**CHAPTER 17**

# Desktop and Web-Based Office Suites

I've long been excited about the mashability and reusability of office suite documents (word processor documents, spreadsheets, and PowerPoint presentations), the potential of which has gone largely unexploited. There are many office suites, but in this chapter I concentrate on the latest versions of Microsoft Office (2007 and 2003) and OpenOffice.org (version 2.x). Few people realize that both these applications not only have programming interfaces but also XML-based file formats. In theory, office documents using the respective file formats (OpenDocument and Office Open XML) are easier to reuse and generate from scratch than older generations of documents using opaque binary formats. And as we seen throughout the book, knowledge of data formats and an API means having opportunities for mashups. For ages, people have been reverse engineering older Microsoft Office documents, whose formats were not publicly documented – but recombining office suites should be easier now. In this chapter, I will also introduce you to emerging space of web-based office suites, specifically ones that are programmable. We'll also look briefly at how to program the office suites. .

This chapter:

* Shows how to do some simple parsing of ODF and Office Open XML documents
* Shows how to create a simple document in both ODF and OpenXML
* Demonstrates some simple scripting of Microsoft Office and OO.o
* Lays out what else is possible by manipulating the open document formats

## Mashup Scenarios for Office Suites

Why would mashups of office suite documents be interesting? For one, they contain much of knowledge and communication embeedd in digital form. Sometimes they are in narratives (such as world documents), sometimes in semi-structured forms (such as spreadsheets). To repurpose that knowledge, it is sometimes a matter of reformatting that document into another format. Other times, it's about extracting valuable pieces (for instance, all the references I have in my book might be extracted into a reference database.)

Some use case scenarios for the programmatic creation and reuse of office documents I have had in mind are:

* *reusing Powerpoint.* How many of you out there have collections of Powerpoint presentations which draw from a common collection of digital assets (pictures, outlines) and complete slides? Can we build a system of personal information management so that PPT are constructed as virtual assemblages of slides, dynamically associated with assets?
* *repurposing my book* (write once, publish everywhere.) I'm currently writing this manuscript in Microsoft Office 2007. I'd like to republish this book in (X)HTML, Docbook, PDF, wiki markup. How would I repurpose the Word manuscript into those formats?
* creating an educational website in which data is downloaded to spreadsheets – not only as static data elements by as dynamic simulations. There's plenty of data out there – can we write programs to translate them into the dominant data analysis tool used by everyone: spreadsheets, whether they are on the desktop or in the cloud?
* *instant Powerpoint from Flickr*. I'd like to download a Flickr set as a Powerpoint presentation. (This scenario seems to fit a world in which Powerpoint is a dominant presentation program. Even if Tuffte hates it, a Flickr to PPT translator might make it easier to show those vacation pictures at your next company presentation.)

# World of Document Markup

This chapter focuses on XML-based document markup languages in two dominant group of office suites: Microsoft Office 2007 and OpenOffice.org. There are plenty of other markup language, which covered well in the Wikipedia:

* http://en.wikipedia.org/wiki/Document_markup_language
* http://en.wikipedia.org/wiki/List_of_document_markup_languages
* http://en.wikipedia.org/wiki/Comparison_of_document_markup_languages

# OpenDocument Format (ODF)

OpenDocument is "an OASIS Standard and a published ISO and IEC International Standard referred to as ISO/IEC 26300:2006."[1] ODF is used most prominently in OpenOffice.org (http://en.wikipedia.org/wiki/OpenOffice.org) and KOffice (http://www.koffice.org/), among other office suites. For a good overview of the file format, consult J. David Eisenberg' excellent book on ODF: *OASIS OpenDocument Essentials*, which is available for download as a PDF (free of charge) or for purchase.[2]

The goal of this section is to get you jumpstarted into the issues of parsing and creating ODF files programmatically.

---

[1] http://en.wikipedia.org/wiki/Opendocument

[2] http://books.evc-cit.info/OD_Essentials.pdf) Page was down – use: http://web.archive.org/web/20060521030544/http://books.evc-cit.info/index.html OASIS hosts the book at http://develop.opendocumentfellowship.org/book/

For this section, I am assuming you have OpenOffice.org v 2.2 installed.

A good way to understand the essentials of the file format is to create a simple instance of an ODF file and then analyze it:

1. Fire up OpenOffice.org Writer, type in "Hello World", and save the file as `helloworld.odt.`[3]

2. Open the file in an zip utility (such as WinZip on the PC). One easy way to do so is to change the file extension from odt to zip so that the operating system will recognize it as a zip file. You will see that it's actually a zip-format file when you go to unzip it. (See the list of files in Figure 17-??)

*Figure 17-1 Unzipping helloworld.zip See that an OpenDocument Writer file produced by OpenOffice.org is actually in the zip format.*

You'll see some of the files that can be part of an ODF file:

* content.xml
* styles.xml
* meta.xml
* settings.xml
* META-INF/manifest.xml
* mimetype
* Configuration2/accelerator/
* Thumbnails/thumbnail.png

You can also use your favorite programming language to generate a list of the files, such as Python or PHP. The following Python code

```
import zipfile
z = zipfile.ZipFile(r'[path_to_your_file_here]')
z.printdir()
```

generates

```
File Name                                             Modified             Size
mimetype                                       2007-06-02 16:10:18           39
Configurations2/statusbar/                     2007-06-02 16:10:18            0
Configurations2/accelerator/current.xml        2007-06-02 16:10:18            0
Configurations2/floater/                       2007-06-02 16:10:18            0
Configurations2/popupmenu/                     2007-06-02 16:10:18            0
```

[3] Available at http://examples.mashupguide.net/ch17/helloworld.odt

```
Configurations2/progressbar/                  2007-06-02 16:10:18           0
Configurations2/menubar/                      2007-06-02 16:10:18           0
Configurations2/toolbar/                      2007-06-02 16:10:18           0
Configurations2/images/Bitmaps/               2007-06-02 16:10:18           0
content.xml                                   2007-06-02 16:10:18        2776
styles.xml                                    2007-06-02 16:10:18        8492
meta.xml                                      2007-06-02 16:10:18        1143
Thumbnails/thumbnail.png                      2007-06-02 16:10:18         945
settings.xml                                  2007-06-02 16:10:18        7476
META-INF/manifest.xml                         2007-06-02 16:10:18        1866
```

The equivalent functionality in PHP can be had with the php zip library (see http://us2.php.net/zip ):

```
<?php
# see http://us2.php.net/manual/en/language.types.string.php for the use of ' -- the
\ doesn't need to be escaped
$zip = zip_open('[path_to_your_file]');
while ($entry = zip_read($zip)) {
  print zip_entry_name($entry) . "\t". zip_entry_filesize($entry). "\n";
}
zip_close($zip);
?>
```

from which you get:

```
mimetype        39
Configurations2/statusbar/      0
Configurations2/accelerator/current.xml 0
Configurations2/floater/        0
Configurations2/popupmenu/      0
Configurations2/progressbar/    0
Configurations2/menubar/        0
Configurations2/toolbar/        0
Configurations2/images/Bitmaps/ 0
content.xml     2776
styles.xml      8492
meta.xml        1143
Thumbnails/thumbnail.png        945
settings.xml    7476
META-INF/manifest.xml   1866
```

Generating a simple ODF file using OpenOffice.org gives you a simple file from which you can build. However, it's useful to boil the file down even further. A question to ask here is what a *minimal example of an ODF file.* But how to figure out what the minimal ODF document is? (I would add that if we have a hard time determining what a minimal valid instance of document format is, it's going to be difficult for a group of vendors and developers to achieve operability in the documents they generate.)

Here are some possible ways to figure out the minimal instance:

* see whether the ODF specification, and specifically use the ODF schema , along with the appropriate tool/library can be used to generate a minimal instance.
* do a bit of trial-and-error generate a ODF file and chop pieces as much as possible while feeding it to the ODF validator to see how far I can cut the file.

Let's take a look at both approaches and see how far we get.

## The ODF Spec: does it answer the question of a minimal instance?

The ODF specification is housed at:

`http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=office`

OO.o 2.2 seems to be using the ODF 1.0 specificaltion (you can look at the XML declaration of content.xml

The specification contains an Relax NG schema for ODF. Links to the schemas , stemming from the oasis-open.org page include:

* The schema for office documents, "extracted from chapter 1 to 16 of the specification" – v 1.0[4]
* "the normative schema for the manifest file used by the OpenDocument package format" – v 1.0[5]
* " the strict schema for office documents that permits only meta information and formatting properties contained in this specification itself" – v 1.0[6]

I wonder whether a minimal instance of an ODF can be strictly derived from the shema itself. (In other words, are there libraries/tools to which I say pass them a Relax NG schema and say return me a minimal instance? I've not found any so far. The closest I've come in that department are the following references:

* There is supposed to be a Sun XML Instance Generator – but I can't find any code to download. (Closest I get is the https://msv.dev.java.net/
* http://www.stylusstudio.com/xml_generator.html points to a commericall XML Generator
* http://xmlbuddy.com/2.0/features.html mentions an instance generator.
* https://relax-ng.dev.java.net/ is a set of Java tools that might be able to generate a minimal instance of a Relax-NG schema.

## Trial and Error Search for an ODF Minimal Instance

Eisenberg (p. 13) gives his answer to the quesion:

> *The only files that are actually necessary are content.xml and the METAINF/manifest.xml file. If you create a file that contains word processor elements and zip it up and a manifest that points to that file, OpenOffice.org will be able to open it successfully. The result will be a plain text-only document with*

---

[4] http://www.oasis-open.org/committees/download.php/12571/OpenDocument-schema-v1.0-os.rng

[5] http://www.oasis-open.org/committees/download.php/12570/OpenDocument-manifest-schema-v1.0-os.rng

[6] http://www.oasis-open.org/committees/download.php/12569/OpenDocument-strict-schema-v1.0-os.rng

 

> *no styles. You won't have any of the meta-information about who created the file or when it was last edited, and the printer settings, view area, and zoom factor will be set to the OpenOffice.org defaults.*

Let's verify Eisenberg's assertion. If you create a `odt` file with only the same content.xml as helloworld.odt and edit METAINF/metadata.xml to reference only content.xml and the METAINF directory:

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest:manifest
xmlns:manifest="urn:oasis:names:tc:opendocument:xmlns:manifest:1.0">
 <manifest:file-entry manifest:media-type="application/vnd.oasis.opendocument.text"
manifest:full-path="/"/>
 <manifest:file-entry manifest:media-type="text/xml" manifest:full-
path="content.xml"/>
</manifest:manifest>
```

thus creating a odt that consists of only those two files,[7] you will find that such a file will load successfully in OpenOffice.org 2.2 and the OpenDocumentViewer[8] -- giving credence to the assertion that in OO.o 2.2 at least, you don't need any more than content.xml and META-INF/manifest.xml.

---

You can download and install the OpenDocument Validator on a Unix system [9] or run the online version.[10]

---

Nonetheless, the ODF Validator (http://opendocumentfellowship.org/validator) doesn't find the file to be valid though, producing the following error message:

```
1. warning
   does not contain a /mimetype file. This is a SHOULD in OpenDocument 1.0
2. error
   styles.xml is missing
3. error
   settings.xml is missing
4. error
   meta.xml is missing
```

Since the online validator dies on one of the Fellowship's test files,[11] we can see there are some unresolved problems with the Validator and/or the test files produced by the OpenDocument Fellowship.

---

[7] http://examples.mashupguide.net/ch17/helloworld_min_odt_1.odt

[8] http://opendocumentfellowship.org/odfviewer

[9] http://opendocumentfellowship.org/projects/odftools

[10] http://opendocumentfellowship.org/validator

[11] http://testsuite.opendocumentfellowship.org/testcases/General/DocumentStructure/SingleDocumentContents/testDoc/testDoc.odt via http://testsuite.opendocumentfellowship.org/testcases/General/DocumentStructure/SingleDocumentContents/TestCase.html

 

If you insert skeletal styles.xml, settings.xml, and meta.xml, you can convince the ODF Validator to accept the resulting odt as a valid documentat.. That is, create a odt with the following files. (Strictly speaking, the namespace declearations are extraneous – but they are useful to have once you start plugging chunks of ODF):

**meta.xml:**

```
<?xml version="1.0" ?>
<office:document-meta office:version="1.0"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:meta="urn:oasis:names:tc:opendocument:xmlns:meta:1.0"
xmlns:office="urn:oasis:names:tc:opendocument:xmlns:office:1.0"
xmlns:ooo="http://openoffice.org/2004/office"
xmlns:xlink="http://www.w3.org/1999/xlink"/>
```

**settings.xml:**

```
<?xml version="1.0" ?>
<office:document-settings office:version="1.0"
xmlns:config="urn:oasis:names:tc:opendocument:xmlns:config:1.0"
xmlns:office="urn:oasis:names:tc:opendocument:xmlns:office:1.0"
xmlns:ooo="http://openoffice.org/2004/office"
xmlns:xlink="http://www.w3.org/1999/xlink" />
```

**styles.xml:**

```
<?xml version="1.0" ?>
<office:document-styles office:version="1.0"
xmlns:chart="urn:oasis:names:tc:opendocument:xmlns:chart:1.0"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:dom="http://www.w3.org/2001/xml-
events" xmlns:dr3d="urn:oasis:names:tc:opendocument:xmlns:dr3d:1.0"
xmlns:draw="urn:oasis:names:tc:opendocument:xmlns:drawing:1.0"
xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"
xmlns:form="urn:oasis:names:tc:opendocument:xmlns:form:1.0"
xmlns:math="http://www.w3.org/1998/Math/MathML"
xmlns:meta="urn:oasis:names:tc:opendocument:xmlns:meta:1.0"
xmlns:number="urn:oasis:names:tc:opendocument:xmlns:datastyle:1.0"
xmlns:office="urn:oasis:names:tc:opendocument:xmlns:office:1.0"
xmlns:ooo="http://openoffice.org/2004/office"
xmlns:oooc="http://openoffice.org/2004/calc"
xmlns:ooow="http://openoffice.org/2004/writer"
xmlns:script="urn:oasis:names:tc:opendocument:xmlns:script:1.0"
xmlns:style="urn:oasis:names:tc:opendocument:xmlns:style:1.0"
xmlns:svg="urn:oasis:names:tc:opendocument:xmlns:svg-compatible:1.0"
xmlns:table="urn:oasis:names:tc:opendocument:xmlns:table:1.0"
xmlns:text="urn:oasis:names:tc:opendocument:xmlns:text:1.0"
xmlns:xlink="http://www.w3.org/1999/xlink" />
```

**content.xml:**

```
<?xml version="1.0" ?>
<office:document-content office:version="1.0"
xmlns:chart="urn:oasis:names:tc:opendocument:xmlns:chart:1.0"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:dom="http://www.w3.org/2001/xml-
events" xmlns:dr3d="urn:oasis:names:tc:opendocument:xmlns:dr3d:1.0"
xmlns:draw="urn:oasis:names:tc:opendocument:xmlns:drawing:1.0"
```

```
xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"
xmlns:form="urn:oasis:names:tc:opendocument:xmlns:form:1.0"
xmlns:math="http://www.w3.org/1998/Math/MathML"
xmlns:meta="urn:oasis:names:tc:opendocument:xmlns:meta:1.0"
xmlns:number="urn:oasis:names:tc:opendocument:xmlns:datastyle:1.0"
xmlns:office="urn:oasis:names:tc:opendocument:xmlns:office:1.0"
xmlns:ooo="http://openoffice.org/2004/office"
xmlns:oooc="http://openoffice.org/2004/calc"
xmlns:ooow="http://openoffice.org/2004/writer"
xmlns:script="urn:oasis:names:tc:opendocument:xmlns:script:1.0"
xmlns:style="urn:oasis:names:tc:opendocument:xmlns:style:1.0"
xmlns:svg="urn:oasis:names:tc:opendocument:xmlns:svg-compatible:1.0"
xmlns:table="urn:oasis:names:tc:opendocument:xmlns:table:1.0"
xmlns:text="urn:oasis:names:tc:opendocument:xmlns:text:1.0"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <office:body>
    <office:text>
      <text:p>
        Hello World!
      </text:p>
    </office:text>
  </office:body>
</office:document-content>
```

**manifest.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest:manifest
xmlns:manifest="urn:oasis:names:tc:opendocument:xmlns:manifest:1.0">
  <manifest:file-entry manifest:media-type="application/vnd.oasis.opendocument.text"
manifest:full-path="/"/>
  <manifest:file-entry manifest:media-type="text/xml" manifest:full-
path="content.xml"/>
  <manifest:file-entry manifest:media-type="text/xml" manifest:full-
path="meta.xml"/>
  <manifest:file-entry manifest:media-type="text/xml" manifest:full-
path="settings.xml"/>
  <manifest:file-entry manifest:media-type="text/xml" manifest:full-
path="styles.xml"/>
</manifest:manifest>
```

you can a minimal and valid ODF document. See:
`http://examples.mashupguide.net/ch17/helloworld_min_odt_2.odt`

As an exercise to the reader, I leave it to you to generate minimal instances of the spreadsheet (.ods), presentation (.odp), graphics (.odg), and math (.odf) documents.

Why should we care about minimal instances of ODF (and later OOXML) documents? ODF and OOXML are complicated markup formats. One of the best ways to figure out how to create formats is to use a tool such as OO.o and Microsoft Office to generate what you

want, save the file, unzip the file, extract the section of the document you want, and plug that stuff into a minimalist document that you know is valid. That's why we are looking at boiling ODF down to its essense. Alas I wish there were a more definitive way to know what is here is actual minimal. But it's a good start. I've uploaded the file to

## So what to do?

ODF is a promising format but is immature. We can wait until some better libraries are in place. In the meantime, you can make some good progress by using a tool such as OpenOffice.org to generate by hand a document along the lines of what you want to generate programmatically. Unzip the file, extract the relevant pieces, and use that as a template that your program will fill out. You might be able to get things working without worrying about issues of validity, depending on what you are trying to accomplish.

## The single XML document format – how to generate an instance?

I know that there is supposed to a single XML version of an ODF document – but I've not figured out how to generate an instance.

## Resolving the Validity/Minimal Instance Issues

Followup I plan to do to resolve some the unresolved issues in this chapter:

* Contact Alex Hudson, the author of the ODF Validator: `http://www.alexhudson.com/contact`
* Daniel Carrera, the informal lead of ODF Fellowship: `http://daniel.carrera.name/about-me/`
* contact J. David Eisenberg, the author of the book on ODF (http://books.evc-cit.info/index.html)
* follow up on the post I made on the ODF developer list: http://lists.opendocumentfellowship.org/pipermail/odf-discuss/2007-June/002110.html
* chat with folks on IRC: http://opendocumentfellowship.org/about_us/contact/irc
* report a bug in example 2.1 of http://opendocumentfellowship.org/files/api-for-odfpy.odt – *doc should be textdoc.*

DanielC's advice: " My advice to get started with ODF is to use OOo to save a blank document with the styles you want, and then just insert your content inside <office:text>"

---

Note: You can work with the ODF validator: http://opendocumentfellowship.org/validator to see how far we can boil our document down and still have a valid ODF file. The tool is also downloadable: http://opendocumentfellowship.org/projects/odftools

---

 

### Some Other Useful References

ODF has some basic similarities to the OpenOffice 1.0 document format -- but they are not the same. Documentation of the exact similarities and differences between the OpenOffice 1.0 format (http://xml.openoffice.org/general.html) and the ODF format would be incredibly useful – but I've not yet found such a document. Because of the similarities, some of the old tutorials related to OpenOffice.org 1.0 are still potentially useful, such as:

`http://www.xml.com/pub/a/2005/01/26/hacking-ooo.html`

You can find a list of implementation of ODF at

`http://opendocumentfellowship.org/applications`

The collection of sample ODF documents can come in handy:

`http://testsuite.opendocumentfellowship.org/`

## API kits for working with ODF

In the previous sections, we looked at the approach of working directly with the ODF specification and the Validator and using trial and error to generate valid ODF files. In this section, we move up the abstraction ladder and look at using libraries/API kits/wrapper libraries that work with ODF. Such libraries can be a huge help if they are implemented well and reflect conscientious effort on the part of the authors to wrestle with some of the issues we discuss in the previous section.

A good list of tools that support ODF is:

`http://en.wikipedia.org/wiki/OpenDocument_software`

Another good list is:

`http://opendocumentfellowship.org/development/tools`

Let me highlight some useful bits:

* `http://opendocumentfellowship.org/projects/odfpy` According to documentation for odfpy: "Odfpy aims to be a complete API for OpenDocument in Python. Unlike other more convenient APIs, this one is essentially an abstraction layer just above the XML format. The main focus has been to prevent the programmer from creating invalid documents. It has checks that raise an exception if the programmer adds an invalid element, adds an attribute unknown to the grammar, forgets to add a required attribute or adds text to an element that doesn't allow it."
* OpenDocumentPHP (`http://opendocumentphp.org/`) – which is at the early stages of development
* There's a .NET library that is for sale: `http://www.independentsoft.de/odf/`

* I would think that there should be some decent Java libraries since OpenOffice.org, probably the single best parser and generator of ODF documents, is largely written in Java. I don't know of standalone Java libraries that use the logic built in OO.o – but there are libraries that depend on OO.o itself (see below).

## odfpy

Maybe using ODFPY will help us generate minimal documents easily. Let's give it a try. To use it, follow the documentation at

```
http://opendocumentfellowship.org/files/api-for-odfpy.odt
```

To install the library:

```
svn export http://opendocumentfellowship.org/repos/odfpy/trunk odfpy

python setup.py install
```

To generate a Hello world document:

```
from odf.opendocument import OpenDocumentText
from odf.text import P

textdoc = OpenDocumentText()
p = P(text="Hello World!")
textdoc.text.addElement(p)
textdoc.save("helloworld_odfpy.odt")
```

you will get helloworld_odfpy.odt with the following file structure:

```
File Name                                    Modified             Size
mimetype                                 2007-06-04 14:46:50        39
styles.xml                               2007-06-04 14:46:50       451
content.xml                              2007-06-04 14:46:50       514
settings.xml                             2007-06-04 14:46:50       383
meta.xml                                 2007-06-04 14:46:50       429
META-INF/manifest.xml                    2007-06-04 14:46:50       854
```

But the generated instance doesn't validate, even though OO.o 2.2 has no problem reading the file. For many practical purposes, this may be OK – though it'd be nice to know that a document coming out of odfpy is valid since that's the stated design goal of odfpy.

## OpenDocumentPHP

In this subsection, I use OpenDocumentPHP version 0.5.1, which you can get from

```
http://downloads.sourceforge.net/opendocumentphp/OpenDocumentPHP-0.5.1.zip
```

Documentation of the API is currently auto-generated:

```
http://opendocumentphp.org/static/apidoc/svn/
```

Unzip the file in your PHP library area. To see a reasonably complicated example of what you can do, consult the samples in OpenDocumentPHP/samples.

Here I will write a simple helloworld generated document right now to demonstrate how to get started with the library.

```
<?php
require_once 'OpenDocumentPHP/OpenDocumentText.php';
$text = new
OpenDocumentText('D:\Document\PersonalInfoRemixBook\examples\ch17\helloworld_opendoc
umentphp.odt');
$textbody = $text->getBody();
$paragraph = $textbody->nextParagraph();
$paragraph->append('Hello World!');
$text->close();
?>
```

I think that this code should work – but it causes the same error as the sample program. So at this point, I'd wait until OpenDocumentPHP moves a bit further along.

# Leveraging OO.o to generate ODF

If you are willing and able to have OpenOffice.org installed on your computer, it is posible to use OO.o itself as a big library of sorts to parse and generated your ODF documents – and to convert ODF to and from other formats. Libraries/tools that use this approach include:

* JOOC Java Library (http://jooreports.sourceforge.net/?q=jooconverter)
* OOoLib – Perl and Python libraries that use OO.o (http://sourceforge.net/projects/ooolib/)

In Win32 oriented systems, you can access OpenOffice.org via a COM interface. For instnace, the following Python running the win32all library will generate a new .odt document by scripting OO.o:

```
import win32com.client

objServiceManager = win32com.client.Dispatch("com.sun.star.ServiceManager")
objServiceManager._FlagAsMethod("CreateInstance")
objDesktop = objServiceManager.CreateInstance("com.sun.star.frame.Desktop")
objDesktop._FlagAsMethod("loadComponentFromURL")

args = []
objDocument = objDesktop.loadComponentFromURL("private:factory/swriter", "_blank",
0, args)
objDocument._FlagAsMethod("GetText")
objText = objDocument.GetText()
objText._FlagAsMethod("createTextCursor","insertString")
```

```
objCursor = objText.createTextCursor()
objText.insertString(objCursor, "The first line in the newly created text
document.\n", 0)
```

---

Note: It will take some amount of research to make effective use of these interfaces.

---

# Ecma Office Open XML (OOXML)

Now we turn to a competing file format: Office Open XML. The Wikpedia provides a good overview of the specification that undergirds Microsoft Office 2007 is

`http://en.wikipedia.org/wiki/Office_Open_XML`

The Office Open XML specification has been made into an ECMA standard (ECMA-376). The specification can be found at:

`http://www.ecma-international.org/publications/standards/Ecma-376.htm`

Note that the standard runs to 6000 pages – in case you want to read it!

ECMA provides an overview white paper:

`http://www.ecma-international.org/news/TC45_current_work/OpenXML%20White%20Paper.pdf`

Getting hard, easy-to-digest information on OOXML is challenging. I recommend the following, more colloquial overviews that you might find useful

* 5 Cool Things You Must Know About the New Office 2007 File Formats (`http://www.devx.com/MicrosoftISV/Article/30907/2046`)
* http://openxmldeveloper.org/default.aspx might have useful tutorials on the subject.

In working with Office Open XML, it's good to heed the following warning: "Open XML is a new standard. So new, in fact, that the schemas are still being edited and haven't been published by Ecma yet. And there are no books out on Open XML development, although that will surely change in the next year."[12] Although ECMA has published schemas, I still find it a challenge to get my head around the details of OOXML.

The Office Open XML format has a predecessor in the Microsoft Office 2003 XML format. In the book *Office 2003 XML*, the following was given as a minimalist document Office 2003 XML document:

```
<?xml version="1.0"?>
<?mso-application progid="Word.Document"?>
<w:wordDocument
  xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml">
  <w:body>
    <w:p>
      <w:r>
        <w:t>Hello, World!</w:t>
```

---

[12] http://openxmldeveloper.org/articles/LearningOnline.aspx accessed on June 5, 2007.

```
      </w:r>
    </w:p>
  </w:body>
</w:wordDocument>
```

This document is actually readable by Microsoft Office 2007, though in "compatibility mode". Can we get a valid document by using the Microsoft Office 2003 document and updating the namespace of the document? That is, Can we just update the namespace for w?

```
xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
```

to generate

```
<?xml version="1.0"?>
<?mso-application progid="Word.Document"?>
<w:wordDocument
  xmlns:w=" http://schemas.openxmlformats.org/wordprocessingml/2006/main ">
  <w:body>
    <w:p>
      <w:r>
        <w:t>Hello, World!</w:t>
      </w:r>
    </w:p>
  </w:body>
</w:wordDocument>
```

No – that doesn't work. We can certainly keep pushing in this direction by looking through the spec and schema. However, the the most promising lead right now is to see what gets written out by a simple little C# script at

```
http://blogs.msdn.com/dmahugh/archive/2006/06/27/649007.aspx
```

I downloaded the free (as in free beer) Microsoft Visual Studio C# (Express Edition) to run the script and made a small change to update the namespace – from

```
http://schemas.openxmlformats.org/wordprocessingml/2006/3/main
```

to

```
http://schemas.openxmlformats.org/wordprocessingml/2006/main
```

With that change, I was able to generate a simple Office Open XML document file (`http://examples.mashupguide.net/ch17/helloworld_simple.1.docx`) that is acceptable by Microsoft Office 2007. (This doesn't prove that the file is valid but only that we are on the right track in terms of generating OOXML.)

Unzipping and studying the file gives you insight into what goes into a minimal instance of OOXML. The list of files is:

```
File Name                                    Modified            Size
word/document.xml                      2007-06-04 16:43:44        246
[Content_Types].xml                    2007-06-04 16:43:44        346
_rels/.rels                            2007-06-04 16:43:44        285
```

Let's look at the individual files. The first is the `document.xml` file in the `word` directory, which holds the "content" of the document and corresponds most closely to `content.xml` in ODF.

```
<?xml version="1.0" encoding="utf-8"?>
<w:document
xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
  <w:body>
    <w:p>
      <w:r>
        <w:t>Hello World!</w:t>
      </w:r>
    </w:p>
  </w:body>
</w:document>
```

The `.rels` file in the `_rels` directory contains information about "relationships" among the various files that make up the package of files (a bit like the `META-INF/meta.xml` file in ODF):

```
<?xml version="1.0" ?>
<Relationships
xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
  <Relationship Id="rId1" Target="/word/document.xml"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/officeDocume
nt"/>
</Relationships>
```

The final file in the package is `[Content_Types].xml`:

```
<?xml version="1.0" ?>
<Types xmlns="http://schemas.openxmlformats.org/package/2006/content-types">
  <Default ContentType="application/vnd.openxmlformats-
officedocument.wordprocessingml.document.main+xml" Extension="xml"/>
  <Default ContentType="application/vnd.openxmlformats-package.relationships+xml"
Extension="rels"/>
</Types>
```

These files should give you a feel of what's in OOXML. To learn more, take a look at the following resources:

* The "Ecma Office Open XML Format Guide" is an official high level conceptual/marketing overview of OOXML. [13]
* http://openxmldeveloper.org/articles/directory.aspx lists tutorial articles that are gathered by the OOXML community.

---

[13] http://office.microsoft.com/en-us/products/HA102057841033.aspx

* http://openxmldeveloper.org/articles/OpenXMLsamples.aspx has sample OOXML documents.
* http://msdn2.microsoft.com/en-us/library/bb187361.aspx gives the object model of Microsoft Office 2007.
* http://en.wikipedia.org/wiki/User:Flemingr/Microsoft_Office_2003_XML_formats documents the older Office 2003 XML format, which has some family resemblance to OOXML – though an unclear on e to me.
* Brian Jones of Microsoft has written some clear tutorials on generating spreadsheets in OOXML: http://blogs.msdn.com/brian_jones/archive/2007/05/29/simple-spreadsheetml-file-part-3-formatting.aspx

### Viewers/Validators for OOXML

Presumably a big point of OOXML is being able to read and generate documents that are readable in the latest versions of Microsoft Office without having to directly manipulate the object models of Microsoft Office. Yet, it's always helpful to have tools that view and validate OOXML documents – other than Microsoft Office 2007 itself. Some promising tools are:

* Open XML Package Explorer, which lets you browse and edit Open XML packages and validate against the ECMA final schemas. (http://www.codeplex.com/PackageExplorer)
* If you are using Microsoft Office XP and 2003, you can download a "Microsoft Office Compatibility Pack for Word, Excel, and PowerPoint 2007 File Formats" to read and write OOXML.[14] This Compatiblity Pack will also enable you to use the free Microsoft Office Word Viewer 2003 and Microsoft Office Excel Viewer 2003 to view Word 2007 and Excel 2007 files.[15]

## Comparing ODF and OOXML

I will not get into surveying the complicated and often heated comparisons made between ODF and OOXML other than to refer you to the following articles, which in turn provide more references:

* http://en.wikipedia.org/wiki/Comparison_of_OpenDocument_and_Office_Open_XML_formats
* http://weblog.infoworld.com/realitycheck/archives/2007/05/odf_vs_openxml.html

gives a flavor of the conflation of political, economical, PR, technical issues.

---

[14] http://www.microsoft.com/downloads/details.aspx?FamilyId=941b3470-3ae9-4aee-8f43-c6bb74cd1466&displaylang=en

[15] http://support.microsoft.com/kb/925180

# Online Office Suites

Web-based offices suites are emerging, in addition to the traditional desktop office suites and their respective file formats. Prominent examples of such applications include the Zoho Office Suite (http://zoho.com/) and Google Docs and Sheets (http://docs.google.com/). There are others, of course. See a list of online spreadsheets, for instance:

`http://en.wikipedia.org/wiki/List_of_online_spreadsheets`

I will focus specifically on using a programmable online spreadsheet – specifically Google Spreadsheet – in this section. The Google Spreadsheet has an API (which we will use in a mashup later in the chapter):

`http://code.google.com/apis/spreadsheets/overview.html`

# Usage Scenarios for Programmable Online Spreadsheets

What might one want to do with an online spreadsheet? Here are a few examples I brainstormed:

* Tracking one's weight, finances, or time and sharing that info with your family and friends – or not!
* Having bots calculate data that they put into my spreadsheets that I can then analyze. For instance, if I wanted to track my stock portfolio, I could use the StrikeIron fee-based real-time stock quote service to calculate the value of my portfolio. (I might think twice before storing that portfolio info online, however – but this is feasible in principle.) .
* Build an application to track and disseminate grades
* Manage a wedding database
* Build a project management tool which I can update and read with the API.
* Backup a list of my del.icio.us bookmarks in a spreadsheet form
* my library books
* build online charts: http://imagine-it.org/google/spreadsheets/makechart.htm

There are many other applications. Consider StrikeIron SOA Express™ for Excel (http://www.strikeiron.com/tools/tools_soaexpress.aspx) as a source of hints about what people might do with Google Spreadsheet API – if you started to think of the Google spreadsheet as Excel into the cloud but account for its lack of some of Excel's current internal exstensibility (i.e., macros.) (There is no equivalent to Google Mapplets for the Spreadsheet or VBA macros – yet.)

 

The application I will demonstrate in detail is copying my amazon.com wishlist to a spreadsheet to more easily take that information with me (say to a real-life bookstore or library) – and current price.

## Google Spreadsheet API

Let's figure out how to use the Google Spreadsheet API, focusing specifically on PHP and Python wrapper libraries. You can also directly manipulate the feed protocol:

`http://code.google.com/apis/spreadsheets/developers_guide_protocol.html`

### PHP

The main page for the service is:

http://code.google.com/apis/spreadsheets/overview.html

Builds on GData. For PHP, that means we can use the Zend Googld Data Client Library and follow the documentation at

`http://framework.zend.com/manual/en/zend.gdata.spreadsheets.html`

The library is under development and is a bit buggy. Here's I discovered so far. I downloaded 1.0.0RC1 and unzipped it to examples.mashupguide.net as
`http://examples.mashupguide.net/lib/ZendFramework-1.0.0-RC1/`

Alas none of the GData demos work out of the box in this release.

Things worked a bit better in a previous version of the Zend GData framework – once I commented out require_once('Zend.php'). Try them at
`http://examples.mashupguide.net/lib/ZendFramework-0.9.3-Beta/demos/Zend/Gdata/`

I've also downloaded the latest version of the ZendFramework (under svn control) to track development of the project, which I hope will respond to bug reports. (http://examples.mashupguide.net/lib/ZendFrameworkCurrent/trunk/)

Basic conclusion: Wait until the full release of the ZendFramework 1.0.0 and then try again....

### Python

Google provides a Python GData library and sample code to access the Google spreadsheet. You can either download specific releases (from `http://code.google.com/p/gdata-python-client/downloads/list`) or access the svn repository

```
svn checkout http://gdata-python-client.googlecode.com/svn/trunk/
gdata-python-client
```

I will use the svn image as of June 5, 2007.

Note the dependencies on other libraries, especially ElementTree, which is not part of the standard Python libraries until version 2.5.[16]

---

[16] http://code.google.com/p/gdata-python-client/wiki/DependencyModules

I highly recommend reading the documentation on the Google site specific to the Python library:

`http://code.google.com/apis/spreadsheets/developers_guide_python.html`

Once you have the Python GData library installed, you can try out some code samples – using the Python interprester -- to teach yourself how it works:

First the obligatory imports:

```
import gdata.spreadsheet.service
```

Let's then declare some convenience functions and variables

```
GoogleUser = "[your Google email address]"
GooglePW = "[your password]"
```

Define the following convenience function:

```
def GSheetService(user,pwd):
    gd_client = gdata.spreadsheet.service.SpreadsheetsService()
    gd_client.email = user
    gd_client.password = pwd
    gd_client.source = 'amazonWishListToGSheet.py'
    gd_client.ProgrammaticLogin()

    return gd_client
```

Instantiate a Google Data client for your spreadsheet:

```
gs = GSheetService(GoogleUser,GooglePW)
sheets = gs.GetSpreadsheetsFeed()
```

To get a list of the spreadsheets, their titles and ids:

```
map(lambda e: e.title.text + " : "  + e.id.text.rsplit('/', 1)[1],sheets.entry)
```

yields something like the following (which is based on my own spreadsheets):

```
['My Amazon WishList : o06341737111865728099.3585145106901556666', 'Udell Mini-
Symposium May 1, 2007 : o06341737111865728099.1877210150658854761', 'weight.journal
: o06341737111865728099.6289501454054682788', 'Plan :
o10640374522570553588.5762564240835257179']
```

Note the key for the spreadsheet "My Amazon WishList" – the spreadsheet we'll be reading from and writing to:

`o06341737111865728099.3585145106901556666`

In the browser, if I'm logged in as the owner of the spreadsheet, I can access:

```
http://spreadsheets.google.com/feeds/spreadsheets/private/full/o06341737111865728099
.3585145106901556666
```

Otherwise, I get a 404 error. Now I need to get the ID of the one worksheet in the "My Amazon Wishlist" spreadsheet.

```
gs.GetWorksheetsFeed(key="o06341737111865728099.3585145106901556666").entry[0].id.te
xt
```

returns

```
http://spreadsheets.google.com/feeds/worksheets/o06341737111865728099.35851451069015
56666/private/full/od6
```

and

```
gs.GetWorksheetsFeed(key="o06341737111865728099.3585145106901556666").entry[0].id.te
xt.rsplit('/', 1)[1]
```

gets you the worksheet id:

```
od6
```

There are two ways to get at the data – either in a "list-based" way that gets you rows or in a "cell-based" way that gets you by a range of cells. We will use the row-based method, which depends on the assumption that the first row is the header row.

For testing purposes, I created a spreadsheet with the header row and one line of data that I entered:

```
ASIN   DetailPageURL  Title  Author Date Added    Price  Quantity Desired

1590598385    http://www.amazon.com/gp/product/1590598385/       Smart and Gets
Things Done: Joel Spolsky's Concise Guide to Finding the Best Technical Talent
(Hardcover)   Joel Spolsky  6/5/2007      13.25  1
```

```
lfeed =
gs.GetListFeed(key="o06341737111865728099.3585145106901556666",wksht_id="od6")
```

returns a feed for the rows (there's only one). You can see the "content" of the row with

```
lfeed.entry[0].content.text
```

which is

```
'ASIN: 1590598385, DetailPageURL:
http://www.amazon.com/gp/product/1590598385/ref=wl_it_dp/103-8266902-
5986239?ie=UTF8&coliid=I1AOWT8LH796DN&colid=1U5EXVPVS3WP5, Author: Joel Spolsky,
Date Added: 6/5/2007, Price: 13.25, Quantity Desired: 1'
```

```
lfeed.entry[0].custom
```

holds the data that has been mapped from namespace-extended elements in the entry (See http://code.google.com/apis/spreadsheets/developers_guide_protocol.html#listFeedExample). Specifically:

```
map(lambda e: (e[0],e[1].text), lfeed.entry[0].custom.items())
```

returns

```
[('asin', '1590598385'), ('dateadded', '6/5/2007'), ('detailpageurl',
'http://www.amazon.com/gp/product/1590598385/ref=wl_it_dp/103-8266902-
5986239?ie=UTF8&coliid=I1AOWT8LH796DN&colid=1U5EXVPVS3WP5'), ('author', 'Joel
Spolsky'), ('quantitydesired', '1'), ('price', '13.25'), ('title', "Smart and Gets
Things Done: Joel Spolsky's Concise Guide to Finding the Best Technical Talent
(Hardcover) ")]
```

Now let's look at adding another row of data. Let's see whether we can just duplicate the row by creating a dictionary of the first row and stick it into the second row.

```
h = {}
for (key,value) in lfeed.entry[0].custom.iteritems():
  h[key] = value.text
```

h now is:

```
{'asin': '1590598385', 'dateadded': '6/5/2007', 'detailpageurl':
'http://www.amazon.com/gp/product/1590598385/ref=wl_it_dp/103-8266902-
5986239?ie=UTF8&coliid=I1AOWT8LH796DN&colid=1U5EXVPVS3WP5', 'author': 'Joel
Spolsky', 'quantitydesired': '1', 'price': '13.25', 'title': "Smart and Gets Things
Done: Joel Spolsky's Concise Guide to Finding the Best Technical Talent (Hardcover)
"}
```

To add the new role:

```
gs.InsertRow(row_data=h,key="o06341737111865728099.3585145106901556666",wksht_id="od
6")
```

To clear the second row that we just added, first we need to get an update lfeed that reflects the current state of the spreadsheet/worksheet:

```
lfeed =
gs.GetListFeed(key="o06341737111865728099.3585145106901556666",wksht_id="od6")
gs.DeleteRow(lfeed.entry[1])
```

---

Note: the Google Spreadsheet Data API is under active development and is still in the process of maturation

---

Since this is a pretty RESTful service based on Atom, that you could write your own fairly generic code to manipulate the feeds instead of relying on the Google library. I am not convinced that you wouldn't just end up having to replicate what the Google library does anyhow.

# Mashup: Amazon wishlist and Google Spreadsheet Mashup

To show how to use the Google Spreadsheet for a simple mashup, I will show you how to write code that will transfer the contents of an Amazon Wishlist to a Google Spreadsheet. Why do that? I use my wishlist to keep track of books and other stuff that I find interesting. If the wishlist belonged to someone else, I might want to download it into a spreadsheet to make it easier to generate a paper shopping list I could use.

## Accessing the Wishlist through the Amazon ECS web service

First, a word about how you can use awszone.com to help you formulate the right Amazon ECS query to get the information you are looking for. I figured out that I wanted to use the "ListLookup" query by using

```
http://www.awszone.com/scratchpads/aws/ecs.us/ListLookup.aws
```

Furthermore, I was using a `ListType=WishList` and the `ListID=1U5EXVPVS3WP5`. The URL for web interface to an amazon wishlist is:

```
http://www.amazon.com/gp/registry/wishlist/[ListID]/
```

I can get info about the list with the following query:

```
    convenience function to return all the text in an array of nodes
    """
    rc = ""
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE:
            rc = rc + node.data
    return rc

GoogleUser = "[YOUR ID]"
GooglePW = "[YOUR PASSWORD]"
GSheet_KEY = "o06341737111865728099.3585145106901556666"
GWrkSh_ID = "od6"

# a sample row for testing the insertion of a row into the spreadsheet
GS_Example_Row = {'asin': '1590598385', 'dateadded': '6/5/2007', 'detailpageurl':
'http://www.amazon.com/gp/product/1590598385/ref=wl_it_dp/103-8266902-
5986239?ie=UTF8&coliid=I1AOWT8LH796DN&colid=1U5EXVPVS3WP5', 'author': 'Joel
Spolsky', 'quantitydesired': '1', 'price': '13.25', 'title': "Smart and Gets Things
Done: Joel Spolsky's Concise Guide to Finding the Best Technical Talent (Hardcover)
"}
GS_HEADER = ['ASIN', 'DetailPageURL', 'Title', 'Author', 'Date Added', 'Price',
'Quantity Desired']
GS_KEYS = ['asin', 'detailpageurl', 'title', 'author', 'dateadded', 'price',
'quantitydesired']

class GSheetForAmazonList:
    def __init__(self,user=GoogleUser,pwd=GooglePW):
        gd_client = gdata.spreadsheet.service.SpreadsheetsService()
        gd_client.email = user
        gd_client.password = pwd
        gd_client.source = 'amazonListToGsheet.py'
        gd_client.ProgrammaticLogin()
        self.gd_client = gd_client
    def setKey(self,key):
        self.key = key
    def setWkshtId(self,wksht_id):
        self.wksht_id = wksht_id
    def listSpreadsheets(self):
        """
        return a list with information about the spreadsheets available to the user
        """
        sheets = self.gd_client.GetSpreadsheetsFeed()
        return map(lambda e: (e.title.text , e.id.text.rsplit('/',
1)[1]),sheets.entry)
    def listWorkSheets(self):
        wks = self.gd_client.GetWorksheetsFeed(key=self.key)
        return map(lambda e: (e.title.text , e.id.text.rsplit('/', 1)[1]),wks.entry)
    def getRows(self):
        return self.gd_client.GetListFeed(key=self.key,wksht_id=self.wksht_id).entry
    def insertRow(self,row_data):
        return
self.gd_client.InsertRow(row_data,key=self.key,wksht_id=self.wksht_id)
```

```
    def deleteRow(self,entry):
        return self.gd_client.DeleteRow(entry)
    def deleteAllRows(self):
        entrylist = self.getRows()
        for entry in entrylist:
            self.deleteRow(entry)


AMAZON_LIST_ID = "1U5EXVPVS3WP5"
AMAZON_ACCESS_KEY_ID = "[AMAZON_ACCESS_KEY_ID]"

class amazonWishList:

    def __init__(self,listID=AMAZON_LIST_ID,amazonAccessKeyId=AMAZON_ACCESS_KEY_ID):
        self.listID = listID
        self.amazonAccessKeyId = amazonAccessKeyId
        self.getListInfo()


    def getListInfo(self):

        aws_url =
"http://ecs.amazonaws.com/onca/xml?Service=AWSECommerceService&Version=2007-05-
14&AWSAccessKeyId=%s&Operation=ListLookup&ListType=WishList&ListId=%s" %
(self.amazonAccessKeyId, self.listID)
        f = openAnything.openAnything(aws_url)
        dom = minidom.parse(f)
        self.title = getText(dom.getElementsByTagName('ListName')[0].childNodes)
        self.listLength =
int(getText(dom.getElementsByTagName('TotalItems')[0].childNodes))
        self.TotalPages =
int(getText(dom.getElementsByTagName('TotalPages')[0].childNodes))
        return(self.title, self.listLength, self.TotalPages)

    def ListItems(self):
        """
        a generator for the items on the Amazon list
        """

        import itertools
        for pageNum in xrange(1,self.TotalPages):
            aws_url =
"http://ecs.amazonaws.com/onca/xml?Service=AWSECommerceService&Version=2007-05-
14&AWSAccessKeyId=%s&Operation=ListLookup&ListType=WishList&ListId=%s&ResponseGroup=
ListItems,Medium&ProductPage=%s" % (self.amazonAccessKeyId,self.listID,pageNum)
            f = openAnything.openAnything(aws_url)
            dom = minidom.parse(f)
            f.close()
            items = dom.getElementsByTagName('ListItem')
            for c in xrange(0,10):
                yield items[c]
```

```
    def parseListItem(self,item):
        from string import join
        from decimal import Decimal

        itemDict = {}

        itemDict['asin'] = getText(item.getElementsByTagName('ASIN')[0].childNodes)
        itemDict['dateadded'] =
getText(item.getElementsByTagName('DateAdded')[0].childNodes)
        itemDict['detailpageurl'] =
getText(item.getElementsByTagName('DetailPageURL')[0].childNodes)

        # join the text of all the author nodes, if they exist
        authorNodes = item.getElementsByTagName('Author')
        # blank not allowed
        itemDict['author'] = join(map(lambda e: getText(e.childNodes), authorNodes),
", ") or ' '

        itemDict['quantitydesired'] =
getText(item.getElementsByTagName('QuantityDesired')[0].childNodes)

        titleNodes = item.getElementsByTagName('Title')
        # blank title not allowed
        itemDict['title'] = join(map(lambda e: getText(e.childNodes), titleNodes),
", ") or ' '

        # to fix -- not all things have a LowestNewPrice
        itemDict['price'] =
str(Decimal(getText(item.getElementsByTagName('LowestNewPrice')[0].getElementsByTagN
ame('Amount')[0].childNodes))/100) or ' '

        return itemDict


def main():

    gs = GSheetForAmazonList(user=GoogleUser,pwd=GooglePW)
    gs.setKey(GSheet_KEY)
    gs.setWkshtId(GWrkSh_ID)

    aWishList =
amazonWishList(listID=AMAZON_LIST_ID,amazonAccessKeyId=AMAZON_ACCESS_KEY_ID)
    items = aWishList.ListItems()
    print "deleting all rows..."
    gs.deleteAllRows()
    for item in itemsd:
        try:
            h = aWishList.parseListItem(item)
            print h['asin']
        except Exception, e:
            print "Error %s parsing %s" % (e, item.toprettyxml("  "))
        try:
```

```
            gs.insertRow(h)
        except Exception, e:
            print "Error %s inserting %s" % (e, h['asin'])


if __name__ == '__main__':
    main()
```

Things to note about this code:

* The GSheetForAmazonList class provides convenience methods for the Google GData library.
* The error handling is essential since not all wishlist items necessarily have all the pieces of data requested. It's important for the code to keep moving even if data is missing.
* At least in the Python GData interface to Google Spreadsheets, you can't insert blank cells.
* The amazonWishList.ListItems is a Python generator, which creates an iterator to parcel out the Amazon items one at a time. See http://www.ibm.com/developerworks/library/l-pycon.html?t=gr,lnxw16=PyIntro for a tutorial on Python generators.

## Variation: Amazon WishList to Microsoft Excel Via COM

With code to access the Amazon wishlist in hand, you can COM programming to generate an Excel spreadsheet with the same information. To learn more about the details about how to do so, consult Chapter 12 of *Python Programming on Win32*[18]

```
from amazonListToGSheet import GS_HEADER, amazonWishList, AMAZON_LIST_ID,
AMAZON_ACCESS_KEY_ID, GS_KEYS
from win32com.client import Dispatch

# fire up the Excel application
xlApp = Dispatch("Excel.Application")
xlApp.Visible = 1
xlApp.Workbooks.Add()

# write the headers
col = 1

def insertRow(sheet,row,data,keys):
    col = 1
    for k in keys:
        sheet.Cells(row,col).Value = data[k]
        col += 1
```

[18] http://www.oreilly.com/catalog/pythonwin32/chapter/ch12.html

```
for h in GS_HEADER:
    xlApp.ActiveSheet.Cells(1,col).Value = h
    col +=1
# now loop through the amazon wishlist

aWishList =
amazonWishList(listID=AMAZON_LIST_ID,amazonAccessKeyId=AMAZON_ACCESS_KEY_ID)
items = aWishList.ListItems()

row = 2
for item in items:
    try:
        p = aWishList.parseListItem(item)
        print p['asin']
    except Exception, e:
        print "Error %s parsing %s" % (e, item.toprettyxml("  "))
    try:
        insertRow(xlApp.ActiveSheet,row,p,GS_KEYS)
        row += 1
    except Exception, e:
        print "Error %s inserting %s" % (e, p['asin'])
```

# Zoho APIs

Zoho (http://zoho.com) has been generating a good amount of attention for its online office suite, which is the most comprehensiveone available right now.[19] Among its offerings are Zoho Writer (a word processor), Zoho Sheet (a spreadsheet), and Zoho Show.

There are currently APIs for Writer, Sheet, and Show:
`http://writer.zoho.com/public/help/zohoapi/fullpage`

Currently, the APIs do not talk deeply to pieces of documents. There are storage APIs that let you upload and download documents. To access pieces, you have to use the techniques shown in the rest of this chapter to parse and write documents. (For instance: http://writer.zoho.com/public/help/userView.AddWorkbook/noband) I suspect that API calls to access parts of documents will follow at some point in the future.

# Summary

* There's huge potential for mashups with both desktop and web-based office suites, though there are barriers
* Tthere's a lot of work that has gone into the file formats ODF and OOXML but it's hard to figure out how to make sense of some basic aspects of the file formats.

---

[19] http://www.technologyreview.com/Biztech/18816/ provides a useful analysis of the Zoho vs Google Docs battle and how they compare.

* The Google Spreadsheet API is fairly easy to use; Google has spent a fair amount of effort not only building the underlying REST protocol but also writing API kits in various languages, and providing usable documentation.