Python-based XML identification demo

Johan van der Knijff, KB / National Library of the Netherlands, 11 July 2011

Disclaimer

Introduction

This document describes a simple script that demonstrates the identification of XML files using the standard Python libraries for XML parsing. The script can be used for analysing either one file at a time, or all files in a user-defined directory tree.

The main purpose of the demo is to demonstrate how the identification is done (using the functions 'isXML' and 'parseFile'), and how these functions are used. In addition, the script gives an indication of the computational performance that can be achieved.

Dependencies

In order to use the "isXMLDemo.py" software, you'll need a recent version of the (free) Python interpreter. The software should work with Python versions 2.6 and more recent. The code is fully compatible with Python 3. If you don't have Python already, you can download it from the following location:

http://www.python.org/download/

If you're using the software under Linux/Unix, Python is already be pre-installed on your system (although older versions may not work).

Installation

Windows

On Windows-based systems, simply copy the file "isXMLDemo.py" to any location you like. If Python is installed correctly, files with a ".py" extension are by default associated with the Python interpreter.

Unix/Linux

The installation under Unix/Linux involves the following steps:

1. Copy the file “isXMLDemo.py” to any location you like.

2. Use the following command to convert DOS/Windows linebreaks to Unix ones:

```
dos2unix isXMLDemo.py
```

3. Make the file executable using the following command:

```
chmod 755 isXMLDemo.py
```

You can now test the installation by typing

```
./isXMLDemo.py
```

If all goes well, you will now see something like this:

```
isXMLDemo.py version 11 July 2011 (JvdK)

Checks file (or collection of files in directory tree) for XML well-formedness

USAGE: isXMLDemo.py <fileIn> <fileOut>

 fileIn         : input file or directory
                    If fileIn is a directory, all files within that
                    directory and its sub-directories will be analysed
 fileOut        : output file
```

If this doesn’t work, check that you’re using the correct command line interpreter, which is set in the first line of the script:

```
#! /usr/bin/env python
```

Here, the command line interpreter is set to environment variable ‘python’, which points to the Python interpreter on most systems. If this isn’t the case on your system, or if you want to use an alternative on, just modify this line.

Using the demo

The functionality of the script is very simple: for any given file object, it attempts to establish whether the format of the file is XML. This is done by throwing the contents of each file at a standard XML parser. The following outcomes are possible:

1. The contents of the file are parsed successfully. This means that it contains well-formed XML.

2. The contents of the file cannot be parsed, which results in the XML parser raising an exception. This means that the file does *not* contain well-formed XML.

The results of the analysis are written to a formatted text file.

Usage

The script takes two command line arguments:

```
isXMLDemo.py <fileIn> <fileOut>
```

With:

fileIn	: input file or directory
fileOut	: output file

If *fileIn* is a directory, the script will recursively analyse all files in that directory and its sub (sub … sub) directories.

Usage examples

Single file:

```
isXMLDemo.py rubbish.pdf resultRubbish.csv
```

Result: file “rubbish.pdf” is analysed, and results are written to “resultRubbish.csv”.

Directory tree:

```
isXMLDemo.py D:\myRandomRubbish\ resultRubbish.csv
```

Result: all files in directory “myRandomRubbish” are analysed, and results are written to “resultRubbish.csv”.

Output format

The output file is a comma-delimited text file. Each line contains two items: the name of the analysed file (and its path), followed by the result of the analysis. The analysis result can be any of the following:

1. ‘isXML’: file contains well-formed XML
2. ‘noXML’: file *doesn’t* contain well-formed XML
3. ‘readError’: file could not be read (which could be the result of ‘permission denied’ errors, among other things)

An example:

```
"d:\aipSamples\prdrm_4188\unpacked\SIP_toc.xml",isXML
"d:\aipSamples\prdrm_4188\unpacked\SIP_toc_1.xml",isXML
"d:\aipSamples\prdrm_0647\unpacked\Original metadata\fd1.gif",noXML
"d:\aipSamples\prdrm_6459\unpacked\OriginalEpublication\main.pdf",noXML
```