Task documentation JSN: JSON2XML in PHP 5 for IPP 2016/2017
Name and surname: Jozef Urbanovský
Login: xurban66

1 Assignment

The goal was to create a script in PHP 5, which interprets an input file in JSON format to XML. The resulting XML output contains elements meeting the requirements of an XML naming rules and format. The program utilizes a number of libraries available on Merlin, e.g. `JSON Functions` to read an input file and `XMLWriter` to output processed text in XML format.

2 Implementation

The program as whole, is implemented in a single file, that can be divided into smaller subroutines, which are described below. Each error is being handled by built-in class `ErrorException` with suitable error message and exit code. Individual errors are caught by `try...catch` construction.

2.1 Parse command line options

For this task a simple command line parser was written. Extensive use of regular expressions was key in this implementation. Each argument is being inspected whether it is syntactically correct and meets the required format. This function also verifies whether arguments are set only once and have correct dependencies. Arguments with their respective parameters are stored in an array for further needs. If argument is not set and is supposed to have a default value, it is handled in further inspections. Should an error occur while parsing arguments `ErrorException` is called and program is terminated.

2.2 File manipulation

Both input and output file are handled in function `file_handler()`. Contents of input file are loaded by `file_get_contents()` and converted to a mixed array by `json_decode`. Output of this function is later checked whether input file was in correct JSON format, should it not be, exception is evoked. Output file which is going to contain XML is opened for write, alternatively created and its handle is later used by XMLWriter.

2.3 Conversion

Conversion of an input JSON file is split into a few functions. Most fundamental ones are `xml`, `xml_object`, `xml_array` and `xml_element` with their respective types. `XMLWriter` is created and initialized to use indent in XML and eventually header is not set otherwise. Decoded input consisting of objects and arrays. These types are processed and further decoded thanks to recursive calls of their respective functions. Each element type supported has its own function to be handled within. Conversion of individual elements also accounts for its validity in XML format, by calling function `validate_xml_name` before writing to an output file, which happens to also substitute unsuitable characters. If element does not meet requirements of XML format (even after substitution) exception is evoked.

3 Extensions

3.1 Padding

To implement this extension parser was expanded to accept padding option and to check its dependency on index-items. To function `xml_array` have been added variable that specifies minimal number of characters needed to output. Afterwards given variable was used in function `str_pad` to determine length to which padding is going to add zeroes from left if needed.

4 Conclusion

Created script was properly tested by set of supplementary test provided to students. Additional 37 tests were written. All testing was automatized with a use of bash script to check output and error codes. Output was afterwards verified with `JExamXML` with reference outputs. All tests were executed on Ubuntu 16.04 OS and server Merlin with CentOS.