Authoring for Interoperability

robert_weir@us.ibm.com

Different uses of documents

- Document as the end product, e.g., reports, white papers, customer presentations, etc.
- Documents as an analysis/collaboration surface.
- Document/Applications, with macros and scripts and other forms of automation.
- Documents integrated with the business process, via transformation, forms, custom XML, etc.

Each of these has different interoperability needs

Interop is not always a user priority

- In many cases documents are exchanged within a single organization among known users running the same editors.
  - Interop not a priority
- In some cases the documents are tightly tied to an organizations business process via integrated scripting.
  - Interop not a priority
- In some cases a document is created by a single user for their own use.
  - Interop not a priority

Interop is one of many goals

- Other goals include:
  - Business goals
    - Flexibility
    - Extensibility
  - Productivity goals
    - Ease of ad-hoc use
    - Familiar authoring practices

  - Of course, some times interop is the primary goal.

An analogy: C programming

- There is the set of conforming C programs:
  - int *x = (int *) 42;
  - int x,y; memcpy(&x,&y,4);
- And then there is the set of portable C programs
- But not all programs need to be portable.

Conforming versus Portable

- We all learned this as programmers
  - Some things that are allowed in C are discouraged in portable code.
    - Assuming size of integers, byte ordering
    - Structure alignment/padding
    - Writing to code segments
    - Values of uninitialized memory
- Some things are undefined or implementation-defined in C, and programmers know that these should be avoided in portable code.

What helps programmers

- Education on portable programming practices
- Tools that warn when non-portable constructs are used, e.g., lint -w4, especially when integrated into the IDE
- Isolate and conditionalize platform dependencies
- Use of portable libraries and frameworks

Undefined/Implementation-Defined in ODF

- Line breaking algorithms
- Page breaking algorithms
- Scripts/Macros
- String to number conversion in spreadsheets
- The exact feature set supported by an application

What is Portable ODF?

- A constrained subset of ODF that is expressive and useful, but far more portable across implementations.
  - Could be informal authoring guidelines
  - Could be supported by the editor
  - Could be a formal profile standard

What is Portable ODF?

- Features that are not portable are excluded:
  - Extensions
  - OLE embeddings
  - macros/scripts
  - Absolute page/content references
- Other features are constrained:
  - Use only widely-supported fonts
  - Use named styles rather than direct attributes

How editors can help

- If a user wants to create a portable document
  - Encourage the use of named styles
  - Discourage direct application of attributes
  - Discourage non-portable constructs
  - Explicitly save default values into document
  - Scan document for problems
- Perhaps have a 'portable document' mode in the editor that users can switch into?
- Provide a template that encourages portable documents.
- Bonus: a portable document will likely be more accessible at the same time!

Summary

- We will <u>never</u> make it so that <u>all</u> ODF documents will be interoperable. If we tried, we would need first to remove all flexibility and extensibility from the format.
- It is possible, though difficult, to ensure that all <u>reasonable</u> (non-pathological) ODF documents are interoperable. However, there may be many pathological users.
- A complementary approach is to encourage authors to create Portable ODF documents, via education, tooling and/or a profile standard.