Dokumentace úlohy CSV: CSV2XML v Perl do IPP 2011/2012
Jméno a příjmení: Jan Hrivnák
Login: xhrivn01

Dokumentace k projektu IPP 2011/2012, CSV: CSV2XML

Analýza zadání

Cílem tohoto projektu bylo vytvořit skript v jazyce Perl, který bude převádět zdrojový soubor ve formátu CSV na výstupní soubor XML. Podobu výsledného souboru je možné upravovat spuštěním skriptu se zadanými přepínači. Je možné vytvořit i výstupní soubor neodpovídající přesné specifikaci XML z důvodu možnosti spojení více výstupních souborů do jediného.

Postup řešení

Po spuštění skriptu se jako první provádí kontrola zadaných parametrů. Stejně tak jsou definovány důležité proměnné, které jsou potřeba dále ve skriptu. Vzhledem k implementovanému rozšíření padding se zde také počítá, na kolik míst bude potřeba doplňovat tisknuté indexy řádků.

Dále se připraví pole pro názvy sloupců, které se poté využívá při tisku výsledného XML(název sloupců se generuje tedy pouze jednou, nikoliv na každém řádku. Název colX se generuje znovu pouze v případě spuštění s rozšířením padding, když je v řádku více sloupců než v hlavičce-aby bylo možné ošetřit případ kdy je v hlavičce 9 sloupců, tj. zarovnává se na jedno místo, ale v jednom z řádků je sloupců 10 a zápis colX se musím zarovnat na dvě místa). V případě zadání parametru *-h* se tyto názvy generují z prvního řádku vstupního souboru, v opačném případě se názvy generují jako colX podle zadání.

Dále se prochází vstupní soubor po řádcích a rovnou se generuje výstupní soubor. Během tohoto vypisování se samozřejmě upravují příslušné nepovolené znaky a kontroluje se, zda je povolen jejich výskyt v daném místě. Stejně tak se kontroluje počet sloupců v jednotlivých řádcích. Případy, kdy je mnoho nebo naopak málo sloupců v daném řádku, se přímo na místě ošetřují tiskem chybového hlášení, nebo v případě zotavování z chyb parametrem *-e*, se zde upraví i výpis od požadované podoby. V případě úspěšného dokončení této části se uzavřou oba používané soubory.

Za zmínku k mému řešení rovněž stojí nepřílišné rozdělování do funkcí. Skript je tvořen víceméně jednou velkou funkcí. Zvláště je vyčleněno pouze několik menších částí, které by se jinak v kódu opakovaly (úprava & entit, přepočet čísla u rozšíření padding, tisk chybového …). Tento způsob řešení jsem zvolil především z důvodů přehlednosti. Skript vykonává jednotlivé části postupně a nijak se mezi nimi nepřeskakuje, tudíž rozdělením na více funkcí bych nic moc nezískal. Takto se aspoň dobře v kódu orientuje, když člověk ví, že např. parametry musí hledat na začátku, zpracování hlavičky uprostřed apod. Z toho také plyne způsob komentování zdrojového kódu, kdy je komentář uveden přímo u jednotlivých sporných či neúplně jasných konstrukcí.

Nyní blíže popíši některé důležité části mého řešení.

Zpracování parametrů

K zpracování parametrů příkazové řádky využívám knihovnu *GetOptions* umožňující jednodušší ošetření správného zadání parametrů. Kontroluje se zde například vícenásobné zadání některého z parametrů. Dále zde probíhá kontrola kombinace zadaných parametrů, či v případě zadání parametru *--help* rovnou také tisk nápovědy. Provádím zde také nastavení některých výchozích hodnot parametrům, které nebyly zadány. Stejně tak se zde provádí kontrola hodnot těch parametrů, které byly zadány. Především root a line elementu. Jako poslední činnost se zde provádí úprava parametru *-s* a jeho specializaci ze zadání využitím pseudo-hodnoty TAB, pro činnost skriptu změněné na \t.

Zpracování vstupního souboru

Následuje otevření výstupního souboru, případně nastavení výstupu na STDOUT.
Ke zpracování vstupního souboru využívám knihovnu *Text::CSV*. Hlavní tělo skriptu je while cyklus pomocí funkce *getline()* zmíněné knihovny. Toto řešení jsem zvolil především kvůli umožnění jednoduššího načtení celého vstupního řádku i v případě, že obsahuje znak nového řádku (\r\n) uvnitř uvozovek. K tisku výsledného souboru využívám knihovnu *XML::Writer*. Pro samostatný obsah elementů, ale využívám zvláštní metodu této knihovny-*raw*, která nijak neupravuje vypisované entity jako & apod. O toto ošetření se starám sám ručně.

Na začátku tohoto cyklu je kontrola na načítání prvního řádku souboru. V případě zadání parametru –h se generuje hlavička podle prvního řádku a pokračuje se další iterací cyklu (načtení dalšího řádku ze vstupu). V opačném případě se k číslování názvů sloupců využívá proměnná *pocet_sloupcu* a další cyklus. V tomto případě se dále pokračuje zpracováváním prvního řádku jako každého jiného.

Zpracování řádku vstupu je vyřešeno cyklem foreach přes všechny sloupce daného řádku. Kontroluje se zde také zotavování z chyb v případě špatného počtu sloupců, ale této části se budu ještě věnovat dále. Každý zpracovávaný sloupec se rovnou tiskne, je však obalen hodnotami z pole s názvy sloupců.

Zotavování z chyb

Tato část se skládá ze dvou částí a je implementována přímo v daném místě použití. První část je na začátku cyklu přes všechny sloupce a spouští se v případě, že se již vytisknuly všechny sloupce dané hlavičkou, avšak na vstupu řádku stále ještě něco je. V tomto případě se skript dělí několik a podmínkami podle zadaných parametrů. Buď se zde hlásí chyba a skript končí nebo se vypíš zbývající sloupce a pokračuje se další iterací řádku. V případě tisku i zbývajících sloupců se při každé průchodu tiskne pouze jeden sloupec a celý cyklus se volá znovu, avšak pomocí stejných podmínek se dostane do stejné části ošetření chyb.

Toto řešení jsem zvolil jako lepší, neboť by bylo možné zde přidat uvnitř další cyklus tisknoucí rovnou všechny sloupce, čímž by se ušetřil výpočet všech předchozích podmínek. Moje řešení ale obsahuje pouze jeden cyklus načítání ze vstupního souboru a je tedy přehlednější a lépe pochopitelné.

Další podobná část ošetření chyb se nachází po výstupu z cyklu načítajícího sloupce. Využívá proměnných *pocet_vypsanych* a *pocet_sloupcu_v_hlavicce*. Pokud se již z cyklu vyskočilo, ale nejsou vytisknuty všechny sloupce z hlavičky, tak se postupuje podle zadaných parametrů chybou nebo vypsáním zbývajících sloupců.

Rozšíření padding

Jelikož se toto rozšíření týká různých částí kódu, vytvořil jsem si na jeho řešení funkci *prepocet*, která dostává jako parametr zadané číslo, které se převádí, a také maximální hodnotu, které číslo v daném místě může nabývat. Na základě těchto dvou hodnot vrací řetězec, obsahují zleva správný počet nul, který je ale minimální možný.

Závěr

Jakožto uživatel převážně systému Windows, jsem projekt již od počátku testoval přímo na referenčním školním serveru merlin. Měl jsem tak zajištěnou plnou funkčnost po odevzdání a nemusel řešit problémy s nekompatibilitou mého řešení. K testování jsem ze začátku používal svoji vlastní sadu testů, po zveřejnění ukázkových testů jsem pouze upravil detaily skriptu, aby výstupy odpovídaly přesně požadovaným výstupům.