Task documentation: JSN: JSON2XML in PHP 5 for IPP 2015/2016
Name and surname: Martin Krajňák
Login: xkrajn02

Problem analysis

The goal of this task, is to create a script, which properly converts data stored in JSON format to XML. The output of the script may vary, as it depends on parameters entered by user. Script is able to work with files, but it is also able to handle input data from standard input and write output data to standard output of command line. Every XML file is created to meet XML standards.

Solution

Setup

At first, script creates instance of class `Options opt`. This class is providing all available options, most of them are used to properly format XML output. This is achieved by boolean variables for every option possible. For example if `-n` parameter is entered, value `$opt→generate_header` is set to false. There are also variables, which define default names of XML tags and options. This instance is passed to every part of the script, so it can easily alter the output by user's demands.

Parsing arguments

Next thing, which is done by the script is parsing arguments and properly setting every option defined by user. In certain cases it is needed to extract the values entered in to the command line. This applies to parameters `--input`, `--output`, `-r`, `--array-name`, `--item-name`, `--start`, `-h`. This functionality is handled through regular expressions (library PCRE), which is also used to check validity of XML elements entered by user. If any of processed arguments are not recognized, script will end with an error message. When this occurs, it is recommended to use `--help` parameter to see a proper form of every option that could be written by user.

Input data

There are two supported ways to deliver input data to the script. First, is to use `--input` option with the name of JSON file. It will open the file (assuming user has rights to read the file), read the content and process it. Second option is to execute it excluding option `--input`. Afterwards, it will wait until user inserts some data. Processed data are checked, whether they are valid JSON data or not, and transformed to object by function `json_decode`. This function is also very important, because it is able to check validity of input JSON data. If input data are not valid, the script ends with an error.

Writing data

Data, which need to be written, could be in different forms – objects, arrays, values. Script is iterating through every piece of data, therefore the type of data is recognized and written out depending on its type. There are some special cases, that need special handling. Those are cases, where data are stored deep inside arrays or objects. For this purpose are used recursive functions, that are able to properly handle every possible case.
At the beginning, function `write_json_to_xml` is checking, whether input object or array is empty. This way the script is not necessarily iterating and is able to write out the output quickly, without any loops or recursions. Different types of data values, need different treatment. Numerical values could be stored in input JSON as a string, an integer, a float or a number with exponent. The script is using the proper set of conditions to properly recognize datatype and write it correctly. Float numbers are also converted to integers and rounded down automatically. Boolean values are parsed as 1 or 0 from input. It is converting them to string values true or false before writing them out. User can define own names of certain tags. Default names of tags are stored in object `Opt` as string values. If user uses parameters to change them, they are changed and checked to verify whether they meet conditions for valid XML tags.

This is again achieved by regular expressions, that are checking if new names start and have all characters valid. Same principle is used in case with `-c` option, where certain characters are replaced in string values. The most problematic character is &, but the function `write_raw` provided solution for this case.

Testing and development

This tool was developed on Linux platform, using open source code editor Atom and versioning system Github. For testing, I was using a test script written in PHP, with JExamTool to compare and evaluate the output. Script was tested simultaneously on Merlin server but after the project grew bigger I decided to use CI integration provided by circleCI. Therefore every line of code I added was automatically tested and archived(`is_it_okay` test included).