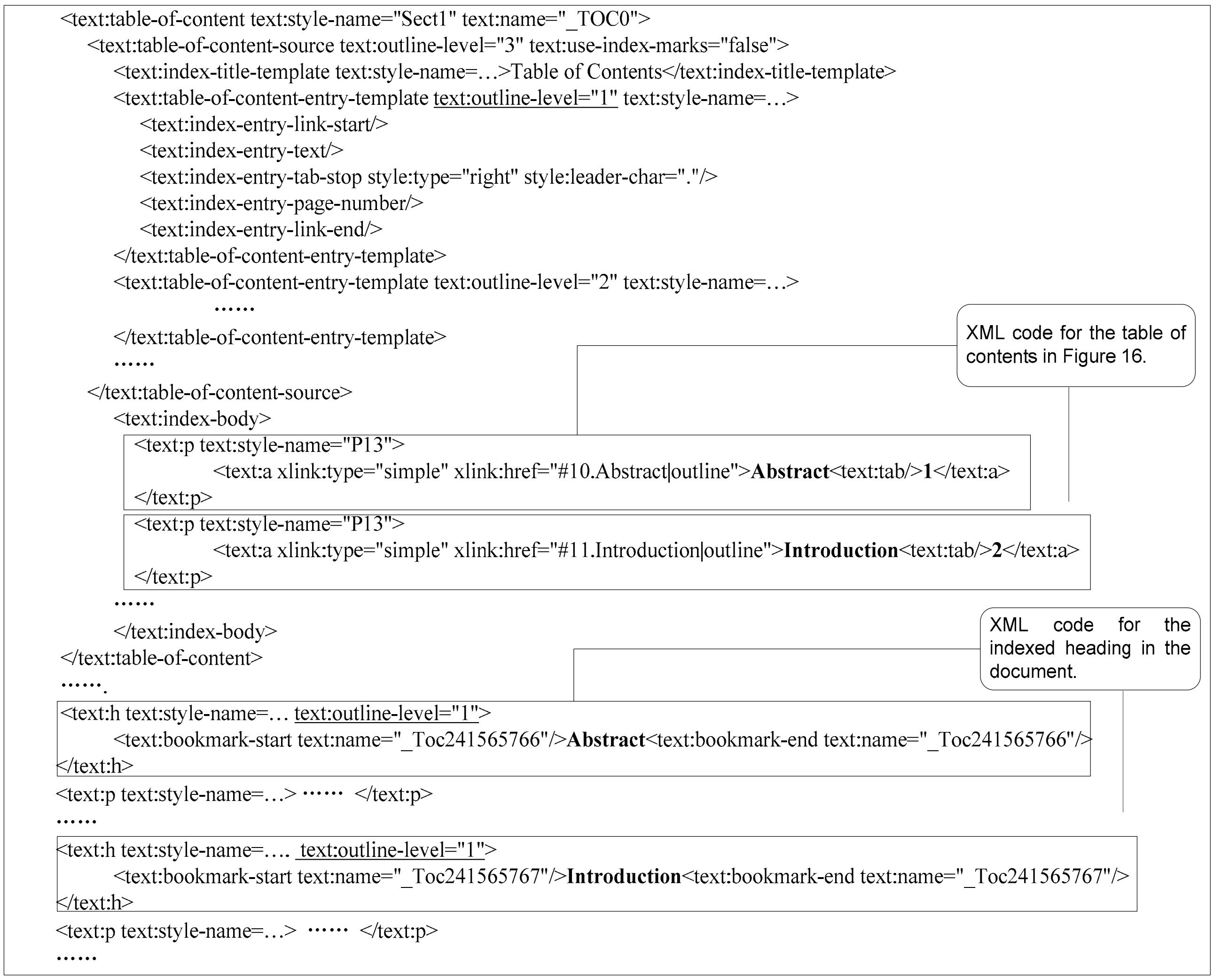


Figure 73 — XML structure of <*table-of-content*> in ODF

Sample XML code of the use case introduced in subclause 5.2.10 is shown in Figure 74.



```
<text:table-of-content text:style-name="Sect1" text:name="_TOC0">
  <text:table-of-content-source text:outline-level="3" text:use-index-marks="false">
    <text:index-title-template text:style-name=…>Table of Contents</text:index-title-template>
    <text:table-of-content-entry-template text:outline-level="1" text:style-name=…>
      <text:index-entry-link-start/>
      <text:index-entry-text/>
      <text:index-entry-tab-stop style:type="right" style:leader-char="."/>
      <text:index-entry-page-number/>
      <text:index-entry-link-end/>
    </text:table-of-content-entry-template>
    <text:table-of-content-entry-template text:outline-level="2" text:style-name=…>
            ……
    </text:table-of-content-entry-template>
    ……
  </text:table-of-content-source>
    <text:index-body>
     <text:p text:style-name="P13">
            <text:a xlink:type="simple" xlink:href="#10.Abstract|outline">Abstract<text:tab/>1</text:a>
     </text:p>
     <text:p text:style-name="P13">
            <text:a xlink:type="simple" xlink:href="#11.Introduction|outline">Introduction<text:tab/>2</text:a>
     </text:p>
    ……
    </text:index-body>
</text:table-of-content>
…….
<text:h text:style-name=… text:outline-level="1">
    <text:bookmark-start text:name="_Toc241565766"/>Abstract<text:bookmark-end text:name="_Toc241565766"/>
</text:h>
<text:p text:style-name=…> …… </text:p>
……
<text:h text:style-name=…. text:outline-level="1">
    <text:bookmark-start text:name="_Toc241565767"/>Introduction<text:bookmark-end text:name="_Toc241565767"/>
</text:h>
<text:p text:style-name=…> …… </text:p>
……
```

XML code for the table of contents in Figure 16.

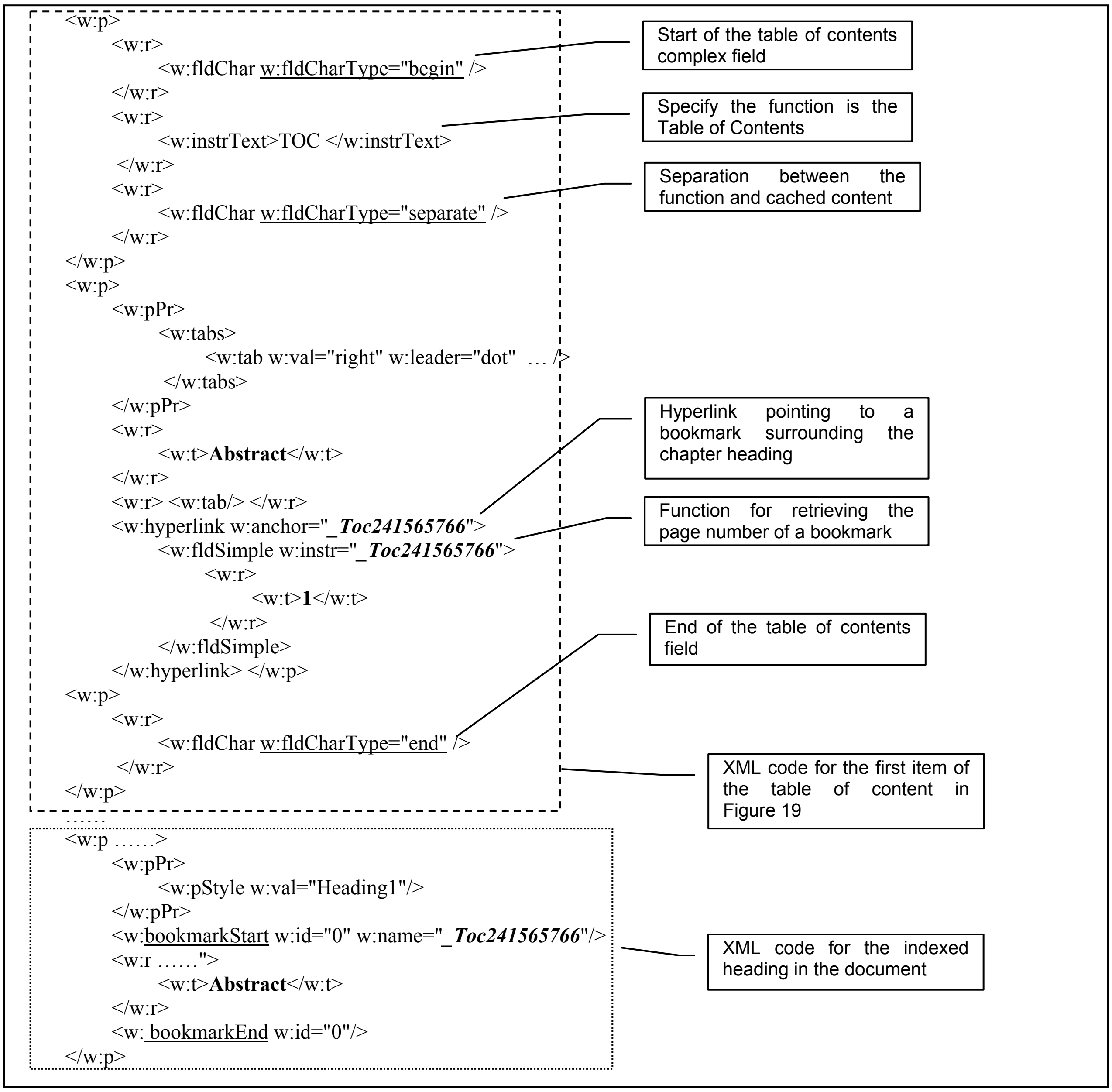
XML code for the indexed heading in the document.

Figure 74 — Excerpt from the XML code of <*table-of-content* > in ODF

7.2.6.2 Indices in OOXML

The concepts of tables of contents and indices are implemented as *dynamic content fields* in OOXML. There are two types of fields, simple and complex. Simple fields are able to wrap a single run. The run text stores a cached version of the field data from the last time the fields were updated. Complex fields can surround multiple runs. Both the simple and complex fields use functions to define their dynamic data. The keyword *TOC* means for example that the function returns the table of contents.

In Figure 75 sample XML code of the use case introduced in subclause 5.2.10 is shown to illustrate the structure of an index in OOXML.



```
<w:p>
      <w:r>
            <w:fldChar w:fldCharType="begin" />
      </w:r>
      <w:r>
            <w:instrText>TOC </w:instrText>
       </w:r>
      <w:r>
            <w:fldChar w:fldCharType="separate" />
      </w:r>
</w:p>
<w:p>
      <w:pPr>
            <w:tabs>
                  <w:tab w:val="right" w:leader="dot"  … />
             </w:tabs>
      </w:pPr>
      <w:r>
            <w:t>Abstract</w:t>
      </w:r>
      <w:r> <w:tab/> </w:r>
      <w:hyperlink w:anchor="_Toc241565766">
            <w:fldSimple w:instr="_Toc241565766">
                  <w:r>
                        <w:t>1</w:t>
                   </w:r>
            </w:fldSimple>
      </w:hyperlink> </w:p>
<w:p>
      <w:r>
            <w:fldChar w:fldCharType="end" />
       </w:r>
</w:p>
……
<w:p ……>
      <w:pPr>
            <w:pStyle w:val="Heading1"/>
      </w:pPr>
      <w:bookmarkStart w:id="0" w:name="_Toc241565766"/>
      <w:r ……">
            <w:t>Abstract</w:t>
      </w:r>
      <w: bookmarkEnd w:id="0"/>
</w:p>
```

Start of the table of contents complex field
Specify the function is the Table of Contents
Separation between the function and cached content
Hyperlink pointing to a bookmark surrounding the chapter heading
Function for retrieving the page number of a bookmark
End of the table of contents field
XML code for the first item of the table of content in Figure 19
XML code for the indexed heading in the document

Figure 75 — Excerpt from the XML code of <*table of content*> in OOXML

7.2.6.3 Summary

In ODF different types of elements are used for different types of index entries. Although these elements have a similar structure, the definition of the index source for different types of indices is different. In OOXML all kinds of indices are implemented by the same mechanism, namely dynamic content fields. However, a function must be specified for special fields. Detailed information is given in subclause 6.2.7.

7.2.7 Change tracking and collaboration support

7.2.7.1 Change tracking and collaboration support in ODF

In ODF all tracked changes in text documents are represented by the <*tracked-changes*> element. The element whose XML structure is shown in Figure 76 is defined as follows:

- The *@track-changes* attribute determines whether or not the track and record changes for this document should be recorded.
- A sequence of zero or more <*changed-region*> elements; for every changed region of a document there is one entry in the list of tracked changes. Every <*changed-region*> element has an *@id* attribute. The elements that mark the start and end of a region use this *@id* to identify the region to which they belong.

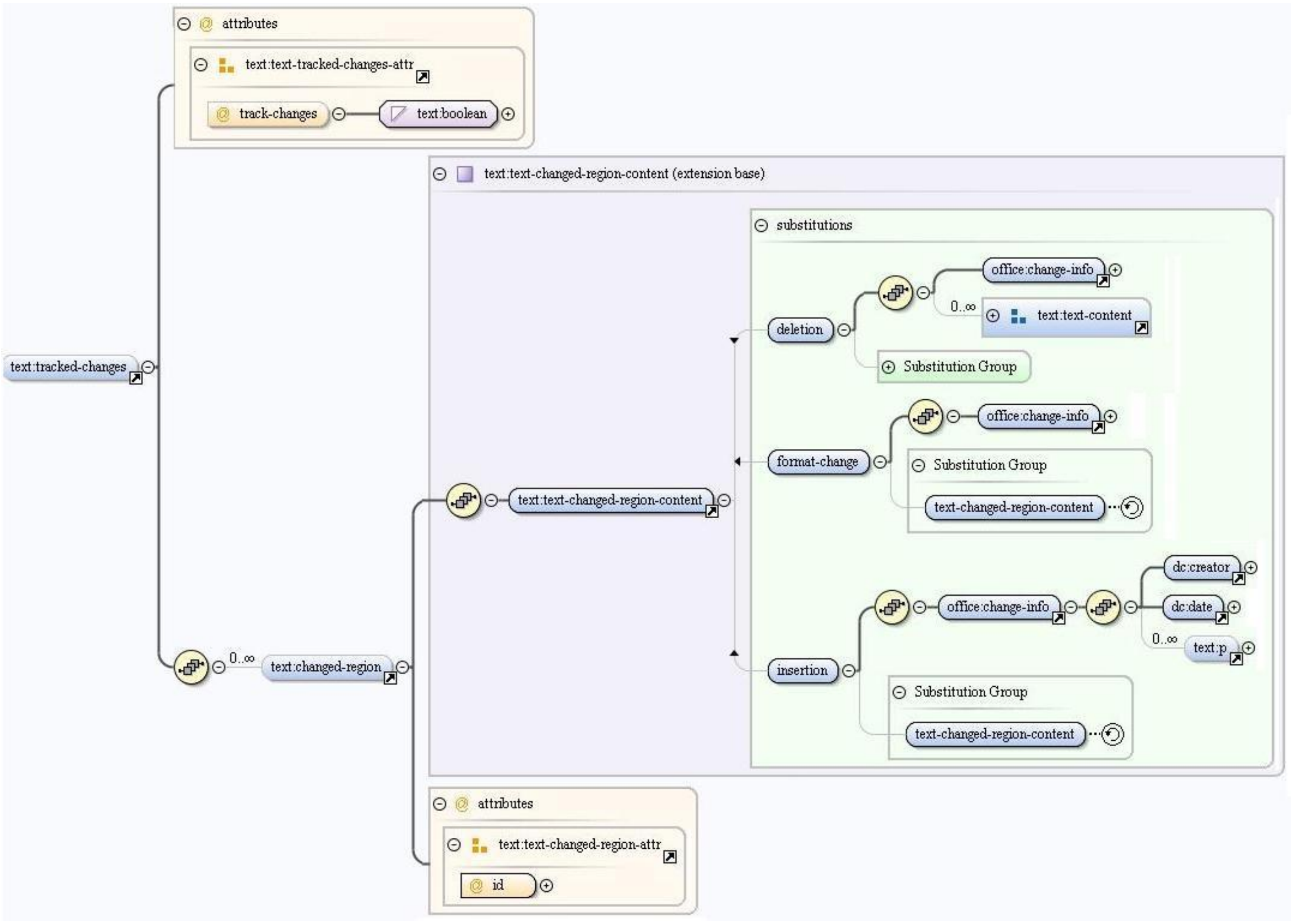
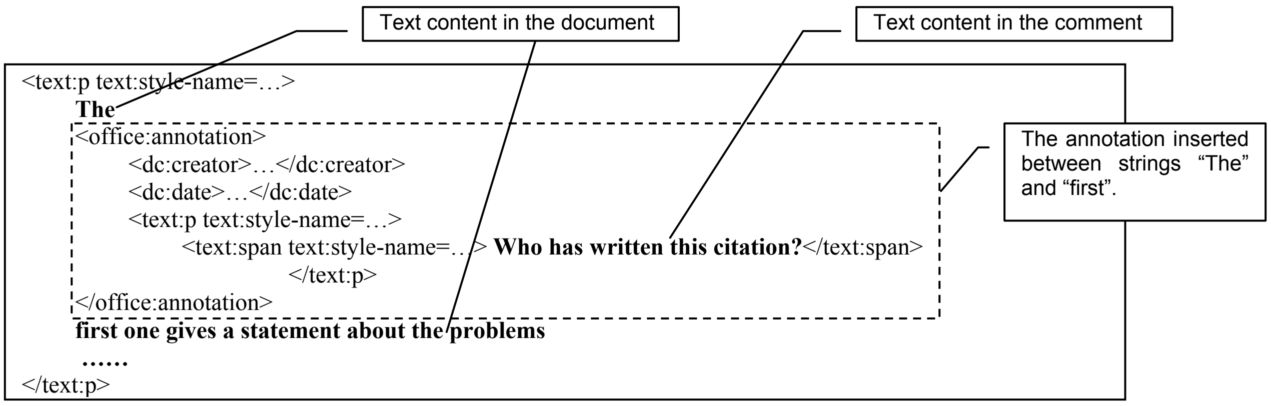


Figure 76 — XML structure of <*tracked-changes*> in ODF

The types of region content can be *insertion*, *deletion* or *format change*. The XML structures of their <*change-info*> elements are identical. The location of each <*changed-region*> is defined by a <*change*> element whose *@region-id* attribute has the same value as the *@id* of the <*changed-region*>. The inserted content can be a piece of text within a paragraph, a whole paragraph, or a whole table. The inserted content is part of the text document itself and is marked by a <*change-start*> and a <*change-end*> element. The deleted content is represented by a <*text-content*> element whose structure is shown in Figure 65.

In ODF an annotation is represented by an <*annotation*> element inserted at a selected point. The text in an annotation is contained in a sequence of zero or more paragraph <*p*> or list elements <*list*>. An <*annotation*> element includes optional elements <*creator*>, <*date*> and <*date-string*> to record the author and the creation date and time of the annotation. Figure 77 shows sample XML code defining how an annotation is inserted between the strings “The” and “first”.


Text content in the document
Text content in the comment
The annotation inserted between strings “The” and “first”.

```
<text:p text:style-name=…>
    The
    <office:annotation>
        <dc:creator>…</dc:creator>
        <dc:date>…</dc:date>
        <text:p text:style-name=…>
            <text:span text:style-name=…> Who has written this citation?</text:span>
                </text:p>
    </office:annotation>
    first one gives a statement about the problems
    ……
</text:p>
```

Figure 77 — Excerpt from the XML code of an annotation in ODF

7.2.7.2 Change tracking and collaboration support in OOXML

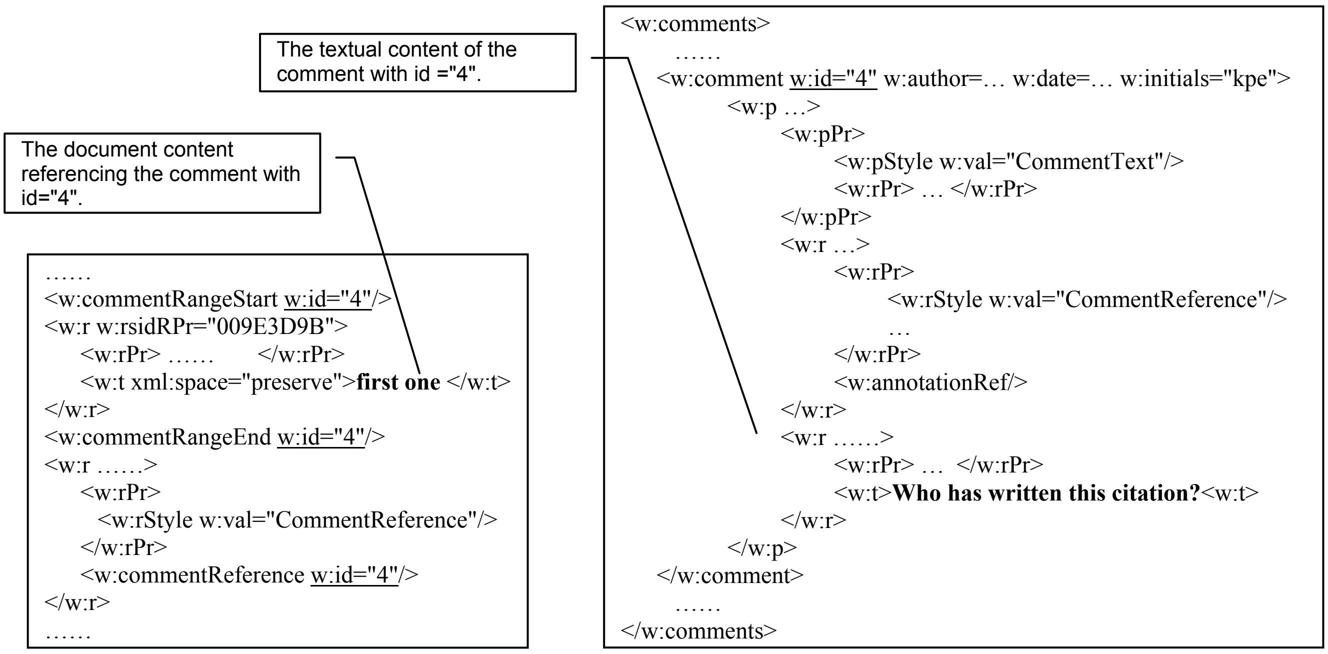
Within an OOXML document the following types of revisions can be used to track the changes to a document: deletions, moves, changes to run, paragraph, table, numbering and section properties, and changes to custom XML markup. There are 17 different types of elements to represent these different types of changes.

In OOXML annotations refer to various types of supplementary markup which can be stored inside or around a region of text within the document's contents. The types of supplementary markup include: comments, revisions, spelling and/or grammatical errors, bookmark information and optional editing permissions.

A comment in a document is divided into two components:

- Comment *anchor*; the text on which the comment applies. It is the cross structure annotation which defines the region of text on which the comment in anchored.
- Comment *content*; the contents of the comment. It is the actual content stored in the comment part.

In the use case depicted in Figure 24 the string “first one” is a region of text tied to a comment. The XML code of the comment anchor is shown in Figure 78(a). The <*commentRangeStart*> and <*commentRangeEnd*> elements delimit the run content to which the comment with the "@id=4" applies. The <*commentReference*> element links the selected content to a comment in the comments part with the associated "@id=4". The XML code of the comment *content* is shown in Figure 78(b). A comment can contain an arbitrary amount of block level content like paragraphs and tables.


The textual content of the comment with id ="4".
The document content referencing the comment with id="4".

```
……
<w:commentRangeStart w:id="4"/>
<w:r w:rsidRPr="009E3D9B">
    <w:rPr> ……    </w:rPr>
    <w:t xml:space="preserve">first one </w:t>
</w:r>
<w:commentRangeEnd w:id="4"/>
<w:r ……>
    <w:rPr>
      <w:rStyle w:val="CommentReference"/>
    </w:rPr>
    <w:commentReference w:id="4"/>
</w:r>
……
```

```
<w:comments>
    ……
  <w:comment w:id="4" w:author=… w:date=… w:initials="kpe">
        <w:p …>
            <w:pPr>
                <w:pStyle w:val="CommentText"/>
                <w:rPr> … </w:rPr>
            </w:pPr>
            <w:r …>
                <w:rPr>
                    <w:rStyle w:val="CommentReference"/>
                    …
                </w:rPr>
                <w:annotationRef/>
            </w:r>
            <w:r ……>
                <w:rPr> …  </w:rPr>
                <w:t>Who has written this citation?<w:t>
            </w:r>
        </w:p>
  </w:comment>
    ……
</w:comments>
```

(a) XML code of the comment anchor (b) XML code of the comment content

Figure 78 — Excerpt from the XML code of comments in OOXML

7.2.7.3 Summary

Both International Standards support change tracking that stores information about the author, date and the changed content. In ODF all tracked changes are represented by one element. In OOXML specific elements are used to track different types of changes. Therefore OOXML offers more complex mechanisms to support change tracking than ODF. A detailed comparison is given in subclause 6.2.8.

Comments in ODF are inserted at certain points in the document. In OOXML comments consist of anchor and content parts that are associated via an *@id* attribute.

7.2.8 Section and page layout

7.2.8.1 Section and page layout in ODF

A text section is a named region of paragraph level text content. Sections start and end on paragraph boundaries and can contain any number of paragraphs. Sections have two uses in ODF:

- They can be used to assign certain formatting properties to a region of text.
- They can be used to group text that is automatically acquired from some external data source.

Sections can contain regular text content, see Figure 65, or the text can be contained in an external file and linked to the section. Sections support two ways of linking to external content:

- A resource identified by an XLink, represented by a <*section-source*> element;
- Dynamic Data Exchange (DDE), represented by a <*dde-source*> element.

In ODF a text section has properties defining the *section style*, such as text columns, background colour or pattern, and the configuration of *notes*. Properties about pages are defined in page layouts and master pages. The <*page-layout*> element specifies the physical properties of a page. This element contains a <*page-layout-*

properties> element which specifies the formatting properties of the page and two optional elements specifying the properties of headers and footers.

A *master page* is a template for pages in a document. It contains a reference to a <*page-layout*> and static content such as headers, footers, or background graphics that are displayed on all pages in the document that use the master page.

7.2.8.2 Section and page layout in OOXML

In OOXML, sections are groups of paragraphs that have a specific set of properties used to define the pages. The layout of a page within a section is controlled by the section's properties. For example each section can have a specific page orientation and its own headers and footers. As shown in Figure 79 the <*sectPr*> element defines properties such as <*footnotePr*>, <*endnotePr*> and <*pgSz*> to specify the properties of footnotes, endnotes and the page size.

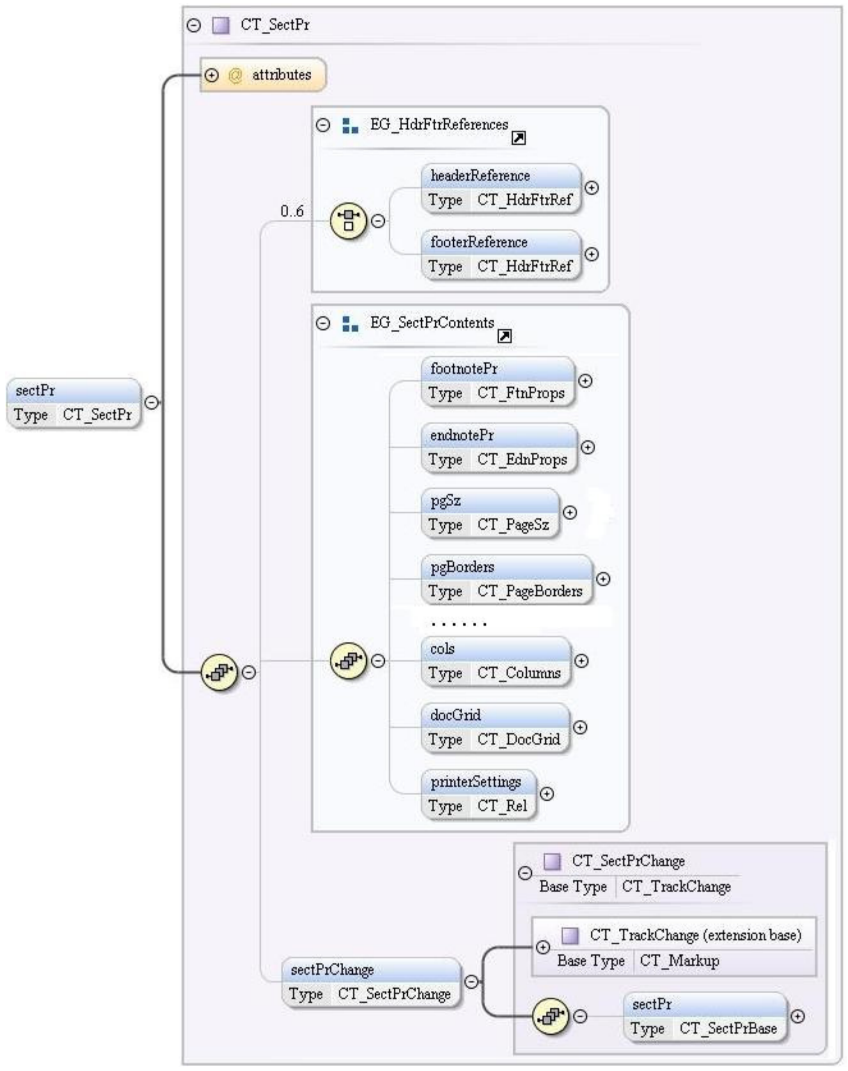


Figure 79 — XML structure of <*sectPr*> in OOXML

A <*sectPr*> element can be applied in three different ways:

- When stored as the last child element of the *body* element mentioned in Figure 66, the <*sectPr*> element defines the section properties for the final section of the document.
- When stored as the last child element of the last *paragraph* in a section which is not the final section of the document, the <*sectPr*> element defines this section's properties.
- When specified as a child element of <*sectPrChange*>, the <*sectPr*> element specifies a set of section properties that were modified to track all revisions when the document was set.

7.2.8.3 Summary

Sections are groups of paragraphs. In both International Standards sections are used to assign formatting properties to different parts of a document. However, there are many differences:

- In ODF sections can contain text from some external data sources. Sections can also be write-protected or hidden. Such features are not defined at section level in OOXML.
- In ODF sections are allowed to contain another section. This is not possible in OOXML.
- In ODF a list can only belong to one section. In OOXML a list is allowed to start in one section and end in another section.
- In OOXML sections are used to define the properties of page layouts. The layout information in ODF is described in page layouts and master pages, but not in sections.

7.3 Spreadsheet documents

7.3.1 Logical structure

7.3.1.1 Spreadsheet documents in ODF

In ODF spreadsheet documents are composed of a *prelude*, *main content* and an *epilogue* as shown in Figure 80. The content of spreadsheet documents mainly consists of a sequence of tables. The spreadsheet document prelude contains the document's form data, change tracking information, calculation setting for formulas, validation rules for cell content and declarations for label ranges. The XML structure of a table embedded in a spreadsheet document is the same as in a word processing document. For details refer to subclause 7.2.4. The epilogue of spreadsheet documents contains declarations for named expressions, database ranges, data pilot tables, consolidation operations and DDE links.

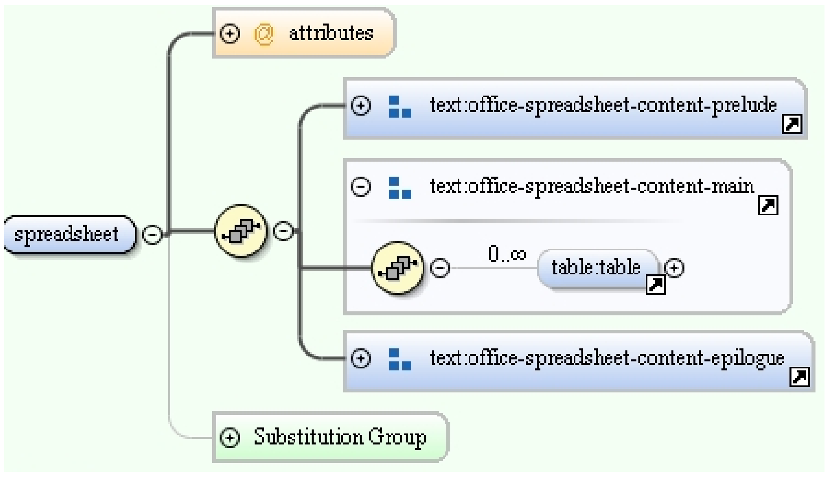

attributes
text:office-spreadsheet-content-prelude
text:office-spreadsheet-content-main
spreadsheet
0..∞
table:table
text:office-spreadsheet-content-epilogue
Substitution Group

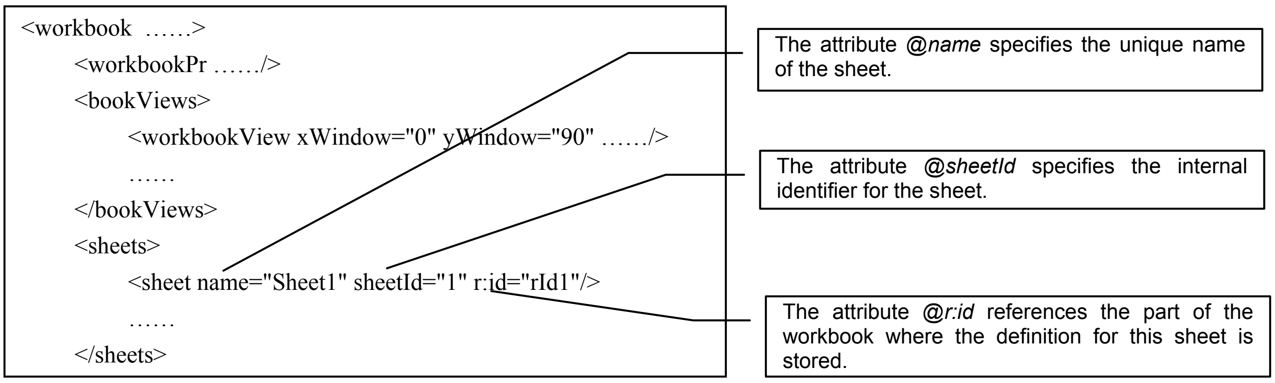
Figure 80 — Structure of a spreadsheet document in ODF

7.3.1.2 Spreadsheet documents in OOXML

In OOXML the <*workbook*> element, the root element of a spreadsheet document, contains elements and attributes that encompass the data content of the workbook. Some sample XML code from a workbook is shown in Figure 81. The workbook's child elements have their own sub clause references.

- The <*workbookPr*> element stores basic workbook settings such as the date system to use, file protection settings, calculation settings, and smart tag behaviors.
- The <*calcPr*> element defines the collection of properties the application uses to record calculation status and details.
- The <*bookViews*> element specifies the collection of workbook views in the enclosing workbook. Each view can specify a window position, filter options and other configurations.
- The element <*sheets*> represents the collection of sheets in the workbook. Each element <*sheet*> refers to a sheet. The element <*sheet*> has three required attributes @*name*, @*sheetId* and @*id*.

The sheets are the central structure within a workbook. They contain text, numbers, dates, formulas and other elements of a workbook. The type of sheet can either be a worksheet, dialog sheet or chart sheet. A worksheet is a two dimensional grid of cells that are organized into rows and columns. Inside a worksheet the data can be split up into three distinct sections. The first section contains sheet properties. The second contains the data, using the required <*sheetData*> element. Various supporting features such as sheet protection and filter information can be found right after <*sheetData*>.



```
<workbook ……>
    <workbookPr ……/>
    <bookViews>
        <workbookView xWindow="0" yWindow="90" ……/>
        ……
    </bookViews>
    <sheets>
        <sheet name="Sheet1" sheetId="1" r:id="rId1"/>
        ……
    </sheets>
```

The attribute @*name* specifies the unique name of the sheet.

The attribute @*sheetId* specifies the internal identifier for the sheet.

The attribute @*r:id* references the part of the workbook where the definition for this sheet is stored.

Figure 81 — Excerpt from the XML code of an empty spreadsheet document in OOXML

7.3.1.3 Summary

The concept *table* in ODF is similar to the concept *worksheet* in OOXML. The basic structure is a two dimensional grid of cells that are organized into rows and columns. In ODF the XML structure of a table is identical in word processing and spreadsheet documents. In OOXML tables and worksheets are defined separately.

7.3.2 Table contents

As mentioned above, the concept *table* in ODF is similar to the concept *worksheet* in OOXML. Each horizontal set of cells in a table/worksheet is called a *row*. Each row has a heading numbered sequentially. Each vertical set of cells in a table/worksheet is called a *column*. Each column has an alphabetic heading. Each *cell* is identified by a cell reference; a combination of its column and row headings.

The cell is the primary place in which data is stored and operated on. A cell can have a number of properties such as numeric, text, date or time formatting, alignment, font, colour and border. Instead of data, a cell can contain a formula, which is an instruction for calculating the associated data. A spreadsheet document can contain additional features such as comments, hyperlinks, images and sorted and filtered tables.

7.3.2.1 Table contents in ODF

In ODF a cell is represented by the <*table-cell*> element, a child element of <*table-row*> as mentioned in subclause 7.2.4. The structure of <*table-row*> is shown in Figure 82. The @*value-type* attribute in ODF specifies the type of value that can appear in a cell. It may contain one of the following values: float percentage or currency (numeric types), date, time, a boolean or a string.

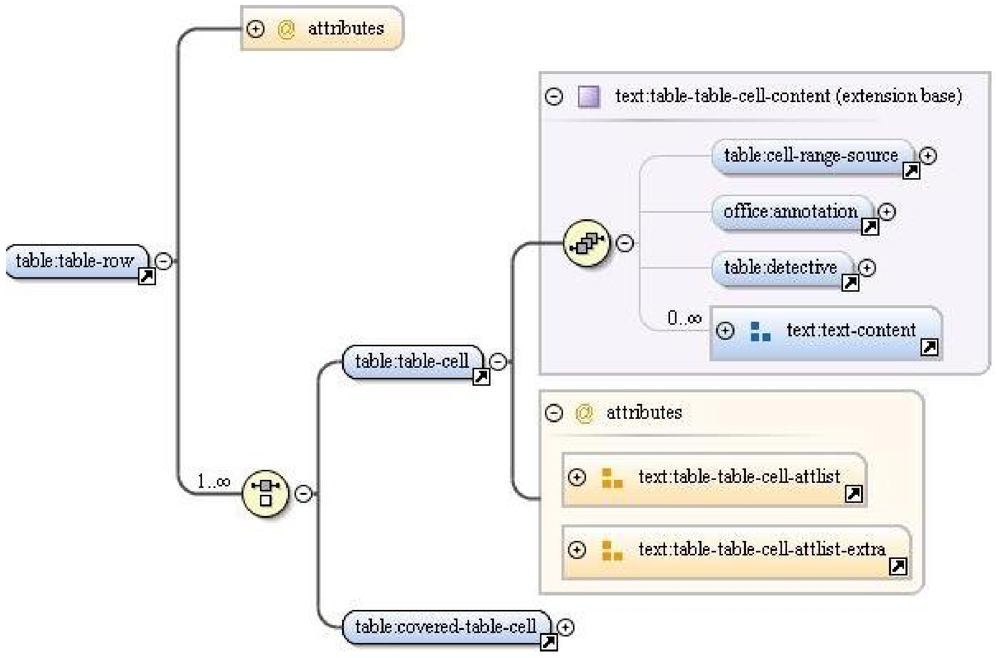


Figure 82 — XML structure of <*table-row*> in ODF

7.3.2.2 Table contents in OOXML

In OOXML the element <*sheetData*> contains information about each cell of a worksheet. Its structure is shown as Figure 83. The information about a cell's location (reference), value, data type, formatting, and formula is defined by the <*c*> element. The @*t* attribute of <*c*> specifies the type of the value in a cell such as a boolean, date, error, number or string. The <*c*> element has a sequence of zero or more child elements:

- The <*f*> element contains the definition of a formula.
- The <*is*> element allows strings to be expressed directly in the cell definition instead of implementing the shared string table.
- The <*v*> element expresses the value contained in a cell. If the cell contains a string this value is an index in the shared string table pointing to the actual string value. Otherwise the value of the cell is expressed directly in this element. Cells containing formulas store the last calculated result of the formula in this element.
- The <*extLst*> element provides a convention for extending spreadsheetML within the markup specification.

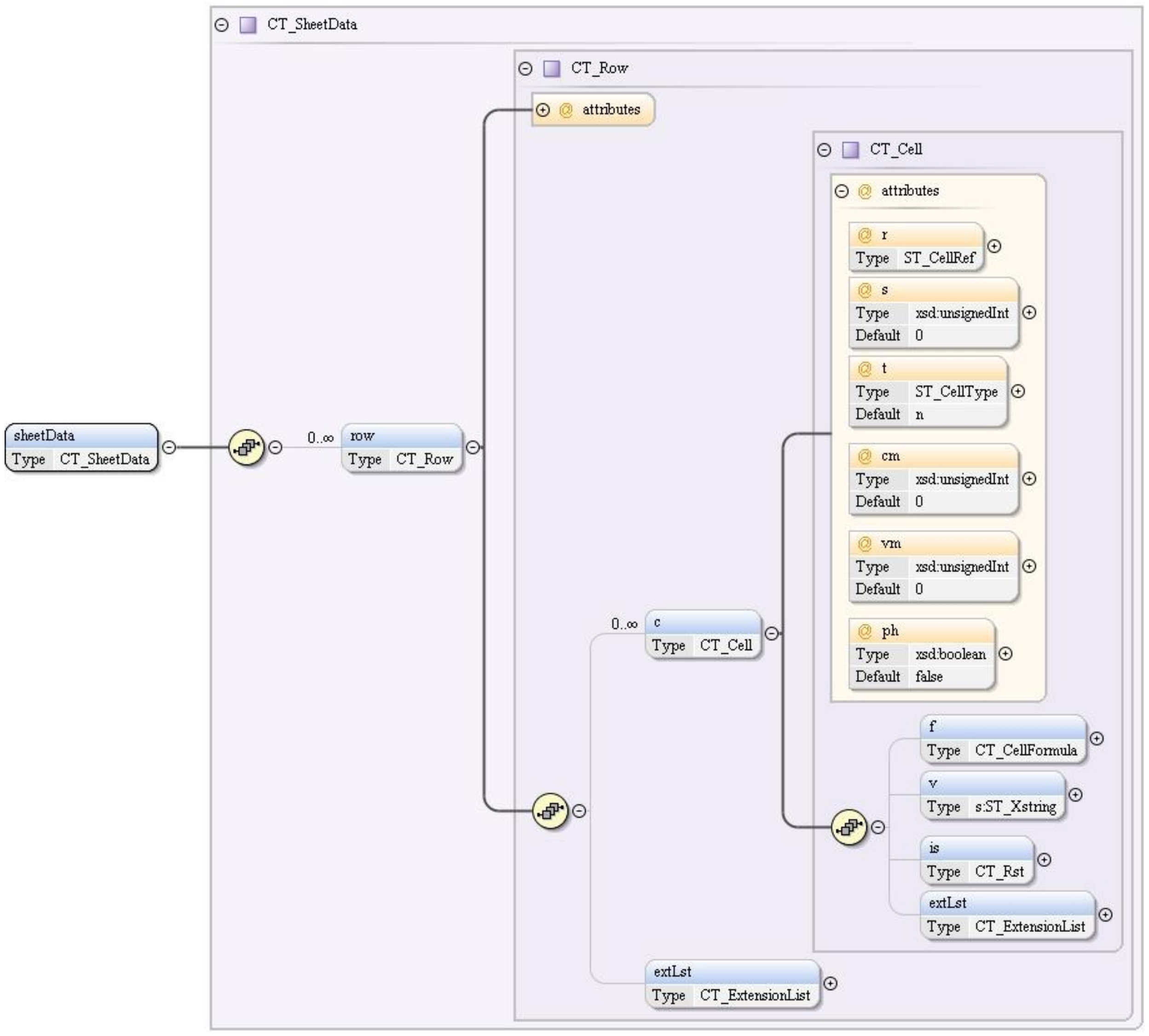


Figure 83 — XML structure of <*sheetData*> in OOXML

7.3.2.3 Summary

In ODF tables in a spreadsheet document are identical to tables in a word processing document. In OOXML a spreadsheet document has a specific definition. It consists of separate parts that implement different functionalities. For example the data of a worksheet is stored in an associated part, all string literals are stored in a single shared string part, and comments are stored in a comments part. Refer to subclause 7.3.4 for detailed information and examples.

7.3.3 Table style

7.3.3.1 Table style in ODF

In ODF the @*style-name* attribute references a table style. Different types of <*style*> elements are used to describe different element styles:

- The *table style* describes the formatting properties of the table such as width and background colour.

- The *table row style* stores the formatting properties of a table row such as height and background colour.
- The *table column style* stores formatting properties of a table column such as width and background colour. It is specified by a <*style*> element with a family attribute value known as the @*table-column*.
- The <*table cell style*> stores the formatting properties of a cell such as background colour, number format, vertical alignment and borders.

Table cell content validations specify validation rules for the contents of table cells. The <*content-validation*> element specifies such validation rules. All validation rules that exist in a document are contained in the <*content-validations*> element. The validation rules themselves are named and referenced from the table cell using their name.

7.3.3.2 Table style in OOXML

In OOXML a style is a named collection of formatting elements.

- A *cell style* specifies number format, cell alignment, font information, cell border specifications, colours and background / foreground fills.
- *Table styles* specify formatting elements for a table's regions. They can, for example, make the header row and totals *bold*, and apply *light gray* fills to alternating rows in the data portion of the table to achieve striped or banded rows.
- *PivotTable styles* specify formatting elements for the regions of a pivot table for example first and second level subtotals, row axis, column axis, and page fields.

7.3.3.3 Summary

A comparison of table styles and formatting functionality in ODF and OOXML is concluded in subclause 6.3.2.

7.3.4 Formulas and calculation

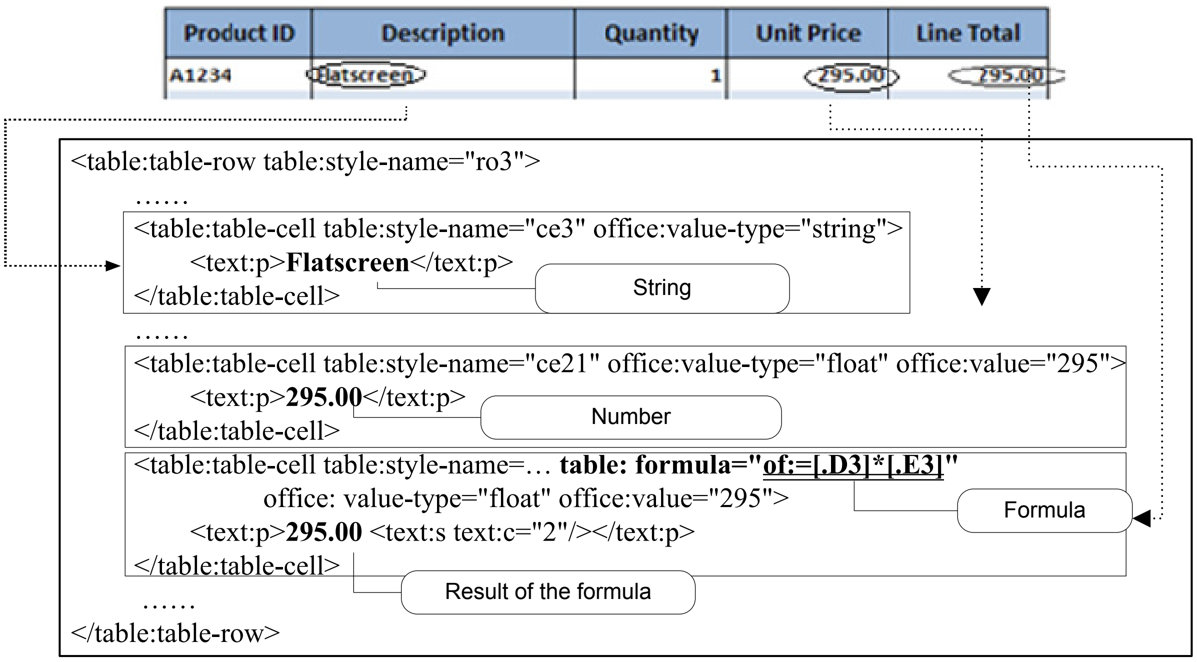
Formulas allow calculations to be performed within table cells. Many similar concepts are used in ODF and OOXML. In the use case introduced in subclause 5.3.3 the values of the column *Description* are strings and the value of the column *Line Total* is defined by *Quantity* plus *Unit Price*. Sample XML code of the example is shown in Figure 84 and Figure 85 to illustrate the different representations in ODF and OOXML.

7.3.4.1 Formulas and calculation in ODF

In ODF every formula should begin with a namespace prefix specifying the syntax and semantics used within the formula. Typically, the formula itself begins with an equals (=) sign and includes arguments such as numbers, text, named ranges, operators, logical operators, function calls and relative or absolute addresses of cells that contain numbers.

7.3.4.2 Formulas and calculation in OOXML

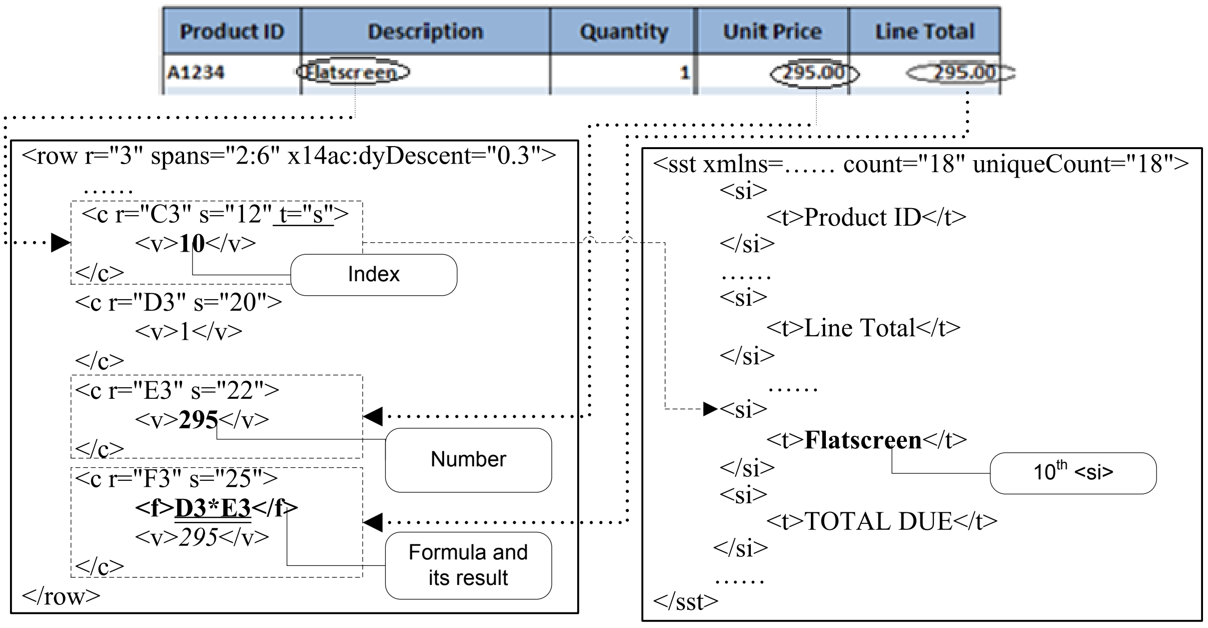
In OOXML a formula is an expression that contains entities such as constants, operators, cell references, calls to functions and names. Examples of predefined formulas are AVERAGE, MAX, MIN, and SUM. A function takes one or more arguments on which it operates, producing a result. For example in the formula SUM(B1:B4), there is one argument, B1:B4, which is the closed range of cells B1–B4. Formulas in OOXML support types like array, error, logical, number, and text.



Product ID	Description	Quantity	Unit Price	Line Total
A1234	Flatscreen	1	295.00	295.00

```
<table:table-row table:style-name="ro3">
      ……
      <table:table-cell table:style-name="ce3" office:value-type="string">
            <text:p>Flatscreen</text:p>
      </table:table-cell>
      ……
      <table:table-cell table:style-name="ce21" office:value-type="float" office:value="295">
            <text:p>295.00</text:p>
      </table:table-cell>
      <table:table-cell table:style-name=… table: formula="of:=[.D3]*[.E3]"
                  office: value-type="float" office:value="295">
            <text:p>295.00 <text:s text:c="2"/></text:p>
      </table:table-cell>
      ……
</table:table-row>
```

Figure 84 — Example of a formula in ODF



Product ID	Description	Quantity	Unit Price	Line Total
A1234	Flatscreen	1	295.00	295.00

```
<row r="3" spans="2:6" x14ac:dyDescent="0.3">
      ……
      <c r="C3" s="12" t="s">
            <v>10</v>
      </c>
      <c r="D3" s="20">
            <v>1</v>
      </c>
      <c r="E3" s="22">
            <v>295</v>
      </c>
      <c r="F3" s="25">
            <f>D3*E3</f>
            <v>295</v>
      </c>
</row>
```

```
<sst xmlns=…… count="18" uniqueCount="18">
      <si>
            <t>Product ID</t>
      </si>
      ……
      <si>
            <t>Line Total</t>
      </si>
            ……
      <si>
            <t>Flatscreen</t>
      </si>
      <si>
            <t>TOTAL DUE</t>
      </si>
      ……
</sst>
```

XML code in sheet1.xml

XML code in sharedStrings.xml

Figure 85 — Example of a formula in OOXML

7.3.4.3 Summary

The formulas and calculations supported in ODF and OOXML have some differences as explained in subclause 6.3.3. The main differences are:

- In ODF the *@formula* attribute of the <*table-cell*> element contains a formula for a table cell. In OOXML a formula is represented by an <*f*> element that contains the text of the formula and a <*v*> element that contains the value of the last evaluation of the formula. This pair of elements is included in a <*c*> element, which is part of a <*row*> element.
- In ODF all content is stored in one file named content.xml. In OOXML string values are not stored in a cell unless they are the result of a calculation. Generally strings are stored in a shared string table.
- Cell ranges are expressed in different ways.

7.3.5 Charts

Charts define a visualisation of numeric data. Charts define the source of the data and how they should be visualised.

7.3.5.1 Charts in ODF

ODF defines 11 basic chart types which are: line, area, circle, ring, scatter, radar, bar, stock, bubble, surface and gantt.

The <*chart*> element represents an entire chart including title, legend, and the graphical object that visualises the underlying data called the *plot area*. Its XML structure is shown in Figure 86. The *@class* attribute specifies the chart type. Charts are always contained within other XML documents. There are two types of chart container documents:

- Containers that do not provide data for the chart; the chart data is contained in a <*table*> element inside the <*chart*> element.
- Containers that provide data for the chart; the chart data is contained in a <*table*> element in the parent document, for example in a spreadsheet or text document. The chart data is specified by the *@cell-rangeaddress* attribute of the <*plot-area*> element. The <*plot-area*> element represents the visualisation container of all data series in the chart.

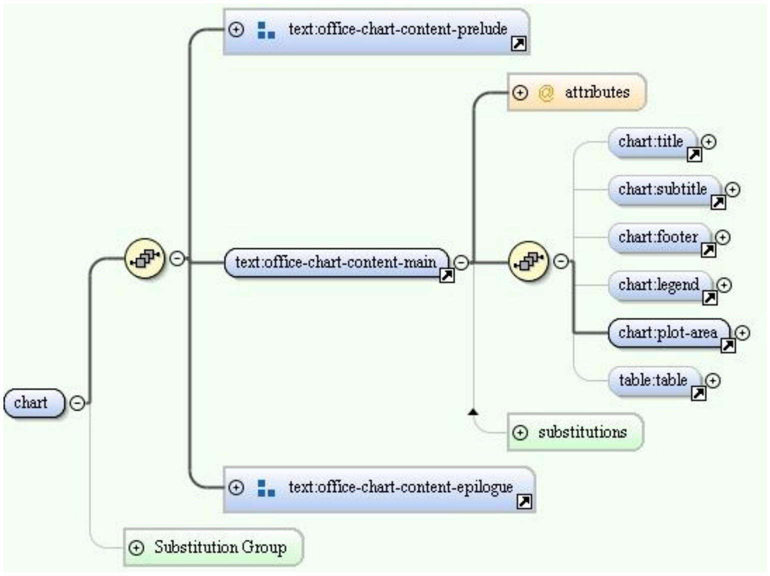


Figure 86 — XML structure of <*chart*> in ODF

7.3.5.2 Charts in OOXML

OOXML defines11 basic charts, which are: column chart, bar chart, line chart, pie chart, area chart, scatter chart, stock chart, surface chart, doughnut chart, bubble chart and radar chart. Most chart types have three dimensional representations. 3D charts have extra properties to describe depth, floor or walls as well as some other rendering effects.

A <*chartSpace*> element specifies overall settings for a single chart. It is the root node for a chart. A chart is represented by a <*chart*> element whose XML structure is shown as Figure 87. The <*plotArea*> element is the only mandatory child element of <*chart*>. It specifies the plot area of the chart. Different chart types have different child elements of <*plotArea*>. The chart XML files can be reused and shared among different applications such as a spreadsheet, presentation and word processing.

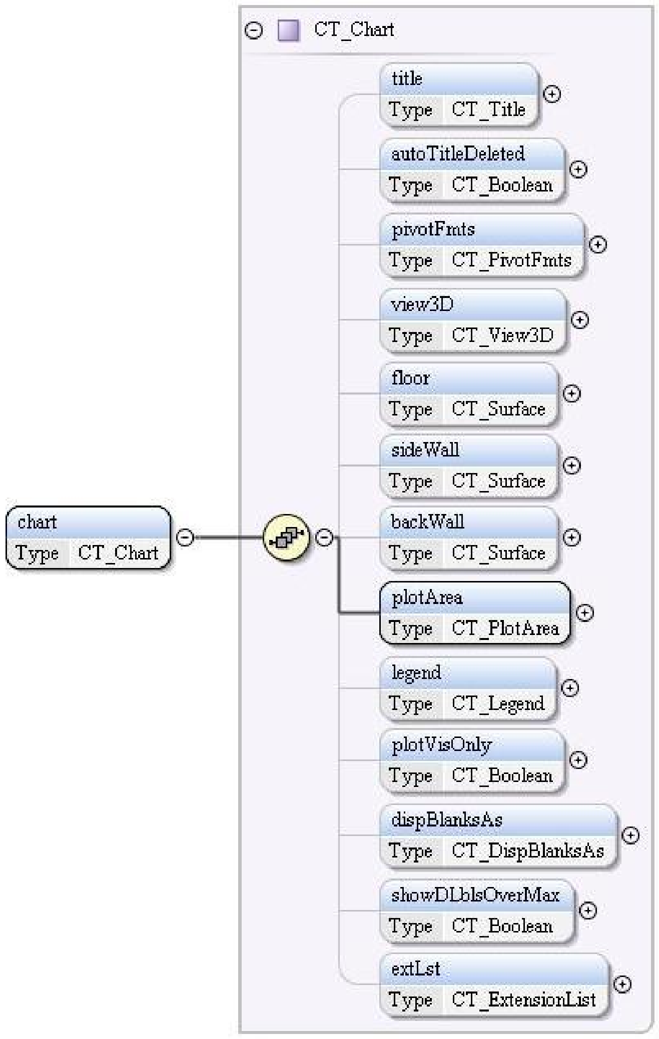


Figure 87 — XML structure of <*chart*> in OOXML

7.3.5.3 Summary

Although the names of chart types are different in ODF and OOXML, most of them find a corresponding type in the other format.

7.4 Presentation documents

7.4.1 Logical structure

7.4.1.1 Presentation documents in ODF

In ODF presentation documents are composed of a prelude, main content and an epilogue as shown in Figure 88.

- The presentation content *prelude* equals that of a drawing document but may contain additional declarations.
- The presentation content *epilogue* may contain presentation settings and elements that implement enhanced table features.
- The main document content contains a sequence of pages represented by the <*page*> element which acts as a container for content.

The <*page*> element contains

- a sequence of office forms (group <*office-forms*>),
- shapes (group <*shape*>),
- animations (choice of element <*animations*> or <*animation-element*>),
- presentation notes (<*notes*>). The <notes> element contains zero or more <*shape*> elements and some additional attributes.

The attributes that may be associated with the <*page*> element are page name, page style, master page, presentation page layout, header declaration, footer declaration, date and time declaration and id.

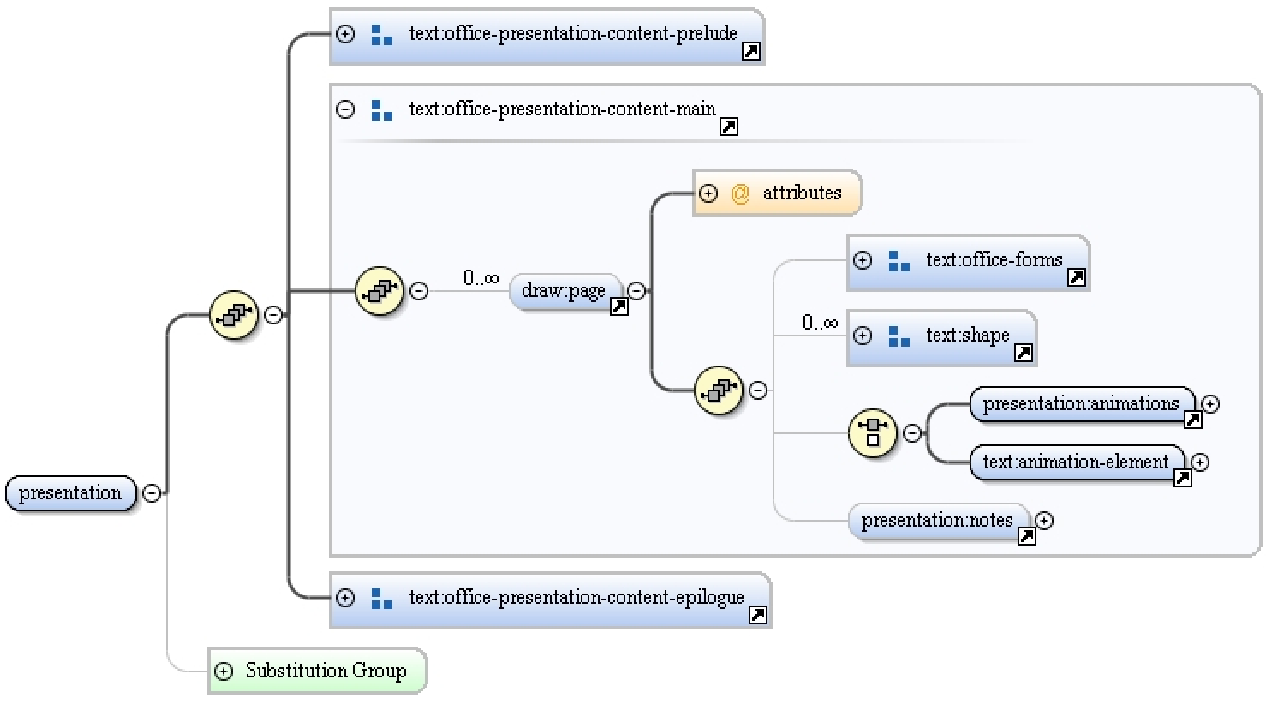


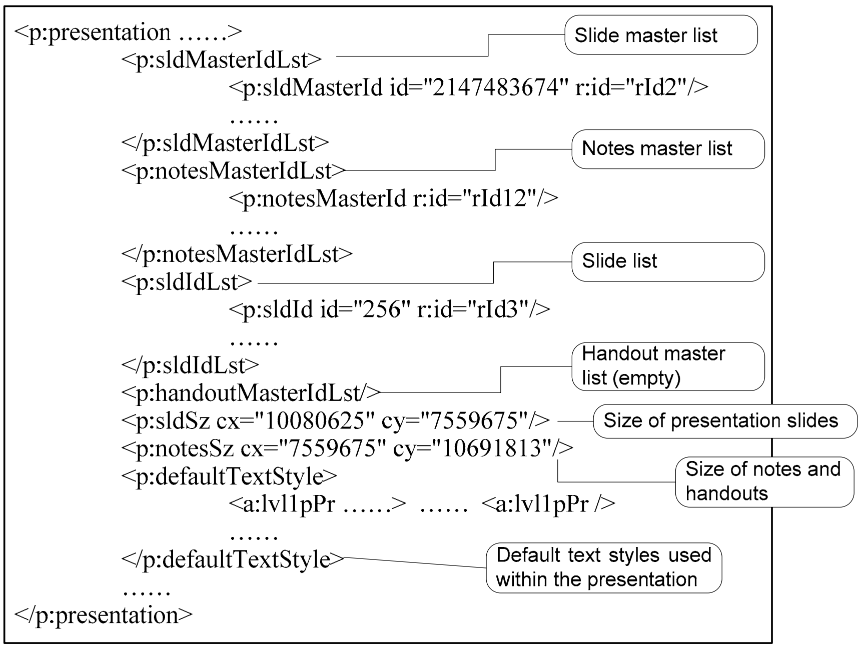
Figure 88 — XML structure of a presentation document in ODF

7.4.1.2 Presentation documents in OOXML

A presentation document starts with a <*presentation*> root element that refers to a slide list, a slide master list, a notes master list, and a handout master list, which refer to all of the corresponding objects in the presentation.

- A *slide* is a frame containing text and/or images, comments, notes and layout definitions. It can be a part of one or more custom presentations. A *comment* is an annotation intended for the person maintaining the presentation slide deck. A *note* is text intended for the presenter or the audience. A *slide layout* defines the visualisation of the elements of a slide.
- The *slide master list* refers to all *slide masters* of the presentation document.
- A *notes master* contains information about the format of notes pages.
- A *handout* is a printed set of slides that can be handed out to an audience for future reference. A *handout master* defines the format of a handout.

A <*presentation*> element has several sub elements describing the properties of the presentation. Some example XML code from a presentation is shown in Figure 89.



```
<p:presentation ……>
          <p:sldMasterIdLst>
                    <p:sldMasterId id="2147483674" r:id="rId2"/>
                    ……
          </p:sldMasterIdLst>
          <p:notesMasterIdLst>
                    <p:notesMasterId r:id="rId12"/>
                    ……
          </p:notesMasterIdLst>
          <p:sldIdLst>
                    <p:sldId id="256" r:id="rId3"/>
                    ……
          </p:sldIdLst>
          <p:handoutMasterIdLst/>
          <p:sldSz cx="10080625" cy="7559675"/>
          <p:notesSz cx="7559675" cy="10691813"/>
          <p:defaultTextStyle>
                    <a:lvl1pPr ……>  ……  <a:lvl1pPr />
                    ……
          </p:defaultTextStyle>
          ……
</p:presentation>
```

Slide master list
Notes master list
Slide list
Handout master list (empty)
Size of presentation slides
Size of notes and handouts
Default text styles used within the presentation

Figure 89 — Excerpt from the XML code of <*presentation*> in OOXML

7.4.1.3 Summary

The structure of presentation documents in ODF and OOXML is very different.

7.4.2 Text formatting

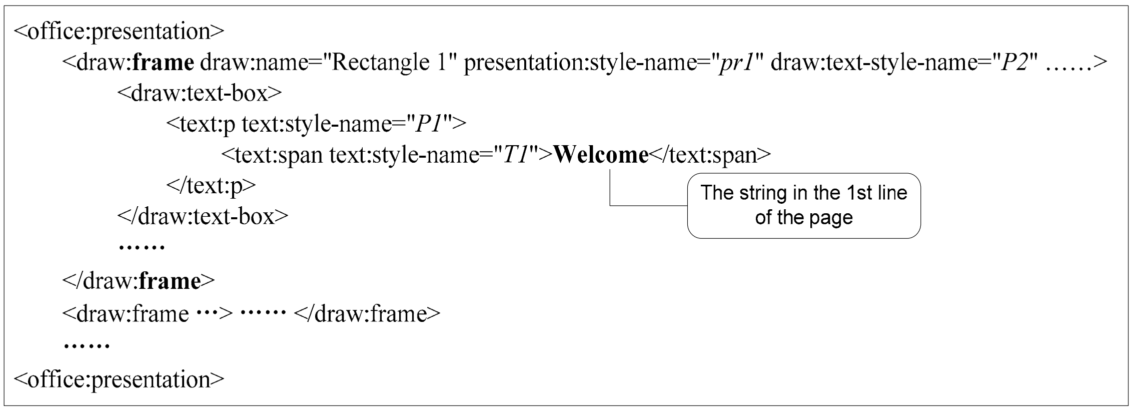
7.4.2.1 Text formatting in ODF

In ODF the <*shape*> group defines different kinds of shapes such as rectangle, line, poly line and circle as well as a <*frame*> element. Most drawing shapes contain text which may contain paragraphs and lists.

The <*frame*> element is a container that contains enhanced content like text boxes, images or objects. The <*frame*> element can be used in many places. It may include text box elements <*text-box*> to place text. A

<*text-box*> element contains attributes and a sequence of zero or more <*text-content*> elements, similar to word processing documents; see Figure 65.

Sample XML code from the use case introduced in subclause 5.4.2 is shown in Figure 90. The page consists of a sequence of <*frame*> elements which contain text content. In Figure 90 the italic strings denote style names.



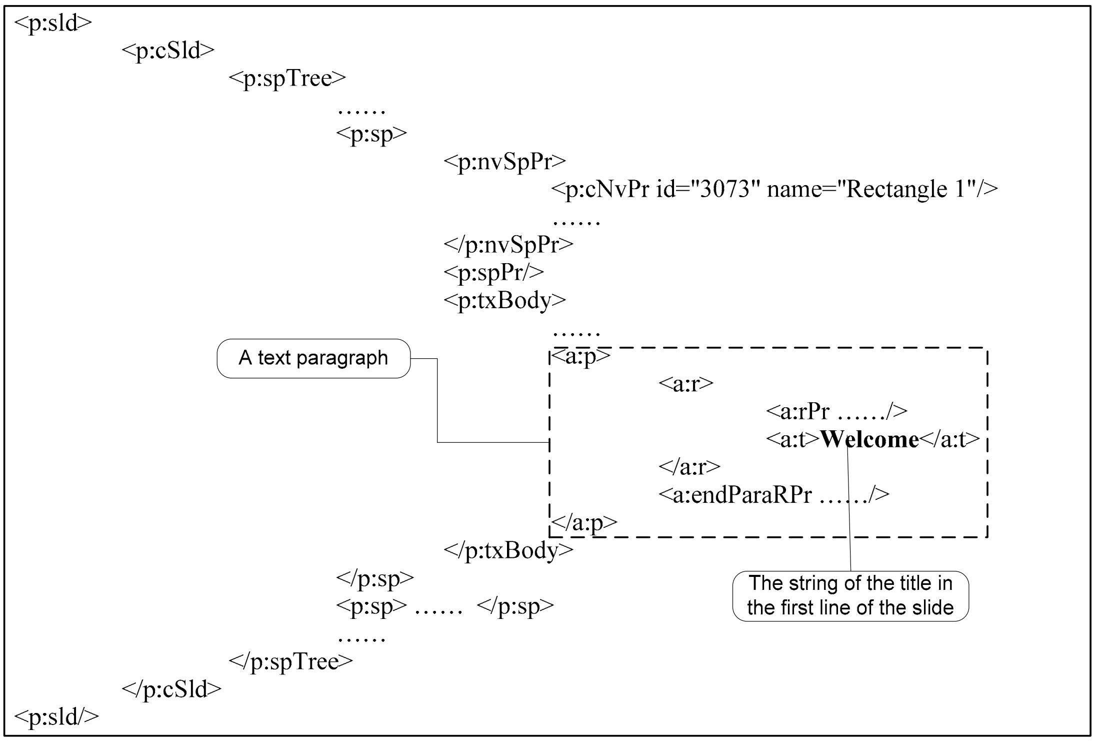
```
<office:presentation>
    <draw:frame draw:name="Rectangle 1" presentation:style-name="pr1" draw:text-style-name="P2" ……>
        <draw:text-box>
            <text:p text:style-name="P1">
                <text:span text:style-name="T1">Welcome</text:span>
            </text:p>
        </draw:text-box>
        ……
    </draw:frame>
    <draw:frame …> …… </draw:frame>
    ……
<office:presentation>
```

The string in the 1st line of the page

Figure 90 — Excerpt from the XML code of <*frame*> in ODF

7.4.2.2 Text formatting in OOXML

In OOXML, the <*sld*> element specifies a slide within the slide list and properties specific to the slide's appearance in the outline view. Sample XML code of the slide introduced in subclause 5.4.2 is given in Figure 91.



```
<p:sld>
    <p:cSld>
        <p:spTree>
            ……
            <p:sp>
                <p:nvSpPr>
                    <p:cNvPr id="3073" name="Rectangle 1"/>
                    ……
                </p:nvSpPr>
                <p:spPr/>
                <p:txBody>
                    ……
                    <a:p>
                        <a:r>
                            <a:rPr ……/>
                            <a:t>Welcome</a:t>
                        </a:r>
                        <a:endParaRPr ……/>
                    </a:p>
                </p:txBody>
            </p:sp>
            <p:sp> …… </p:sp>
            ……
        </p:spTree>
    </p:cSld>
<p:sld/>
```

A text paragraph
The string of the title in the first line of the slide

Figure 91 — Excerpt from the XML code of a slide in OOXML

The <*cSld*> element specifies a container for slide information. The <*spTree*> element specifies all shape-based objects that can be referenced from a given slide. Text and effects are attached to shapes that are contained within the <*spTree*> element. A single shape is specified by the <*sp*> element. As in ODF, shapes can contain text content. All visible text and visible text related properties are contained within the element <*txBody*>. A paragraph included in <*txBody*> is similar to a paragraph in word processing documents.

The <*txStyles*> element can be used to specify text styles within the slide master that will be discussed in subclause 7.4.3.

7.4.2.3 Summary

In ODF and OOXML text in presentations is similar to text in word processing and spreadsheet documents. Although the text containers in ODF and OOXML are different, the text formatting supported by both International Standards is similar. For details refer to subclause 6.4.3.

7.4.3 Master layout

7.4.3.1 Master layout in ODF

In ODF master and layout are defined in styles.

- The <*master-page*> element is used to define master pages as common backgrounds for drawing pages. It specifies the style information of headers and footers, forms, styles, shapes and presentation notes. Each drawing page is directly linked to one master page, which is specified by the *@master-page-name* attribute in the drawing pages style.
- Physical properties like the sizes, borders and orientation of the master page are specified in the <*page-layout*> element together with two optional elements that specify the properties of headers and footers. Both a page and a master can reference a page layout with the *@page-layout-name* attribute.

7.4.3.2 Master layout in OOXML

In OOXML a slide master <*sldMaster*> element contains the definition of formatting, text, and objects that appear on each slide in the presentation that is associated with the slide master. A slide layout is based on a slide master. As shown in Figure 92, a slide master has two main elements:

- The <*cSld*> element specifies the common slide elements such as shapes and their attached text bodies. The structures of <*cSld*> in slides and slide masters are similar.
- The <*txStyles*> element specifies the formatting of the text within each shape. The other properties within a slide master specify colour information, headers and footers, timing and transition information for all corresponding presentation slides, and style information for title text, body text and other slide texts. All types of styles have the same structure.
- The <*sldLayoutIdLst*> element specifies the slide layout identification list. This list is contained within the slide master and is used to determine which layouts are used within the slide master file.

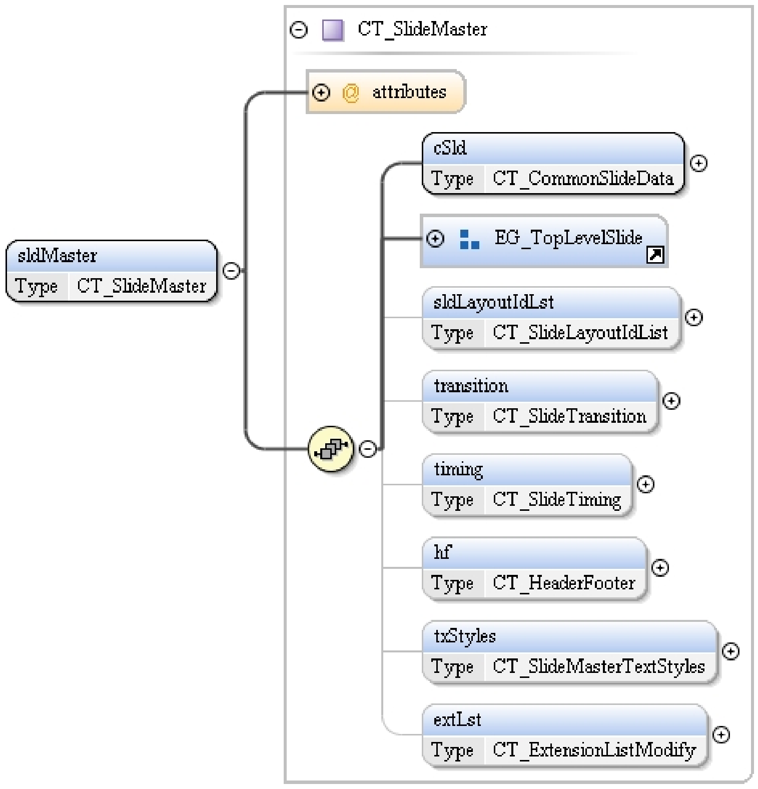


Figure 92 — XML structure of <*sldMaster*> in OOXML

7.4.3.3 Summary

A presentation document contains one or more slide master parts. Subclause 5.4.9 introduces a related use case. Detailed information is given in subclause 6.4.4.

In OOXML a slide master is associated to a layout list. The slide layout contains a template slide design that can be applied to any existing slide. When applied to an existing slide all corresponding content should be mapped to the new slide layout. In ODF a master and a layout contain different style information. A slide refers to a master and a layout.

7.4.4 Animations

7.4.4.1 Animations in ODF

As shown in Figure 88 each page has an optional <*animations*> element, which is a container for animation effects. The structure of the <*animations*> element is shown as Figure 93. If there is a <*show-shape*> element for a shape, this shape is automatically invisible before the effect is executed. The attributes of the <*show-shape*> and <*show-text*> elements are used to specify the *@id* of the shape/text using this effect together with the type, direction, speed and other properties of the effect. The elements <*hide-shape*> and <*hide-text*> make a shape and the text of a shape invisible. The element <*dim*> fills a shape in a single colour. The element <*play*> starts the animation of a shape that supports animation. The element <*sound*> may be used in all animation effects that support sound. The sound file referenced by the *@XLink* attribute is played when the effect is executed.

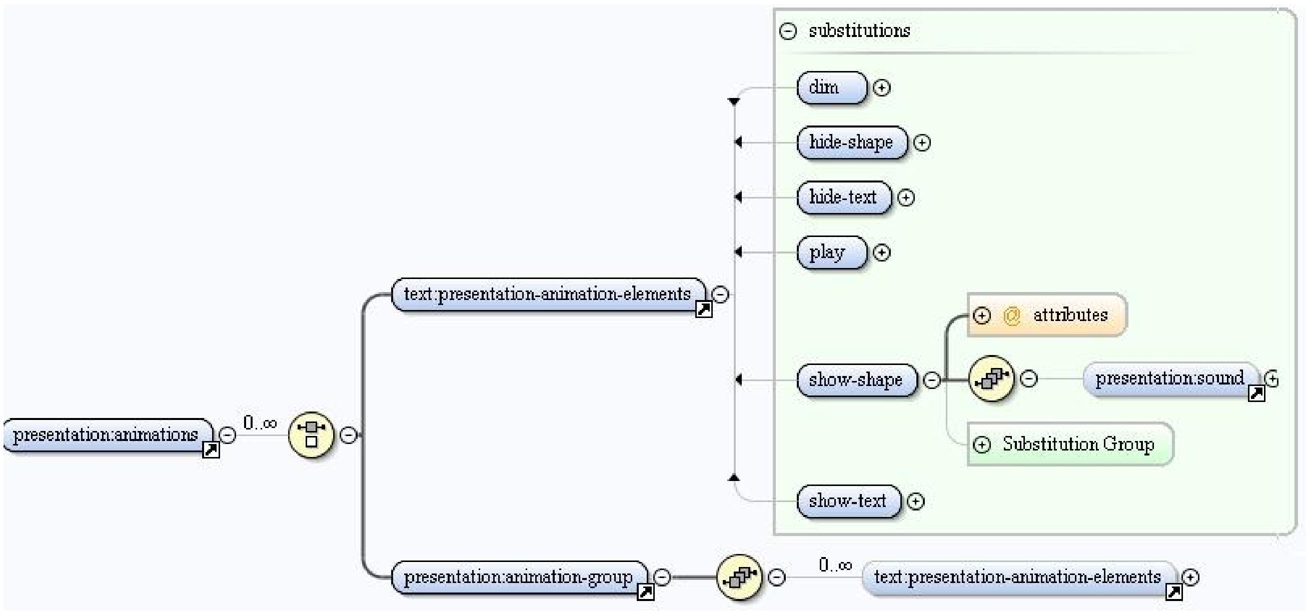


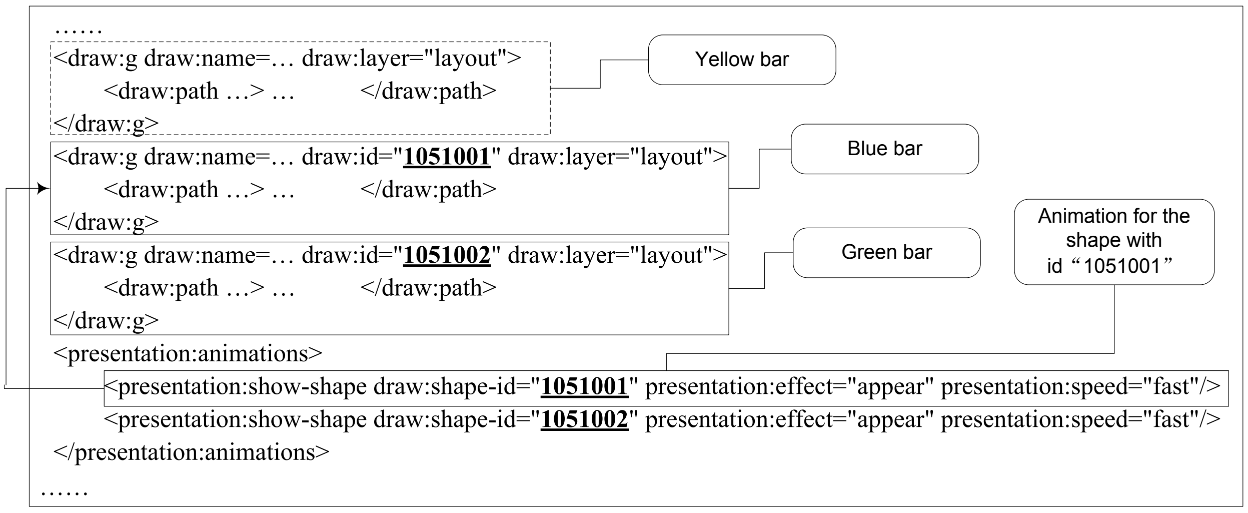
Figure 93 — XML structure of <*animations*> in ODF

Sample XML code of the use case introduced in subclause 5.4.6 is shown in Figure 94. The yellow bar has no animation and the blue and green bars appear one after the other with certain effects which are triggered by a mouse click or shown at timed intervals. The elements <*show-shape*> with id of “1051001” and “1051002” refer to <*g*> elements with the same *@id*.

Synchronized Multimedia Integration Language (SMIL) based shape animations can be used instead of animations specified by the <*animations*> elements if one of the following features is required:

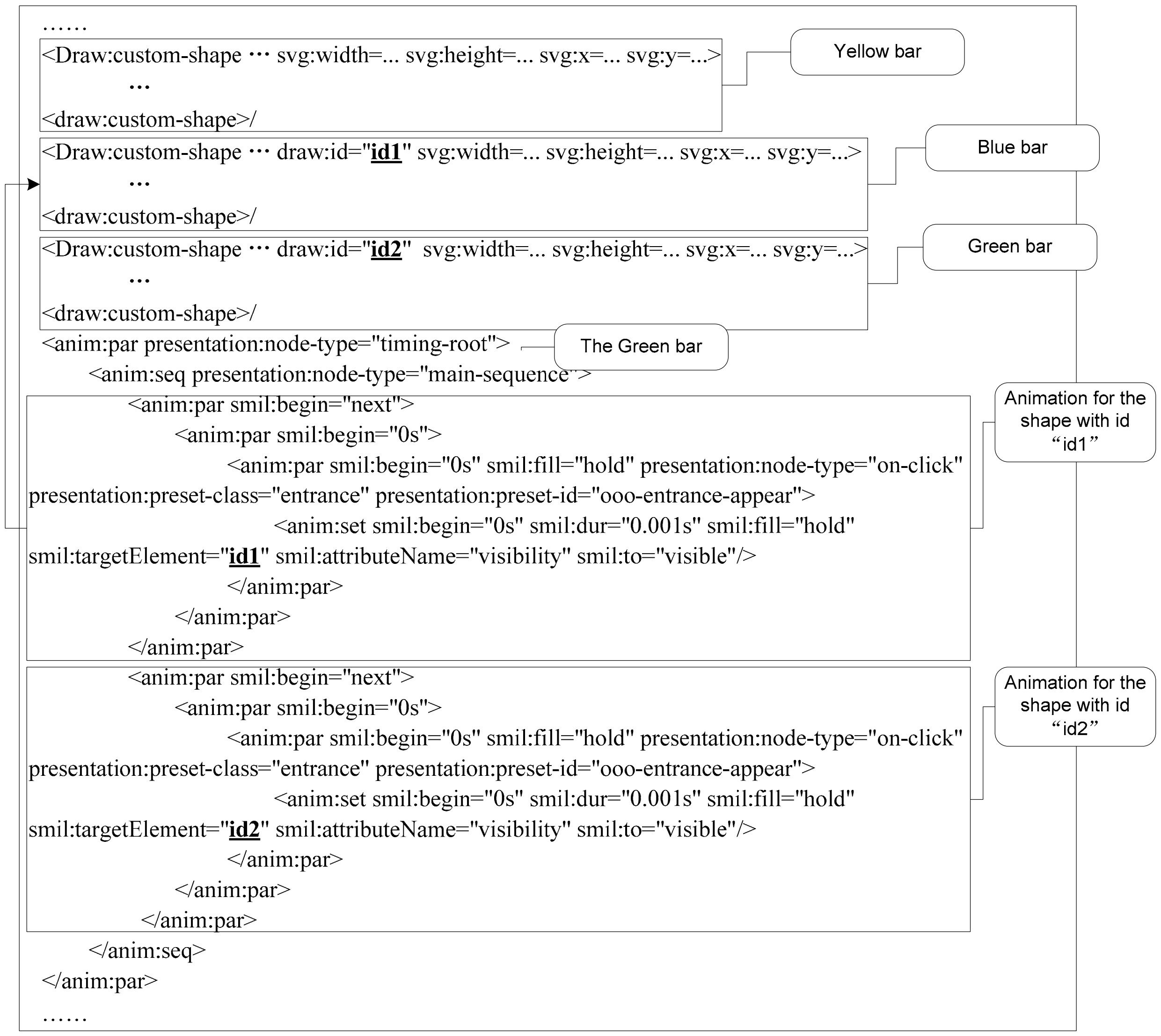
- Multiple animations per shape;
- A mixture of animations starting upon user interaction and starting automatically with each page;
- Multiple animations running at the same time;
- Additional effects "programmed" in XML by combining basic animation elements;
- Document transformations to SVG including SMIL.

The XML code from the same use case using SMIL is shown in Figure 95.



```
……
<draw:g draw:name=… draw:layer="layout">
      <draw:path …> …        </draw:path>
</draw:g>
<draw:g draw:name=… draw:id="1051001" draw:layer="layout">
      <draw:path …> …        </draw:path>
</draw:g>
<draw:g draw:name=… draw:id="1051002" draw:layer="layout">
      <draw:path …> …        </draw:path>
</draw:g>
<presentation:animations>
      <presentation:show-shape draw:shape-id="1051001" presentation:effect="appear" presentation:speed="fast"/>
      <presentation:show-shape draw:shape-id="1051002" presentation:effect="appear" presentation:speed="fast"/>
</presentation:animations>
……
```

Figure 94 — Excerpt from the XML code of <*animations*> in ODF



```
……
<Draw:custom-shape ⋯ svg:width=... svg:height=... svg:x=... svg:y=...>
         …
<draw:custom-shape>/
<Draw:custom-shape ⋯ draw:id="id1" svg:width=... svg:height=... svg:x=... svg:y=...>
         …
<draw:custom-shape>/
<Draw:custom-shape ⋯ draw:id="id2"  svg:width=... svg:height=... svg:x=... svg:y=...>
         …
<draw:custom-shape>/
<anim:par presentation:node-type="timing-root">
      <anim:seq presentation:node-type="main-sequence">
            <anim:par smil:begin="next">
                  <anim:par smil:begin="0s">
                        <anim:par smil:begin="0s" smil:fill="hold" presentation:node-type="on-click"
presentation:preset-class="entrance" presentation:preset-id="ooo-entrance-appear">
                              <anim:set smil:begin="0s" smil:dur="0.001s" smil:fill="hold"
smil:targetElement="id1" smil:attributeName="visibility" smil:to="visible"/>
                        </anim:par>
                  </anim:par>
            </anim:par>
            <anim:par smil:begin="next">
                  <anim:par smil:begin="0s">
                        <anim:par smil:begin="0s" smil:fill="hold" presentation:node-type="on-click"
presentation:preset-class="entrance" presentation:preset-id="ooo-entrance-appear">
                              <anim:set smil:begin="0s" smil:dur="0.001s" smil:fill="hold"
smil:targetElement="id2" smil:attributeName="visibility" smil:to="visible"/>
                        </anim:par>
                  </anim:par>
             </anim:par>
      </anim:seq>
</anim:par>
……
```

Yellow bar
Blue bar
Green bar
The Green bar
Animation for the shape with id "id1"
Animation for the shape with id "id2"

Figure 95 — Excerpt from the XML code of a slide using SMIL animation in ODF

7.4.4.2 Animations in OOXML

In OOXML an animation describes all animation effects that are defined for one slide and also the animation effects that occur during a transition between slides. Animations on one slide are inherently time based and consist of animation effects on objects or text. Slide transition effects appear before any animation on a slide. All elements described in the animations are contained in the <*transition*> and <*timing*> elements within the slide <*sld*> element.

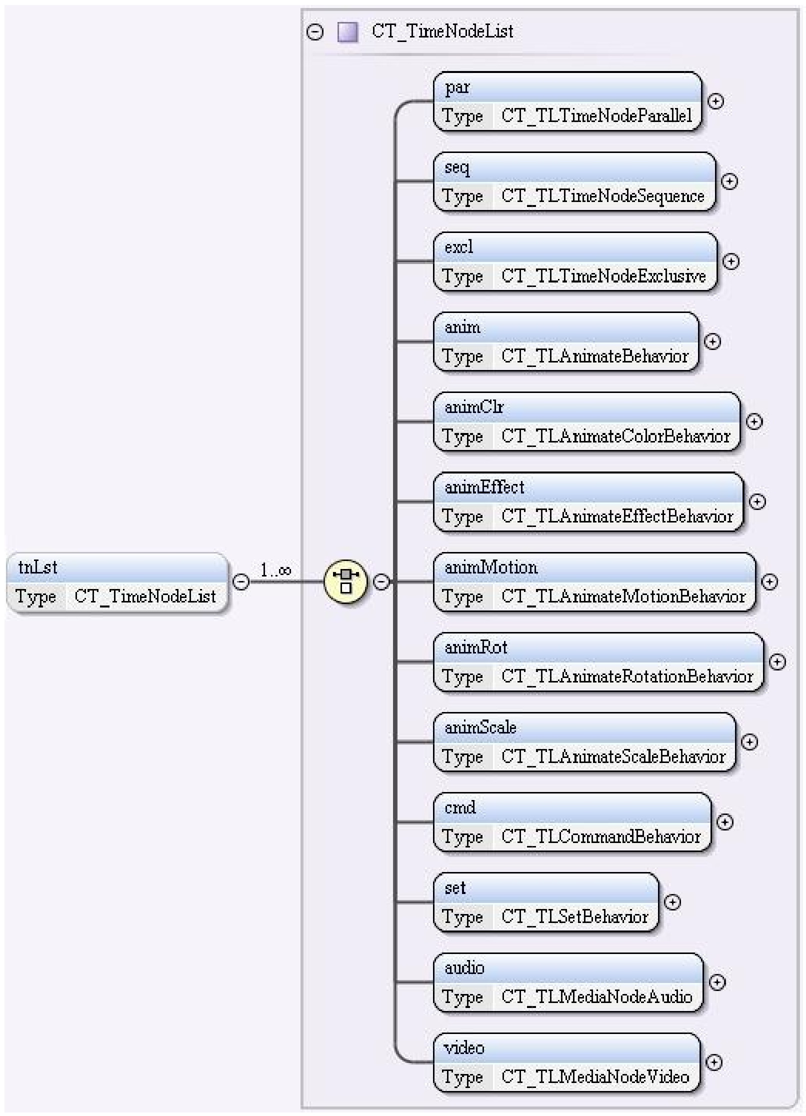

CT_TimeNodeList
tnLst
Type CT_TimeNodeList
1..∞
par
Type CT_TLTimeNodeParallel
seq
Type CT_TLTimeNodeSequence
excl
Type CT_TLTimeNodeExclusive
anim
Type CT_TLAnimateBehavior
animClr
Type CT_TLAnimateColorBehavior
animEffect
Type CT_TLAnimateEffectBehavior
animMotion
Type CT_TLAnimateMotionBehavior
animRot
Type CT_TLAnimateRotationBehavior
animScale
Type CT_TLAnimateScaleBehavior
cmd
Type CT_TLCommandBehavior
set
Type CT_TLSetBehavior
audio
Type CT_TLMediaNodeAudio
video
Type CT_TLMediaNodeVideo

Figure 96 — XML structure of <*tnLst*> in OOXML

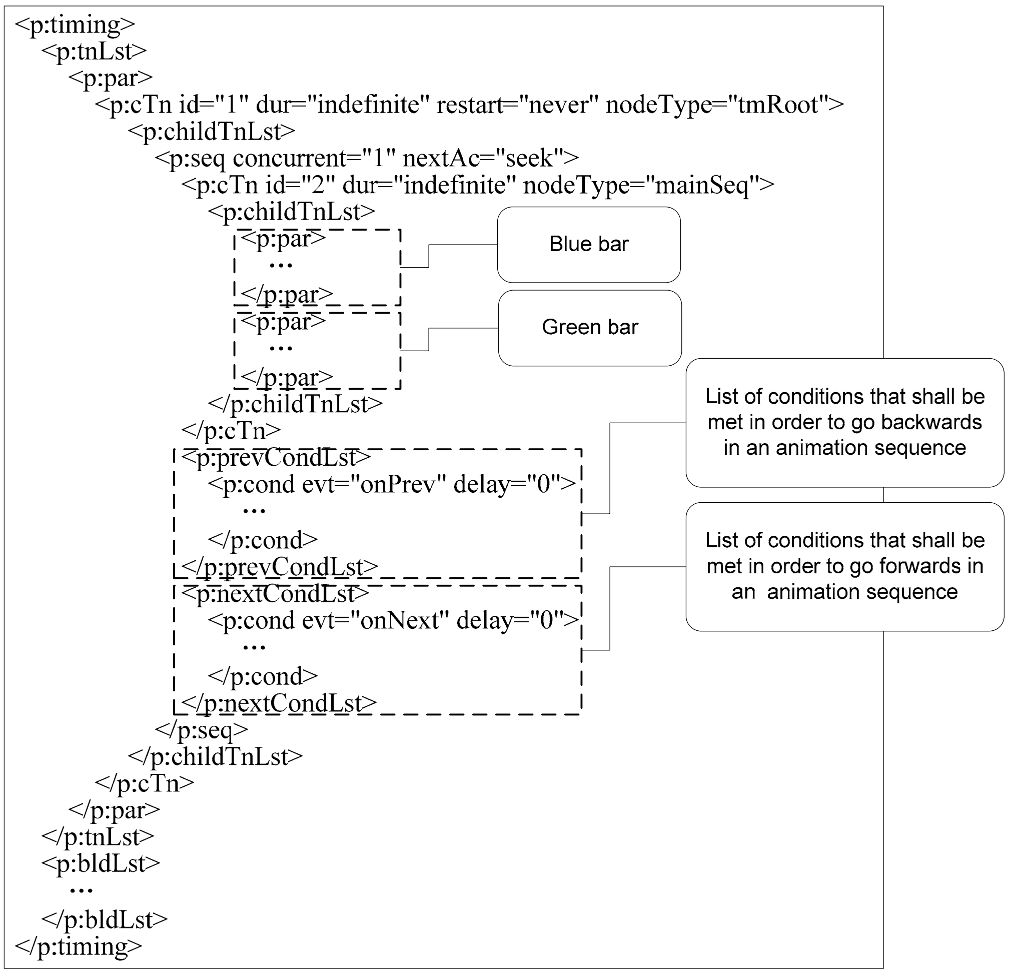
In OOXML a <*timing*> element specifies the timing information for handling all animations and timed events within the corresponding slide. The information is tracked via time nodes within the <*timing*> element, which includes three sub elements:

- The <*extLst*> element specifies the extension list with modification abilities.
- The <*bldLst*> element specifies the list of graphic elements to build. It defines how the different sub shapes or sub components of an object like text, diagrams, and charts are displayed.
- The <*tnLst*> element specifies a list of time node elements used in an animation sequence. It describes the animation behaviors and the timeline. The structure of the <*tnLst*> element is shown in

Figure 96. The timeline is an important aspect for animations on a slide. It moderates the amount of time that the animations use from beginning to end. A timeline is composed of timing nodes that define at which point a certain animation is shown. There are three types of time nodes:

- *Parallel* (<*par*>); a parallel time node that can be activated along with other parallel time nodes.
- *Sequence* (<*seq*>); a sequence time node that can only be activated when the preceding node finishes.
- *Exclusive* (<*excl*>); a time node that is used to suspend all other timelines when it is activated.

The XML code of the use case introduced in subclause 5.4.6 is shown in Figure 97.



```
<p:timing>
  <p:tnLst>
    <p:par>
      <p:cTn id="1" dur="indefinite" restart="never" nodeType="tmRoot">
        <p:childTnLst>
          <p:seq concurrent="1" nextAc="seek">
            <p:cTn id="2" dur="indefinite" nodeType="mainSeq">
              <p:childTnLst>
                <p:par>
                  ...
                </p:par>
                <p:par>
                  ...
                </p:par>
              </p:childTnLst>
            </p:cTn>
            <p:prevCondLst>
              <p:cond evt="onPrev" delay="0">
                ...
              </p:cond>
            </p:prevCondLst>
            <p:nextCondLst>
              <p:cond evt="onNext" delay="0">
                ...
              </p:cond>
            </p:nextCondLst>
          </p:seq>
        </p:childTnLst>
      </p:cTn>
    </p:par>
  </p:tnLst>
  <p:bldLst>
    ...
  </p:bldLst>
</p:timing>
```

Blue bar
Green bar
List of conditions that shall be met in order to go backwards in an animation sequence
List of conditions that shall be met in order to go forwards in an animation sequence

Figure 97 — Excerpt from the XML code of animated slides in OOXML

7.4.4.3 Summary

Shapes and other graphic entities can be animated in a presentation document. The animation is executed when the slide is displayed during a presentation. Subclause 5.4.6 introduces a related use case. Detailed information is given in subclause 6.4.2.

ODF defines and supports presentation animation in two ways. Each slide can have an optional <*animations*> element for simple animation effects or SMIL can be used for complex animations. The schema for animation in OOXML is loosely based on the syntax and concepts from SMIL. The definition of animations in OOXML is more complex than in ODF and allows more effects to be defined.

7.5 Summary

Clause 7 analyses and compares the XML representations of several features of ODF and OOXML documents. Although ODF and OOXML use many similar concepts like *paragraph*, *table*, or *master*, the XML representations of these concepts are different; this reduces the translation fidelity between the two International Standards. According to these differences the complexity of corresponding translations between OOXML and ODF can be divided into three types.

- *Easy type*; components of this type have direct and obvious relationships. They are easy to translate from one format into the other. For example, simple paragraphs and tables are of the *easy type*.
- *Moderate type*; components of this type do not correspond directly with each other or they use different XML structures. From a logical point of view, most components and features have corresponding components and features. For example, page layout can be translated with moderate effort.
- *Difficult type*; components of this type are very difficult to translate or cannot be translated at all. OOXML and ODF use different concepts and design ideas to implement these components and their features. For example, change tracking and collaborations are difficult to translate. While tracked changes are represented by one element in ODF, OOXML uses nearly twenty elements to track the changes of different elements such as run, paragraph, table and numbering.

8 Translation

8.1 Introduction

This section describes the three types of *translation complexity* introduced in the previous section and gives typical examples of each type. Guidelines for evaluating translatability between the two International Standards will be derived from the use cases introduced in clause 5, the evaluation of features and functions in clause 6, and the examples of structures, features and translations given in clauses 7 and 8.

8.2 Translation complexity

As explained in clause 7, ODF and OOXML use different approaches to describe the logical structure and features of documents. Similar features and the corresponding functionality of ODF and OOXML documents are implemented in different ways. Therefore it is necessary to define mappings between these implementations in order to be able to translate between both document formats. In this section we define the translation complexities which depend on the complexity of the structures and the translation rules between the two document formats. The translation complexity can be *easy* for 1:1 or 1:n mappings between corresponding elements and attributes, *moderate* for n:m mappings between corresponding elements and attributes, and *difficult* if complex mapping algorithms are required.

The *easy type* uses a 1:1 or 1:n mapping between the two formats. In a 1:1 mapping XML elements and attributes can be mapped directly to corresponding entities. In a 1:n mapping one XML element or attribute in format A can be mapped to n elements attributes in format B respectively. In other words, when a translation occurs, format A’s features and functions can be translated directly to format B's features and functions and vice versa. Figure 98 shows an example of a 1:1 mapping or 1:n mapping type.

In the following three figures, A and B denote *formats*. The colours dark and light blue represent structures such as *elements* and *attributes*. Shapes represent *features*. For instance, if A denotes ODF then B denotes OOXML and vice versa. Both shapes in Figure 98 are circles. This implies that the two formats use similar structures. For this structure format A is a superset of B which means the elements and attributes in format B can be mapped to format A, while format A's elements and attributes can be partially mapped to format B.

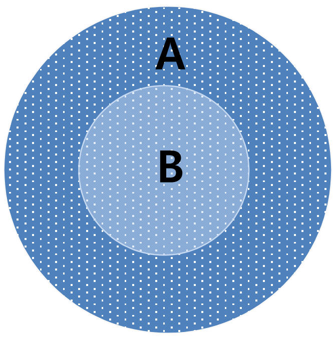


Figure 98 — *Easy* translation complexity

The *moderate type* uses n:m mapping. Both formats support similar features. As depicted in Figure 99, the features are implemented using different elements or attributes in the two formats. In other words when a translation occurs a feature in format A, consisting of n elements and attributes, will be mapped to a corresponding feature in format B, consisting of m elements and attributes.

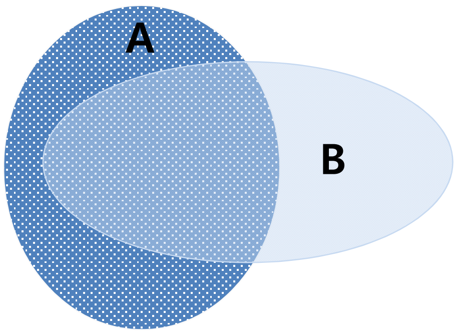


Figure 99 — *Moderate* translation complexity

The *difficult type* uses complex algorithms for translations of features between both formats. As depicted in Figure 100, the features in formats A and B are different even though the elements and attributes used for their implementation in each format can overlap. Therefore it is only possible to implement a feature existing in format A by applying a complex algorithm that works with format B elements and attributes. This mapping is only possible in the direction from format A to format B, round tripping is only possible if the translation processes is traced within the document. In the worst case a feature cannot be translated between the two formats.

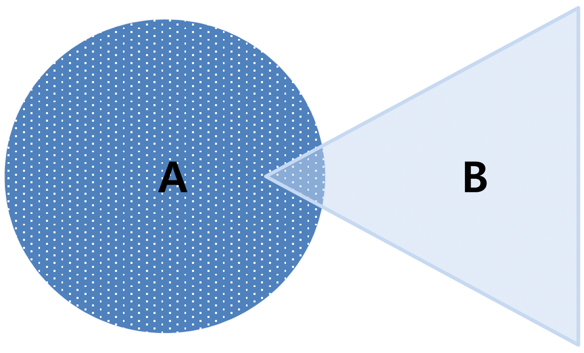


Figure 100 — *Difficult* translation complexity

8.3 Sample translations

8.3.1 Easy translation

8.3.1.1 Text and paragraph formatting

In this translation example, we will consider common functionalities of text and paragraph formatting in word processing documents. The selected functionalities are bold, italic, underline, strikethrough, font size, line spacing, and justification. Figure 101 shows which functionalities of text and paragraph formatting are used in the sample sentence. Figure 102 and Figure 103 show the sample sentence in ODF and OOXML using the same text and paragraph formatting functionalities.

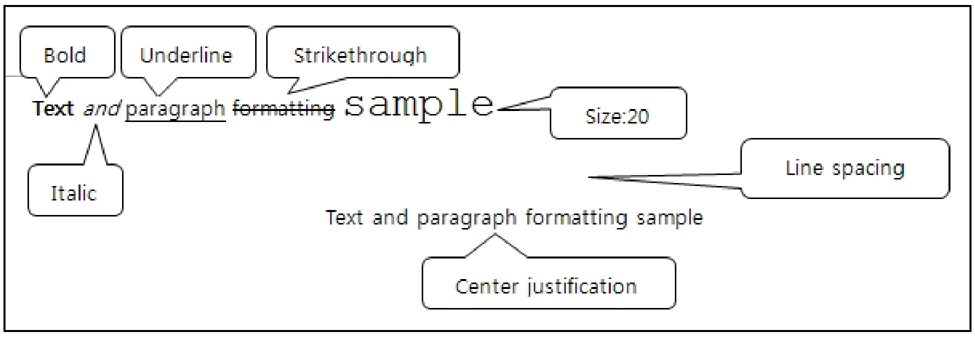

Bold
Underline
Strikethrough
Text and paragraph formatting sample
Size:20
Italic
Line spacing
Text and paragraph formatting sample
Center justification

Figure 101 — Selected text formatting functionalities

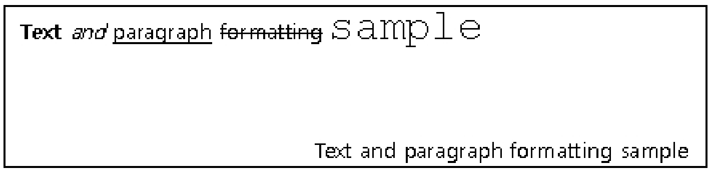

Text and paragraph formatting sample
Text and paragraph formatting sample

Figure 102 — Text formatting of an ODF document

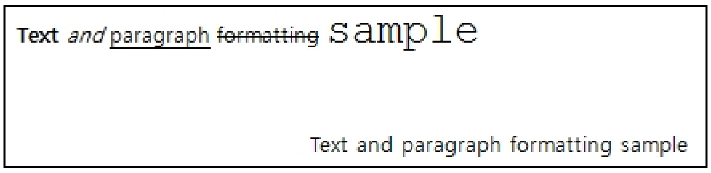

Text and paragraph formatting sample
Text and paragraph formatting sample

Figure 103 — Text formatting of an OOXML document

Although the selected functionality is the same in both formats, the structures of the instance documents are different. As described in subclause 7.2.3 and depicted in Figure 104, OOXML mixes styles and content in serial order while ODF uses separate elements for styles and contents.

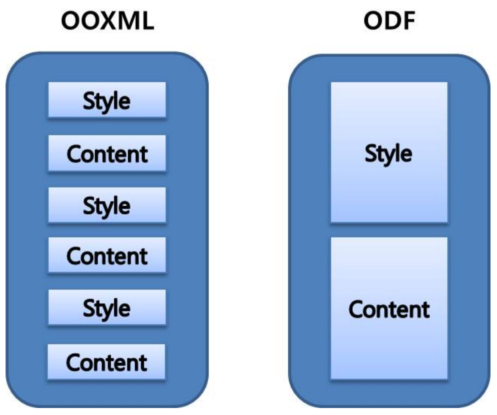


Figure 104 — Paragraph formatting in OOXML and ODF

Table 13 shows how paragraph formatting can be translated between the OOXML elements and the corresponding ODF elements and attributes.

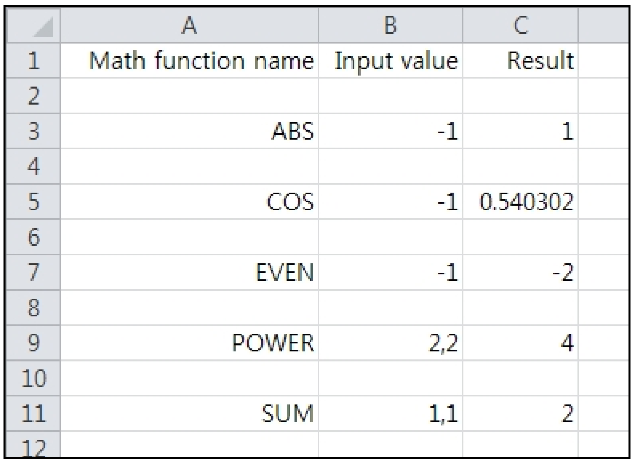
Table 13 — Translation rules for paragraph formatting

Functionality	OOXML	ODF
Bold	Element : b	Element- <*style:text-properties*> Attribute- @*fo:font-weight*="bold"
Italic	Element: i	Element- <*style:text-properties*> Attribute- @*fo:font-style*="italic"
Underline	Element: u	Element- <*style:text-properties*> Attribute- @*style:text-underline-style*="solid"
Strikethrough	Element: strike	Element- <*style:text-properties*> Attribute- @*style:text-line-through-style*="solid"
Font size	Element: sz	Element- <*style:text-properties*> Attribute- @*fo:font-size*="20pt"
Justification	Element: jc	Element- <*style:paragraph-properties*> Attribute- @*style:justfy-single-word*="false"
Line spacing	Element: spacing	Element- <*style:paragraph-properties*> Attribute- @*fo:line-height*="250%"

This example is characterized by a simple mapping between attributes, but not every "source has a target": some attributes or attribute values can't be translated. Therefore the visual appearance of the translated documents may be different and the result of round tripping will not always be an identical document.

8.3.1.2 Math functions in spreadsheets

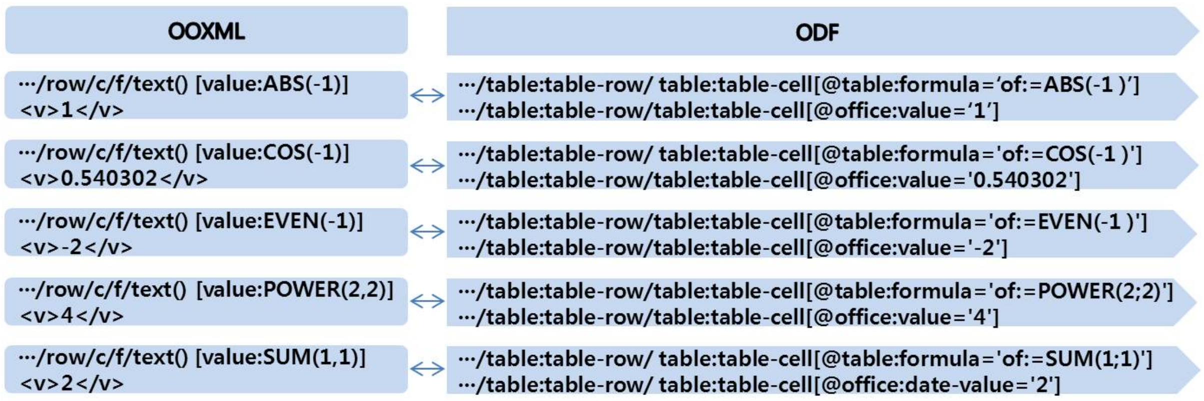
This example focuses on the translation of mathematical functions such as ABS, COS, EVEN, POWER and SUM that are typically used in spreadsheets as shown in Figure 105. The sample mathematical functions of both formats can be almost identically translated between both formats.



	A	B	C
1	Math function name	Input value	Result
2			
3	ABS	-1	1
4			
5	COS	-1	0.540302
6			
7	EVEN	-1	-2
8			
9	POWER	2,2	4
10			
11	SUM	1,1	2
12			

Figure 105 — Sample mathematical functions in a spreadsheet

As explained in subclauses 7.3.2 and 7.3.4 the implementations of mathematical functions in the two formats are different. OOXML uses predefined formulas and the element <*f*> while ODF uses the <*table-cell*> element and the @*formula* attribute. Another difference is that ODF does not support value types to distinguish types such as *floats* and *strings* while OOXML does. Figure 106 shows the elements, attributes and values that are used in the two formats.



OOXML		ODF
…/row/c/f/text() [value:ABS(-1)] <v>1</v>	⇔	…/table:table-row/ table:table-cell[@table:formula=‘of:=ABS(-1)’] …/table:table-row/table:table-cell[@office:value=‘1’]
…/row/c/f/text() [value:COS(-1)] <v>0.540302</v>	⇔	…/table:table-row/ table:table-cell[@table:formula='of:=COS(-1)'] …/table:table-row/table:table-cell[@office:value='0.540302']
…/row/c/f/text() [value:EVEN(-1)] <v>-2</v>	⇔	…/table:table-row/table:table-cell[@table:formula='of:=EVEN(-1)'] …/table:table-row/table:table-cell[@office:value='-2']
…/row/c/f/text() [value:POWER(2,2)] <v>4</v>	⇔	…/table:table-row/table:table-cell[@table:formula='of:=POWER(2;2)'] …/table:table-row/table:table-cell[@office:value='4']
…/row/c/f/text() [value:SUM(1,1)] <v>2</v>	⇔	…/table:table-row/ table:table-cell[@table:formula='of:=SUM(1;1)'] …/table:table-row/ table:table-cell[@office:date-value='2']

Figure 106 — Translation of mathematical functions in a spreadsheet

Table 14 lists the corresponding elements, formulas and attributes for the sample function in both formats. It can be recognized that an easy 1:1 translation can be defined between the corresponding entities.

Table 14 — Mathematical functions and corresponding entities in OOXML and ODF

Formula	OOXML	ODF
ABS	Element : f Predefined formula: <f>ABS(-1)</f>	Element- *<table:table-cell>* Attribute- *@table:formula*="of:=ABS(-1)"
COS	Element: f Predefined formula: <f>COS(-1)</f>	Element- *<table:table-cell>* Attribute- *@table:formula*="of:=COS(-1)"
EVEN	Element: f Predefined formula: <f>EVEN(-1)</f>	Element- *<table:table-cell>* Attribute- *@table:formula*="of:=EVEN(-1)"
POWER	Element: f Predefined formula: <f>POWER(2,2)</f>	Element- *<table:table-cell>* Attribute- *@table:formula*="of:=POWER(2;2)"
SUM	Element: f Predefined formula: <f>SUM(1,1)</f>	Element- *<table:table-cell>* Attribute- *@table:formula*="of:=SUM(1;1)"

Many mathematical functions exist in both formats and can be translated easily. However, as described in subclause 6.3.3 and shown in Table 15, some functions exist only in one format. Some functions can be computed from an existing function, for example COT = COS/SIN. Other functions only exist in one format and cannot be translated. There the translation *complexity* for mathematical functions is easy but the translatability *level* is only medium.

Table 15 — Mathematical functions that exist only in one format[14]

OOXML only	ODF only
MDETERM	BESSELI
MINVERSE	BESSELJ
MMULT	BESSELK
ROMAN	BESSELY
SUMPRODUCT	COMBINA
SUMX2MY2	CONVERT
SUMX2PY2	CONVERT_ADD
SUMXMY2	COT
	COTH
	COUNTBLANK
	COUNTIF
	DELTA
	ERF
	ERFC
	GCD_ADD
	GESTEP
	ISEVEN
	ISODD
	LCM_ADD

[14] ODF 1.0 does not define any formula language. For this reason the list of ODF functions has been generated by OpenOffice.org, the quasi reference implementation of ODF 1.0. A complete list of the supported mathematical functions including several functions that are not supported by OpenOffice.org is introduced in ODF 1.2.

This example is characterized by a simple mapping between elements and attributes but not every “source has a target”. Some elements or attributes can’t be translated. Therefore the structure and content of translated documents may be different and the result of round tripping will not always be an identical document.

8.3.2 Moderate translation

8.3.2.1 Slide blending and effects in presentations

Figure 45 illustrates slide blending in presentation documents. Slide blending and effects in presentations are usually composed of a *transition type*, *duration time* and *directions* of slide effects. As explained in subclause 7.4.4, ODF and OOXML have different structures to express various types of slide blending and effects. ODF uses elements such as <*transition*> and <*transitionFilter*> for slide blending and attributes such as @*type*, @*subtype*, @*direction* and @*dur* to control start time, duration time, effect directions and repetition. OOXML uses <transition> elements as parents and <*blinds*>, <*circle*>, <*dissolve*>, <*pull*>, and <*push*> elements as children in order to define a transition type. Each transition type has an attribute @*dir* to define the direction of the effect. Another difference is that ODF may use SMIL. Figure 107 and Figure 108 illustrate the difference of the two formats.

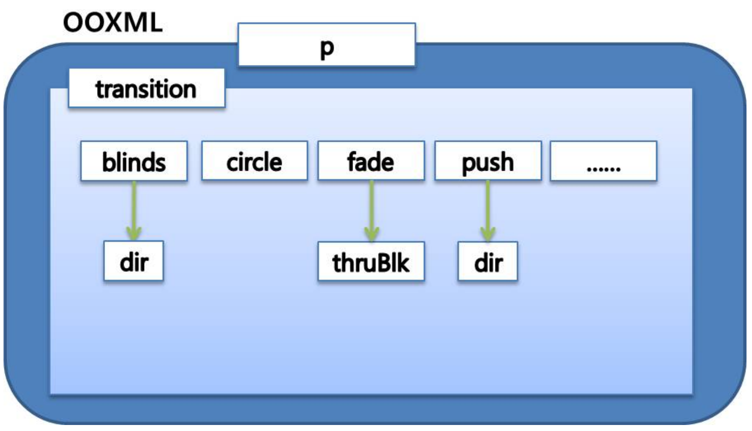


Figure 107 — Slide blending and effects in OOXML

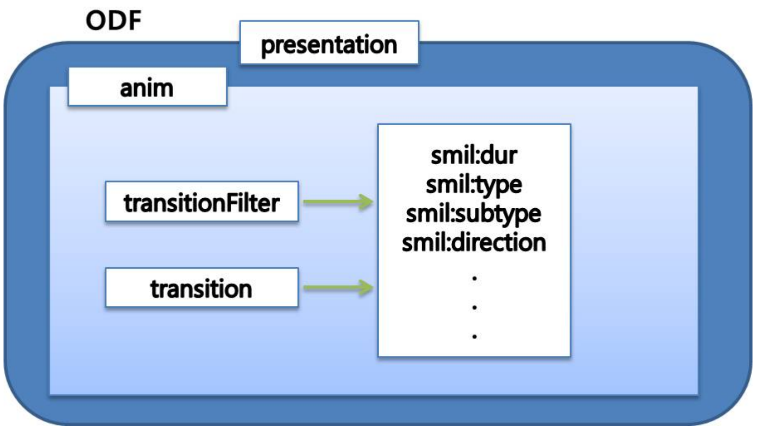


Figure 108 — SMIL based slide blending and effects in ODF

Figure 109 shows the mapping of slide blending and effects that are used in the example described above.

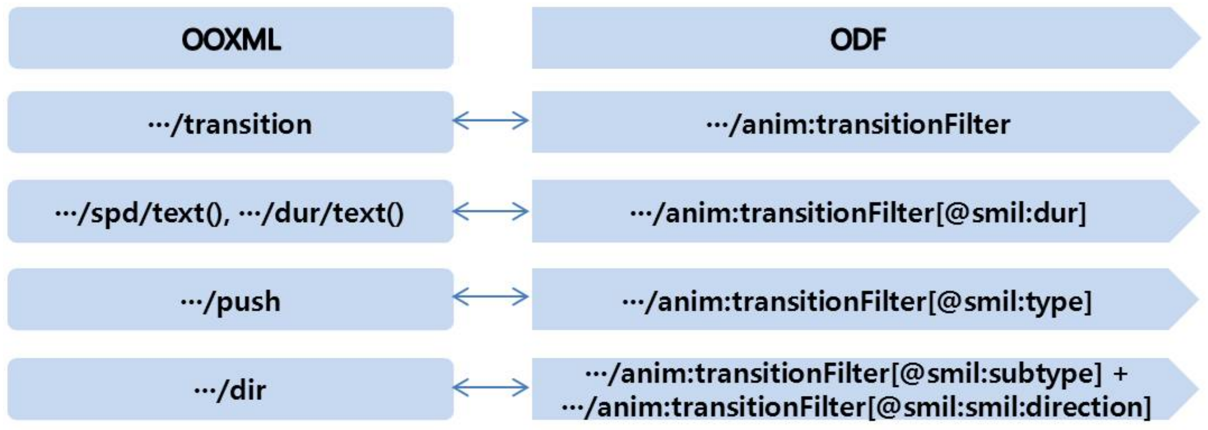

OOXML
ODF
…/transition
…/anim:transitionFilter
…/spd/text(), …/dur/text()
…/anim:transitionFilter[@smil:dur]
…/push
…/anim:transitionFilter[@smil:type]
…/dir
…/anim:transitionFilter[@smil:subtype] +
…/anim:transitionFilter[@smil:smil:direction]

Figure 109 — Translations of slide blending

This example is characterized by the implementation of similar features using different logical structures. Therefore it is necessary to define translation rules between corresponding sets of elements and attributes. Due to the different expressiveness of both formats it is not possible to translate any slide blending and effect.

8.3.2.2 Headers and footers in word processing documents

The logical structure and translatability of headers and footers has been discussed in subclauses 7.2.8 and 6.2.3. OOXML uses a <*section*> element which is a child of the <*body*> element. A section defines page layout, headers and footers. The header and footer refer to header.xml and footer.xml files through <*relationship*> elements as shown in Figure 110. ODF uses a reference to a <*style*> file inside the <*body*> element. The <*style*> file defines the <*page-layout*> element that in turn defines <*header style*> and <*footer style*> as shown in Figure 111.

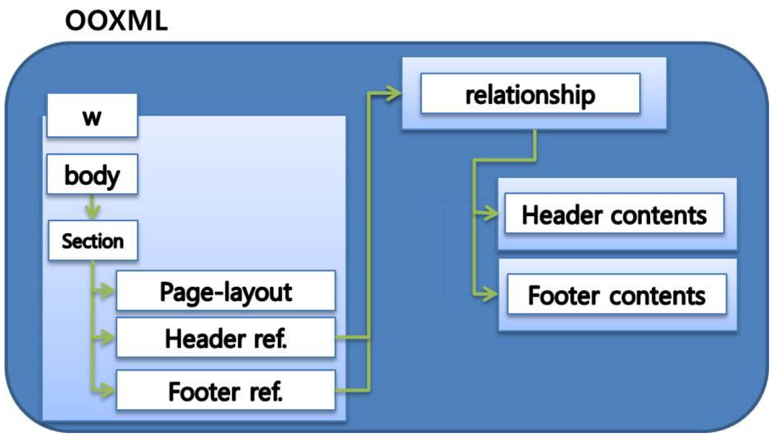


Figure 110 — Headers and footers in OOXML

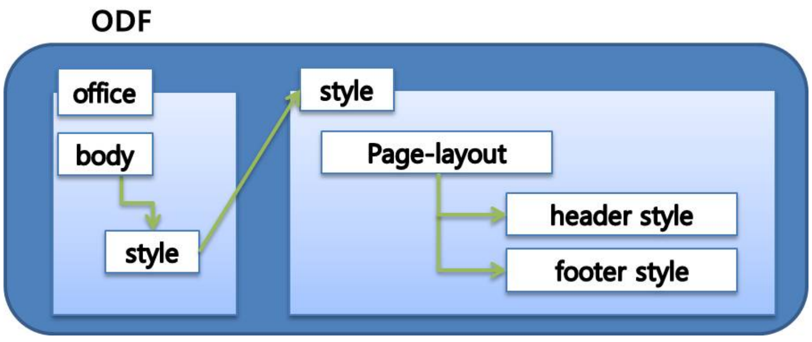
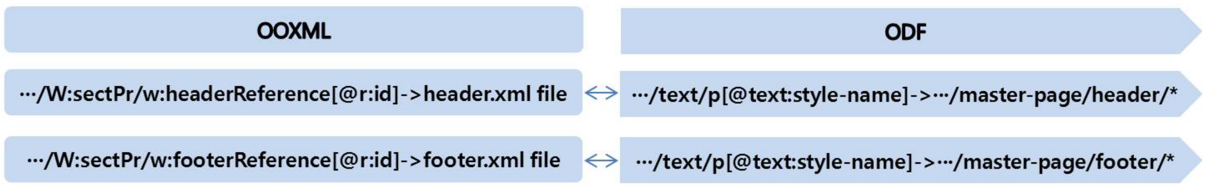


Figure 111 — Headers and footers in ODF

Figure 112 illustrates sample mappings of headers and footers between both formats.



OOXML		ODF
…/W:sectPr/w:headerReference[@r:id]->header.xml file	<->	…/text/p[@text:style-name]->…/master-page/header/*
…/W:sectPr/w:footerReference[@r:id]->footer.xml file	<->	…/text/p[@text:style-name]->…/master-page/footer/*

Figure 112 — Translation of headers and footers

This example is characterized by the implementation of similar features using different concepts including the utilization of different logical and physical structures. Therefore it is necessary to define complex translation rules between corresponding sets of elements and attributes. Because both formats allow different documents parts to be a member of a header or footer is not possible to define a total mapping between both formats.

8.3.2.3 Formatting of spreadsheets

Figure 39 gives an example of formatting features in spreadsheets such as text colour, numbers, and dates. Even though the translatability of these features is high, the concepts used to implement these features are different as shown in Figure 113 and Figure 114.

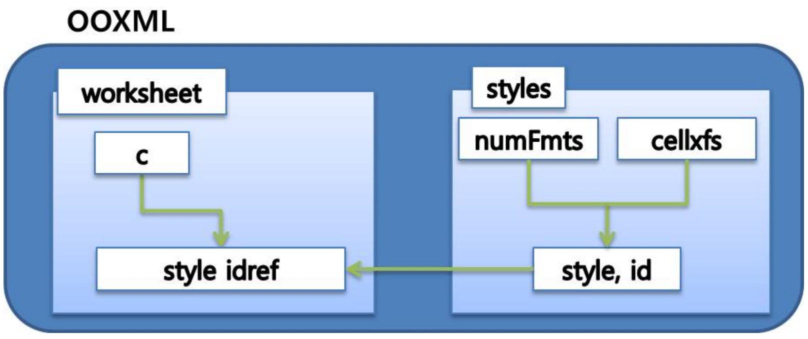


Figure 113 — Formatting of spreadsheets in OOXML

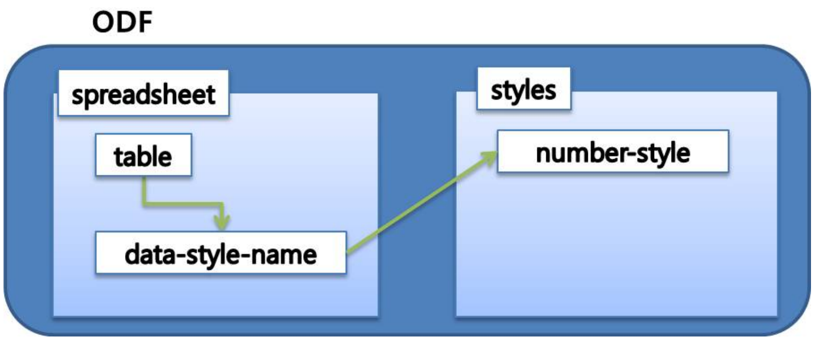


Figure 114 — Formatting of spreadsheets in ODF

OOXML combines the definitions of the formatting features in the <*formatCode*> while ODF distributes the definitions of the formatting features to several elements as shown in Figure 115 and Figure 116. Therefore the <*formatCode*> in OOXML maps to several sub elements of <*number-style*> in ODF.

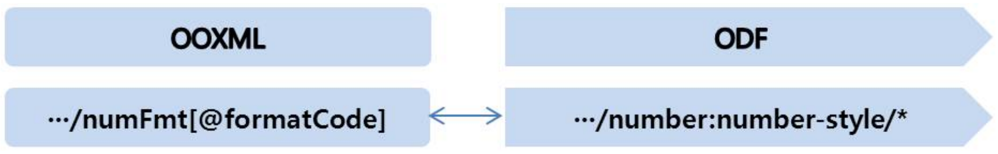
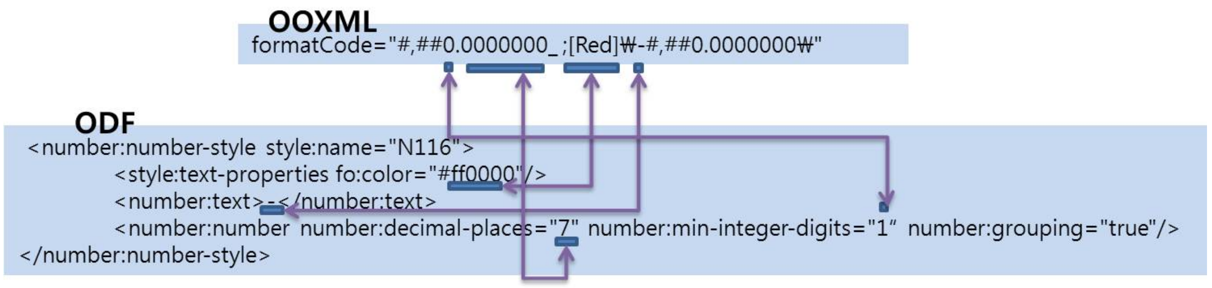


Figure 115 — Translation of formatting features

Figure 116 shows a sample mapping of corresponding formatting features of a spreadsheet document. This example is characterized by the implementation of similar features using different concepts including the utilization of different logical and physical structures. Therefore it is necessary to define complex translation rules between corresponding elements and attribute values.


OOXML

```
formatCode="#,##0.0000000_;[Red]₩-#,##0.0000000₩"
```

ODF

```
<number:number-style style:name="N116">
        <style:text-properties fo:color="#ff0000"/>
        <number:text>-</number:text>
        <number:number number:decimal-places="7" number:min-integer-digits="1" number:grouping="true"/>
</number:number-style>
```

Figure 116 — Translation of formatting features in XML

8.3.2.4 Master layout in presentations

Figure 51 and Figure 52 illustrate the basic templates of master slides and the master layout in both formats. Translatability and the structure of masters have been discussed in subclauses 6.4.4 and 7.4.3. ODF defines <*masterslide*> and <*layout*> separately while OOXML defines <*layout*> as a child element of <*masterslide*> as shown in Figure 117.

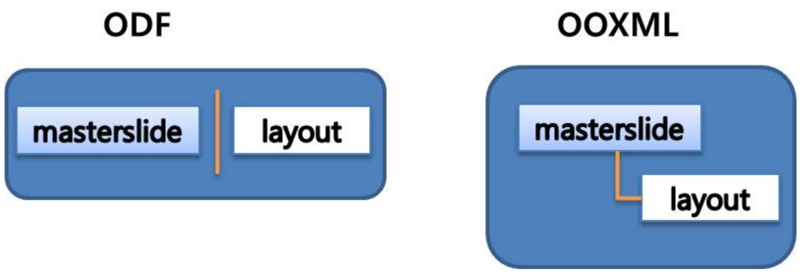


Figure 117 — Master layouts in ODF and OOXML

As depicted in Figure 118 and Figure 119, the assignment of a layout definition to a slide is implemented in different ways in both formats. Layouts assignments in ODF presentation documents are based on three independent entities while layout assignments in OOXML are based on four partially dependent entities.

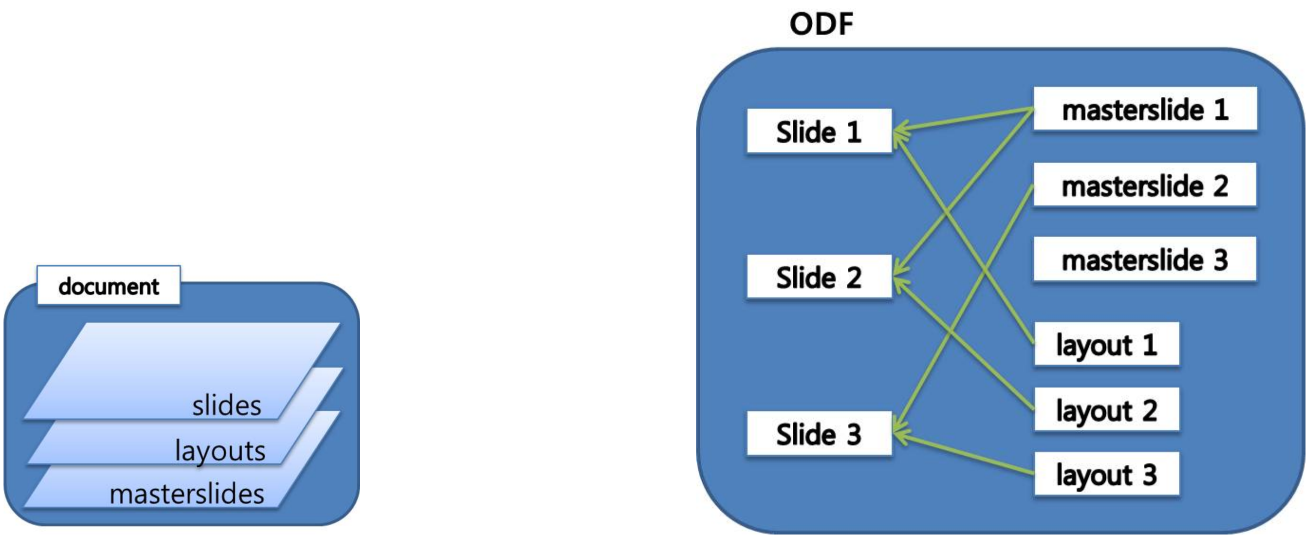


Figure 118 — Assignment of a layout definition in ODF

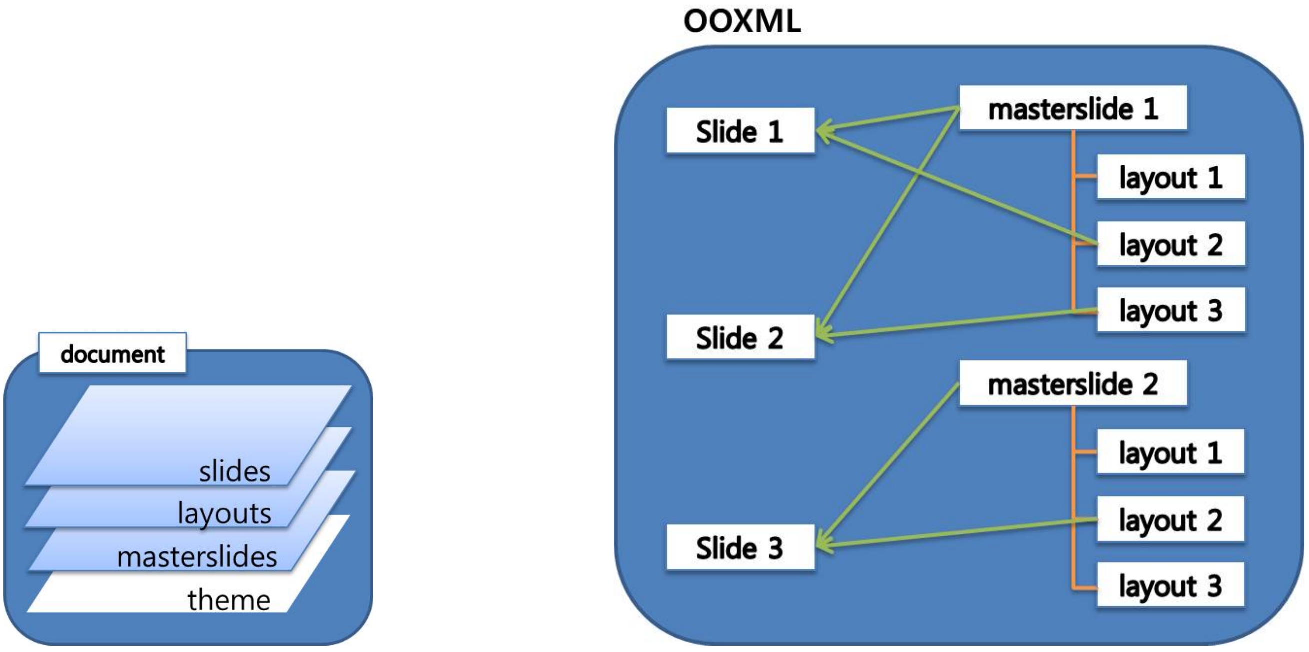
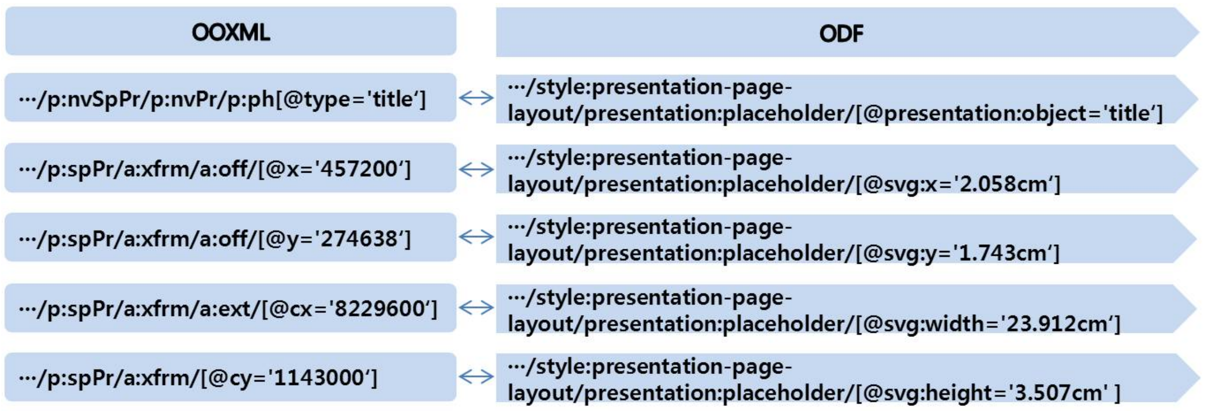


Figure 119 — Assignment of a layout definition in OOXML

In the master slide depicted in Figure 51, the boxes start with the sentence “click to add…” everywhere where the author can insert his textual content. These so-called *placeholders* have to be translated between both formats as shown in the example in Figure 120.



OOXML		ODF
…/p:nvSpPr/p:nvPr/p:ph[@type='title']	<->	…/style:presentation-page-layout/presentation:placeholder/[@presentation:object='title']
…/p:spPr/a:xfrm/a:off/[@x='457200']	<->	…/style:presentation-page-layout/presentation:placeholder/[@svg:x='2.058cm']
…/p:spPr/a:xfrm/a:off/[@y='274638']	<->	…/style:presentation-page-layout/presentation:placeholder/[@svg:y='1.743cm']
…/p:spPr/a:xfrm/a:ext/[@cx='8229600']	<->	…/style:presentation-page-layout/presentation:placeholder/[@svg:width='23.912cm']
…/p:spPr/a:xfrm/[@cy='1143000']	<->	…/style:presentation-page-layout/presentation:placeholder/[@svg:height='3.507cm']

Figure 120 — Translation of placeholders

Table 16 lists the existing placeholders in OOXML and ODF. There are comparable and specific placeholders in both formats. This example is characterized by the implementation of similar features using different concepts including the utilization of different logical and physical structures. Therefore it is necessary to define complex translation rules between corresponding high level concepts and low level elements and attributes.

Table 16 — Placeholder lists in OOXML and ODF

OOXML	ODF
title	title
body	outline
ctrTitle	subtitle
subtitle	text
dt	graphic
sldNum	object
ftr	chart
hdr	table
obj	orgchart
chart	page
tbl	notes
clipArt	handout
dgm	header
media	footer
sldImg	date-time
pic	page-number

8.3.3 Difficult translations

8.3.3.1 Equations

Figure 37 depicts an example of a mathematical equation. ODF represents mathematical equations utilizing MathML while OOXML utilizes the shared markup language OMML. In OOXML shared part types can refer to both MathML and OMML even though OOXML uses only OMML as its native format for formulas. OMML is contained within elements of the document file. In ODF MathML can be stored in an external referenced file as shown in Figure 125.

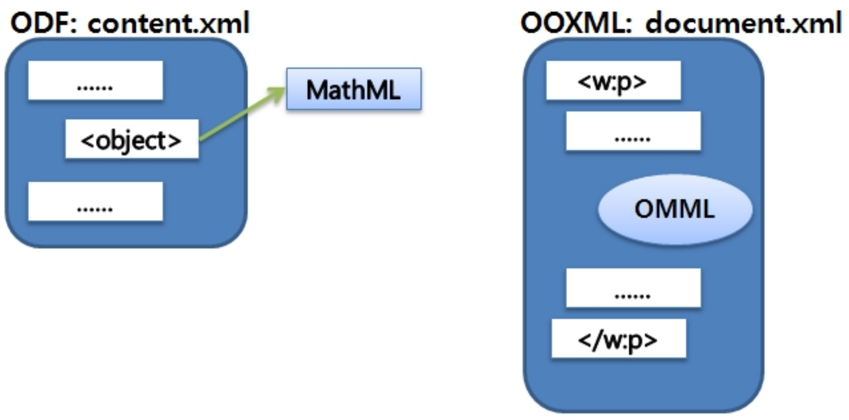


Figure 125 — Equations in ODF and OOXML

This example is characterized by the utilization of different languages to define specific content. Concepts and structure are different in both formats. One possible way to define a translation is to use a graphic representation of the equation. In this case the result of a translation process looks like an equation but it is simply a character string or a graphic and cannot be further edited as an equation. Therefore the definition of translation rules requires complex algorithms or is otherwise impossible.

8.4 Guidelines for evaluating translatability

This TR focuses on an analysis and comparison of the two International Standards ISO/IEC 26300:2006 and ISO/IEC 29500:2008. As already mentioned in the introduction, ISO has published several corrigenda and amendments to both standards. OASIS has published errata and new versions 1.1 and 1.2 of ODF. Therefore this TR can only give a snapshot analysis of the basic versions of both ISO/IEC International Standards. Evaluations of other variants or versions of the two International Standards will produce different evaluation results. Nevertheless the translatability levels and complexity that have been introduced in this report can still be applied.

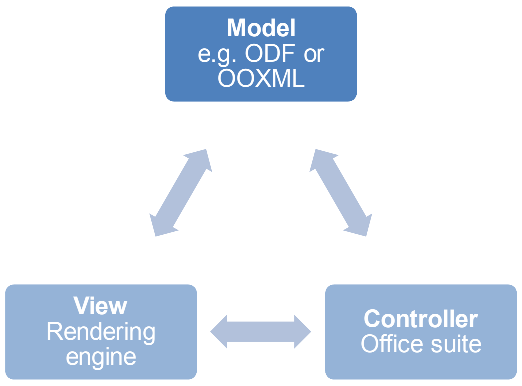


Figure 121 — MVC pattern applied to the architecture of office suites

There is a huge difference between evaluating the translatability between the two International Standards and evaluating the interoperability between office applications. As depicted in the model-view-controller pattern in Figure 121 a comparison between the International Standards focuses on the *model*. If the two models are incompatible it is hard or even impossible to define a translation between certain model elements. The TR focuses directly on the identification and categorization of such incompatibilities between the models.

Nevertheless it is possible to have identical *views* on incompatible parts of the models. The mathematical expression discussed in subclause 8.3.3.1 can have identical views for a MathML, OMML, or even a graphical representation within the models. As a result the structural properties of a document cannot be translated although the visual fidelity of specific translations may be excellent.

Office suites (*controller*) are free to implement a whole International Standard or only a subset of a standard. Additionally they are free to provide some functionality that can be mapped to the standard (model) but that is not directly supported by the International Standard. An example of such behaviour is the support for the CMYK colour model in OpenOffice.org that is mapped to equivalent RGB colours in the model. A user may not be aware which features are implemented by the controller and which features are directly supported by the model. For this reason the combination of model, view and controller, in other words the office suites together with their input and output filters and their rendering engines, have to be improved to give a better impression of document interoperability to the end user.

Every office suite runs in a technical IT-environment and references external resources such as user preferences, dictionaries, glossaries, bibliographies, and font definitions. Documents may refer to external files such as other (sub) documents, XML data, audio, video, and database files. These dependencies have a great impact on the translatability or at least on the view of the documents. Even if it is possible to translate all features that are represented in the document (model) it may happen that the visual fidelity of the target document is poor because the rendering software (view) is running in a different environment. Additionally all files that are referenced from the source document must be available in the environment of the target document and the references have to be adjusted. For these reasons it may be hard for an end user to distinguish between the translatability of standards and the interoperability of applications.

Therefore the evaluation of the translatability of the two document standards covers only one aspect of the general goal of interoperability and every assessment has to distinguish clearly between the different aspects including model, view, and controller.

For these reasons we believe that there is no simple and unique solution to obtaining comprehensive (MVC) interoperability. Translation between document formats covers only one important but quite theoretical aspect of improving document interoperability. The interoperability of the rendering engines and the interoperability of the office suites are of similar importance from a practical, user oriented point of view. However, nobody should expect that documents can be 100% translated between document formats. A generic translator will be able to map approximately (up to) 80% of the most common features. Furthermore, the number of translators grows quadratic considering the number of formats and their versions. Thus the governance of the translators will become a crucial challenge because their number will increase quickly.

8.4.1 Translation fidelity

In subclause 4.2 a set of document properties such as presentation instructions, document content, dynamic content, metadata, annotation & security and document parts & structure have been introduced as a framework for the definition of translation fidelity. Following this approach the translatability of a document has to be evaluated along a number of different dimensions depending on the intention of the translation.

When the focus of the translation process is on the preservation of the *visual appearance* of the document the presentation instructions and the content are of special importance. Typical use cases are short and long time storage of a document and workflows with read-only access to the translated document like displaying the document in different environments and on different devices. Typically the target format of such translations will be PDF/A or some eBook format.

When the focus of the translation process is on the *preservation of content and structure* of the document the static and dynamic content together with the document's parts are of special importance. Typical use cases are workflows with read/write access to the translated document. If the changes performed during the workflow should be traced, annotations and security have to be considered. If the visual appearance should be preserved, the presentation instructions have to be considered, too. These use cases are typical for translations between office formats such as OOXML and ODF.

The value of metadata will probably change during a translation process due to the nature of metadata. Nevertheless the structure of metadata should be preserved during a translation if required.

For these reasons the translation fidelity cannot be measured using an absolute measure. The fidelity depends strongly on the reasons for and the goals of the translation. It is absolutely necessary to know why a document should be translated before the translatability of a document and its features can be evaluated.

The following conclusion can be drawn from the discussion of the *translation complexity* in subclauses 7.5 and 8.2. In case of *easy* and *moderate* complexity a subset of the functionality of format A can be mapped to the equivalent functionality in format B. In most cases round tripping is possible. With *difficult* translations some functionality of format A can be mapped to the corresponding functionality in format B but round tripping and read/write access cannot be ensured.

Whether for example the mapping of an attribute with real values in the interval [0,1] to an integer attribute with the values {0,1} provides sufficient fidelity or not depends on the intention of the document's authors or the application context. Therefore high fidelity can only be guaranteed if a document uses only those features und functionalities that are identical in both formats, even for easy translations. The translatability levels introduced in clause 6 can be used as a guideline to estimate how well a document feature can be translated between both formats.

It is preferable that the developers of office applications provide information about their methods used to implement the various features of the supported International Standards in a manner that Microsoft does for word processing[15], presentation[16], and spreadsheet[17] documents. From this information power users can estimate the set of translatable document features, at least for the considered applications.

8.4.2 Document interoperability

Interoperability of documents depends on a common understanding of the syntax, semantics and structure of document formats. Both ISO/IEC 26300:2006 and ISO/IEC 29500:2008 are based on XML which provides a common syntax. In any case document features expressed by element names and attribute names/values could be ambiguous and inconsistent: Both International Standards don't share the same semantics. Identical functionalities could be implemented utilizing different structures: Both International Standards don't require the same implementation strategy. To achieve interoperability, these differences have to be resolved and appropriate translation rules have to be defined as far as possible. Due to commercial interest, the adaptation of both document standards in the near future seems to be unrealistic. However providing guidelines such as those presented in this Technical Report can improve interoperability and help to avoid unnecessary problems.

Document interoperability is influenced by document standards, application software and user environments. Therefore harmonizing the International Standards is not sufficient to achieve complete interoperability. In order to process a document consistently, the application software has to be harmonized and the user environments have to be aligned. Reference implementations that support specific features are a prerequisite for the definition and evaluation of standardized behaviour and presentation fidelity. Without reference implementations it is impossible to define the intention of the document's producer and without being able to define this intention it is impossible to compare it with the consumer's perception. Interoperability is always measured between different entities, one of which has to act as a reference system.

To improve document interoperability, joint efforts have to be made by developers, users and standards bodies. It is important to:

- Carry out *conformance* tests ensuring that the software is implementing the International Standard in a correct way:
    - Validate documents;
- Carry out *interoperability* tests and assessments ensuring maximum interoperability between documents and application software respectively;
- Provide test *libraries* with best practice documents and templates for selected application areas;
- Approach interoperability at *higher abstractions*:
    - Identify interoperable subsets or profiles of the International Standards;
    - Develop document templates which only use interoperable features of the International Standards;
    - Harmonize the semantics of document formats with ontologies and meta models;

[15] Differences for wordprocessing documents: http://office.microsoft.com/en-gb/word-help/differences-between-the-opendocument-text-odt-format-and-the-word-docx-format-HA010283563.aspx?mode=print

[16] Differences for presentation documents: http://office.microsoft.com/en-us/powerpoint-help/differences-between-the-opendocument-presentation-odp-format-and-the-powerpoint-pptx-format-HA010287723.aspx?mode=print

[17] Differences for spreadsheet documents: http://office.microsoft.com/en-us/excel-help/differences-between-the-opendocument-spreadsheet-ods-format-and-the-excel-xlsx-format-HA010287722.aspx?mode=print

- Use adequate *implementation strategies:*
    - Avoid the usage of unspecified default values in both the International Standard and the application software;
    - Choose implementation options carefully to ensure that the software is implemented in an interoperable way.

9 Examples and tools

The TR focuses on the translatability of the two *document formats* ODF and OOXML. The interoperability of supporting *office applications* has not been analysed. In practice a typical user is often interested in finding solutions for the latter problem. For this reason this section gives a short overview on the tools available at the time of writing.

Generic translation rules between ODF and OXML are typically defined between the particular XML schemas and executed on the corresponding XML instances stored in the document's package. From this assumption it can be concluded that only schema valid documents can be translated, otherwise the translation rules cannot be executed. For this reason it is necessary to be able to validate documents. The following online validators are available for ODF:

- OpenDocument Fellowship provides an OpenDocument validation service for ODF 1.0 documents at http://opendocumentfellowship.com/validator
- Oracle provides an ODF validator supporting ODF 1.0, ODF1.1, and ODF 1.2 documents at http://tools.services.openoffice.org/odfvalidator/
- Alex Brown from *Griffin Brown Digital Publishing Ltd.* provides the Office-o-tron validator for ODF 1.0 documents at http://www.probatron.org:8080/officeotron/officeotron.html
- The *Package Explorer* provided by Wouter van Vugt from CodeCounsil supports validation against ODF 1.0, 1.1 and 1.2.

The following validators are available for OOXML documents:

- Alex Brown from *Griffin Brown Digital Publishing Ltd.* provides the office-o-tron validator for transitional OOXML documents at http://www.probatron.org:8080/officeotron/officeotron.html
- Jesper Stocholm provides an ISO/IEC 29500 validator service using the latest up-to-date version of the base schemas of ISO/IEC 29500 at http://29500.idippedut.dk/
- The *Microsoft OpenXML SDK-Tool* contains an ISO/IEC 29500 (transitional) validator
- The *Package Explorer* provided by Wouter van Vugt from CodeCounsil supports validation against ECMA 376-1, ECMA 376-2 (transitional), and ISO/IEC 29500 strict and transitional.

In addition XML tools like Oxygen or XMLSpy can be used to perform syntactic XML and even semantic validation of documents. Tool dependent translation rules that take the characteristics of the tools into consideration do not presume that the documents are valid. They can be specific to the appropriate tools and take tool specific implementations of the International Standards into consideration.

When translating between the two document formats, the ODF "reference implementations" OpenOffice 3.* and LibreOffice 3.* respectively are able to read and translate OOXML documents into ODF. Microsoft Office 2010 supports both the ODF and OOXML document formats as native storage and exchange formats. Therefore it is able to read and write ODF documents as well as transitional OOXML documents. The same is true for *Hancom Office 2010*, a Korean Office Suite provided by Haansoft Corporation. A comprehensive overview on existing office suites and the supported document formats is given in the English Wikipedia[18].

[18] http://en.wikipedia.org/wiki/Office_suite

Release 4 of the *OpenXML / ODF translator*, a plug-in for Microsoft Office 2007, was released in June 2010 by DIaLOGIKa. The translator is able to read ODF files and map them to ISO/IEC 29500:2008 transitional.

This compilation of tools shows that the major vendors provide ODF ←→ OOXML translation tools. Yet after looking a little bit deeper into the documentation or statistics of the translators it becomes apparent that only a subset of document features has been translated. For several (sub)-functionalities, using the terminology of this TR, the functionality could be seen as *incompatible*, or as *not available* in the target format, or as simply *not having been implemented*. Some functionalities are "lost during the translation process" of their parent/master entities. Some documentations state that a feature or functionality has been translated in a specific way; it has been re-implemented using other functionalities of the target format. Because neither ODF nor OOXML support any tracing attributes that specify the source of a set of XML elements, round trip translation is restricted to simple functionalities.

10 Conclusion

The *"ODF - OOXML Translation – Guidelines"* Technical Report contains four major technical clauses. Clause 5 (Use cases) introduces scenarios describing typical situations occurring when word processing documents, presentations or spreadsheets are exchanged between office suites that are using different external storage formats such as the ISO/IEC International Standards OOXML and ODF. Based on the translation types and document properties explained in clause 4 (Basic principles), expected and observable behaviour of the translation process, or the author's intensions and the reader's perceptions as introduced in the OASIS interoperability model, are described. The comparison of both behaviours is used as a metric for the fidelity of the translation process. Some use cases focus on the layout of a document before and after the translation process. Such *visual interoperability* or *presentation fidelity* is an important criterion for many users but it depends more on the rendering engines used by the particular office suites than on the translatability of the *presentation instructions* supported by the document formats. Of even higher importance is the possibility to preserve the other document properties identified in clause 4 such as document content, dynamic content, meta data, annotations and security, and document parts.

Clause 6 (Features and functionality) of the TR summarizes the features and functionalities that are necessary to implement the document properties introduced in the use cases. The tables introduced in clause 6 summarize how the features, functionalities and sub functionalities that are used to implement the document properties can be translated between ODF and OOXML. *Translatability levels* have been defined for every (sub)-functionality indicating if the translatability of the functionality is *low*, *medium*, or *high*. The tables give a comprehensive overview on the document models used in ODF and OOXML. They collect document features and functionalities that belong together, independent of the location in the standard that introduces these features. Nevertheless these locations are referenced to support the reader who wants to dive deeper into the definition of a specific feature. The limitation on features with high translatability eases the development of portable documents and document templates.

Clause 7 (Representation and XML structure) derives some examples from the use cases introduced in clause 5, showing how both International Standards implement specific features. This section goes into XML details and explains for example, how paragraphs and tables are defined in ODF and OOXML and how functionalities such as alignment, border, hyphenation override, indentation, line spacing, shading, and text direction are specified. The understanding of such XML details is a prerequisite for the understanding of the translation strategies introduced in clause 8 (Translation). This section defines *translation complexity,* which depend on the complexity of the structures and translation rules between the two document formats. The translation complexity can be *easy*, *moderate*, and *difficult.* Concluding the examples for the translation complexity, clause 8 introduces guidelines for the evaluation of translatability between ODF and OOXML. These guidelines define requirements and restrictions on the utilization of document features to be able to define feature preserving one-way and round trip translations between both document formats. They consider the intention of the editors of a document concerning the requested fidelity, starting from the preservation of the visual appearance and ending with the preservation of the logical structure and content of a document.

10.1 Resume

The authors of the TR have learned a lot about both ISO/IEC International Standards and document interoperability during the preparation of the Technical Report. Like for many other interoperability related problems it is impossible to define a generic solution of the translation problem. On the other hand several proprietary solutions already exist. Of course, as long as no standardized translation rules have been defined, every solution will be proprietary. But is it realistic to wait for standardized rules; to define translation rules for two moving targets? Probably not! But it seems to be realistic to introduce subsets or profiles of document features that are important for specific application areas and that avoid fancy features that may be nice to use but prevent interoperability. For such subsets a corresponding document model including mappings to available document formats can be formally defined and validated. Such an approach solves the problem resulting from different versions of the document formats and eases the definition of translation rules between the different formats.

Different tools will in many cases produce different results of a translation between the two document formats. Therefore the tool must be carefully chosen depending on the given requirements, the available environments and the intention of the document's producers and consumers.

Another lesson learned is that a comprehensive documentation of how a standard is interpreted and implemented helps a lot to understand the behaviour of the appropriate office suites and the implementation of filters and translation rules. Therefore it seems to be desirable to provide comprehensive documentation such as the description of the ODF implementation in Microsoft Office[19], the community forum of OpenOffice.org[20], and to further activities like the OASIS interoperability and conformance TC[21]. It should be identified clearly which parts of a standard allow different implementations and probably have to be refined in later versions.

The authors hope that this Technical Report transfers a lot of this expert knowledge to the reader. The report should guide users creating and exchanging documents and templates between tools supporting both International Standards. It should encourage standards bodies as well as the developers of office suites to translate some of the ideas into future versions of the standards and products.

[19] DII information about ODF 1.1 implementation: http://www.documentinteropinitiative.org/OASISODF1.1/reference.aspx

[20] OpenOffice community forum: http://user.services.openoffice.org/en/forum/

[21] OASIS interoperability TC: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=oic

Bibliography

[1] ISO/IEC 10746-1:1998, *Information technology — Open Distributed Processing — Reference model: Overview*

[2] ISO/IEC 10746-3:1996, *Information technology — Open Distributed Processing — Reference Model: Architecture*

ICS 35.060; 35.240.30

Price based on 156 pages