An OASIS Open Document Format Interoperability and Conformance (OIC) TC White Paper

The State of Interoperability Report

Version 2.0

Committee Note Draft 01

06 August 2013

Author:
Bart Hanssens (bart.hanssens@fedict.be), Fedict

Technical Committee:
OASIS Open Document Format Interoperability and Conformance (OIC) TC

Chair:
Bart Hanssens (bart.hanssens@fedict.be), Fedict

Document URIs

This version:

http://docs.oasis-open.org/oic/StateOfInterop/v2.0/cnd01/StateOfInterop-v2.0-cnd01.odt

http://docs.oasis-open.org/oic/StateOfInterop/v2.0/cnd01/StateOfInterop-v2.0-cnd01.html

http://docs.oasis-open.org/oic/StateOfInterop/v2.0/cnd01/StateOfInterop-v2.0-cnd01.pdf

Latest version:

http://docs.oasis-open.org/oic/StateOfInterop/v2.0/StateOfInterop-v2.0.odt

http://docs.oasis-open.org/oic/StateOfInterop/v2.0/StateOfInterop-v2.0.html

http://docs.oasis-open.org/oic/StateOfInterop/v2.0/StateOfInterop-v2.0.pdf

The OASIS ODF Interoperability and Conformance (OIC) TC was created in October 2008 with the stated purpose “to produce materials and host events that will help implementors create applications which conform [to] the ODF standard and which are able to interoperate.”

Table of Contents

Introduction

OASIS OpenDocument Format (ODF) is a standard for office documents, including text documents, spreadsheets and presentations.

ODF 1.0 was published in 2005, ODF 1.1 was published in 2007 and ODF 1.2 in 2011. The OASIS ODF Technical Committee is currently working on ODF 1.3. ODF 1.0 was also approved as ISO/IEC 26300:2006. In 2012, ISO/IEC 26300:2006 was brought into technical alignment with ODF 1.1.

The OASIS ODF Interoperability and Conformance (OIC) TC was created in October 2008 with the stated purpose "to produce materials and host events that will help implementors create applications which conform [to] the ODF standard and which are able to interoperate."

The charter of the OIC TC also calls for it to periodically review the state of conformance and interoperability among ODF implementations, to report on "overall trends in conformance and interoperability", to note "areas of accomplishment as well as areas needing improvement" and to "recommend prioritized activities for advancing the state of conformance and interoperability among ODF implementations."

This State of ODF Interoperability report is the second of the OIC TC's reports on interoperability, and as such provides an overview of the topic and discusses the baseline level of achievement. Future reports will focus on progress achieved beyond this baseline.

Conformance and Interoperability

ODF Conformance

Conformance is the relationship between a product and a standard. A standard defines provisions that constrain the allowable attributes and behaviors of a conforming product. Some provisions define mandatory requirements, meaning requirements that all conforming products must satisfy, while other provisions define optional requirements, meaning that where applicable they must be satisfied.

Conformance exists when the product meets all of the mandatory requirements defined by the standard, as well as those applicable optional requirements. For example, validity with respect to the ODF schema is a mandatory requirement that all conformant ODF documents must meet. However, support of the "`fo:text-align`" attribute is not required. Nevertheless, there are optional requirements that constrain how this feature must be implemented by products that do support that feature, namely that its value is restricted to "`start`", "`end`", "`left`", "`right`", "`center`" or "`justify`".

A standard may define requirements for one or more conformance targets in one or more classes, in which case a product may be said to conform to a particular conformance target and class. For example, ODF 1.1 defines requirements for an ODF Document target as well as an ODF Consumer target. ODF 1.2 defines requirements for an additional conformance class for the ODF Document target, namely the Extended ODF Document class.

There are several validators to assess conformance of ODF Documents:

1. The Apache ODF Toolkit contains a ODF Validator component[1], featuring a java applet and an online validator hosted by the OpenDoc Society.

2. Alex Brown has developed Office-o-tron[2], a web and command-line application that accepts ODF or OOXML packages and validates the XML within.

3. OfficeShots.org[3] is a project founded by the OpenDoc Society and the "Netherlands Open in Connection"-program. It uses several validators to test the conformance of a document and generates screenshots of the same document as displayed in various editors.

These tools primarily look at document validity, an XML concept, which is a necessary but insufficient condition for document conformance. However, we believe that XML validation, combined with additional static analysis, is a promising approach to automate conformance testing of ODF documents.

[1] http://incubator.apache.org/odftoolkit/conformance/ODFValidator.html
[2] http://code.google.com/p/officeotron/
[3] http://officeshots.org/

ODF Interoperability

According to ISO/IEC 2382-1[4], “Information Technology Vocabulary, Fundamental Terms”, interoperability is “The capability to communicate, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units”.

From the perspective of ODF, the document is the data which is transferred, and the functional units are the software applications which create, edit, view and manipulate these documents. Where the document can be successfully transferred among such applications, without the user needing to be concerned with the unique characteristics of each application, then interoperability is high. Conversely, where the user needs to be aware of the quirks of each application, there interoperability is poor.

Since the capabilities of ODF applications extend beyond the common desktop editors, and include other product categories such as web-based editors, mobile device editors, document converters, content repositories, search engines, and other document-aware applications, interoperability will mean different things to users of these different applications. However, to one degree or another, interoperability consists of meeting user expectations regarding one or more of the following qualities when transferring documents:

1. The visual appearance of the document at various levels, e.g., glyph, run, line, block, page, etc.
2. The structure of the document as revealed when the user attempts to edit the document, e.g., headers, paragraphs, lists, tables.
3. The behaviors and capabilities of internal and external links and references.
4. The behaviors and capabilities of embedded images, media and other objects.
5. The preservation of document metadata.
6. The preservation of document extensions.
7. The integrity of digital signatures and other protection mechanisms.
8. The runtime behaviors manifest from scripts, macros and other forms of executable logic.

In any given user task, one or more of these qualities may be of overriding concern.

[4] http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=7229

Conformance and Interoperability

The relationship between conformance and interoperability is subtle and often confused. On one hand, it is possible to have good interoperability, even with non-conformant documents.

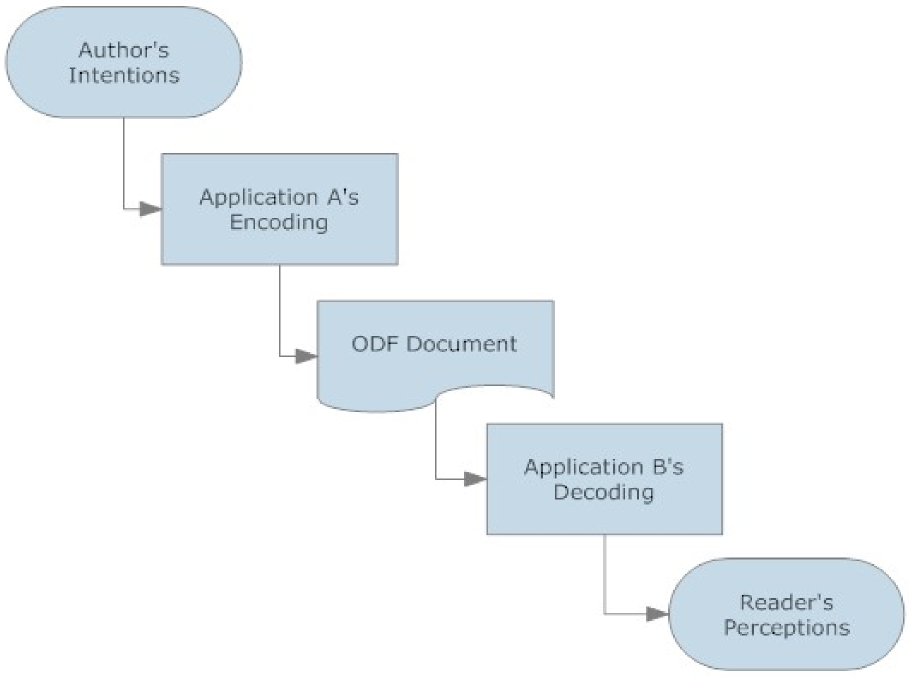
Take HTML, for example. A study of 2.5 million web pages found that only 0.7% of them conformed to the HTML standard. The other 99.3% of HTML pages were not conformant. So conformance is clearly not a prerequisite to interoperability. On the other hand, web browsers require significant additional complexity to handle the errors in non-conformant documents. This complexity comes at a tangible cost, in development resources required to write a robust (tolerant of non-conformance) web browser, and well as less tangible liabilities, such as greater code complexity which typically results in slower performance and decreased reliability.

So although conformance is not required for interoperability, we observe that interoperability is most efficiently achieved in the presence of conforming applications and documents. However, this is an ideal alignment that is rarely achieved, since standards have defects, applications have bugs and users make mistakes. So, in practice, achieving a satisfactory degree of interoperability almost always requires additional efforts beyond mere conformance.

An Interoperability Model

Sources of Interoperability Problems

A model for describing document interoperability is given in Figure 1.



Any document exchange scenario will involve these steps:

1. Authoring: The human author of the document express his intentions by instructing a software application, typically via a combination of keyboard and GUI commands. Note that computer-generated documents also have an authoring step, in which case the human intentions are expressed by the author of the document generation software, via his code instructions.

2. Encoding: The software application, when directed to save the document, executes program logic to encode into ODF format a document that corresponds to the instructions given to it by the user.

3. Storage: The ODF document then represents a transferable data store that can be given to another user.

4. Decoding: The receiving user then uses their software application to decode the ODF document, and render it in fashion suitable for User B to interact with it.

5. Interpretation: User B then perceives the document in their software application and interprets the original author's intentions.

Interoperability defects can be introduced at any step in this process. For example:

1. An inexperienced user might instruct the authoring software application incorrectly, so that his instructions do not match his intentions. For example, a user may try to center text by padding a line with extra spaces rather than using the text align feature of their word processor. Or he might try to express a table header by merely applying the bold attribute to the text rather than defining it as a table header.

2. The software application that writes the document may have defects that cause it to incorrectly encode the user's instructions.

3. Due to ambiguities in the ODF specification, the document may be subject to different interpretations by Application A and Application B.

4. The software application that reads the document may have defects that cause it to incorrectly decode and render the document.

5. The user reading the document may incorrectly perceives its contents.

Round-trip Interoperability

Round-trip interoperability refers to scenarios where a document author collaborates with one or more other users, such that the document will be edited, not merely viewed, by those other users, and where the revised document is eventually returned to the original author.

From the perspective of the interoperability model, this introduces nothing new. A round-trip scenario is simply an iteration of the above steps, with the same opportunities for errors being introduced, e.g., A→B→A is the same as A→B, followed by B→A. However, since errors introduced at any step in the process tend to accumulate, a complex round-trip scenario will tend to suffer more in the presence of any interoperability defects.

Also, since the original author is the person who most knows the author's intentions, he will also be the most sensitive to the slightest alterations in content or formatting introduced into his document. So minor differences that might not have been noticed when read by a second user will be more obvious to the original author. This sensitivity to even the smallest differences will tend to cause the perception of round-trip interoperability to be lower.

Approaches to Improving ODF Interoperability

Steps

Improving interoperability generally follows three steps:

1. Define the expected behavior
2. Identify defects in implementations
3. Fix the defects

The primary definition of expected behavior for the rendering of ODF documents is the published ODF standard.

However, there are other sources of expected behavior, and meeting these expectations, where they do not conflict with the ODF standard, are, from the user's perspective, very important as well. For example:

1. Approved Errata to the ODF standard
2. Other publications of the OASIS ODF TC, such as the Accessibility Guidelines
3. Draft versions of ODF, where they clarify the expected behavior
4. Interoperability Advisories[5], as may be published by the ODF Interoperability and Conformance TC.
5. Common sense and convention, which often provides a shared set of expectations between the user and the application vendor. For example, the ODF standard does not define the exact colorimetric value of the color "red", though undoubtedly users would be surprised to see their word processor render it as yellow.

[5] This is a work in progress, a few Candidate Interop Advisories can be found on the – informal – wiki https://wiki.oasis-open.org/oic/InteropAdvisories

Tools and Techniques

There are several tools, processes and techniques that have been suggested for identifying interoperability defects, including:

1. Automated static testing of ODF documents, which could range in complexity from simple XML validation to more involved testing of conformance and portability.

2. “Atomic” conformance tests, which test the interoperability of individual features of the ODF standard at the lowest level of granularity. The ODF Fellowship's[6] OpenDocument Sample Documents was an early example of this approach. The approach used by Shah and Kesan[7] falls into this category as well.

3. Scenario-based testing, which combine multiple ODF features into documents which reflect typical real-world uses.

4. “Acid” tests aim to give the end-user a quick view how well their application supports the standard. This approach was popularized in the Web Standards Project to improve browser interoperability. Sam Johnston has created a prototype[8] ACID test for spreadsheet formulas.

5. Plugfests, face-to-face and virtual. This approach has proven useful in interactive testing among vendors in the ODF Plugfests[9].

6. OfficeShots.org[10] allows an end-user to upload a document and compare how it will render in different applications. This can allow the user to see potential interoperability problems before they distribute their document.

7. User-submitted bug reports, sent to their application vendor, can help the vendor prioritize areas which are in need of improvement.

8. Public comments, submitted to the OASIS ODF TC[11], which report ambiguities in the specification which can impact interoperability. Such comments can be resolved in errata or revisions of the standard.

6 http://opendocumentfellowship.com/
7 http://moritzlaw.osu.edu/students/groups/is/files/2012/02/Kesan.pdf
8 http://samj.net/2008/06/opendocument-odf-acid-test-proof-of.html
9 http://odfplugfest.org/history/
10 http://officeshots.org/pages/about
11 https://www.oasis-open.org/committees/comments/index.php?wg_abbrev=office

As enumerated above, there are a variety of approaches to identifying interoperability defects. At this point it is not clear which of these techniques will, over the long term, be the most effective. Some require more substantial up-front investment in automation development, but this investment also permits a degree of test automation.

Some approaches require substantial efforts in analysis of the ODF standard and construction of individual test cases. But once these test cases are designed, they can be executed many times at low cost. There are also techniques that require far less advance preparation and are suitable for formal and informal testing at face-to-face plugfests.

These options are familiar in the field of software quality assurance (SQA), and from that field we are taught to consider several factors:

1. What is the “defect yield” of each testing approach? The defect yield is the number of defects found per unit of testing time, and is a measure of the efficiency of any given approach.

2. What is the test coverage that can be achieved by any given approach? Test coverage would indicate what fraction of the features of ODF are tested by that approach.

3. Once an initial investment is made, how expensive will it be to update test materials as new ODF versions and new application versions are released?

4. How well does the testing approach prioritize the testing effort, so that the defects that most impact interoperability for the most number of users are found quickly?

The goal should be to identify an approach that finds the most number of defects with the least effort, with an emphasis on those defects that have the greatest real-world impact. It is well-known from SQA research that that the optimal approach often involves a blend of different complementary techniques.

Of course, interoperability will not improve merely by testing. The results of testing must feed forward into the vendors' development plans, so these defects are fixed and the fixes make it into the hands of end-users. Nothing improves until vendors change their code. Merely talking about the problem doesn't make it go away.

Interoperability "Roadmap"

Current and Previous Versions of the ODF Specification

Specification	Feature	Comment
ODF 1.1	Accessibility improvements	Arguably a feature instead of an interoperability fix, the possibility of e.g. adding alternate text to pictures also allows for easier processing of documents by machines (in this case, better indexing).
ODF 1.2	Numbered lists	Exchanging documents with numbered lists is challenging in ODF 1.1 implementations.
	Tables in presentations	ODF 1.1 does not permit tables in presentations, so some implementations used a work-around by adding a spreadsheet to a presentation, which was not supported by other implementations.
	"Open Formula"	While ODF 1.1 did of course allow spreadsheet formulas, there was little information on the syntax and no information on the semantics of these formulas, leading to incompatible implementations.
	Digital signatures	ODF 1.1 did not specify digital signatures. However, at least one code base did support signatures using XML-DSIG.

Suggestions for Future Versions

Some suggestions for improving interoperability in future versions of the ODF specification.

Specification	Feature	Comment
ODF 1.3	Change tracking	While ODF has basic change tracking features, several competing ways of implementing more advanced change tracking exist. Preferrably one method should be chosen.
Beyond 1.3	Real SVG support	ODF uses its own version of SVG, which makes interoperability with other standards supporting SVG more challenging. Aligning ODF with the SVG specification could be beneficial.

Improvements since the first Report

Numbered Lists

Exchanging documents with numbered lists (including outline numbering) between older implementations can be challenging, especially if the document contains numbered lists interrupted by paragraphs and bulleted lists, or uses outline numbering with sublevels.

Many implementation-specific issues have meanwhile been solved. In addition, ODF 1.2 introduced an attribute "`text:continue-list`" that can be used to specify the ID of the list that is to be continued, making even more complex numbering schemes possible.

Tables in Presentations

Until ODF 1.2, the specification did not directly support tables in presentation documents.

A work-around used by some ODF 1.1 implementations was to embed a spreadsheet in a presentation, along with a preview of said spreadsheet (stored inside the ODF document as an image). But this did not work well across all implementations: some implementations only rendered the preview of the spreadsheet, and did not support editing the spreadsheet table itself.

Starting with ODF 1.2, a `<table:table>` can be inside a `<draw:frame>`, removing the need for this work-around.

Spreadsheet Formulas

Implementations vary in their ability to parse spreadsheet formulas written by other implementations.

When an implementation does not support a formula syntax, typically the values are shown when "`office:value`" is present, but formulas themselves are removed. A simple example is a `SUM` function that calculates the sum of 2 cells. When opening a spreadsheet, a user may not notice any difference because the values are still there. But when changing the values of these 2 cells, the value of the cell originally containing the `SUM` function will not be updated accordingly because the formula has been removed.

Adoption of OpenFormula (part of ODF 1.2) as the interchange format for spreadsheet formula expressions is thus encouraged. Luckily, almost all current versions of ODF code bases now support the ODF 1.2 OpenFormula syntax.

Charts

Exchanging charts between ODF implementations has been improved.

First of all, ODF spreadsheet implementations store charts as embedded documents, using the "`xlink:href`" attribute to link the spreadsheet to the chart. However, some implementations add a leading "`./`" to the value of this attribute, while others add a trailing "`/`", which resulted in some implementations not being able to load the charts stored by other implementations. This is discussed in the OASIS OIC TC Interop Advisory 00002[12], and implementations nowadays tend to be more liberal in the way they accept the syntax of embedded documents.

Second, cell ranges for chart data can be defined in multiple ways in ODF 1.1, which could lead to incorrect rendering of charts. ODF 1.2 provides some more guidance in this area, deprecating the "`table:cell-range-address`" attribute within the `<chart:plot-area>` element. This is discussed in the OASIS OIC TC Interop Advisory 00004[13], and implementations nowadays tend to be more liberal in the way they accept cell ranges.

12 http://tools.oasis-open.org/version-control/browse/wsvn/oic/Advisories/00002-subdoc_trailing_slash/trunk/description.html

13 http://tools.oasis-open.org/version-control/browse/wsvn/oic/Advisories/00004-chart_cell_ranges/trunk/description.html

Some Areas for Improvements

Based on initial testing in the OASIS ODF Interoperability and Conformance TC, as well as scenario-based testing at the ODF Plugfests, several feature areas have been identified as needing improvement in one or more ODF implementations.

Change Tracking

Type: specification + implementation challenge

The ODF 1.2 specification has only limited change tracking support. For instance, tracking changes in tables may result in a table with superfluous empty rows when rows are added and removed again. The OASIS OpenDocument Advanced Document Collaboration SC[14] is therefore working on a markup vocabulary that supports advanced change tracking, and a technical document that explains how to convert the ODF 1.2 change tracking mechanism into this new markup.

When an ODF 1.1 or 1.2 document containing change tracking information is loaded in an implementation that does not support change tracking, the `<text:tracked-changes>` element and its content can be removed.

Also, the content of deleted paragraphs, stored in `<text:p>` elements inside `<text:tracked-changes>`, can become visible at the top of the first page of the document (because `<text:tracked-changes>` is stored before all other `<text:p>`s), which can be very confusing for the user. The OASIS OIC TC Interop Advisory 00001[15] therefore recommends that implementations that do not support change tracking, should ignore the content of this element.

Proprietary Image Format for Previews

Type: implementation challenge

ODF allows to store a rendered presentation of embedded documents inside the document package, most notably previews of charts embedded in spreadsheets which is referenced by a `<draw:image>` element inside a `<draw:frame>`. Some implementations still use a proprietary legacy format to store the preview, even though the ODF specification recommends using a more widely supported open format like PNG or SVG.

Implementations using the legacy format should consider switching to PNG or SVG. Meanwhile, implementations not supporting the legacy format could ignore the preview and try to render the embedded object instead (if they support the embedded object type to begin with).

[14] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=office-collab
[15] http://tools.oasis-open.org/version-control/browse/wsvn/oic/Advisories/00001-non_inline_text/tags/latest/description.htm

Application-defined Default Settings

Type: implementation + user expectation challenge

Different implementations use different default settings, most notably the default color schemes for chart graphs and default minimum / maximum values for axes. These default settings are not always preserved in the document itself, surprising some users that expect “pixel-perfect” respresentation of their document across implementations.

Implementations could provide an option to write out all these implementation-specific default values when saving a document. It is understood that users may want to override these settings nevertheless, but at least this option would offer users a choice.

RDF

Type: implementation challenge

ODF 1.2 specification supports RDF for storing enhanced metadata, but most code bases don't support it. The result is that RDF-markup may be removed when editing an ODF document using an implementation that does not support RDF. In addition, even when an implementation does support RDF at the storage level, it may not provide a user interface for viewing/editing the existing RDF data.

At the moment, there is not much that can be done to improve interoperability between various implementations: users should be aware that exchanging RDF-data between different code bases may not work.

Digital Signature with Namespace Prefix

Type: implementation challenge

ODF 1.2 supports XML-DSIG for digital signatures, but some implementations only accept signature elements without a namespace prefix, i.e. they do parse `<Signature xmlns=..` but ignore `<ds:Signature xmlns:ds=..` elements.

This is discussed in the OASIS OIC TC Interop Advisory 00008[16]: implementations supporting digital signatures are encouraged to implement the parsing of namespace-prefixed signature elements . Meanwhile, implementations creating digital signatures could store these elements with the default namespace binding for increased interoperability.

[16] http://tools.oasis-open.org/version-control/browse/wsvn/oic/Advisories/00008-DSig_Namespaces/trunk/description.html

XForms

Type: implementation challenge

ODF 1.2 supports XForms for advanced forms, but few code bases support it. Implementations not supporting XForms sometimes render the form controls as plain text, or remove the form altogether.

At the moment, there is not much that can be done to improve interoperability between various implementations: users should be aware that exchanging forms between different code bases may not work.

Text Rotation in Shapes

Type: specification + implementation challenge

The `<draw:text-rotate-angle>` element does not specify which direction to rotate the shape. Some implementations rotate clockwise while others rotate counter-clockwise. This is also discussed in ODF Jira issue OFFICE-3750[17], and a comment is made suggesting to follow SVG's clockwise direction of rotation.

Angles in Shapes

Type: implementation challenge

The “`draw:formula`” attribute may contain trigonometric functions, and while the specification clearly states that angles must be specified in degrees, most - if not all - implementations use radians instead.

This is reported as ODF Jira issue OFFICE-3823[18]: the specification is to be changed in an ODF 1.2 Errata document, using radians as the angle unit.

17 https://tools.oasis-open.org/issues/browse/OFFICE-3750
18 https://tools.oasis-open.org/issues/browse/OFFICE-3823

Acknowledgments

The following individuals have participated in the creation of this report and are gratefully acknowledged.

- Andreas Guelzow, Individual
- Dennis Hamilton, Individual
- Bart Hanssens, Fedict
- John Haug, Microsoft
- Chris Rae, Microsoft
- Louis Suarez-Potts, Individual

Revision History

Version	Date	Changes
WD1	2013-01-09	Initial draft
WD2	2013-02-06	Changed template to OASIS White Paper template
WD3	2013-02-20	Verified information in first chapters Removed “Problem and Solutions” chapter Added more information to “Priority Areas” / “Change Tracking” Added “Priority Areas” / “Mobile Platform Support”
WD4	2013-03-08	Added links in “Tools and Techniques” chapter Added “Interoperability Roadmap” section
WD5	2013-04-10	Splitted “Interoperability Roadmap” into “Current” and “Future” table Added more information to “Improvements since the first Report”
WD6	2013-05-15	Added “Improvements since the first Report” / “Charts” Removed “Priority Areas” / “Mobile Platform Support” Removed “Priority Areas” / “Database Fields” Renamed “Priority Areas” to “Some Areas for Improvements” Added “Proprietary Image Format for Previews” section
WD7	2013-07-24	Added “Revision History” Removed numbering from headers
W8	2013-07-29	Editorial changes Mentioned ISO/IEC 26300:2006 / ODF 1.1 alignment in “Introduction”