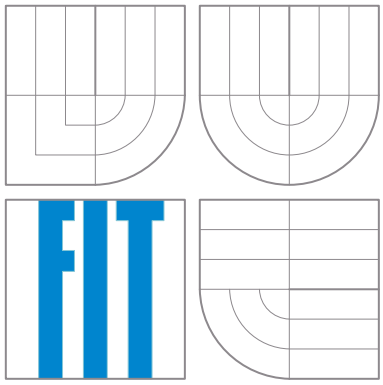

IPP 2014/15 Project 2 Documentation

21. dubna 2015

Tomáš Pružina xpruzi01@stud.fit.vutbr.cz

Fakulta Informačních Technologií
Vysoké Učení Technické v Brně

Obsah

1 Introduction **1**

2 Design **1**

3 Implementation overview **1**

3.1 Argument parsing 1

3.2 CSV parsing 1

3.3 XML conversion 1

4 Implementation details **2**

5 Extension: padding **2**

6 Ending thoughts **2**

1 Introduction

The main goal of this project was to create a python script which would convert file in 'Comma Separated Values' format into the Extensible Markup Language format (XML) as specified in projects formal requirements.

2 Design

Projects implementation was fairly straight-forward, commandline arguments are parsed via pythons argparse standard library, each optional argument parameter is furtherchecked for format correctness. Conversion of CSV input onto XML output is done on the fly, line by line.

Since the conversion follows straight-forward program flow, all of the critical program exceptions cause immediate program execution termination.

While python3 supports UTF-8 natively, all of the parsing functions (such as csv.arg_parse, open) are setup to use exclusively utf-8 encoding.

Source code as well as comments and documentation are written entirely in english.

3 Implementation overview

3.1 Argument parsing

Upon script execution, argument parsing function is called. This initializes all the variables required for script execution as well as prepares input/output, verifies that input file exists (if –input= option was given) and that output destination is writeable (if –output= option was given).

After argparse standard function parses and matches argument options onto program variables, program checks for incompatible argument options.

This is followed by argument parameters checks, such as that -s=separator really only contains a single character as defined by project requirements.

Option –help is generated automatically via argparse library as well as handling of unsupported arguments.

3.2 CSV parsing

CSV file format parsing is implemented via Python's standard 'csv' library. While this library allows for many dialets and options, minimal preset was chosen and more advanced parsing is done during conversion process manually. This includes conversion of characters unsupported by XML format, such as '<','>','&', etc.

3.3 XML conversion

XML format comes in many flavors and implementation in this project is based off provided reference output format. No special python library was used, XML is created line-by-line and stored in memory buffer. If parsing is successful, this buffer is then flushed onto output, be it stdout or specified file.

4 Implementation details

Script is stored within single file, csv.py which declares a single class csv2xml that takes care of entire script functionality.

Upon class initialisation, parse_cmdline function is called. This parses all the command line arguments and also reads input into the memory buffer. This is required for PAD extension (–padding), since requires input preparsing which isn't possible to do when we assume that input is read in streamed manner.

After argument parsing and initialisation, csv2xml function. This function contains main bulk of scripts functionality and spans nearly 2 hundreds of LoC. It didn't made much sense to split it up, since most of the code is only used once and functionality that is used more than once (such as padding helper function) is implemented in separate helper functions.

The function contains lenghty initialisation that consideres several program arguments, but the main body is fairly simple and made of two nested cycles, one that parses each line of CSV file and one that parses each column.

As this is done, the output is continously generated, untill there is nothing left to be read and output is stored or an error occurs and program terminates with appropriate error code.

5 Extension: padding

Padding extension implementation required two-pass input parsing. First pass was needed to count number of all the lines and columns on each line in CSV input, second pass was used during conversion itself and XML output generation. After all of the counters were initialized during parsing, with little help from helper functions, all of the indices were padded with leading zeroes to provide uniform and compact output.

Padding extension however made the code less readable and slightly harder to understand due to aditional logic required due to error handling options (-e, –all-fields).

6 Ending thoughts

This project also was my first hands-on experience with python3, an as such, I don't expect that my implementation was always using the best approach and took all advantages language has to offer.

For development, I used Gentoo Linux, vanilla Python 3.2.4 and Eclipse IDE extended by PyDev plugin which allowed for easier debugging. Program was also staticly analysed with Pylint program.

While JExamXML was officially recommended, initially I only used standard unix diff against my results and reference output.

Testing was done on top of provided reference tests, script was first and foremost tuned to satisfy basic reference tests.