Dokumentace úlohy XQR: XML Query v PHP 5 do IPP 2015/2016
Jméno a příjmení: Jan Ondruch
Login: xondru14

IPP Project 1 (XQR) Documentation

Assignment

The goal of this project was to create a script in a programming language PHP 5, that parses input data formatted as XML text. Data selection is specified by a given query. Using additional program arguments, it's possible to specify further output options, such as an XML file header, or a root element wrapping the selection.

Program Logic

Argument parsing

For argument parsing I used an embedded function `getopt`, that saves input arguments in a multi-dimensional array. It checks validity of the arguments, forbidden options, further checking as e.g. a unique occurrence of one argument and saving content into variables for the next processing is worked out in a foreach cycle, iterating through the whole array. If help option is chosen, the program prints out a help message and terminates.

Parser

The parser itself is created as a class in a file *Parser.php* (which is required in the main *xqr.php* file). Construct of the class sets an input file, an output file (*STDIN*, *STDOUT* alternatively) and the input text for processing. Validity and accessibility of the files is checked, file descriptors are set and the XML text – if given by -qf parameter, is saved to a class variable `xmlText`.

Function `parseQuery()` is responsible for the whole XML parsing. Firstly, it creates a new *SimpleXMLElement* class object from the XML text, checking validity of the data, too. As an subsequent action, regular expression representing query's context-free grammar is created and used for the XML data validation, as well as for its parsing. Grammar is divided into 5 parts transferred into an array (`queryParSecs`) parts, each comprising of one of keywords, such as *SELECT*, and a regular expression describing possible sequence of characters, which can occur up to the next keyword. The whole array is then checked in a for cycle using `preg_match` function. Each part is saved to another array, that is eventually compared to the original query. If both of `preg_match` in the cycle and comparison succeed, the query is syntactically valid.

```
[0]=>
  array(2) {
    ["KEYWORD"]=>
    string(6) "SELECT"
    ["REGEX"]=>
    string(18) "/(?<=SELECT\s)\w+/"
  }
  [1]=>
  array(2) {
    ["KEYWORD"]=>
    string(5) "LIMIT"
   ...
```

Splitting array into separate parts utilizing `preg_split` function, that exactly match the grammar and its elements, prepares the data for a semantic analysis. Hence method *XPath* (XML Path Language) can be used now. On the basis of *XPath* grammar, each part of the query is processed. The full path of the selection is separated into 3 parts – *SELECT*, *FROM* and *WHERE*. All of them compose a part of the final path, saving elements / attributes / literals / conditional operators, utilizing elements of the `queryParSecs` array. In case of the *WHERE* section, multiple *NOT* keyword is dealt with, possibly negating the selection. If *CONTAINS* occurs, it's succeeding literal is checked for validity by means of regular expression. Final path is composed of those 3 key parts, in such order:

```
$xpathFull = $xpathPartFrom . $xpathPartSelect . $xpathPartWhere
```

Limit of eventually selected XML nodes is set by `array_slice` function. Possible ordering of the nodes is not fully implemented, only is checked during the initial syntactic analysis. Additionally, validity of element names is checked, thus collision with keywords is prevented.

Thereafter, parsed XML data is converted into a string in a `GetXmlText` function. If required by arguments, the root element is added to the data, in case of XML header addition (when `-n` parameter is omitted), formal header text is concatenated with the data. Eventually, XML data is outputted.

Errors

Errors are handled by try-catch mechanism. When an error is found, an exception is thrown and the program terminates with a corresponding error code.