EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

Evidenčné číslo: 103004/I/2018/ 3781225405

Aplikácia hľadajúca požadované záznamy v inštalačnom logovacom súbore operačného systému vytvorená v jazyku C++

Diplomová práca

2017 **Bc. Radoslav Rajčan**

EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

Aplikácia hľadajúca požadované záznamy v inštalačnom logovacom súbore operačného systému vytvorená v jazyku C++

Diplomová práca

Študijný program: Informačný manažment
Študijný odbor: 6258 Kvantitatívne metódy v ekonómii
Školiace pracovisko: Katedra aplikovanej informatiky
Vedúci záverečnej práce: Ing. Igor Košťál, PhD.

2017 **Bc. Radoslav Rajčan**

Čestné vyhlásenie

Čestne vyhlasujem, že záverečnú prácu som vypracoval samostatne a že som uviedol všetku použitú literatúru.

Dátum: **.....**

(podpis študenta)

Poďakovanie:
Na tomto mieste by som chcel poďakovať Ing. Igor Košťál, PhD. za čas, ktorý mi venoval pri vypracovávaní diplmovej práce a jeho odborné rady.

Abstrakt

Rajčan, Radoslav: Aplikácia hľadajúca požadované záznamy v inštalačnom logovacom súbore operačného systému vytvorená v jazyku C++. – Ekonomická Univerzita v Bratislave, Fakulta Hospodárskej Informatiky; Katedra Aplikovanej Informatiky - Vedúci záverečnej práce: Ing. Igor Košťál, PhD. – Bratislava: FHI, 2018, 54 strán.

Cieľom záverečnej práce je zanalizovanie vyhľadávacích algoritmov so zameraním sa na ich princíp, význam a použitie a vytvorenie programu v jazyku C++ s implementovaným vybratým vyhľadávacím algoritmom, ktorý podľa požiadaviek používateľa prehľadá inštalačné logovacíe súbory operačného systému a výsledky vyhľadávania zapíše do výstupného diskového súboru.Práca je rozdelená do piatich kapitol. Obsahuje 5 diagramov a 1 prílohu. Prvá kapitola je venovaná súčasnému stavu problematiky na Slovensku a v zahraničí. Taktiež sa v nej venujeme Použitým naštrojom a algoritmom. Záverečná kapitola sa zaoberá štruktúrou a fungovaním programu. Priblížili sme si v nej dôležité časti programu, používatelške prostredie, príklady využitia a ukázali sme ako bude vyzerať textový výstup. Výsledkom našej práce je program, ktorý prehľadáva inštalačné logovacie súbory operačného systému Windows vo formáte XML. Program výsledok zobrazí do tabuľky a zapíše do výstupného textového súboru.

Kľúčové slová: algoritmus, XML, C++, textový výstup, program, lineárne vyhľadávanie.

Abstract

Rajčan, Radoslav: A C++ application seeking the requested records in the setup log file of the operating system. – University of Economics in Bratislava. Faculty of Economic Informatics; Department of applicated informatics - Supervisor: Ing. Igor Košťál, PhD. – Bratislava: FHI, 2018, 54 pages.

In the thesis, we have analyzed search algorithms with a focus on their principles, their meaning and use. As a part of the thesis, the C++ program with an embedded chosen search algorithm will be created by student. This program according to the requirements of the user scans the setup log file of the operating system and writes search results to the output disk file. This work is divided into 5 chapters. It contains 5 diagrams and 1 attachment. The first chapter is dedicated to current state of this problematics in Slovakia and abroad. We are also analyzing used tools and algorithms. The final chapter is dedicated to the structure and processing of the program. We are also analyzing important parts of our program, user interface, exmaples of use and we have also shown the output disk file. The result of our work is a program which searches requested records in the setup log file of the operating system in XML format. The program will preview the results in a table and it will also record it in the output disk file.

Keywords: algorithm, XML, C++, text file, program, linear search

Obsah

Zoznam obrázkov:

Úvod

Každému používateľovi systému Windows sa už asi stalo, že pri jeho práci došlo k nečakaným chybám. Tieto chyby môžu byť spôsobené problémami na strane systému, alebo používaných programov. Systém Windows ponúka rôzne spôsoby ako používateľom s týmito problémami pomôcť. Na tento účel, ale aj pre mnohe iné dôvody, existujú inštalačne súbory Windows. Každá inštlácia je v nich zapísaná ako samostatná udalosť, ktorá obsahuje údaje o inštalovanom balíku. Taktiež sa v nich dáju nájsť informácie o čase inštalácie, zdroji inštalacie a mnohé ďaľšie užitocné infromácie.

Cieľom tejto práce je pripraviť program, ktorý pomôže uživateľovi prehladávať inštalačné súbory Windows zapísane vo výstupnom súbore v jazyku XML. Toto má mnoho výhod. Export inštalačných údajov vo formáte XML patrí medzi prirodzené súčasti systému Window čo umožní prehľadávať súbory z rôznych zdrojov.

Program uživateľovi poskytne jednoduchý prehľad, ktorý bude obsahovať také údaje, ktoré umožnia jednoduché a rýchle rozhodovanie o ďaľšom postupe. Tento postup môže byť napríklad nová inštlácia balíkov, ktoré sa nenainštalovali správne, alebo sa inštalovali počas obdobia keď sa začali vyskytovať chyby, Taktiež sa budeme sústrediť na to aby bol náš program čo najmenej náročný na pamäť a výpočtové zdroje.

Pre uživateľa vytvoríme také spôsoby vyhľadávania aby vedel jednoducho a rýchlo nájsť potrebné údaje. Za týmto účelom vytvoríme predpovolené vyhľadávanie, ktoré sa bude dať spustiť pomocou jednoduchej roztváracej ponuky. Taktiež poskytneme možnosť vyhľadávať pomocou textového reťazca a dátumu. Možnosť vyhľadávania podľa dátumu nastavíme tak aby sa dalo vyhľadávať udalosti s rovnakým dátumom, ale aj všetky udalosti, ktoré majú skorší alebo neskorší dátúm ako bude zadaný. Vďaka týmto možnostiam chceme dosiahnuť vysokú flexibilitu programu. Výstup zapíšeme okrem vizuálneho rozhrania programu aj do výstupného textového súboru.

V prvej časti našej práce sa budeme venovať súčasnému stavu problematiky. Priblížime si aj nástroje, ktoré budeme používať. Taktiež sa budeme venovať algoritmom, keďže správny výber vyhľadávacieho algoritmu je veľmi dôležitý.

Následne si zhrnieme ciele, ktoré chceme v našej práci dosiahnuť . Taktiež si priblížime metódy, ktoré sme použili pri písaní našej práce.

Vo výsledkoch práce si priblížime štruktúru nášho programu. Do podrobna popíšeme dôležité časti a funkcie, ktoré naprogramujeme. Potom si popíšme aké možnosti náš program ponúka v používateľskom prostredí. Na konci výsledkov budú praktické príklady využitia programu a ukážka textového výstupu.

V závere práce zhrnieme výsledky a vyhodnotíme, do akej miery sme dokázali splniť naše ciele.

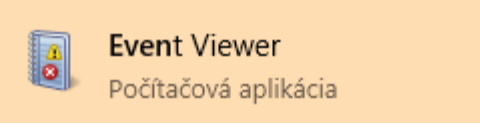
1. Súčasný stav riešenej problematiky doma a v zahraničí

Prvým krokom pri skúmani a tvorbe programu na spracovanie inštalačných súborov je analýza. Potrebujeme preskúmať, ako sú uložene a ako sa k ním dá pristupovať. Cieľom tejto časti je nájsť najlepši spôsob a techniku tvorby nášho programu. Začneme skúmanim už existujúcich riešení. Ďaľej sa budeme zaoberať Jazykom XML, v ktorom je napísaný náš vstupný súbor a nakoniec sa pozrieme na vyhľadávacie agoritmy.

Základný prostriedok na spracovanie inštalačných súborov Windows je Event Viewer. Je ľahko prístupný, kedže je súčasťou systému Windows. Poskytuje základné funkcie na triedenie a zobrazenie. Event Viewer bude bližšie popísaný v nasjledujúcej časti tejto práce. XML výstup z Event Viewera bude našim vstupým súborom. Výhodou nášho programu bude jeho schopnosť spracúvať dáta z rôznych počítačov bez toho, aby sme k nim mali priamy prístup.

Existuje mnoho programov, ktoré sa venujú triedeniu logovacích súborov Windows. Tieto niesu primárnym cieľom našej práce.

1.1 Event Viewer



Obrázok 1 – Event Viewer logo

Event Viewer (Prehliadač udalostí systému Windows) zobrazuje protokoly aplikácií a systémových správ, chyby, informačné správy a upozornenia. Ak chcete spustiť prehliadač udalostí, otvorte ponuku štart a napíšte “Event Viewer“. Po stlačení klávesy Enter so otvorý. Zobrazovač udalostí sa dá spustiť aj z priečinka Nástroje pre správu.

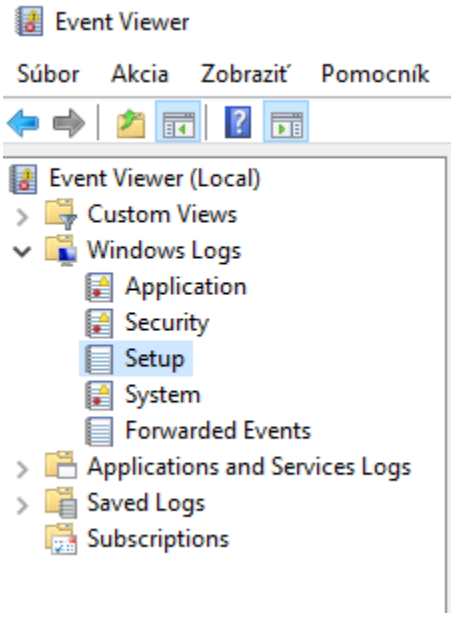
Pri prezeraní často narazíte na mnohé chybové správy, tých sa však netreba báť. Vo všeobecnosti môžete ignorovať všetky chyby a upozornenia, ktoré sa zobrazujú v prehliadači udalostí, za predpokladu, že váš počítač funguje správne.

Udalosti (Events) sú umiestnené v rôznych kategóriách. Napríklad časť Application obsahuje záznamy o udalostiach aplikácií a časť System obsahuje záznamy systémových udalostí systému Windows.

Event Viewer môže byť skutočne užitočný v prípade, že máte problém s počítačom. Napríklad, ak počítač často ukazuje bluescreen, alebo sa náhodne reštartuje. Event Viewer môže poskytnúť informácie o príčine. Napríklad chybová udalosť v sekcií System vás môže informovať o tom, ktorý hardvérový ovládač havaroval, čo vám môže pomôcť nájsť chybový ovládač alebo chybnú hardvérovú súčasť.

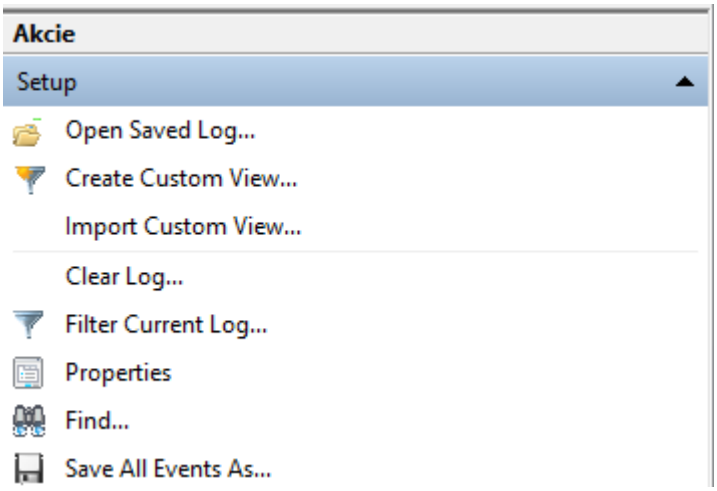
Event Viewer má aj iné skvelé využitia. Systém Windows napríklad zaznamenáva čas spustenia počítača a zapíše ho ako udalosť, takže môžete použiť Event Viewer napríklad na zistenie presného trvania štartu počítača.

1.1.1 Event Viewer – Windows Logs



Obrázok 2 – Event Viewer – Setup

Windows Logs - Setup vytvára záznamy pre všetky akcie, ktoré sa uskutočňujú počas inštalácie. Ak máte problémy s inštaláciou alebo s aktualizáciami systému Windows, záznamy Setup obsahujú informácie na hľadanie týchto problémov.

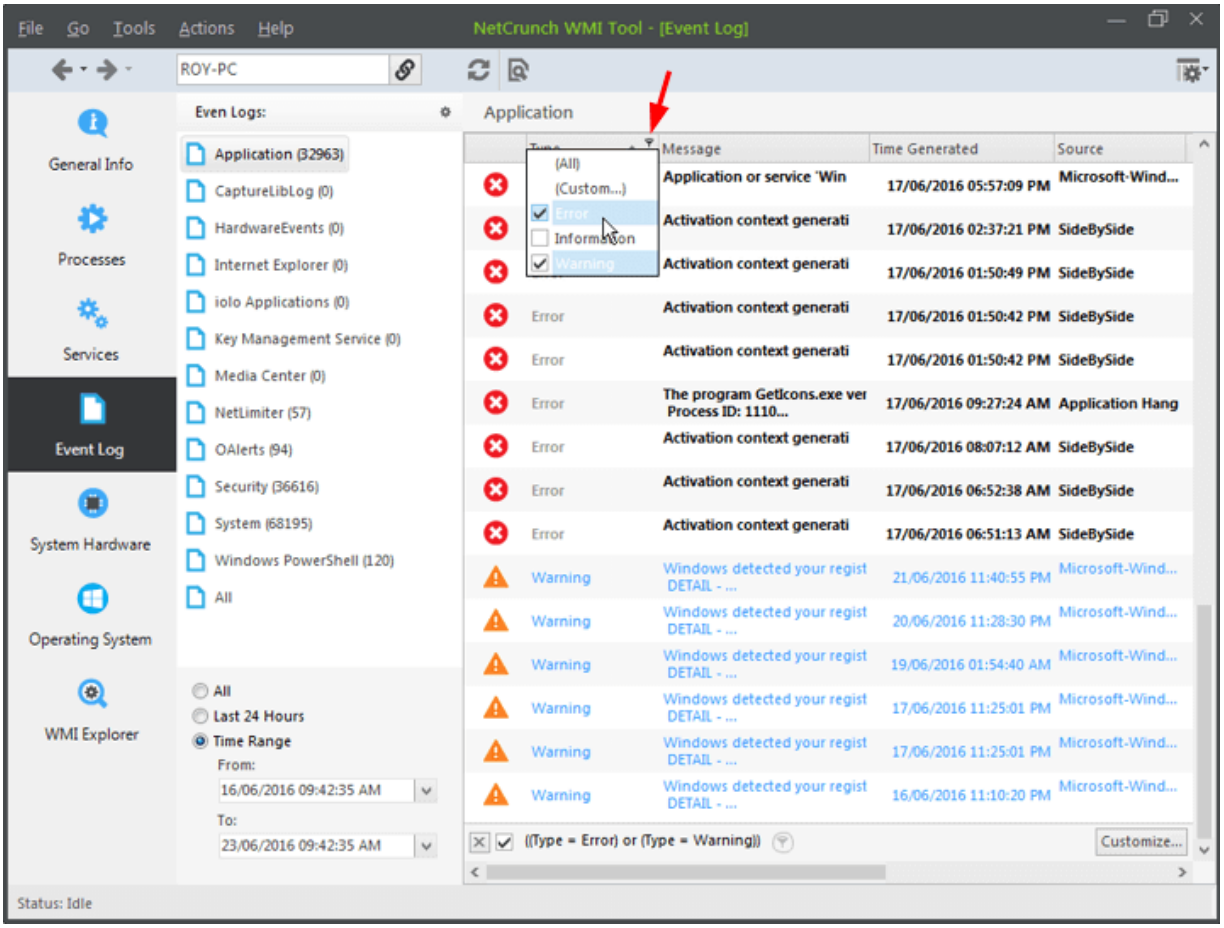


Obrázok 3 – Event Viewer – Setup – Akcie

Export z Event Viewera sa dá uložiť pomocou funckie „Save All Events As“. Na uloženie Windows ponúka 4 typy súborov v rátane .xml súboru. Pri ukladaní sa dá vybrať aj jazyk v ktorom budú systémové hlasenia vypísané.

1.2 WMI Tools

WMI Tools je všeobecný nástroj na prezeranie a vyhodnocovanie informácií z Windows Manažment Inštrumentov (WMI). Tieto informácie sa týkaju hardwaru, softwaru a procesov Windows. Možnost prezerania Windows Event logov je tým pádom len jedna z mnohých funkcií v ponuke. Pre použitie programu je nutná inštalácia, co nieje zvyok pri programoch tochto typu. Po Spustení programu sa dá jedným kliknutím vstúpit do časti, ktorá sa venuje prehliadaniu udalostí. Udalosti sú automaticky nahrané a zobrazené.



Obrázok 4 – VMI Tools úvodná obrazovka
(https://img.raymond.cc/blog/wp-content/uploads/2012/04/windows_event_viewer.png)

V ponuke je aj možnosť jednoduchého filtrovania, prípadne zmena časového obdobia za ktoré sú udalosti vypísané.

1.3 XML

V tejto časti sa pozrieme na jazyk XML (Extensible Markup Language). Jazyk XML bol vytvorený ako zjednocujúci faktor s cieľom, aby moderné aplikácie boli šikovnejšie, všestrannejšie a výkonnejšie. Ked sa spolu rozprávajú ľudia, mnohé informácie dokážeme pochopiť z reči tela alebo tónu hlasu. Počítač však toto nedokáže. XML počítaču umožnuje napodobniť tieto naše výhody. Toto je možné vďaka vlastnostiam jazyka XML.

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
- <Events>
  - <Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
    - <System>
        <Provider Guid="{BD12F3B8-FC40-4A61-A307-B7A013A069C1}" Name="Microsoft-Windows-Servicing"/>
        <EventID>1</EventID>
        <Version>0</Version>
        <Level>0</Level>
        <Task>1</Task>
        <Opcode>0</Opcode>
        <Keywords>0x8000000000000000</Keywords>
        <TimeCreated SystemTime="2017-08-10T06:30:36.597233700Z"/>
        <EventRecordID>45</EventRecordID>
        <Correlation/>
        <Execution ThreadID="5844" ProcessID="4840"/>
        <Channel>Setup</Channel>
        <Computer>Uzivatel-PC</Computer>
        <Security UserID="S-1-5-18"/>
      </System>
```

Obrázok 5 – XML – Príklad

1.3.1 XML – Vlastnosti

a) rozšíriteľnosť

XML je rozšíriteľný. Umožňuje definovať vlastné značky, poradie, v ktorom sa vyskytujú, a ako majú byť spracované alebo zobrazené. Ďalším spôsobom, ako uvažovať o rozšíriteľnosti, je zvážiť, že XML nám umožňuje rozšíriť našu predstavu o tom, čo je dokument. Môže to byť súbor, ktorý žije na serveri, alebo to môže byť prechodná časť dát, ktorá preteká medzi dvoma Počítačovými systémami. V oboch prípadoch je možné dáta ukladať pomocou jazyka XML.

b) značenie

Základnou črtou XML sú jej značky alebo prvky. Údaje v XML sú auto-popisujúce alebo samo-definujúce, čo znamená, že štruktúra dát je vložená s údajmi. Pri príchode dát nie je potrebné predbežne stavať štruktúru na ich ukladanie. Štruktúra je dynamicky vysvetlená priamo v XML. Toto umožnuje zdielať informácie konzistentným spôsobom. Prvky (uzly), ktoré sa používajú v XML sú veľmi podobné elementom, ktoré sa už dlhú dobu používajú v jazyku HTML. XML však umožňuje definovať vlastnú sadu značiek.

c) meta-jazyk

Je dôležité uvedomiť si, že XML nie je len jazykom. XML je meta-jazyk: jazyk, ktorý nám umožňuje vytvárať alebo definovať iné jazyky. Napríklad s XML môžeme vytvoriť ďalšie jazyky ako je RSS, MathML (matematický značkovací jazyk) a dokonca aj nástroje ako XSLT.

1.3.2 XML – Štruktúra

<PackageIdentifier>**KB4025342**</PackageIdentifier>

Obrázok 6 – XML – Riadok

Základné prvky XML obsahujú úvodnú značku, záverečnú značku a ľubovolný obsah. Úvodná značka začína ľavým uhlom (<), nasleduje názov prvku, ktorý obsahuje písmená a čísla (bez medzery) a končí s pravým uhlom (>). Obsah XML zvyčajne nemá jednotný charakter dát. Môže pozostávať z obyčajného textu, iných prvkov XML alebo ďalších exotických vecí, ako sú entity XML, komentáre a pokyny na spracovanie. Na konci je záverečná značka, ktorá má rovnaký pravopis a veľké písmená ako úvodná značka, ale s jednou malou zmenou: pred názvom prvku je lomítko.

Celý súbor je štrukturovaný ako strom. Na obrázku 6 to vyzerá nasledovne:

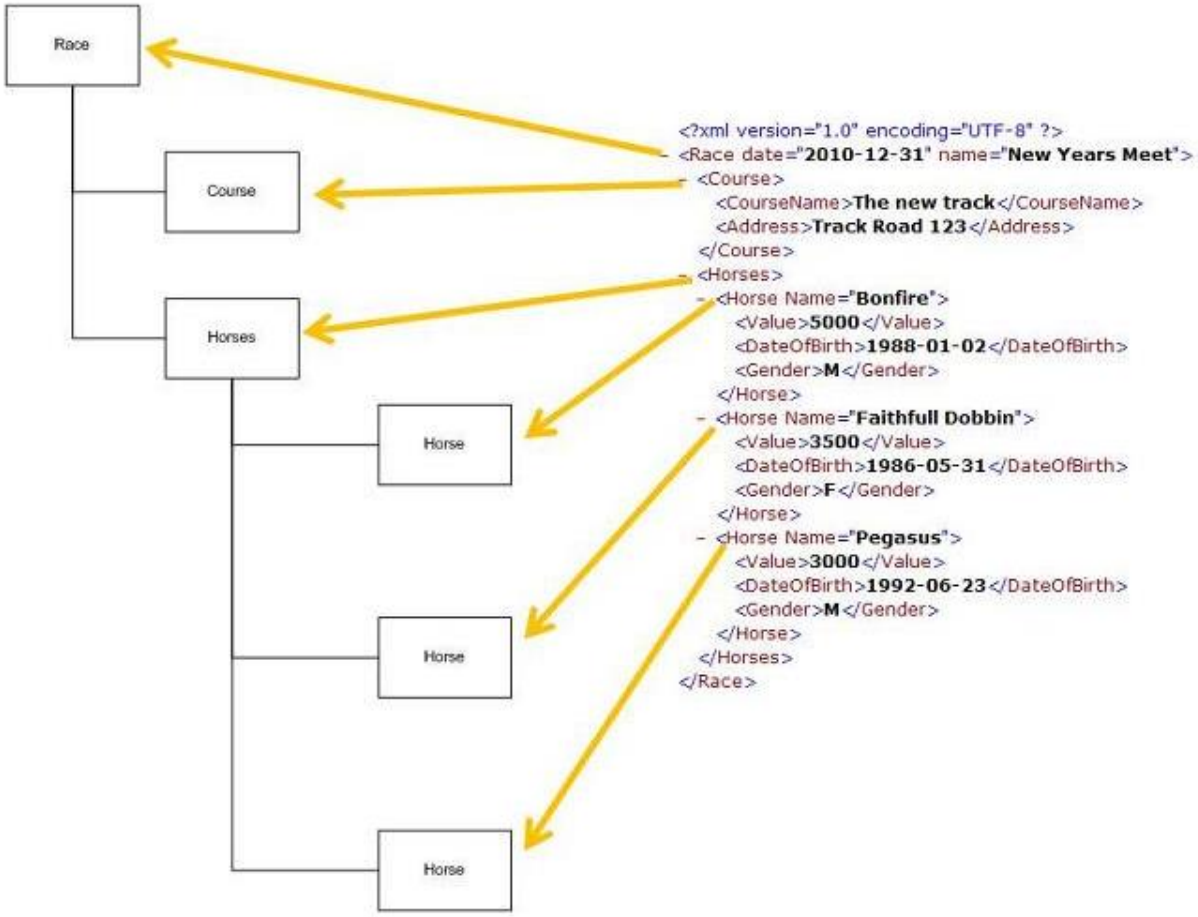
1) Prvok "Race" je horným prvkom dokumentu, obsahuje dva atribúty "date" a "name". Taktiež pod neho patria aj dva komplexné podradené prvky "Course " a " Horses ".

2) Komplexný prvok "Course" obsahuje dva jednoduché prvky "CourseName" a "Address".

3) Komplexný prvok "Horses" obsahuje množstvo zložitých prvkov "Horse".

4) Každý komplexný prvok "Horse" obsahuje atribút "Name" a tri jednoduché prvky "Value", "DateOfBirth" a "Gender".

5)Štruktúru stromu dokumentu určujú zložité prvky a ich vzájomné prepojenie.



Obrázok 7 – XML – Strom
(https://www.websydian.com/olddoc/v61/source/Websydian%20v6.1/TransacXML/wsyxml/xmlstructure.htm)

1.4 Vyhľadávacie algoritmy

Pri práci s počítačom sa často stretávame s väčším objemom dát. To podnietilo vznik vyhľadávacích algoritmov. Ich veľké množstvo je spôsobené rôznou povahou dát. Voľba vhodného algoritmu je kľúčová pre urýchlenie prehľadávania, kedže vhodný algoritmus môže mnohonásobne zmenšiť výpočtovú naročnosť. Pre náš program je možné zvoliť viacero vhodých vyhľadávacích algoritmov, tie najzákladnejšie si teraz priblížime.

1.4.1 Lineárne vyhľadávanie

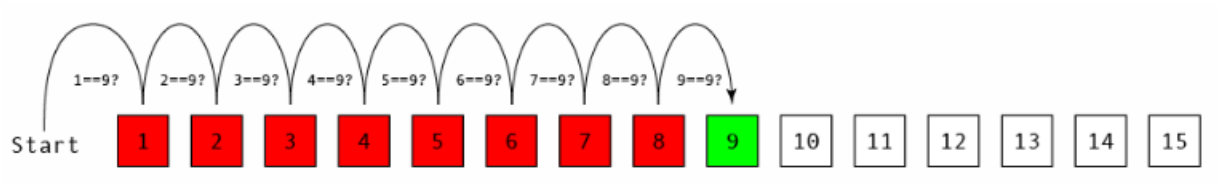
Je vyhľadávací algoritmus, ktorý sa najčastejšie používa pri nezoradených dátach s cieľom vyhľadania určitej hodnoty. Algoritmus postupne prechádza každý prvok zhora nadol, až kým nenájde požadovanú hodnotu alebo do konca súboru. Zložitosť lineárneho vyhľadávania je $O^{(n)}$. Program potrebuje tým väčšiu náročnosť, čím je hľadaný prvok bližšie ku koncu. Ak sa hľadaný prvok nachádza na začiatku, tak stači iba jedno porovnanie.
Medzi hlavné výhody patrí univerzálnosť, úplnosť, jednoduchosť a prehľadnosť.

Univerzálnosť lineárneho vyhľadávania spočíva v tom, ze sa dá použit na akékoľvek dáta (utriedené aj neutriedené). Úplnosť znamená, že lineárne vyhľadávanie vždy dosiahne výsledok.

Nevýhodou lineárneho vyhľadávania je jeho výpočtová náročnosť, ktorá rastie lineárne s veľkosťou prehľadávaného súboru.

Príklad:

```
int linearne(int arr[], int n, int x)
{
    int i;
    for (i = 0; i < n; i++)
        if (arr[i] == x)
         return i;
    return -1;
}
```



Obrázok 8 – Lineárne vyhľadávanie
(http://peterstratton.com/img/LinearSearch/LinearSearch.png)

1.4.2 Binárne vyhľadávanie

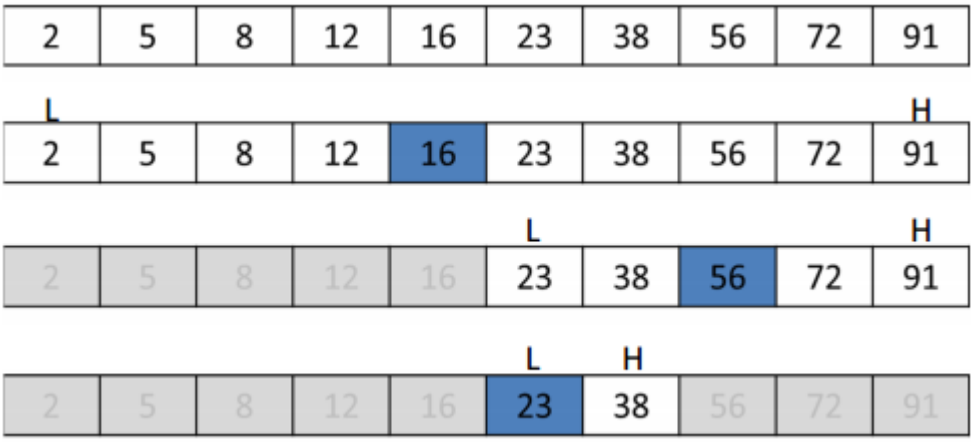
Je takisto ako lineárne vyhľadávanie jedným zo základných tipov vyhľadávacích algoritmov. Používa sa len pri zoradených dátach, ak dáta niesu alebo nemožu byť zoradené, tak binárne vyhľadávanie nemusí dôjsť ku správnemu výsledku. Algoritmus najprv rozdelí súbor na dve polovice. Potom sa pozrie či sa hľadaný prvok nachádza v strede. Ak nedôjde ku zhode tak tieto hodnoty číselne alebo inak porovná a na základe tejto informácie prehľadáva buď vrchnú alebo spodnú polovicu. Takto pokračuje až kým sa nedostane k požadovanému výsledku. Zložitosť binárneho vyhľadávania je $O^{(\log n)}$.

Výhoda binárneho vyhľadávania spočíva v jeho nižšej výpočtovej zložitosti oproti lineárnemu vyhľadavaniu. Pokiaľ sú naše dáta správne utriedené a hľadaná hodnota sa v ňom nachádza, tak binárne vyhľadávanie vždy dôjde k výsledku.

Nevýhodou binárneho vyhľadávania je potreba usporiadať dáta a jeho zložitosť, ktorá je stále vysoká oproti iným špecializovaným algoritmom.

Príklad:

```
int binarne(int arr[], int l, int r, int x)
{
   if (r >= l)
   {
        int mid = l + (r - l)/2;
         if (arr[mid] == x)
            return mid;
         if (arr[mid] > x)
            return binarne(arr, l, mid-1, x);
         return binarne(arr, mid+1, r, x);
   }
    return -1;
}
```



Obrázok 9 – Binárne vyhľadávanie
(https://www.geeksforgeeks.org/wp-content/uploads/gq/2014/01/binary-search1.png)

1.5 Spôsoby implementácie vyhľadávacích algoritmov

Kedže pri našej práci budeme potrebovať nájsť viacero výsledkov, tak sa musíme zamyslieť nad spôsobom opakovania nášho algoritmu. Toto môžme dosiahnuť za pomoci rekurzie alebo za pomoci iterácie. Tieto pricípy si teraz zanalyzujeme.

1.5.1 Rekurzia

Je spôsob opakovania funkcie alebo procedúry, kde vo vnútry je podmienka, ktorá môže znova zavolať samotnú funkciu. Toto volanie môže byť priame, volanie seba samého, alebo nepriame, kde funkcia volá funkciu dva a tá opakovane zavolá pôvodnú funkciu. Toto sa bude opakovať až kým nedôjdeme k výsledku alebo na koniec súboru. Rekurzia ponúka elegantné riešenia, ale predstavuje aj mnohé riziká. Rekurzívne funkcie majú tendenciu rýchlejšie vyčerpávať prostriedky procesora aj pamäte.

Príklad:

```
void rekurzia(int i) {
  cout << "cislo je: " << i << endl;
}

int main() {
```

```
for(int i=5; i<15; i++) {
  rekurzia(i);
}

return 0;
}
```

1.5.2 Iterácia

Je spôsob opakovania, kde si na začiatku zistíme koľko krát sa bude naša funkcia alebo procedúra opakovať a potom spravíme cyklus, ktorý sa presne toľko krát zopakuje. Narozdiel od rekurzie pri iterácií dopredu vieme koľko prostriedkov budeme potrebovať a máme zaručené, že funkcia sa po n opakovaniach zastaví. Pokiaľ sa k výsledku dostaneme pred posledným cyklom, opakovanie je možné jednoducho zastaviť. Iterácia sa nedá použiť ak dopredu neviem zistiť koľko opakovaní budeme potrebovať.

Príklad:

```
int main () {
   for( int a = 5; a < 15; a = a + 1 ) {
      cout << "hodnota je: " << a << endl;
   }

   return 0;
}
```

2. Cieľ práce

Cieľom diplomovej práce je navhrnúť a implementovať program v riadenom jazyku C++, ktorý umožní prehľadávať, triediť a ukladať vyhľadávanie v inštalačných súboroch Windows.

Po vykonaní analýzy existujúcich riešení môžeme zistené poznatky aplikovať pri návrhu nášho programu. Budeme sa sústrediť na funkcie, ktoré umožnia jednoduché a efektívne vyhľadávanie podľa texového reťazca a dátumu. Taktiež sa chceme vyhnúť nedostatkom v analyzovaných systémoch.

Základný cieľ tejto práce je vytvorenie programu, ktorý dokáže jednoducho analyzovať poskytnuté údaje. Na túto analýzu vyžije výhody, ktoré poskytuje obratnosť jazyk C++ a usporiadanosť jazyka XML. Náš program uživateľovi dovolí vybrať si na disku súbor, ktorý bude obsahovať export inštalačných súborov v jazyku XML. Program tento súbor načíta a vypíše základné informácie o jeho obsahu. Poskytnuté informácie by mali uživateľovi poskytnúť dostatočný prehľad na to aby vedel rozhodnuť o ďaľšom postupe pri oprave prípadných chýb, alebo manuálnej inštalácií.

Medzi najdôležitejšie informácie určite patrí dátum inštalácie, poskytovateľ a žiadateľ inštalácie, identifikačné číslo inštalačného balíka a stav inštalácie.

Vďaka týmto informáciam by mal uživateľ vedieť intuitívne určiť, ktoré inštalačné súbory sú dôležité. Program uživateľoví daľej poskytne možnosť triediť inštalačne súbory podľa predvolených návrhov. Každé vyhľadávanie o ktoré uživateľ požiada bude automaticky uložene do textového súboru v priečinku, kde sa nachádza vstupný XML súbor. Program daľej zaručí, že duplicitné vyhľadánie nebude uložené viacnásobne. Uložené vyhľadávanie bude v názve súboru obsahovať informácie potrebné na jeho jednoduché rozoznanie. Vo vnútry súboru bude časová pečiatka.

3. Metodika práce a metódy skúmania

Na začiatok dôkladne preskúmame záznamy inštalačných logovacích súborov operačného systému Windows. Preskúmame ako vyzerá export vo formáte XML a určíme, ktoré atribúty sú dôležité pre uživateľov.

Pri návrhu programu si ujasníme jeho koncepciu. Naverhneme vzťahy medzi funkciami programu tak, aby sme sa vedeli dostať k správnemu výstupu v najkratšom možnom čase. Taktiež musíme brať ohľad na pamäťové nároky aby náš program zbytočne nečerpal systémove zdroje. Pri tvorení návrhu si nakreslíme data flow diagramy pre funkcie, ktoré budú zodpovedné za dôležité funkcie programu.

Po vyjasnení vzťahov môžeme určiť algoritmus, ktorým budeme inštalačné logovacie súbory prehľadávať. Rozhodli sme sa použiť lineárne vyhľadávanie s opakovaním cez iterácie, kedže správnych výsledkov môže byť viac, čo nás núti prechádzať všetky komplexné prvky a atribúty vstupujúceho súboru XML.

Navrhneme konkrétnu štruktúru údajov zobrazovaných počas behu programu, ako aj štruktúru údajov, ktoré budú zapísane do výstupného diskového súboru. Keď budeme mať pevne určenú štruktúru všetkých potrebných súčastí, môžeme navrhnúť ich asociácie, ktoré budeme reprezentovať v kóde programu.

4. Výsledky práce

V našej práci sme vytvorili program, ktorý načíta inštalačné logy systému Windows zo súboru vo formáte XML. Program komunikuje s používateľom cez uživateľské prostedie navrhnuté za pomoci šablóny Windows Forms Application. K dispozícií má predvolené možnosti, ktoré mu pomožu pri špecifikácií požiadaviek prehľadávania. Program vypíše vyhovujúce udalosti do výstupnej tabuľky a následne aj vytvorí textový súbor s plnými výsledkami vyhľadávania. V nasledujúcich podkapitolách si priblížime architektúru, vybrané časti kódu, príklady použitia a výsledky vyhľadávania.

4.1 Analýza vstupného súboru s udalostami vo forme XML

Pred tým ako začneme s popisom programu si priblížime ako vyzerá náš vstupný súbor.

```
- <Events>
  - <Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
    - <System>
        <Provider Guid="{BD12F3B8-FC40-4A61-A307-B7A013A069C1}" Name="Microsoft-Wi
        <EventID>1</EventID>
        <Version>0</Version>
        <Level>0</Level>
        <Task>1</Task>
        <Opcode>0</Opcode>
        <Keywords>0x8000000000000000</Keywords>
        <TimeCreated SystemTime="2017-08-10T06:30:36.597233700Z"/>
        <EventRecordID>45</EventRecordID>
        <Correlation/>
        <Execution ThreadID="5844" ProcessID="4840"/>
        <Channel>Setup</Channel>
        <Computer>Uzivatel-PC</Computer>
        <Security UserID="S-1-5-18"/>
      </System>
    - <UserData>
      - <CbsPackageInitiateChanges xmlns="http://manifests.microsoft.com/win/2004/08/w
          <PackageIdentifier>KB4025342</PackageIdentifier>
          <InitialPackageState>Superseded</InitialPackageState>
          <IntendedPackageState>Absent</IntendedPackageState>
          <Client>DISM Package Manager Provider</Client>
        </CbsPackageInitiateChanges>
      </UserData>
    - <RenderingInfo Culture="sk-SK">
        <Message>Inicializujú sa zmeny pre balík KB4025342. Aktuálny stav je Nahradenie.
        <Level>Information</Level>
        <Task/>
        <Opcode>Info</Opcode>
        <Channel/>
        <Provider>Microsoft-Windows-Servicing</Provider>
        <Keywords/>
      </RenderingInfo>
  </Event>
```

Obrázok 10 – Jedna udalosť vstupného súboru

Na obrázku vidíme jednu udalosť vybranú z inštalačného súboru vo formáte XML. Môžeme si všimnúť, že každá udalosť má tri komplexné prvky: System, UserData a RenderingInfo. Každý s týchto komplexných prvkov má väčšie množstvo atribútov. Dôležité je tiež to, že komplexný prvok UserData má v sebe podprvok CbsPackageInitiateChanges. Toto nám bude komplikovať spracovanie.

Komplexný prvok System obsahuje údaje o systéme kde bol balík inštalovaný, taktiež obsahuje čas inštalácie, ktorý neskôr využijeme pri vyhľadávaní. Komplexný prvok UserData obsahuje datailné informácie o inštalovanom balíku. Komeplxný prvok RenderingInfo obsahuje údaje o zdroji inštalovaného balíka.

Po dôkladnej analýze sme sa rozhodli, že do našej výstupnej tabuľky vypíšme nasledové atribúty: EventID, Provider, Client, TimeCreated (ktorý pred vypísaním upravíme tak aby obsahoval len dátum inštalácie), PackageIdentifier, PC Name a Initial State.

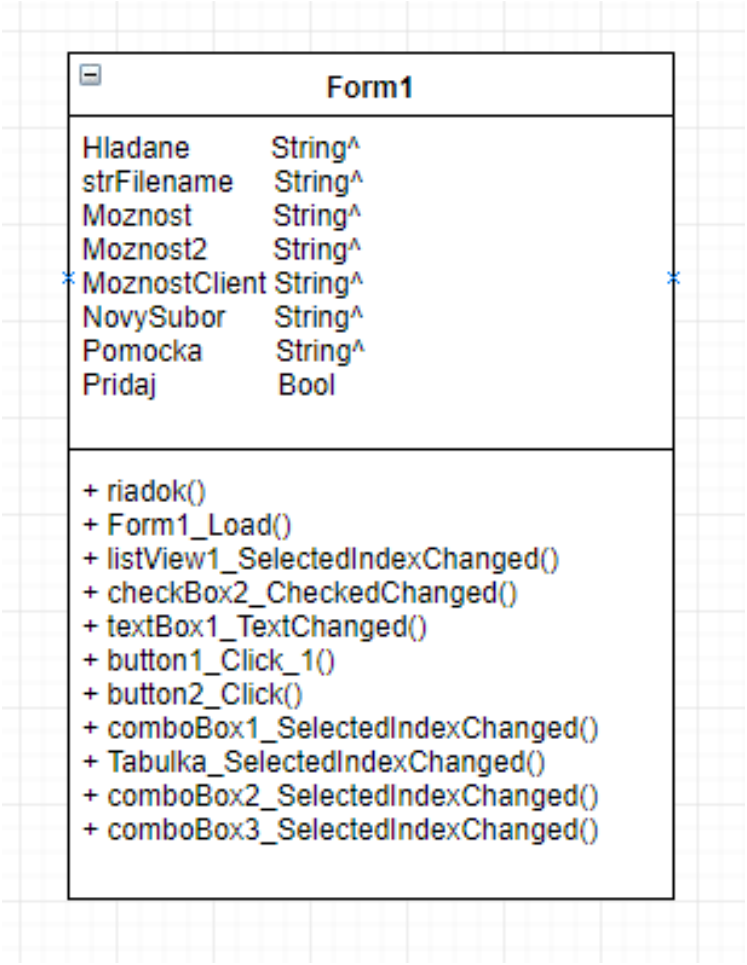
4.2 Výber algoritmu

Pre náš program sa najviac hodí lineárne vyľadávanie. Výstupné súbory programu Windows síce sú usporiadané, ale toto usporiadanie nieje vhodné pre binárny algoritmus. Usporiadať tento súbor dodatočne by bolo neúsporné z výpočtových aj pamäťových dôvodov.

Ďalším dôležitým aspektom nášho algoritmu je spôsob opakovania. Súbory, ktoré budeme prehľadávať budú mať premenlivú dĺžku a hľadaný text sa v ňom môže nachádzať mnoho krát. Taktiež sa tam nemusí nachádzať vôbec. Z týchto dôvodov sa ako najvýhodnejšia javí rekurzia. Avšak, pri spustení programu potrebujeme vždy vypísať dôležité informácie o každej inštalácií. Z tohto dôvodu budeme musieť vždy prejsť celý súbor, čo nám umožní spočitať množstvo záznamov. Tým pádom sa môžeme vyhnúť rekurzií pri opakovani celej funkcie a vybrať si na to lineárny algoritmus. Pri prehľadávaní komplexných prvkov však nemusí vždy byť rovnaký počet atribútov. Z tohto dôvodu budeme na prehľadávanie komplexných prvkov používať rekurziu.

4.3 Diagram tried

Náš program ja založený na jednej hlavnej triede From1, ktorú automaticky generovala šablóna Windows Forms Application. Zvyšné triedy sa používajú na uloženie dát zo vstupného súboru.



Obrázok 11 – Hlavná trieda Form1

4.4 Globálne premenné použité v našom programe

V našom programe sme použili 8 globálnych premenných. Sedem z nich je používa dátový typ String^ a jedna je typu Bool. Teraz si každú z nich problížime.

Globálna premenná Hladane bola vytvorená pre textový input zadaný uživateľom. Táto premenná musela byť založená na globálnej úrovni lebu je využivaná vo viacerých funkciách programu. Napríklad pri posune hľadaného reťazca do funckie riadok, ktorá je zodpovedná za tvorbu textového výstupu.

Globálna premenná strFilename bude obsahovať názov a cestu k vstupnému súboru, ktorý zadá uživateľ. Táto premenna musí byť globálna lebo ju budú požívať všetky funkcie, ktoré prehľadávajú vstupný súbor.

Globálne premenné Moznost, Moznost2 a MoznostClient budú obsahovať vstup z roztváracích ponúk nášho programu. Premenná Moznost bude obsahovať informáciu o zvolenom prednastavenom vyhľadávaní. Premenná Moznost bude obsahovať informáciu o zvolenom spôsobe vyhľadávania podľa dátumu. MoznoClient bude obsahovať informáciu o filtrovaní pomocou atribútu Client.

Globálna premenná NovySubor je potrebná pre zaslaniu koretného názvu výstupného súboru z funkcií, ktoré vyhľadávajú v inštalačných suboroch do funkcie riadok, ktorá následne vytvorí textový výstup.

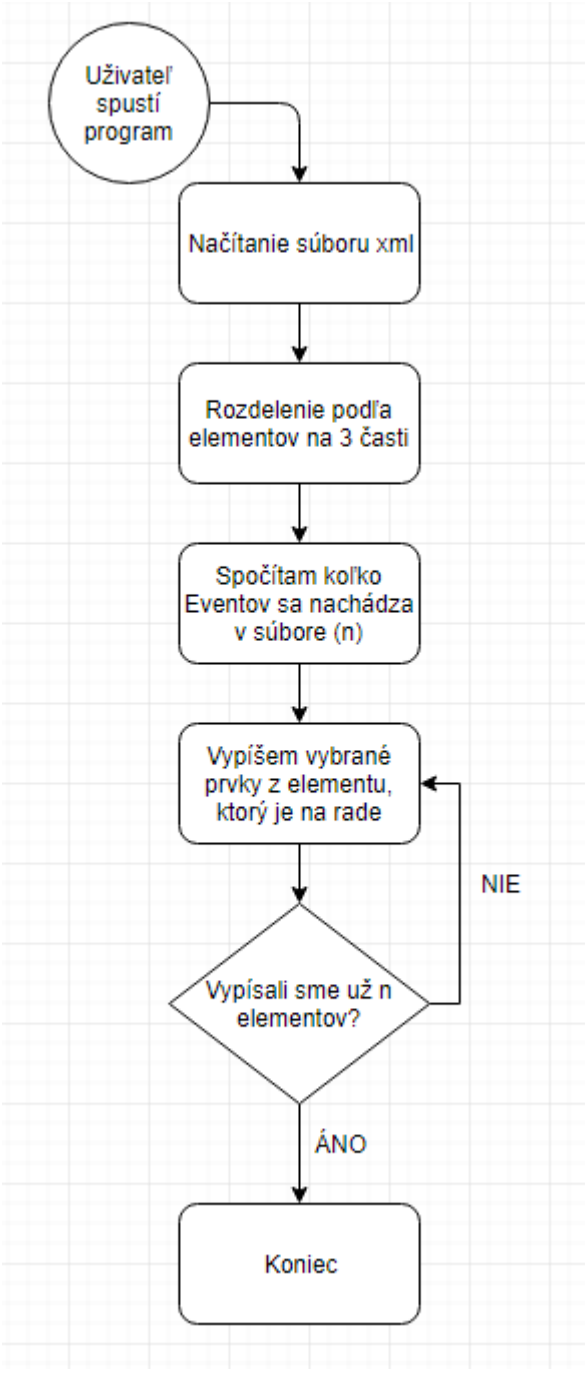
Globálna premenná Pomocka sa používa pri delení dátumu na rok, mesiac a deň.

Globálna premenná Pridaj obsahuje informáciu o nutnosti vypísať hlavičku pri zapisovaní do textového výstupného súboru.

4.5 Dôležité funkcie programu

4.5.1 Vypísanie všetkých udalostí pri štarte programu

Náš program si po spustení od uživateľa vypýta cestu k logovaciemu inštalačnému súboru, ktorý musí byť uložený vo formáte XML. Na toto sme použili funkciu triedy OpenFileDialog. Ak uživateľ poskytol akceptovateľnú cestu, program ju uloží do premennej typu String^. Súbor potom rozdelí na 3 časti poďla komplexných prvkov. Tieto prvky sú System, UserData a RenderingInfo. Každý z týchto komplexných prvkov obsahuje viacero atribútov do ktorých bude vstupovať pri vyhľadávaní. Pomocou funkcie Count si program spočíta množstvo komplexných prvkov typu System, čo mu umožní použiť cyklus for pri vypisovaní. Následne program cyklicky prechádza celý súbor a vypíše atribúty: EventRecordID, Provider, Client, Date, PackageIdentifier, Computer a InitialPackageState do výstupnej tabuľky. Tento cyklus prebehne presne toľko krát koľko napočítal komplexných prvkov System.

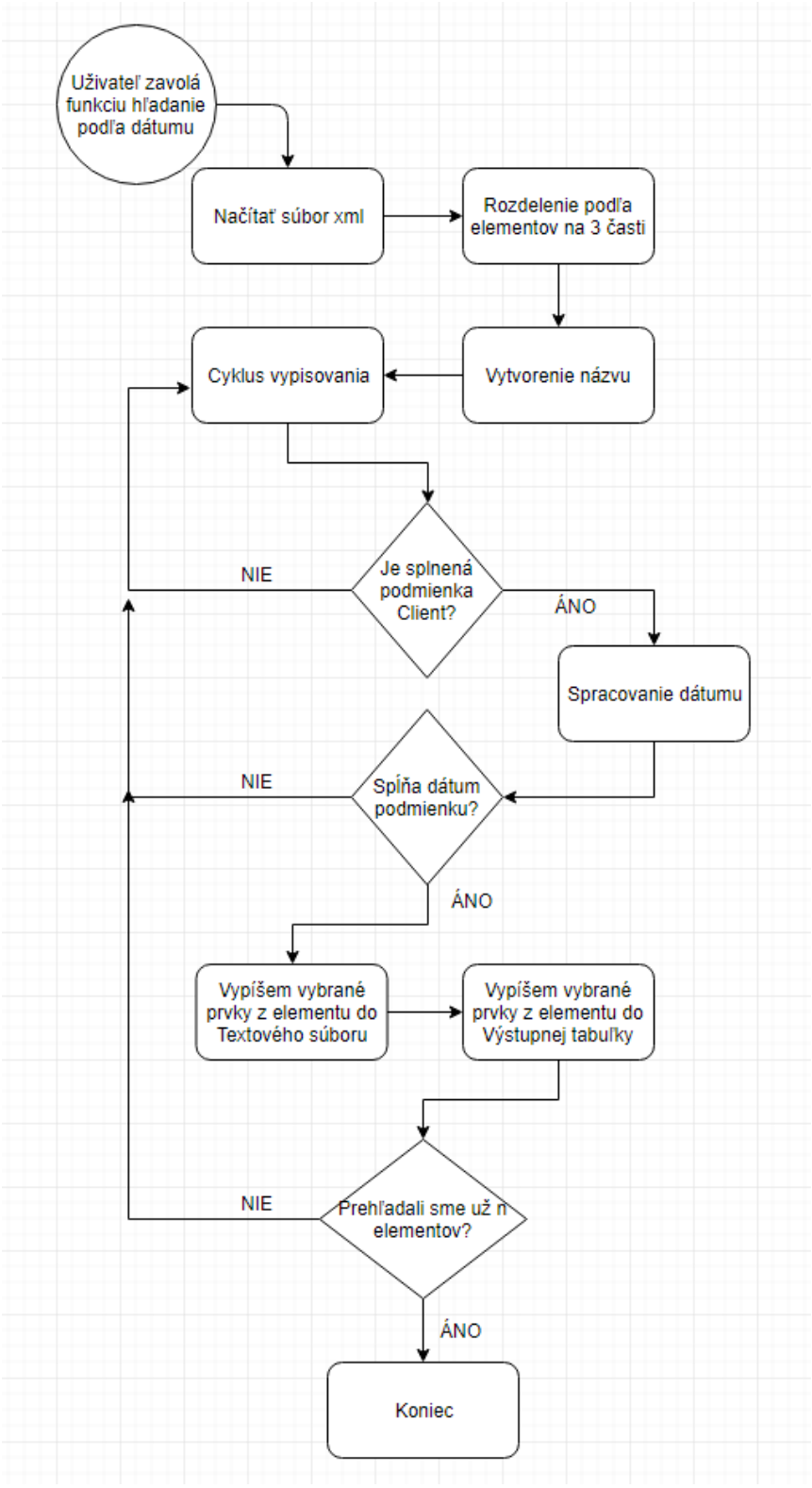


Obrázok 12 – Diagram funckie Vypísanie všetkých udalostí pri štarte programu

4.5.2 Vyhľadávanie udalostí podľa dátumu

Kedže pri každom vyhľadaní musí program vytvoriť export, preto začneme zistením dátumu a času. Na toto sme použili funkciu DateTime localDate, ktorá zistí aktuálny čas a dátum v počítači. Keď máme dátum, môžeme postupne vystavať budúci názov súboru. Názov bude obsahovat cestu k uživateľom vybratému súboru, názov súboru, aktuálny dátum, čas a informácie o hľadanej položke. Program následne vypíše celý názov pre vizuálnu kontrolu.

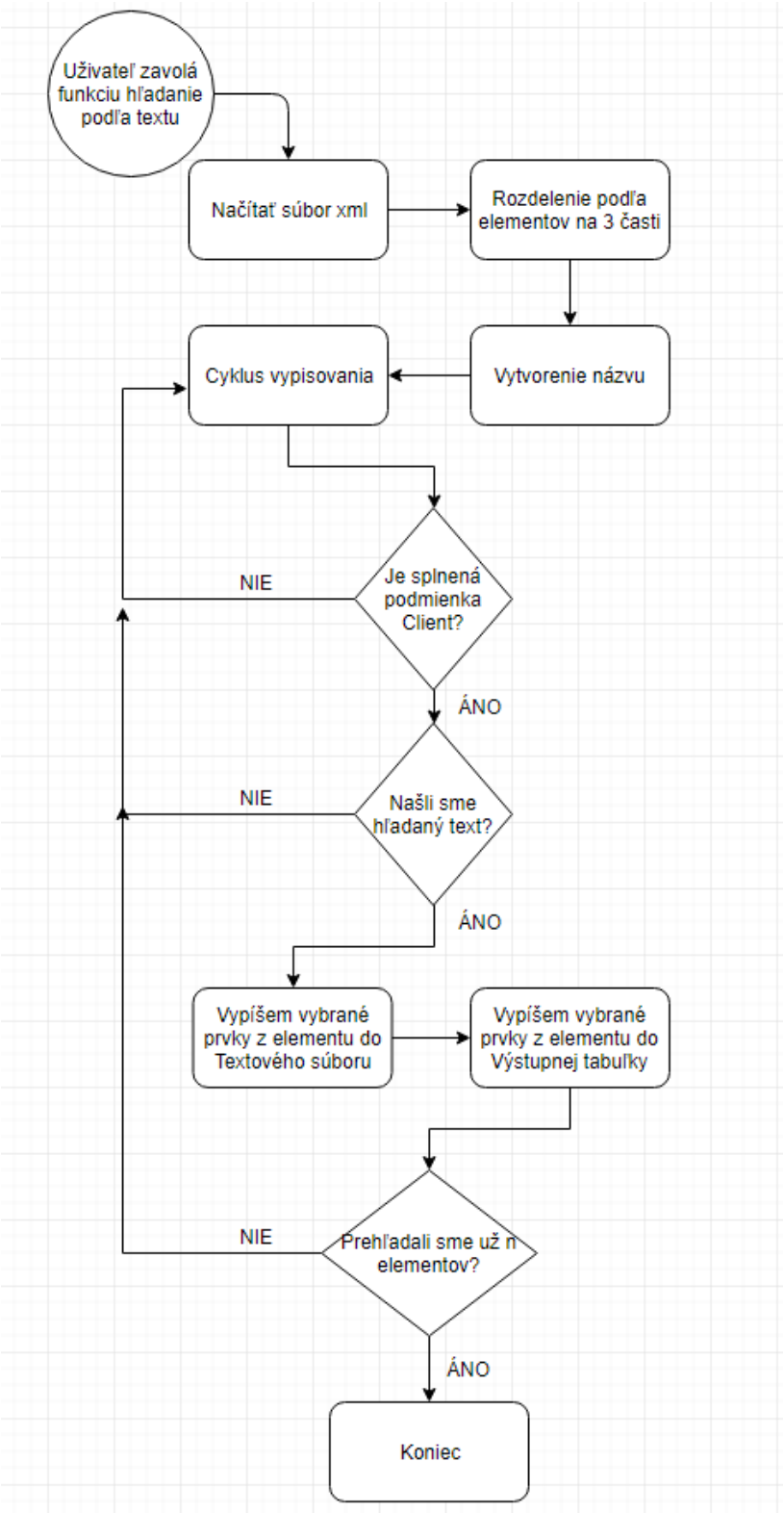
Súbor je následne rozdelený na 3 časti poďla komplexných prvkov, čo využijeme pre zjednodušený prístup. Na rade je cyklus, ktorý prehľadá súbor po prvkoch(event). V prvej časti program kontroluje či je splnená podmienka filtrovania podľa Client. Ak sa podmienka naplní tak začne s vyhľadávaním. Hladaný dátum sa nachádza v globálnej premennej Hladane. Program najprv nájde dátum vytvorenia prvku a rozdelí ho na rok, mesiac a deň. To isté potom spraví s hladaným dátumom. Po porovnaní je výsledok uložený do premennej resultF a spracovaný podľa toho či hladáme rovnaký, minulý alebo budúci dátum. Ak sú všetky podmienky splnené, tak je zavolaná funkcia riadok, ktorá vytvorí a vpíše výsledok do textového súboru. Následne je výsledok vpísaný aj do výstupnej tabuľky.



Obrázok 13 – Diagram funckie Vyhľadávanie udalostí podľa dátumu

4.5.3 Vyhľadávanie udalostí podľa textu

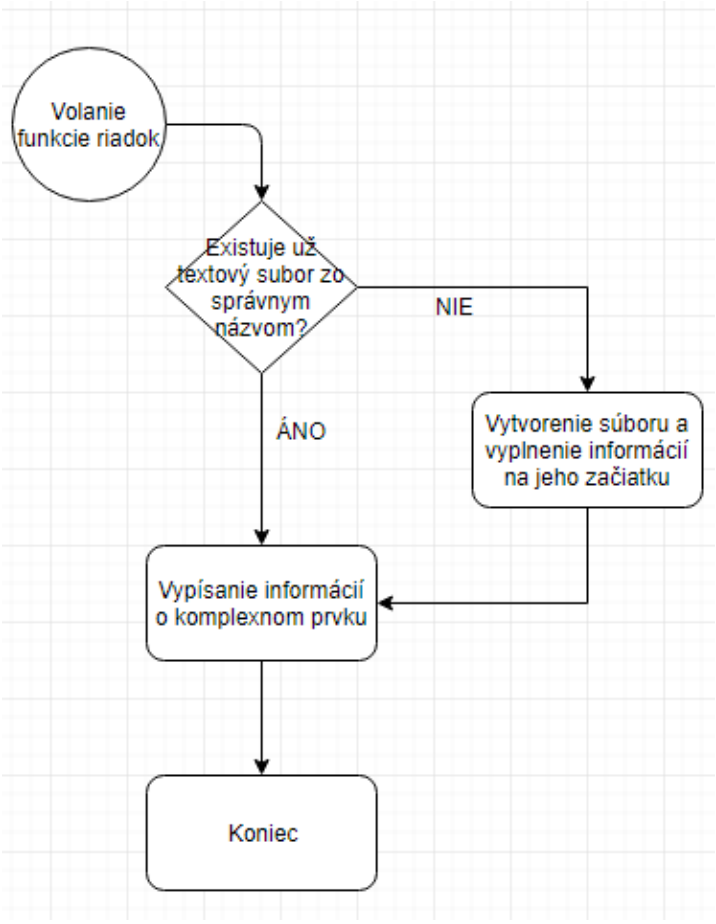
Vyhľadávanie podľa textu sa z veľkej časti podobá vyhľadávaniu podľa dátumu. Rozdiel prichádza po splnení podmienky Client. Hľadaný text sa takisto nachádza v premennej Hladane. Program začne od vrchu prehľadávať každý prvok a atribút, podmienka je splnená ak je hľadaný reťazec nájdený aspoň raz. Po úspešnom nájdení sa do premennej NasloX vloži hodnota TRUE. Vďaka tomu bude program vedieť, že má vypísať daný komplexný prvok.



Obrázok 14 – Diagram funckie Vyhľadávanie udalostí podľa textu

4.5.4 *Zapisovanie udalostí do textového súboru*

Funkcia, ktora vytvorí a následne vypĺňa textový výstup sa v našom programe nazíva riadok. Na vstupe potrebuje 2 premenné. Prvá premenná je typu String^. Musí obsahovat názov súboru a systémovú cestu kde bude uložený. Druhá premenná je typu integer, podľa nej program urči aké je ID komplexného prvku, ktorý sa chystá vypísať. Na vytvorenie a zapisovanie do výstupného súboru používame funkcie triedy StreamWriter. Program najprv vytvorí súbor s korektným názvom a vyplní hlavičku. Potom vypíše ID komplexného prvku, ktorý ide spracovať a následne ho celý vypíše. Na rozdiel od výstupnej tabulky je do výstupného súboru vypísaný plný obsah prvku. Každé ďaľšie volanie zapisuje do existujúceho súboru. Hlavička je tiež vypísaná len raz.



Obrázok 15 - Diagram funckie Zapisovanie udalostí do textového súboru

4.6 Vybrané časti zdrojového kódu

4.6.1 Načítanie vstupu od uživateľa

```
MoznostClient = comboBox3->Text;
                                    if (MoznostClient == "Client DSIM Package")
                                    {
                                           MoznostClient   =   "DISM   Package   Manager
Provider";
                                    }
                                    if (MoznostClient == "Client Windows Agent")
                                    {
                                           MoznostClient = "WindowsUpdateAgent";
                                    }
                                    if (MoznostClient == "Ziadne")
                                    {
                                           MoznostClient = "Ziadne";
                                    }
```

Zdrojový kód pre načítanie vstupu od uživateľa

Hore uvedený kód načíta uživateľom vybranú možnosť filtrovania podľa atribútu Client.

Vybraný text je načitaný do premennej MoznostClient. Nasledujú tri podmienky, ktoré upravia text tak aby sa dal porovnať s textom v atribúte Client.

4.6.2 Zapisovanie udalostí do výstupnej tabuľky

```
if (result == 0)
                              {
                                    String^ x = Systemnode1["EventRecordID"]->InnerText;
                                    ID = true;
                                    riadok (Systemnode1, x);
                                    Pridaj = true;
                                    ID = false;
                                    riadok (Systemnode2, x);
                                    x = "x";
                                    riadok (Systemnode3, x);

                                    String^ Help = Systemnode1["TimeCreated"]->OuterXml;
                                    String^ Help2 = Help->Substring(25,10);
                                    ListViewItem     ^     lviProperty     =     gcnew
ListViewItem(Systemnode1["EventRecordID"]->InnerText);
                                    lviProperty->SubItems->Add(Systemnode3["Provider"]-
>InnerText);
```

```
                                    lviProperty->SubItems->Add(Systemnode2["Client"]-
>InnerText);
                                    lviProperty->SubItems->Add(Help2);
                                    lviProperty->SubItems-
>Add(Systemnode2["PackageIdentifier"]->InnerText);
                                    lviProperty->SubItems->Add(Systemnode1["Computer"]-
>InnerText);

                                    if (Systemnode2["InitialPackageState"] != nullptr)
                                    {
                                          lviProperty->SubItems-
>Add(Systemnode2["InitialPackageState"]->InnerText);
                                    }
                                    Tabulka->Items->Add(lviProperty);
                              }
```

Zdrojový kód pre zapisovanie udalostí do výstupnej tabuľky

Hore uvedená časť kódu zapíše jeden komplexný prvok do výstupnej tabuľky a postupne volá funkciu riadok, ktorá urobý to isté pre výstupný textový súbor.

Program najprv zistí ID komplexného prvku, ktorý chce vypísať a uloží ho do premennej x. Tá je potom spolu s názvom súboru poskytnutá pri volaní funkcie riadok. Globálne premenné ID a Pridaj su postupne upravované tak aby funkcia riadok vedela či má alebo nemá na začiatku napisať ID komplexného prvku s ktorým pracuje. Program potom začne zapisovať do výstupnej tabuľky za pomoci funkcií triedy ListViewItem. Dátum si vystrihneme z atribútu TimeCreated cez metódu Substring. Zvyšné dáta nieje potreba upravovať a sú vypísané v celku.

```
lviProperty->SubItems->Add(Systemnode2["Client"]->InnerText);
```

Každý takýto riadok kódu najprv vstúpy do atribútu, v tomto prípade je to atribút Client, a pripraví jeho textovú časť na vypísanie. Časť `lviProperty->SubItems->Add` je zodpovedná za umiestnenie na správne miesto v tabuľke.

```
Tabulka->Items->Add(lviProperty);
```

Tento riadok potom všetko naraz vypíše pomocou metódy Items->Add.

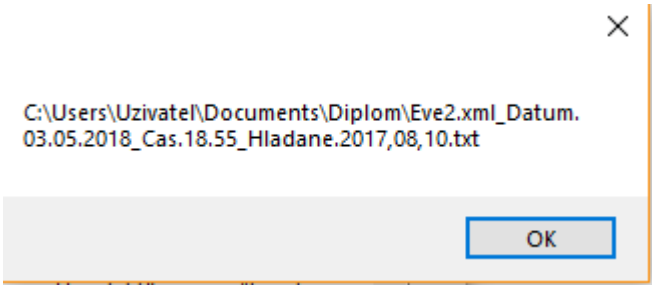
4.6.3 *Tvorenie názvu výstupného súboru*

```
        DateTime localDate = DateTime::Now;
        CultureInfo^ culture = gcnew CultureInfo("de-DE");
        Pomocka = (localDate.ToString(culture));
        String^ DoNazvu = Pomocka->Substring(0,10);
        String^ DoNazvu2 = Pomocka->Substring(11,2);
        String^ DoNazvu3 = Pomocka->Substring(14,2);
        NovySubor = (strFilename + "_Datum. " + DoNazvu + "_Cas." + DoNazvu2 + "." +
DoNazvu3 + "_Hladane." + Hladane + ".txt");
        MessageBox::Show(NovySubor);
```

Zdrojový kód pre tvorenie názvu výstupného súboru

Hore uvedená časť kódu je zodpovedná za zistenie dátumu a času, toto dosiahne vďaka metóde DateTime::Now. Táto metóda zistí aktuálny čas a dátum podľa času operačného systému. Kedže operačný systém môže mať rôzne formáty, museli sme dátum prispôsobiť tak, aby vyhovoval nášmu programu. Toto sme dosiahli vďaka metóde CultureInfo. Táto metóda preformátuje dátum tak aby nám vyhoval. Pre náš program sme vybrali Nemecký formát času. Tento formát je vhodný lebo má vždy rovnakú dĺžku. Taktiež sme sa chceli vyhnúť 12 hodinovému formátu. Tri nasledujúce riadky postupne vytiahnu dátum, hodiny a minúty samostatne pomocou metódy Substring. Toto je dôležité keďže čas je oddlený pomocou znaku dvojbodka, čo je invalidný znak pre názvy súborov v operačnom systéme Windows. Kebyže ho nedostránime tak program nedokáže vytvoriť diskovú súbor.

Keď máme pripravené všetky potrebné veci pre tvorbu názvu, spoločne ich uložíme do bunky NovySubor, ktorá má formát String^. Následne tento už kompletný názov vypíšeme pre vizuálnu kontorlu.



Obrázok 16 – Vizuálna kontrola umiestnenia súboru a jeho názvu

Môžeme vidieť, že názov momentálne obsahuje aj cestu, kde bude súbor uložený. Táto cesta však súčasťou názvu nebude.

4.6.4 Rozdelenie udalostí na komplexné prvky

```
docProperties->Load(strFilename);
        XmlElement ^ elmProperty = docProperties->DocumentElement;
        XmlNodeList ^ lstProperties1 = elmProperty->GetElementsByTagName("System");
              XmlNodeList         ^         lstProperties2         =         elmProperty-
>GetElementsByTagName("UserData");
              XmlNodeList         ^         lstProperties3         =         elmProperty-
>GetElementsByTagName("RenderingInfo");
              pocet = lstProperties1->Count;
              for( j = 0; j < pocet; j = j + 1 )
              {
                     XmlNode ^ Systemnode1 = lstProperties1->Item(j);
                     XmlNode ^ Systemnode2 = lstProperties2->Item(j)->FirstChild;
                     XmlNode ^ Systemnode3 = lstProperties3->Item(j);
```

Zdrojový kód pre rozdelenie udalostí na komplexné prvky

Pred tým ako začneme vstupovať do súboru XML si ho rozdelíme na menšie časti. Toto nám umožní jednoduchšie vstupovať do atribútov udalostí.

Najprv si pomocou metódy Load načítame súbor XML do premennej docProperties. Potom do nej vstúpime a vytvoríme triedu elmProperty typu XmlElement, kde načítame všetky udalosti. V ďalšom kroku vytvoríme zoznam uzlov a uložíme ho do triedy typu XmlNodeList.

Potom si z nej vytvoríme ďalšie tri triedy, ktoré budú obsahovať komplexné prvky udalostí. Každá z týchto troch tried bude pridelené iné komplexné prvky. Delenie je podľa názvu. Trieda lstProperties1 bude obsahovať všetky komplexné prvky s názvom System. Trieda lstProperties2 bude obsahovať všetky komplexné prvky s názvom UserData. Trieda lstProperties3 bude obsahovať všetky komplexné prvky s názvom RenderingInfo.

Po dokončení delenia zavoláme cyklus for, ktorý bude postupne vstupovať do atribútov komplexných prvkov.

4.6.5 *Filtrovanie podľa atribútu Client*

```
for( j = 0; j < pocet; j = j + 1 )
	{
		Cl = 0;
		if (Systemnode2["Client"]->InnerText == MoznostClient)
			{
				Cl = 1;
			}
		if (MoznostClient == "Ziadne")
			{
				Cl = 1;
			}
		if (Cl == 1)
		{
```

Zdrojový kód pre filtrovanie pomocou podmienky Client

Filtrovanie pomocou podmienky Client je veľmi jednoduché a priame. Každý vyhľadávací cyklus má v sebe tri podmienky if, ktoré skúmaju tri rozdielne kritériá. Prva podmienka zistí či atribút daného uzla obsahuje možnosť, ktorú si uživateľ vybral. Táto uživateľom vybraná hodnota je uložená do premennej MoznostClient. Ak nájde zhodu tak nastaví premennú Cl na 1. Toto bude použité neskôr. Daľej kontroluje či premenná MoznostClient obsahuje reťazec Ziadne. Táto podmienka taktiež nastaví pemennú Cl na 1 po nájdení zhody. Posledná podmienka určuje či má cyklus pokračovat. Pokračuje v prípade, že jedna z prvých dvoch podmienok nastavili premennú CL na 1.

4.6.6 *Vyhľadávanie podľa dátumu a spracovanie vstupu pri vyhladávaní podľa dátumu*

```
String^ tmp = Systemnode["TimeCreated"]->OuterXml;
String^ rok = tmp->Substring(25,4);
		String^ mesiac = tmp->Substring(30,2);
		String^ den = tmp->Substring(33,2);
		result = String::CompareOrdinal( rok, 0, Hladane, 0, 4);
		result2 = String::CompareOrdinal( mesiac, 0, Hladane, 5, 2);
		result3 = String::CompareOrdinal( den, 0, Hladane, 8, 2);
//-------------------------------------------------------------------------
		if (result == 0)
			{
			resultF = 0;
		}else if (result < 0)
			{
			resultF = -100;
		}else if (result > 0)
```

```
                    {
                    resultF = 100;
                    }
//------------------------------------------------------------------------
              if (result2 == 0)
                    {
                    resultF = 0;
              }else if (result2 < 0)
                    {
                    resultF =  resultF - 10;
              }else if (result2 > 0)
                    {
                    resultF = resultF + 10;
                    }
//------------------------------------------------------------------------
              if (result3 == 0)
                    {
                    resultF = 0;
              }else if (result3 < 0)
                    {
                    resultF = resultF - 1;
              }else if (result3 > 0)
                    {
                    resultF = resultF + 1;
                    }
//------------------------------------------------------------------------
              if (Moznost == "="){
                           if (resultF == 0)
```

Zdrojový kód pre Spracovanie vstupu pri vyhľadávaní podľa dátumu

Na začiatku si do premennej tmp vložime atribút TimeCreated. Tento atribút obsahuje dátum inštalácie udalosti. Tento dátum potom postupne delíme na podčasti, ktoré budeme spracovávať. Podčasti, ktoré vytvoríme sú rok, mesiac a deň. Toto delenie je dôležité keď budeme zisťovať či dátum udalosti je skorší alebo neskorší ako uživateľom zadaný dátum.

Riadky, ktoré začínaju premennými result porovnávajú dátum prehľadávanej udalosti oproti uživateľom zadanému dátumu. Metóda CompareOrdinal nám po porovnaní dá nulu, ak sa rovnaju. Kladnu hodnotu, ak je prvé číslo väčšie ako nasledujúce. Zápornú hodnota, pokiaľ je nasledujúce čislo menšie ako prvé.

Zvyšné tri oddelenia if podmienok napĺňajú premennú resultF. Program postupne prezerá hodnoty premenných result1, result2 a result3. Ak je hodnota rovná nule, k premennej resulF je pripočítaná nula. Ak je hodnota vyššia ako nula, k premennej resulf je sa pripočíta buď 100(rok), 10(mesiac), 1(deň). Ak je hľadaná hodnota nižšia, tak sa tieto čísla odpočítajú.

Vďaka tomuto bude hodnota resultF na konci obsahovať nulu, ak sú dátumy zhodné. Zápornú hodnotu, ak je hľadaný dátum nižší ako dátum inštalácie udalosti. Kladnú hodnotu, ak je hľadaný dátum vyšší ako dátum inštalácie hľadanej udalosti.

Keď máme naplnenu premennú resultF, program sa posunie k časti zapisovania. Nasledovná časť kódu je jemne upravená, aby neobsahovala časti, kde sa program venuje zapisovaniu do výstupnej tabuľky a výstupného textového súboru.

```
if (Moznost == "="){
                if (resultF == 0)
                {
                (Vymazané zapísanie)
                }
}else if (Moznost == "<"){
                if (resultF < 0)
                {
                (Vymazané zapísanie)
                }
}else if (Moznost == ">"){
                if (resultF > 0)
                {
                 (Vymazané zapísanie)
                }
```

Zdrojový kód pre Spracovanie premennej resultF

Táto časť zdrojového kódu sa postupne pýta pomocou podmienok if. Pokiaľ je uživateľom zadaná možnosť vyhľadávania rovnakého dátumu, program skontroluje či je premenná resultF nastavená na nulu. Ak áno, začne zapisovať do výstupnej tabuľky a výstupného súboru. Pokiaľ uživateľ hľadá skoršie dátumy, program skontroluje či je hodnota resultF nižšia ako nula. Pokiaľ uživateľ hľadá neskoršie dátumy, program skontroluje či je hodnota resultF vyššia ako nula.

4.6.7 Vyhľadávanie podľa textového reťazca

```
for each(XmlNode ^ node in Systemnode1)
                    {
                        String^ tmp = node->InnerText;
                        Naslo = tmp->Contains( Hladane );
```

```
            if (Naslo == true) {NasloX=true;}
        }
        for each(XmlNode ^ node in Systemnode2)
        {
            String^ tmp = node->InnerText;
            Naslo = tmp->Contains( Hladane );
            if (Naslo == true) {NasloX=true;}
        }
        for each(XmlNode ^ node in Systemnode3)
        {
            String^ tmp = node->InnerText;
            Naslo = tmp->Contains( Hladane );
            if (Naslo == true) {NasloX=true;}
        }
```

Zdrojový kód pre vyhľadávanie podľa textového reťazca

Hore uvedený kód slúži na spracovanie textového vstupu. Rozdelený je na tri časti. Každá časť prehľadáva jeden komplexný prvok udalosti.

Na začiatku sa spustí cyklus for each, ktorý zaručí, že bude prehľadaný každý atribút komplexneho prvku. Po nahraní textového obsahu atribútu do premennej tmp voláme metódu Contains, ktorá zistí či sa hľadaný reťazec Hladane nachádza v atribúte.

Tento proces sa potom zopakuje tri krát, keďže toľko komplexných prvkov sa nachádza v každej udalosti.

4.6.8 *roztváracia ponuka s možnosťou predvolených vyhľadávaní*

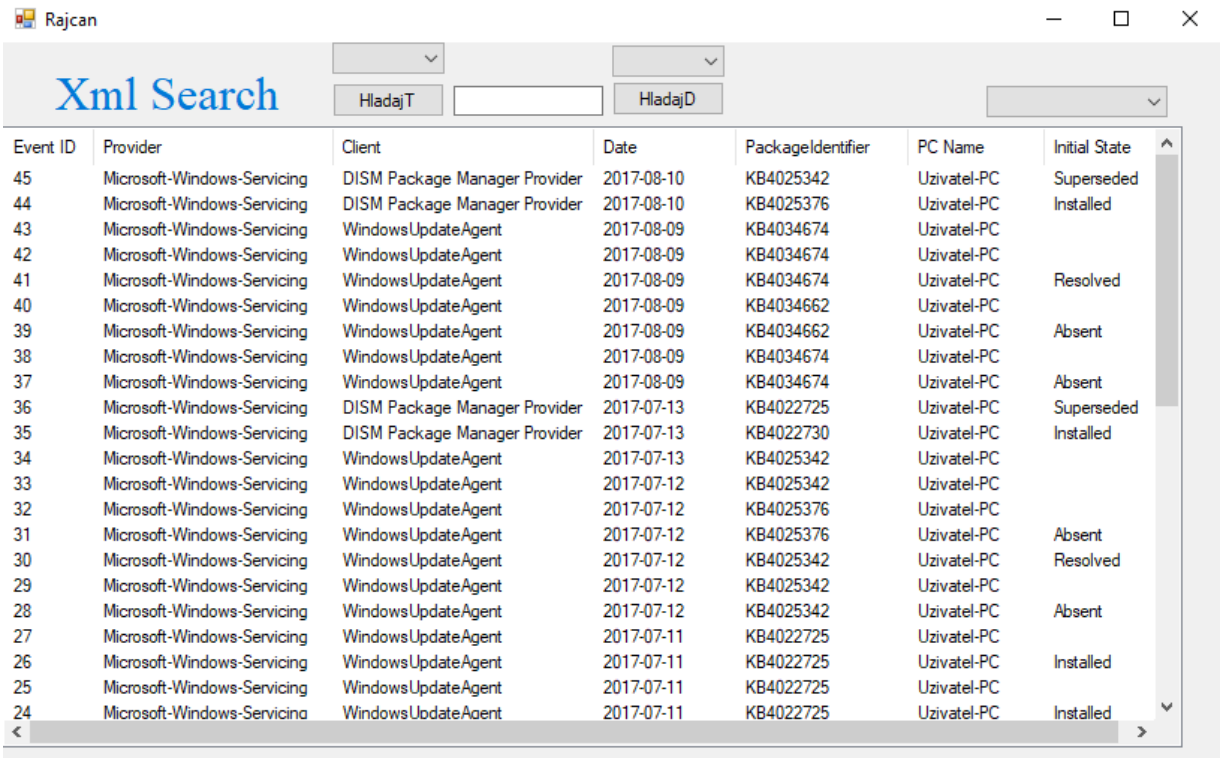
```
private:    System::Void    comboBox2_SelectedIndexChanged(System::Object^       sender,
System::EventArgs^  e)
Moznost2 = comboBox2->Text;
             if (Moznost2 == "Tento Rok") {Hladane="2018";}
             if (Moznost2 == "Predosli Rok") {Hladane="2017";}
             XmlNode ^ Systemnode = lstProperties1->Item(j);
             String^ tmp = Systemnode["TimeCreated"]->OuterXml;
             String^ rok = tmp->Substring(25,4);
             result = String::CompareOrdinal( rok, 0, Hladane, 0, 4);
             if (result == 0)
                    {
```

Zdrojový kód pre roztváraciu ponuku s možnosťou predvolených vyhľadávaní

Kód tejto možnosti by sa dal zhrnúť ako kombinácia načítania zo vstupu uživateľa a vyhľadávania podľa dátumu. Zaujímavosť je v tom, ze program začne konať bez toho aby boli použité možnosti tlačidiel. Toto je zabezpečene tým, že funkcia kde by sa malo riešit načítanie dát plynule pokračuje do vyhľadávania a vypísania udalostí.

Po zvolení možnosti z roztváracej ponuky sa zvolený text z bunky comboBox2 presunie do premennej Moznost2. Nasledujúce podmienky if naplnia globálnu premennú Hladane textom, podľa toho čo zvolil uživateľ. Následne program vyberie a vstúpy do atribútu TimeCreated a vyberie. Mesiac ani deň v tomto prípade nieje potrebný. Nasleduje porovnanie metódou CompareOrdinal a výsledok je zapísaný do premennej result. Pokiaľ sa do premennej uloží nula, znamená to, že udalosť vyhovuje a môže byť vypísaná.

4.7 Popis používateľského rozhrania



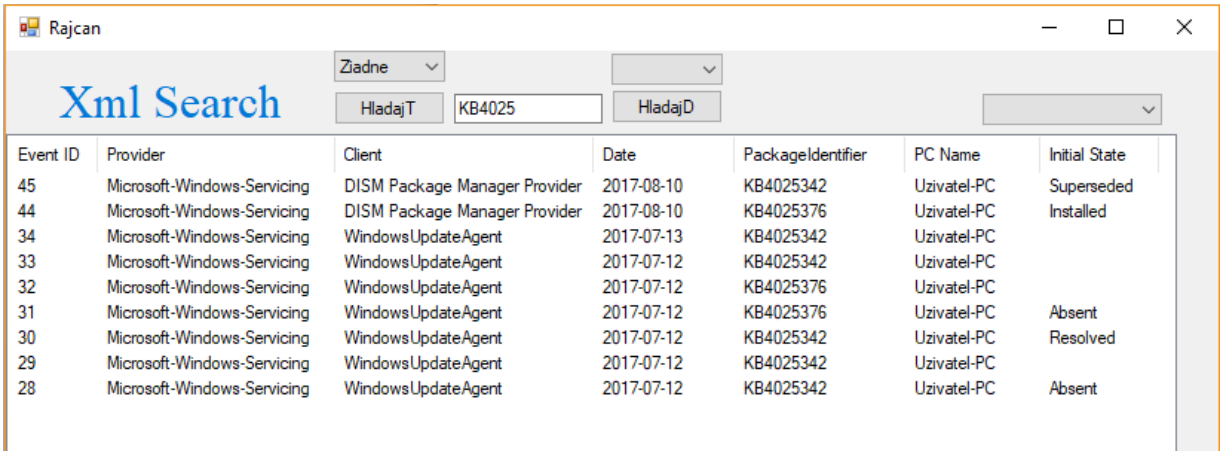
Event ID	Provider	Client	Date	PackageIdentifier	PC Name	Initial State
45	Microsoft-Windows-Servicing	DISM Package Manager Provider	2017-08-10	KB4025342	Uzivatel-PC	Superseded
44	Microsoft-Windows-Servicing	DISM Package Manager Provider	2017-08-10	KB4025376	Uzivatel-PC	Installed
43	Microsoft-Windows-Servicing	WindowsUpdateAgent	2017-08-09	KB4034674	Uzivatel-PC	
42	Microsoft-Windows-Servicing	WindowsUpdateAgent	2017-08-09	KB4034674	Uzivatel-PC	
41	Microsoft-Windows-Servicing	WindowsUpdateAgent	2017-08-09	KB4034674	Uzivatel-PC	Resolved
40	Microsoft-Windows-Servicing	WindowsUpdateAgent	2017-08-09	KB4034662	Uzivatel-PC	
39	Microsoft-Windows-Servicing	WindowsUpdateAgent	2017-08-09	KB4034662	Uzivatel-PC	Absent
38	Microsoft-Windows-Servicing	WindowsUpdateAgent	2017-08-09	KB4034674	Uzivatel-PC	
37	Microsoft-Windows-Servicing	WindowsUpdateAgent	2017-08-09	KB4034674	Uzivatel-PC	Absent
36	Microsoft-Windows-Servicing	DISM Package Manager Provider	2017-07-13	KB4022725	Uzivatel-PC	Superseded
35	Microsoft-Windows-Servicing	DISM Package Manager Provider	2017-07-13	KB4022730	Uzivatel-PC	Installed
34	Microsoft-Windows-Servicing	WindowsUpdateAgent	2017-07-13	KB4025342	Uzivatel-PC	
33	Microsoft-Windows-Servicing	WindowsUpdateAgent	2017-07-12	KB4025342	Uzivatel-PC	
32	Microsoft-Windows-Servicing	WindowsUpdateAgent	2017-07-12	KB4025376	Uzivatel-PC	
31	Microsoft-Windows-Servicing	WindowsUpdateAgent	2017-07-12	KB4025376	Uzivatel-PC	Absent
30	Microsoft-Windows-Servicing	WindowsUpdateAgent	2017-07-12	KB4025342	Uzivatel-PC	Resolved
29	Microsoft-Windows-Servicing	WindowsUpdateAgent	2017-07-12	KB4025342	Uzivatel-PC	
28	Microsoft-Windows-Servicing	WindowsUpdateAgent	2017-07-12	KB4025342	Uzivatel-PC	Absent
27	Microsoft-Windows-Servicing	WindowsUpdateAgent	2017-07-11	KB4022725	Uzivatel-PC	
26	Microsoft-Windows-Servicing	WindowsUpdateAgent	2017-07-11	KB4022725	Uzivatel-PC	Installed
25	Microsoft-Windows-Servicing	WindowsUpdateAgent	2017-07-11	KB4022725	Uzivatel-PC	
24	Microsoft-Windows-Servicing	WindowsUpdateAgent	2017-07-11	KB4022725	Uzivatel-PC	Installed

Obrázok 17 – Používateľské rozhranie

Používateľské rozhranie obsahuje: dve tlačitka (HladajT a HladajD), dve roztváracie ponuky pre spresnenie vyhľadávania, jednu roztváraciu ponuku s možnosťou predvolených vyhľadávaní, jedno textové pole a výstupnú tabuľku.

4.8 Praktické príklady vyhľadávania v inštalačných logovacích súboroch

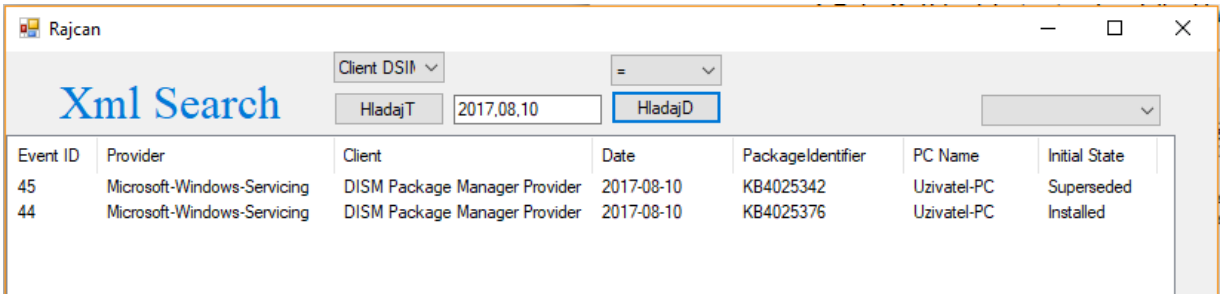
4.8.1 Príklad 1, textové vyhľadávanie



Obrázok 18 – Príklad 1

V tomto príklade sme vyhľadávali všetky inštalácie, ktoré obsahujú textový reťazec KB4025. Takýmto spôsobom si vieme ľahko vyfiltrovať všetky inštalácie, ktorých PackageIdentifier patrí do istej rady. Filtrovanie podľa Client sme nastavili na Ziadne, keďže chceme vypísať všetkých klientov. Na obrázku hore vidíme výsledok tohto vyhľadávania.

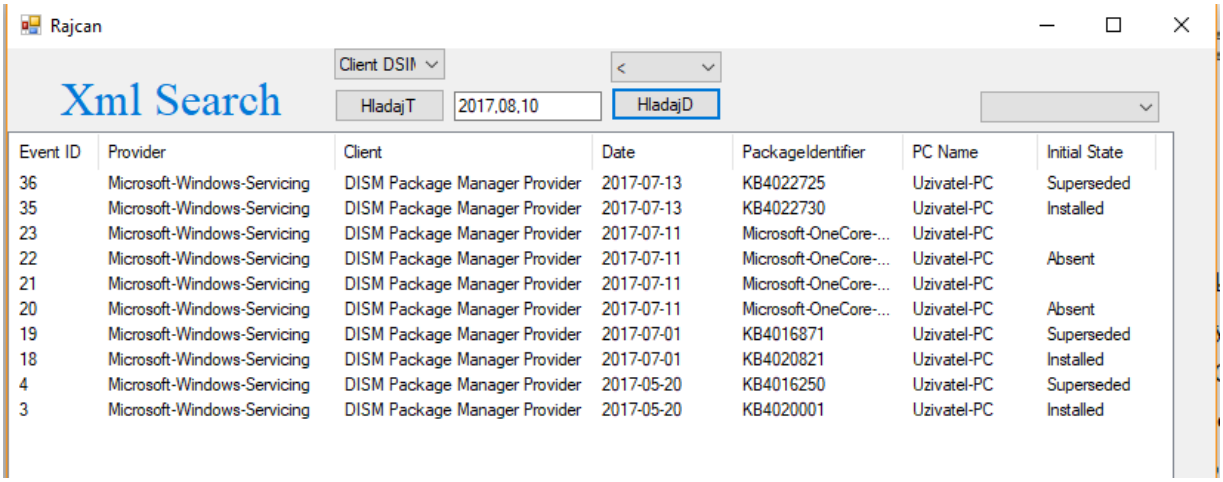
4.8.2 Príklad 2, vyhľadávanie podľa dátumu



Obrázok 19 – Príklad 2a

V tomto príklade sme vyhľadávali podľa dátumu. Vyžiadali sme si všetky inštalačné súbory, ktoré boli inštalované v danom dátume za podmienky, že klient je DSIM Package

Manager Provider. Ako vidíme hore, sú len dve inštalácie, ktoré vyhovujú týmto kritériam. Následne sme zmenili naše zadanie a vypýtali sme si všetky inštalácie so skorším dátumom.



Event ID	Provider	Client	Date	PackageIdentifier	PC Name	Initial State
36	Microsoft-Windows-Servicing	DISM Package Manager Provider	2017-07-13	KB4022725	Uzivatel-PC	Superseded
35	Microsoft-Windows-Servicing	DISM Package Manager Provider	2017-07-13	KB4022730	Uzivatel-PC	Installed
23	Microsoft-Windows-Servicing	DISM Package Manager Provider	2017-07-11	Microsoft-OneCore-...	Uzivatel-PC	
22	Microsoft-Windows-Servicing	DISM Package Manager Provider	2017-07-11	Microsoft-OneCore-...	Uzivatel-PC	Absent
21	Microsoft-Windows-Servicing	DISM Package Manager Provider	2017-07-11	Microsoft-OneCore-...	Uzivatel-PC	
20	Microsoft-Windows-Servicing	DISM Package Manager Provider	2017-07-11	Microsoft-OneCore-...	Uzivatel-PC	Absent
19	Microsoft-Windows-Servicing	DISM Package Manager Provider	2017-07-01	KB4016871	Uzivatel-PC	Superseded
18	Microsoft-Windows-Servicing	DISM Package Manager Provider	2017-07-01	KB4020821	Uzivatel-PC	Installed
4	Microsoft-Windows-Servicing	DISM Package Manager Provider	2017-05-20	KB4016250	Uzivatel-PC	Superseded
3	Microsoft-Windows-Servicing	DISM Package Manager Provider	2017-05-20	KB4020001	Uzivatel-PC	Installed

Obrázok 20 – Príklad 2b

Takéto vyhľadávanie poskytlo omnoho väčšie množstvo výsledkov.

4.9 Textový výstup programu

Po úspešnom prehľadaní vstupného súboru sa vytvorí diskový súbor vo formáte txt. Tento súbor obsahuje hlavičku, kde sú základné informácie o vyhľadávaní, čase vyhľadávania a komplexné prvky, ktoré vyhovujú zadaniu vyhľadávania. Výstupný súbor sa automaticky vytvorí v priečinku kde sa nachádza vstupný súbor. V názve výstupného súboru je názov vstupného súboru, čas vyhľadávania a hľadaný reťazec. V niektorých prípadoch, napríklad pri použití roztváracej ponuky s predvolenými vyhľadávaniami. Ako môže vyzerať výstup su uvidieme v nasledujúcej časti.

4.9.1 Ukážka výstupného súboru

Datum a cas vytvorenia: 2. 5. 2018 23:54:39

Event ID: 45

<System>

<Provider>

<EventID> 1

<Version> 0

<Level> 0

<Task> 1

<Opcode> 0

<Keywords> 0x8000000000000000

<TimeCreated>

<EventRecordID> 45

<Correlation>

<Execution>

<Channel> Setup

<Computer> Uzivatel-PC

<Security>

<CbsPackageInitiateChanges>

<PackageIdentifier> KB4025342

<InitialPackageState> Superseded

<IntendedPackageState> Absent

<Client> DISM Package Manager Provider

<RenderingInfo>

<Message> Inicializujú sa zmeny pre balík KB4025342. Aktuálny stav je Nahradenie. Cieľový stav je Neprítomné. Identifikácia klienta: DISM Package Manager Provider.

<Level> Information

<Task>

<Opcode> Info

<Channel>

<Provider> Microsoft-Windows-Servicing

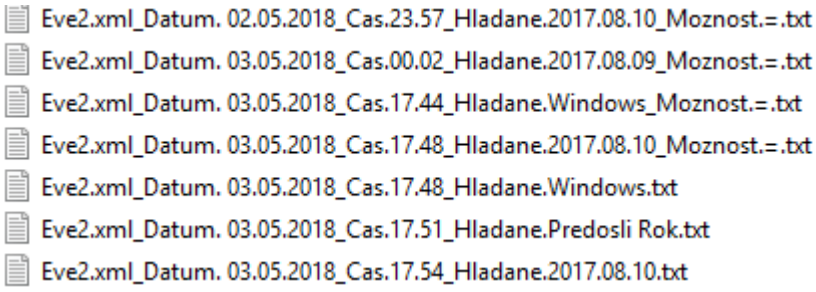
<Keywords>

Event ID: 44

<System>

<Provider>
<EventID> 1
<Version> 0
<Level> 0
<Task> 1
<Opcode> 0
<Keywords> 0x8000000000000000
<TimeCreated>
<EventRecordID> 44
<Correlation>
<Execution>
<Channel> Setup
<Computer> Uzivatel-PC
<Security>
<CbsPackageInitiateChanges>
<PackageIdentifier> KB4025376
<InitialPackageState> Installed
<IntendedPackageState> Absent
<Client> DISM Package Manager Provider
<RenderingInfo>
<Message> Inicializujú sa zmeny pre balík KB4025376. Aktuálny stav je Nainštalované. Cieľový stav je Neprítomné. Identifikácia klienta: DISM Package Manager Provider.
<Level> Information
<Task>
<Opcode> Info
<Channel>
<Provider> Microsoft-Windows-Servicing
<Keywords>

Tu môžeme vidieť štruktúru vyhľadaných udalostí. Taktiež si môžeme prezrieť hlavičku, ktorá sa nachádza na začiatku. Hlavička má jednotnú formu pre všetky druhy vyhľadávania. Keďze každé vyhľadávanie vytvorí nový výstupný súbor, priečinok sa môže veľmi rýchlo naplniť. Duplicitné vyhľadávania ukladané niesu.



Obrázok 21 – Ukážka výstupných súborov v adresári.

5. Záver

V súčasnosti je stále častejšie, že kontrolu chýb v počítači si nevykonávame sami. Na tento účel môžu slúžiť buď zamestnanci kontaktných centier, alebo externé spoločnosti. Toto je spôsobené rýchlym vývojom informačných technológií, ale aj narastajúcim množstvom uživateľov. V budúcnosti môžeme predpokladať dalšie narastanie dôležitosti informačných technológií, keďže ich využitie sa neustále rozširuje. Toto rozširovanie sa týka tak súkromnej ako aj verejnej sféry. Vďaka tomu bude potreba externej pomoci neustále narastať. Rozširujúca sa globalizácia taktiež zvyšuje potrebu spolupráce ľudi z rôznych častí sveta.

Výsledkom našej práce je program, ktorým sme sa snažili zjednodušiť možnosť kontroly inštalačných súborov pre takýchto externých pracovníkov. Náš program nepotrebuje inštaláciu. Na jeho spustenie stačí exe súbor, ktorý generuje Visual Studio.

Po spustení si vyžiada vstupný XML súbor, ktorý môže byť uložený kdekoľvek na disku zariadenia. Toto je veľmi výhodné, kedže to umožňuje kontorlu inštalačných súborov aj bez priameho prístupu k počítaču z ktorého pochádzajú. Tým pádom je možné tento súbor zaslať napríklad e-mailom. Toto by určite ocenili uživatelia z rozličných časových pásiem.

Po načítaní náš program vypíše dôležité údaje o inštalačných súboroch do jednoduchej tabuľky. Táto tabuľka bude uživateľa sprevádzat počas celej práce s programom. Vďaka nej je možne efektívne využivať možnosti prehľadávania, ktoré náš program ponúka. Vyhľadávanie cez reťazec umožní komplexné vyhľadávanie udalostí. Vyhľadávanie pomocou dátumu je veľmi výhodné ak bude uživateľ potrebovať údaje za určité časové obdobie. Toto sa bude dať využit napríklad ak dôjde k chybe počas konkrétneho obdobia, alebo ak bude uživateľ chcieť vidieť len obmedzené množstvo udalostí.

Náš program ponúka možnosti na spresnenie a zjednodušenie vyhľadávania pomocou pripraveného filtra podľa atribútu Client, alebo vďaka prednastavenému vyhľadávaniu na konkrétne roky.

Ak si bude uživateľ potrebovat zobrať výsledky na ďalšie preskúmanie, program automaticky vytvára výstupny subor vo formáte txt. Tento formát je veľmi jednoduchý a dá sa použiť na takmer všetkých zariadeniach. Jeho pamäťová nenáročnosť taktiež prispieva k tomu

aby sa takýto súbor dal ľahko prenášať. Názov tohto súboru dá uživateľovi jednoduchý prehľad o čase a typu vyhľadávania.

Za priaznivý výsledok považujeme jednoduchosť a pamäťovú nenáročnosť. Veľmi pozitívne je aj to, že náš program spĺňa všetky ciele, ktoré sme si vytýčili na začiatku našej práce.

Zoznam použitej literatúry

FunctionX, Inc. 2008. Introduction to XML - The Extensible Markup Language [Online] Dostupné na internete: http://www.functionx.com/vccli/xml/Lesson01.htm. [15.7.2017]

W3schools editor team, 2013. XML Tutorial [Online]
Dostupné na internete: https://www.w3schools.com/xml/default.asp. [2.7.2017]

Sitesbay online informatics editorial, 2016. C++ Programs [Online]
Dostupné na internete: https://www.sitesbay.com/cpp-program/index. [3.7.2017]

HAL9000, 2017 - 5 Alternative Event Viewers To Read Windows Event Logs. s. 16-17 [Online]
Dostupné na internete: https://www.raymond.cc/blog/a-built-in-system-monitor-thats-easier-to-read-than-event-viewer/. [18.9.2017]

Admin team, 8. Január 2013, - Záver diplomovej práce, s. 50-51 [Online]
Dostupné na internete: https://pisanieprac.sk/zaver-diplomovej-prace/. [20.4.2018]

Rôzny autory, 2017 - Algoritmy vyhľadávania, s. 20-21 [Online]
Dostupné na internete: http://www.kiwiki.info/index.php/Algoritmy_vyh%C4%BEad%C3%A1vania. [13.4.2018]

GeeksforGeeks editorial team, 2016 - Linear Search, s. 20 [Online]
Dostupné na internete: https://www.geeksforgeeks.org/linear-search/. [11.4.2018]

GeeksforGeeks editorial team, 2016 - Binary Search, s. 20-21 [Online]
Dostupné na internete: https://www.geeksforgeeks.org/binary-search/. [11.4.2018]

Microsoft team, 2017 - XmlAttributeCollection::ItemOf Property (Int32), praktická časť [Online]
Dostupné na internete: https://msdn.microsoft.com/en-us/library/0ftsfa87(v=vs.110).aspx. [14.4.2018]

Microsoft team, 2017 - XmlAttributeCollection::ItemOf Property (Int32), praktická časť [Online]
Dostupné na internete: https://msdn.microsoft.com/en-us/library/hcebdtae(v=vs.110).aspx. [14.4.2018]

Microsoft team, 2017 - XmlAttributeCollection::ItemOf Property (Int32), praktická časť [Online]
Dostupné na internete: https://msdn.microsoft.com/en-us/library/system.datetime(v=vs.110).aspx. [14.4.2018]

Donald Bell, 2004, The class diagram = An introduction to structure diagrams in UML 2, s. 28 [Online]
Dostupné na internete: https://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/index.html. [27.4.2018]

Manga, 2013, Recursion, s. 22 [Online]
Dostupné na internete: http://www.cplusplus.com/articles/D2N36Up4/. [04.3.2018]

Zoznam príloh

Diplomova-praca-Radoslav-Rajcan.pdf

Rajcan - program diplomovej prace.7z

Rajcan_program.exe

Racan navod.pdf