Міністерство освіти і науки, молоді та спорту України
Міжнародний економіко-гуманітарний університет
імені академіка Степана Дем'янчука

Кафедра математичного моделювання

ДИСЕРТАЦІЯ

на здобуття кваліфікації магістр з інформатики

на тему:

"Розробка модуля функцій парсингу XML і регулярних виразів як важливої технологічної складової АРМ в автосервісі"

(на прикладі ТОВ "Да Тех Рівне")

Робота допущена до захисту
протокол №_6_ від "14"лютого 2012р.
Завідувач кафедри математичного моделювання, доктор фіз.-мат. наук, професор Джунь Й.В.

_____
(підпис завідувача кафедри)

Студента магістратури факультету кібернетики групи 01КІН-М
спеціальності 8.080201
"Інформатика" нампряму підготовки
0802 "Прикладна математика" (8.04030201
"Інформатика в галузі знань 0403
"Системні науки та кібернетика"-перелік 2010р.)

Чернявський
Олексій Ігорович

_____
(підпис студента)

Науковий керівник:
кандидат технічних наук, доцент
Літнарович Р.М.

_____
(підпис керівника)

Рівне - 2012

Зміст

Вступ …..... 3
Розділ I. Огляд технології XML 5
1.1 Мова XML 5
1.2 Існуючі бібліотеки парсингу XML 37
Розділ II. Огляд технології регулярних виразів 39
2.1 Мова регулярних виразів (regular expressions) …..... 39
2.2 Існуючі бібліотеки парсингу регулярних виразів …..... 50
Розділ III. Програмна реалізація модуля парсингу XML …..... 52
3.1 Особливості реалізації 52
3.2 Програмна модель 52
3.3 Комп'ютерна реалізація 54
Розділ IV. Програмна реалізація модуля регулярних виразів 55
4.1 Програмна модель 55
4.2 Комп'ютерна реалізація 55
Розділ V. Реалізація повнофункціональної прикладної програми з використанням розроблених модулів парсингу XML і регулярних виразів .. 57
5.1 Опис програмного продукту 57
5.2 Комп'ютерна реалізація 58
Висновки 60
.
Перелік використаної літератури і джерел 63
Додатки 65

Вступ

Актуальність теми дипломної роботи. Мова XML (eXtensible Markup Language) привабила достатньо уваги серед розробників і користувачів середовища Інтернет, щоб задуматись над питанням використання її в якості основного інструменту для створення Web-програм, зберігання даних. Регулярні вирази в свою чергу є мовою пошуку текстової інформації.

XML і регулярні вирази відносяться до області зберігання, передачі, пошуку і обробки текстової інформації. Оскільки XML є найбільш популярним форматом зберігання даних, очевидним є необхідність програмного модуля функцій парсингу XML. Даний модуль може повторно використовуватись в зовнішніх програмних проектах. Регулярні вирази, з іншого боку, це популярна мова опису шаблонів пошуку текстової інформації.

XML і регулярні вирази широко використовуються в області зберігання, передачі і пошуку текстової інформації, що свідчить про необхідність та актуальність даних технологій для індустрії програмного забезпечення.

Мета роботи. Метою даної роботи є розробка алгоритму швидкого парсингу XML і регулярних виразів.

Для досягнення мети в дипломній роботі поставлені завдання:

- Оглянути існуючі програмні бібліотеки парсингу XML і регулярних виразів;
- Висвітлити теоретичні основи технологій XML і регулярних виразів;
- Розробити ефективні алгоритми парсингу XML і регулярних виразів.

Об’єкт дослідження. Досліджується алгоритм реалізації парсингу XML і регулярних виразів.

Предметом дослідження є стандарт XML, синтаксис мови XML і регулярних виразів.

Методологічною основою написання дипломної роботи є специфікація стандарту, спеціалізовна література і довідники.

Дипломна робота складається з чотирьох розділів. Перший розділ описує

основи мови XML та інших технологій тісно пов'язаних з даною мовою. Також представлений огляд існуючих реалізацій парсерів XML. Зокрема в розділі висвітлена слідуюча інформація: що таке XML, для чого потрібна нова мова розмітки, структура документа XML, правила створення XML-документа, конструкції мови, простір імен, розглядають існуючі бібліотеки парсингу, їхні властивості, переваги і недоліки.

Другий розділ описує основи мови регулярних виразів: історія виникнення, базові принципи, типи машин регулярних виразів, синтакси. Також представлений огляд існуючих реалізацій парсерів регулярних виразів (Regular Expressions).

Третій розділ включає програмну реалізацію розробленого мною модуля парсингу XML: описуються особливості даної реалізації, представлена високорівнева модель алгоритму, вихідний код на мові C, описується технологічна база розробки і тестування модуля.

Четвертий розділ включає програмну реалізацію розроблених мною модуля регулярних виразів: високорівнева модель алгоритму, вихідний код на мові C, технологічна база розробки і тестування модуля.

П'ятий розділ описує реалізацію повнофункціональної прикладної програми в якій в повній мірі застосовані модулі парсингу XML і регулярних виразів. Даний програмний продукт представляє собою реалізацію програми-електронного словника що працює з енциклопедичними і словниковими базами даних в форматі XML. Дана прикладна програма показує в повній мірі ефективність і правильність роботи реалізованих мною модулів парсингу XML і регулярних виразів.

Модуль парсингу XML пройшов тестування з XML-файлами розміром до 300 MB. Тести показали що модуль працює коректно, швидко і не споживає багато пам'яті. Відповідно до результатів тестів можна зробити висновок що розроблений програмний модуль може бути використаний в зовнішніх програмних проектах для парсингу XML і пошуку інформації з використанням регулярних виразів.

Розділ І. Огляд технології XML

1.1 Мова XML

1.1.1 Що таке XML?

XML (eXtensible Markup Language) - це розширювана мова розмітки тексту, запропонована W3C у 1996 році. Це мова, яка повною мірою визначає логічну структуру документа. Задача XML полягає в тому, щоб дані: тексти, зображення або інші частини Web-документа могли бути визначені і структуровані незалежно від платформи , що їх відтворює, постачальника і його програмного забезпечення, наприклад Web-браузерів.

При створенні документів із використанням XML, можуть бути використані індивідуальні елементи і структури для розмітки вмісту документів. Можливо визначити **DTD** (a Document Type Definition), тобто визначення типу документа. DTD визначає те, що можна назвати "граматикою" документа - це список різноманітних елементів і їхніх утворень для використання у визначених документах, у чомусь це нагадує використання **CSS**, тобто можна зробити посилання на DTD, що знаходиться або в мережі або написати його безпосередньо у документі.

Таким чином, вміст документа, його структура, типи використовуваних у ньому елементів і його видгляд визначаться окремо, тобто незалежно один від одного.

Чому XML?

Потрібно сказати, що XML корисний для автоматизованих програмних засобів, що шукають у Web. Недосконалість HTML призвела до того, що мережа перетворилася в мішанину тексту, повну різноманітних елементів і тегів, часто використовуваних, що називається Pro Forma і нічого не значущих.

XML має величезний потенціал для удосконалення гіпертекста.

Наприклад у HTML для створення зв'язку використовується елемент A, XML же дозволяє створити не просто посилання, а наприклад, двонаправлений зв'язок.

Перспектива XML полягає в тому, що він буде використовуватися для опису інших мов розмітки, наприклад, JavaScript, що використовується в HTML-документах.

XML розроблений для того, щоб спростити і полегшити використання SGML, при цьому зберігши його великі можливості по створенню, поширенню і публікації Web-документів мережі [5;118].

1.1.2 Для чого потрібна нова мова розмітки?

Мова розмітки документів - це набір спеціальних інструкцій, називаних тегами, призначених для формування в документах якоїсь структури і визначення відношень між різноманітними елементами цієї структури. Теги мови, або, як їх іноді називають, управляючі дескриптори, у таких документах якимось чином кодуються, виділяються щодо основного вмісту документа і служать у якості інструкцій для броузера.

Всю красу XML можна зрозуміти тільки при порівнянні його з HTML. Формалізована у RFC 1866 у 1995 році, HTML є найбільш популярною мовою розмітки у всьому світі. Термін “розмітка” стосовно до документа означає звичайно усе, що не відноситься до його інформаційного наповнення.

У ранню пору свого розвитку мова HTML підносилася як засіб масштабованого форматування документів, яку можна було б використовувати для обміну інформацією практично на будь-якій платформі. У основі HTML лежить украй проста ідея: ви визначаєте нескладну мову, що описує структуру документа, і чекаєте, коли компанії розроблять програмні засоби, спроможні подавати такі документи в різноманітних операційних середовищах з урахуванням обраних користувачем параметрів. За допомогою HTML можна було б створювати

матеріали, що допускають представлення в будь-якому візуальному або звуковому форматі.

Проте поступово ставало ясно, що ця ідея, незважаючи на свою простоту, йде врозріз з узвичаєною практикою видавничих систем. Традиційний механізм підготування публікацій передбачає, що графічні дизайнери і компоновщики повинні брати до уваги специфічні особливості презентаційного середовища, включаючи розмір листа, якість друку, палітру кольорів і т.п. Виявилося, що переключитися з такого методу на більш простий, при якому автор відповідає лише за вміст і логічну структуру документа, перекладаючи презентаційні обов'язки на користувацькі програми, досить важко.

У файлі HTML у його вихідному виді теги форматування перемішані зі звичайним текстом. Головною особливістю розмітки HTML є, звичайно, можливість вставки посилань на зовнішні документи або на внутрішні розділи того ж самого документа.

HTML процвітав не тільки як адаптована мова розмітки, але й у якості проміжного програмного забезпечення. Завдяки своїй дешевизні і поширеності браузери Web являють собою відмінних клієнтів; за посередництвом HTML вони можуть спілкуватися з найрізноманітнішими серверами.

Проте HTML стикнувся з певними труднощами. Його обмежені можливості форматування намагалися перебороти за допомогою CSS, ініціативи TrueDoc від Bitstream і звісно ж множини специфічних розширень для браузера; а його обмежені можливості в якості проміжного ПО - за допомогою Java, ActiveX і т.п. Проте все це не усуває його фундаментальні недоліки.

По суті, HTML - це технологія представлення інформації, вона описує те, як браузер повинен скомпонувати текст і графік на сторінці. У результаті «те, що ви бачите, - це усе, що ви одержуєте" [2;35]. Немає ніякого способу описати дані незалежно від відображення цих даних (за винятком надзвичайно слабкої системи

ключових слів у заголовку сторінки Web). "Байдужність" до структури документа призводить до того, що пошук або аналіз інформації усередині нього нічим не буде відрізнятися від роботи із суцільним, не розбитим на елементи текстовим файлом. Це головна причина, чому так важко знайти потрібну інформацію за допомогою механізму пошуку.

Клієнт не має ніяких менш прийнятних засобів витягу даних із сторінки Web для подальшої роботи з ними. Далі, на будь-який конкретній сторінці Web клієнт одержує тільки одне представлення конкретної множини даних.

Припустимо, що ви переглядаєте список аукціонів eBay, упорядкований по даті відкриття торгів. Якщо ви захочете глянути на той же список, але відсортований по даті закриття торгів, то вашому браузеру прийдеться посилати новий запит серверу. У свою чергу серверу прийдеться наново відправляти повну сторінку HTML із списком аукціонів. Такого роду маніпулювання даними веде до значного збільшення числа звертань до серверів Web і утруднює, таким чином, їх подальше масштабування.

Інша проблема з HTML у тому, що це «плоска" мова, тобто автори не можуть використовувати її для надання інформації про ієрархію даних. Далі, вона непослідовна і тому утрудняє розбір тексту програмним забезпеченням. Наприклад, хоча більшість відкриваючих тегів, (такі, як <B> або <H1>) має відповідні закриваючі теги, деякі (наприклад, <P>) їх не мають [2;48].

Істотним недоліком HTML можна назвати обмеженість набору його тегів. DTD-правила для HTML визначають фіксований набір дескрипторів і тому в розробника немає можливості вводити власні, спеціальні теги.

Простим рішенням для деяких із перерахованих проблем було би введення додаткових тегів HTML, таких, як <NAME>, <DATE> або <PRICE>. З їхньою допомогою клієнт міг би визначити, що собою являють дані, і відображати їх по-різному або експортувати по запиту користувача. Якщо ж ви вирішите не

чекати зміни стандарту, то майте на увазі, що ви створюєте щось своє, нестандартне і тим самим відмовляєтеся від однієї з головних переваг HTML.

Тому в 1996 році члени робочої групи Консорціуму World Wide Web (W3C) повернулися до розгляду стандартної узагальненої мови розмітки (Standard Generalized Markup Language, SGML), сильно спрощеним нащадком якого є HTML. Запропонована у 1974 році Чарльзом Голдфарбом, SGML являє собою метамову - систему для опису інших мов. Ця мова призначена для створення інших мов розмітки, вона визначає припустимий набір тегів, їхні атрибути і внутрішню структуру документа. При всіх своїх можливостях вона занадто складна для більшості браузеров Web: одна специфікація SGML займає понад 500 сторінок.

Спростивши SGML для використання в Web, група запропонувала XML (рекомендація W3C по статусу на лютий 1998 року). XML – підмножина SGML, причому любий дійсний документ XML є дійсним документом SGML. І, як і SGML, XML - це метамова, що визначає інші мови розмітки для специфічних цілей. Наприклад, мова синхронізованої інтеграції мультимедіа (Synchronized Multimedia Integration Language, SMIL) базується на XML.

Консорціум W3C, закликаючи до використання XML у Web, фактично пропонує кожному сконструювати особисту мову для своїх гіпертекстових документів, причому для різних документів це будуть різні мови.

XML дозволяє визначити формальний синтаксис мови, наприклад правила вкладення елементів. Семантику можна, звичайно, описувати на звичайній англійській мові.

XML використовується для розмітки стандартних документів багато в чому так само, як HTML. Проте XML перевершує його при роботі зі структурованими даними, такими, як результати запиту, метаінформація про вузол Web або елементи і типи схеми.

Документ XML виглядає багато в чому схожим на HTML. Він також складається з текстових фрагментів, анотованих вкладеними в кутові дужки тегами. Проте, на відміну від HTML, зміст тега залежить від регістра, а кожний відкриваючий тег повинен в усіх випадках мати парний закриваючий тег.

XML (*Extensible Markup Language*) - є та мова розмітки, що описує цілий клас об'єктів даних, названих XML- документами. Ця мова використовується в якості засобу для опису грамматики інших мов і контролю за правильністю впорядкування документів. XML не містить ніяких тегів, призначених для розмітки, а просто визначає порядок їх створення. Таким чином, якщо, наприклад, ми вважаємо, що для позначення елемента *rose* у документі необхідно використовувати тег *<flower>*, то XML дозволяє вільно використовувати обумовлений нами тег і ми можемо включати в документ фрагменти, подібні такому:

<flower>rose</flower>

Таким чином, у розробників з'являється унікальна можливість визначати власні команди, що дозволяють їм найбільш ефективно визначати дані, що зберігаються в документі. Автор документа створює його структуру, будує необхідні зв'язки між елементами, використовуючи ті команди, що задовольняють його вимогам і домагається такого типу розмітки, що необхідно йому для виконання операцій перегляду, пошуку, аналізу документа [1;67].

Ще однією з очевидних переваг XML є можливість використання її в якості універсальної мови запитів до сховищ інформації. Сьогодні в глибинах W3C знаходиться на розгляді робочий варіант стандарту XML-QL (або XQL), що, можливо, у майбутньому складе серйозну конкуренцію SQL. Крім того, XML-документи можуть виступати в якості унікального засобу збереження даних, що містить у собі одночасно засоби для розбору інформації й представлення її на стороні клієнта. У цій області одним із перспективних напрямків є інтеграція Java і XML - технологій, що дозволяє використовувати потужність обох технологій при

побудові машинно-незалежних додатків, що використовують, крім того, універсальний формат даних при обміні інформацією [4;82].

XML дозволяє також здійснювати контроль за коректністю даних, що зберігаються в документах, робити перевірки ієрархічних співвідношень усередині документа і встановлювати єдиний стандарт на структуру документів, умістом яких можуть бути самі різноманітні дані. Це означає, що його можна використовувати при побудові складних інформаційних систем, у котрих дуже важливим є питання обміну інформацією між різноманітними додатками, що працюють в одній системі. Створюючи структуру механізму обміну інформації на самому початку роботи над проектом, менеджер може позбути себе в майбутньому від багатьох проблем, пов'язаних із несумісністю використовуваних різноманітними компонентами системи форматів даних.

На основі XML уже сьогодні створені такі відомі спеціалізовані мови розмітки, як SMIL, CDF, MathML, XSL, і список робочих проектів нових мов, що знаходяться на розгляді W3C, постійно поповнюється [4;90].

1.1.3 Структура документа XML

Не обмежуючи автора яким-небудь фіксованим набором тегів, XML дозволяє йому вводити будь-які імена. Ця можливість є ключовою для активного маніпулювання даними.

Приклад для порівняння представлення списку імен і адрес на HTML і на XML.

От фрагмент HTML:

<H1>Editor Contacts</H1>
<H2>Ім'я: Джонатан Эйнджел</H2>
<P>Посада: старший редактор</P>

<P>Видання: Network Magazine</P>

<P>Вулиця і будинок: Гарісона, 600 </P>

<P>Місто: Сан-Франциско</P>

<P>Штат: Каліфорнія</P>

<P>Індекс: 94107</P>

<P>Електронна пошта:

jangel@mfi. com</P>

Теги розміщають дані на екрані, але нічого не повідомляють про їхню структуру.

У випадку XML той же самий фрагмент буде поданий у такий спосіб (і збережений у файлі EDITORS. XML).

```
<?xml version = '1.0' ?>
<?xml-stylesheet type='text/xsl' href='editors.xsl' ?>
<editor_contacts>
  <editor>
    <first_name>Jonatan</first_name>
    <last_name>Andjel</last_name>
    <title>chif editor</title>
    <publication>Network
    Magazine</publication>
    <address>
      <street>Garissona, 600 </street>
      <city>San-Francisko</city>
      <state>California</state>
      <zip>94107</zip>
    </address>
    <e_mail>jangel@mfi.com</e_mail>
  </editor>
```

</editor_contacts>

У XML теги не можуть накладатися, як у HTML, проте вони можуть бути вкладені один в одний. Насправді, вкладення навіть рекомендується як засіб створення ієрархії даних (підпорядковані або рівноправні відношення). Як очевидно з приведеного прикладу, такі елементи, як <first_name> і <e_mail>, містять дані, у той час як інші (<address>) присутні тільки з метою структурування.

Теги початку і кінця елемента є основними використовуваними в XML розмітками, але ними справа не вичерпується. Наприклад, елементам можуть бути привласнені атрибути. Ця можливість аналогічна наявній в HTML, де, наприклад, елементу <table> може бути привласнений атрибут align=”center”. У XML елемент може мати один або більше пов'язаних із ним атрибутів, причому при упорядкуванні документа ви можете видумати їх стільки, скільки побажнете, наприклад <publication topic=”networking” circulation=”controlled”>.

Документи XML можуть містити посилання на інші об'єкти. Посилання являють собою рядок, що починається з амперсанта і закінчується “;”. Ці посилання дозволяють, зокрема, вставити в документ спеціальні символи. Посилання XML на об'єкти надають набагато більше можливостей, тому що вони можуть посилатися на визначені автором розділи тексту в тому ж самому або в іншому документі.

Наприклад, посилання на об'єкти дозволяють застосувати об'єктно-орієнтований підхід при створенні журнальної статті:

<article>
&introduction;
&body;
&sidebar;
&conclusion;
&resources;

</article>

Найпростіший XML- документ може виглядати так, як це показано в Прикладі 1

```
<?xml version="1.0"?>
<list_of_items>
<item id="1"><first/>Перший</item>
<item id="2">Другий <sub_item>підпункт 1</sub_item></item>
<item id="3">Третій</item>
<item id="4"><last/>Останній</item>
</list_of_items>
```

У XML існують відкриваючі, закриваючі і порожні теги (у HTML поняття порожнього тэга теж існує, але спеціального його позначення не потрібно).

Тіло документа XML складається з елементів розмітки (markup) і безпосередньо вмісту документа - даних (content). XML - теги призначені для визначення елементів документа, їхніх атрибутів і інших конструкцій мови.

Любий XML-документ повинен завжди починатися з інструкції <? xml?>, усередині якої також можна задавати номер версії мови, номер кодової сторінки й інші параметри, необхідні програмі-аналізатору в процесі розбору документа [20;47].

1.1.4 Правила створення XML-документа

У загальному випадку XML- документи повинні задовольняти таким вимогам:

- У заголовку документа поміщається оголошення XML, у якому вказується мова розмітки документа, номер її версії і додаткова інформація

- Кожний відкриваючий тег, що визначає деяку область даних у документі обов'язково повинен мати відповідний закриваючий тег
- У XML враховується регістр символів
- Всі значення атрибутів, використовуваних у визначенні тегів, повинні бути взяті в лапки
- Вкладеність тегів у XML строго контролюється, тому необхідно стежити за порядком слідування відкриваючих і закриваючих тегів
- Вся інформація, що розташовується між початковим і кінцевими тегами, розглядається в XML як дані і тому враховуються всі символи форматування

Якщо XML- документ не порушує приведені правила, то він називається *формально-правильним* і всі аналізатори, призначені для розбору XML-документів, зможуть працювати з ним коректно.

З XML-документом пов'язані три рівні коректності:

- *Правильно побудований* XML-документ - це такий, у якому елементи правильно структуровані у вигляді дерева з коректно розставленими відкриваючих і закриваючих тегами.
- *Діючий* XML-документ правильно побудований і містить теги, що відповідають оголошенню типу документа. Він містить тільки елементи і значення атрибутів, що відповідають DTD. Хоча XML-документ може підготовлятися і читатися без DTD, DTD істотно для встановлення дієвості.
- *Синтаксично коректний* XML-документ знаходиться поза контролем XML. Розробник такого документа відповідає за його логічну структуризацію.

Проте крім перевірки на формальну відповідність граматиці мови, у документі можуть бути присутнім засоби контролю над вмістом документа, за дотриманням правил, що визначають необхідні співвідношення між елементами і

формуючою структурою документа. Наприклад, наступний текст, будучи цілком правильним XML- документом, буде абсолютно безглуздим:

<country><title>Russia</title><city><title>Novosibirsk</country></title></city>

Для того, щоб забезпечити перевірку коректності XML-документів, необхідно використовувати аналізатори, що роблять таку перевірку і називаються верифікованими.

На сьогоднішній день існує два способи контролю правильності XML-документа: DTD - визначення (Document Type Definition) і схеми даних (Semantic Schema). Визначення DTD- правил у XML не є необхідністю.

1.1.5 Конструкції мови

Вміст XML- документа являє собою набір елементів, секцій CDATA, директив аналізатора, коментарів, спецсимволів, текстових даних.

Елементи даних

Елемент - це структурна одиниця XML- документа. Вкладаючи слово rose у тэги <flower> </flower> , ми визначаємо непустий елемент, названий <flower>, вмістом якого є *rose*. У загальному випадку в якості вмісту елементів можуть виступати як простий текст, так і інші, вкладені, елементи документа, секції CDATA, інструкції з опрацювання, коментар, - тобто практично будь-які частини XML- документа.

Любий непустий елемент повинен складатися з початкового, кінцевого тегів і даних, між ними заключених. Наприклад, наступні фрагменти будуть елементами:

<flower>rose</flower>

<city>Novosibirsk</city>

,а ці - ні:

<rose>

<flower>

rose

Набором всіх елементів, що містяться в документі, задається його структура і визначаються всі ієрархічні співвідношення. Плоска модель даних перетворюється з використанням елементів у складну ієрархічну систему з множиною можливих зв'язків між елементами. Наприклад, у такому прикладі ми описуємо місце розташування Новосибірських університетів (вказуємо, що Новосибірський Університет розташований у місті Новосибірську, що, у свою чергу, знаходиться в Росії), використовуючи для цього вкладеність елементів XML :

```
<country id="Russia">
 <cities-list>
<city>
<title>Новосибірськ</title>
<state>Siberia</state>
<universities-list>
<university id="2">
<title>Новосибірський Державний Технічний Університет</title>
<noprivate/>
<address URL="www.nstu.ru"/>
<description>дуже гарний інститут</description>
</university>
<university id="2">
<title>Новосибірський Державний Університет</title>
<noprivate/>
<address URL="www.nsu.ru"/>
```

<description>теж не поганої</description>

</university>

</universities-list>

</city>

</cities-list>

</country>

Проводячи пошук у цьому документі, програма клієнта буде спиратися на інформацію, закладену в його структуру - використовуючи елементи документа. Тобто, якщо, наприклад, потрібно знайти потрібний університет у потрібному місті, використовуючи приведений фрагмент документа, то необхідно буде переглянути вміст конкретного елемента <university>, що знаходиться всередині конкретного елемента <city>. Пошук при цьому, природно, буде набагато більш ефективним, ніж знаходження потрібної послідовності по всьому документу.

У XML документі, як правило, визначається хоча б один елемент, названий кореневим і з нього програми-аналізатори починають перегляд документа. У приведеному прикладі цим елементом є <country>

У деяких випадках теги можуть змінювати й уточнювати семантику тих або інших фрагментів документа, по різному визначаючи ту саму інформацію, тим самим надаючи додатку-аналізатору цього документа зведення про контекст використання описуваних даних.

У випадку, якщо елемент не має вмісту, тобто немає даних, які він повинний визначати, він називається порожнім. Необхідно тільки пам'ятати, що початковий і кінцевий теги порожнього елемента ніби об'єднується в один, і треба обов'язково ставити косу риску перед кутовою закриваючою (наприклад**, <empty/>**;)

Коментар

Коментарями є будь-яка область даних, поміщена між послідовностями символів <! -- і --> Коментар пропускаються аналізатором і тому при розборі структури документа в якості значущої інформації не розглядається.

Атрибути

Якщо при визначенні елементів необхідно задати якісь параметри, що уточнюють його характеристики, то є можливість використовувати атрибути елемента. Атрибут - це пару "назва" = "значення", що треба задавати при визначенні елемента в початковому тегу. Приклад:

<color **RGB**="true">#ff08ff</color>

<color **RGB**="false">white</color>

або

<author id=0>Ivan Petrov</author>

Прикладом використання атрибутів у HTML є опис елемента :

Black

Спеціальні символи

Для того, щоб включити в документ символ, використовуваний для визначення яких-небудь конструкцій мови і не викликати при цьому помилок у процесі розбору такого документа, потрібно використовувати його спеціальний символьний або числовий ідентифікатор. Наприклад, < , > " або $(десяткова форма запису), &#x1a (шестнадцатеричная) і т.д.

Директиви аналізатора

Інструкції, призначені для аналізаторів мови, описуються в XML документі за допомогою спеціальних тегів - <*?* і *?* >;. Програма клієнта використовує ці інструкції для керування процесом розбору документа. Найбільше часто інструкції використовуються при визначенні типу документа

(наприклад, <? Xml version="1.0"? >) або створенні простору імен.

CDATA

Розділи символьных даних - це частини документа, аналізовані винятково як символьні дані, що не піддаються розборові, але, у відмінності від коментарів, використовуються застосуванням, виглядають так:

<![CDATA[

Цей текст, навіть якщо він містить інструкції JavaScript або елементи коду HTML, такі, як <B>жирний шрифт</B> або <H1>заголовок</H1>, не піддається граматичному розборові. Замість цього він відображається як їсти.

]]>

Таблиці стилів

Таблиці стилів узагалі, і каскадні таблиці стилів (Cascading Style Sheets, CSS) зокрема, дозволяють відокремити структуру й вміст документа від рівня представлення. У застосуванні до Web і HTML це означає, що мова HTML не містить у собі презентаційних можливостей: характер представлення формується окремими інструментальними засобами.

Технологія CSS помітно спрощує упорядкування і супровід документів. Створивши одну таблицю стилів, ви зможете використовувати її в сотнях документів. Вже в CSS1, першої версії CSS, були передбачені елементи уявлення, узагалі немислимі в HTML (наприклад, регулювання фізичних розмірів шрифтів).

XML/CSS як метод публікації можна зіставити з використанням програмного засобу опрацювання текстів, що підтримує стилі або макрокоманди: XML/CSS здійснює структурування документів, але виникаюча структура не має незалежну загальнодоступну семантику.

CSS можуть служити і для форматирования документів XML, але це не дуже удалий вибір. Головна перевага XML у тому, що вона подає формат

документа, для можливих маніпуляцій, у виді деревоподібної структури. На жаль, CSS не спроможні взаємодіяти з деревом і можуть тільки форматувати документи XML «як вони є". Ви можете вивести документ на екран у будь-якому форматі, але не можете здійснити якесь вибіркове представлення його даних без застосування мови сценаріїв.

Дані обмеження призвели до створення XSL. Найбільше важлива особливість XML і супутньої йому технології розширюваної мови таблиці стилів (Extensible Stylesheet Language, XSL) складається у відділенні форматирования від інформаційного наповнення [3;54].

Таблиці стилів XSL описують, як документи XML повинні перетворюватися в інші формати, такі, як HTML або RTF. Але таблиці стилів XML - це щось більше, ніж просто перетворювачі форматів; вони також надають механізм для маніпулювання даними. Наприклад, дані можна сортувати, робити по ним пошук, видаляти або додавати прямо з браузера.

XSL спроможна також здійснювати умовну трансформацію виведення в залежності від значень різноманітних елементів або атрибутів. Більш того, вона дозволяє запитувати дані з використанням множини різноманітних операторів шаблонів, символів підстановки, фільтрів, булевых операторів і виражень множини. XML і XSL ніяким чином не призначені для заміни SQL, до того ж навряд чи знайдеться багато бажаючих берегти свої бази даних безпосередньо у форматі XML. Проте XSL відчиняє можливість різноманітного пошуку по даним після їх завантаження в браузер. Вам ніколи вже не знадобиться використовувати для пошуку інформації примітивну вмонтовану команду браузера Find.

Значний потенціал XML у якості проміжного програмного забезпечення підкріплюється об'єктною моделлю документа (Document Object Model, DOM), версія 1.0 якиа була прийнята в якості рекомендації W3C у жовтні 1998 року.

Визначення Типу Документів (DTD)

Якщо теги й елементи XML використовуються винятково заради зручності на вашому власному вузлі Web, то не має ніякого значення, що ви даєте цим елементам і тегам імена, зміст яких відрізняється від стандартного і відомий тільки вам. Якщо ж, з іншого боку, ви хочете надавати дані зовнішньому світу й одержувати інформацію від партнерів по бизнесу, те ця обставина набуває величезне значення. Елементи й атрибути повинні вживатися вами точно так само, як і всіма іншими людьми, або принаймні ви повинні документувати те, що робите.

Для цього використовується визначення типів документів (Document Type Definition – DTD). Збережені на початку файла XML або назовні у виді файла *.DTD, ці визначення описують інформаційну структуру документа. DTD перераховують можливі імена елементів, визначають наявні атрибути для кожного типу елементів і описують сполучуваність одних елементів з іншими.

Кожний рядок у визначенні типу документа може містити декларацію типу елемента, іменувати елемент і визначати тип даних, що елемент може містити. Вона має такий вигляд

<!ELEMENT ім'я_елемента
(тип_даних)>

Наприклад, декларація визначає елемент з ім'ям publication, що містить символьні дані (тобто текст).

Декларація визначає елемент з ім'ям special_report, що містить піделементи article_1, article_2 і article_3 у зазначеному порядку, наприклад:

<special_report>
<article_1>XML:час прийшов</article_1>
<article_2>XML перевершує саме себе</article_2>
<article_3>Керування мережами і системами за допомогою

XML</article_3>

</special_report>

Після визначення елементів DTD можуть також визначати атрибути за допомогою команди !ATTLIST. Вона вказує елемент, іменує пов'язаний із ним атрибут і потім описує його припустимі значення.!ATTLIST дозволяє управляти атрибутами і багатьма іншими засобами: задавати значення по замовченню, знищувати пробіли і т.д. DTD можуть також містити декларації !ENTITY, де визначаються посилання на об'єкти, а також декларації !NOTATION, що вказують, що робити з двійковими файлами не у форматі XML.

Серйозне і дещо надзвичайне обмеження DTD полягає в тому, що вони не припускають типізації даних, тобто обмежують дані конкретним форматом (таким, як дата, ціле число або число з плаваючою точкою). DTD використовують інший синтаксис, ніж XML, і не дуже-то інтуїтивно зрозумілі. По названих причинах DTD будуть, напевно, замінені на більш потужні і прості у використанні схеми XML, робота над який ведеться в даний час.

Схеми даних

Схеми даних (Schemas) є альтернативним засобом створення правил побудови XML-документів. У порівнянні з DTD, схеми мають більш потужні засоби для визначення складних структур даних, забезпечують більш зрозумілий засіб опису грамматики мови, спроможні легко модернізуватися і розширюватися. Безумовною перевагою схем є також те, що вони дозволяють описувати правила для XML- документа засобами самого ж XML.

Проте це не означає, що схеми можуть цілком замінити DTD-описи - цей засіб визначення грамматики мови використовується зараз практичними всіма верифікуючими аналізаторами, XML і, більш того, самі схеми, як звичайні XML-елементи, теж описуються DTD. Але серйозні можливості нової мови і її відносної простоти, безумовно, дають підстави підтверджувати, що майбутній стандарт знайде широке застосування в якості зручного й ефективного засобу

перевірки коректності упорядкування документів.

В даний час у W3 консорціумі йде робота над першою специфікацією схем даних [3;47].

Консорціум World Wide Web (W3C) не збирається давати своє благословення ніяким додаткам XML (у термінології XML «додатком" називається опис галузевих термінів за допомогою деякого набору тегов XML). Іншими словами, конкретні вертикальні ринки повинні самостійно узгодити усередині галузі імена для своїх об'єктів. Щоб сприяти відкритості і передбачуваності при упорядкуванні схем XML у вертикальних галузях, Microsoft висунула ініціативу, названу BizTalk. За станом на серпень 1999 року цю ініціативу підтримало понад 25 компанії.

Зокрема BizTalk являє собою не що інше, як суспільний сервер Web, де публікуються всі схеми, запропоновані для використання в різноманітних галузях. Проте BizTalk не ставить своєю ціллю об'єднати всі галузі в спробі скласти одну гігантську схему для усіх використовуваних у якому б то ні було бізнесі даних.

BizTalk складається з трьох окремих елементів. По-перше, це сховище на сервері Web разом із рекомендаціями і тегами XML, використовуваними для додавання нових схем у сховище. По-друге, це розробка програмного продукту, серверу BizTalk. І по-третє, це будуть інтерактивні послуги на базі технології BizTalk [5;116].

Відмова від DTD

У тому, що стосується відображення галузевих даних, BizTalk виходить із безперспективності визначень типів документів (Document Type Definition, DTD). Замість того щоб заохочувати розробку XML DTD, прихильники BizTalk описують свої ієрархії даних за допомогою XML Schema (як передбачається, цей стандарт повинен прийти на заміну DTD).

В даний час W3C намагається узгодити різноманітні підходи до схем, але

запропонована версія стандарту - XML Schema - дає достатньо ясне уявлення про те, як буде виглядати заміна DTD. XML Schema має значно більш широкі можливості, ніж DTD, причому описи даються за допомогою безпосередньо XML, без створення ще однієї системи розмітки, як того потребує DTD.

DTD цілком достатньо для базового визначення документа, але вони мають декілька недоліків. По-перше, вони даються не на XML. З огляду на високий ступінь адаптованості і розширюваність XML, наявність ще одного формату для визначення документів є зайвою.

По-друге, елементи DTD усередині документа XML потребують повного визначення усього, що знаходиться усередині цих елементів. Іншими словами, ніякі піделементи «на перспективу" не припускаються - якщо такі будуть присутні в документі, те, по визначенню, документ не буде бути правильно складеним. Тим часом визначення XML Schema використовують модель відкритого інформаційного наповнення, у котрої невизначені елементи цілком припустимі.

По-третє, DTD обмежуються тільки граматикою і синтаксисом (тобто відношенням одного елемента до іншого), тоді як XML Schema може також задавати безпосередні обмеження на тип даних, що елемент може містити. Це значно спрощує реалізацію передачі даних додатка в порівнянні з більш традиційним текстовим документом. Наприклад, точно так само, як це роблять розроблювачі в мовах програмування, ви можете явно зазначити, що дана область збереження може містити тільки целочисленные дані. Нарешті, розроблювачам, що працюють у середовищах Wintel, буде дуже зручно те обставина, що XML Schema легко відображається на Microsoft Document Object Model. Таким чином, що працює з документами XML програма може запросити у відповідної схеми наявне визначення для елемента документа по своєму виборі. Код виглядає в такий спосіб:

var bookNode = doc. documentElement

Проте як же буде виглядати сам документ, що містить схему, зсередини?

По-перше, він буде містити теги XML, що повідомляють, що це схема, на зразок:

<Schema name="schema_sample_1">

... вміст схеми

</Schema>

Кожний пункт усередині схеми об'являється потім індивідуально, причому особливості кожного елемента розшифровуються за допомогою вкладених тегів, наприклад:

<ElementType name="PERSONA" content="textOnly" />

визначає елемент <Inventor> як здатний містити тільки текстові дані.

Подібні схеми можуть виявитися дуже важкі для читання, але вони легко піддаються розборові за допомогою інструментів XML. Іншими словами, вам не буде потрібно спеціальний редактор для роботи з документом XML Schema, як у випадку DTD.

У випадку правил на базі XML для форматів комерційних даних можна використовувати для відображення однієї схеми на другу вмонтовані функціональні можливості перетворення XML - розширювана мова таблиць стилів (Extensible Stylesheet Language, XSL).

На загальному рівні BizTalk Framework потребує, щоб видавці XML Schema притримувалися визначених рекомендацій. Так, тегам пропонується давати осмислені імена зі зрозумілим нескороченим написанням; ці імена повинні відповідати функціональному призначенню інформації, а не її місцю в приватній структурі даних (наприклад, «PartLocation" замість «PartFieldFourteen"), а інформація, що міститься в тегу, не повинна потребувати спеціального, відмінного від XML, декодування (наприклад, позначення валюти грошової суми повинно зберігатися у виді елемента XML, а не приєднуватися до суми як у «$30US").

Необхідними складовими BizTalk Framework є спеціальні, загальні для всіх галузей теги XML. Ці теги покликані звільнити розроблювачів від турбот із

приводу трьох найважливіших проблем взаємодії додатків. По-перше, від того, як дані передаються з одного додатка в інший; по-друге, від того, як «викликати" інший додаток - відправлення додатку даних у форматі XML повинно бути достатньо; по-третє, від того, у якому порядку повинні випливати елементи даних.

Один із тегів визначає код, за допомогою якого XML програма, що приймає дані у форматі, може встановити, що за схема BizTalk використовується. За допомогою інших тегів додаток може з'ясувати, хто є відправником даних, що відправник від нього хоче і кому дані повинні бути потім передані.

Для забезпечення сумісності документ BizTalk повинний починатися і, відповідно, закінчуватися тегом BizTalk, щоб одержувач знав, що він вступив у сектор BizTalk. Тег MsgType задає простір імен XML (вашу конкретну схему), що визначає припустимі елементи документа. Тому що ваша схема використовує формат даних XML, то тип даних, котрими ви наповняєте свій документ, буде легко встановити. Нарешті, ви можете також вставити блок маршрутних документів, наприклад:

```
<Route>
<From locationID=”11111111”
locationType="DUNS"
 process="" path="" handle="3"/>
<To locationID="222222222"
locationType="DUNS"
process="" path=""
handle="23CF15"/>
</Route>
```

BizTalk Framework нічого не говорить про те, які дані повинні входити в чотирьох атрибута тегів і, вона просто встановлює призначення кожного з них. Теги location ідентифікують мережний вузол (можливо, за допомогою URL), куди направляється документ, у той час як теги process і handle визначають додаток і

конкретний примірник (наприклад, номер транзакции), до якого відносяться дані. Тег path служить свого роду вмістилищем, де проміжні сервери можуть берегти відомості про дату й іншу інформацію, щоб маршрут (і за допомогою розширення зворотний маршрут) був видимий усім серверам уздовж шляху.

Бізнес-модель BIZTALK

Microsoft випустить серверний продукт для регулювання обміну BizTalk-сумісними повідомленнями XML між партнерами по бізнесу (бета-версія наприкінці осені 1999 року; готовий продукт повинний вийти після Windows 2000).

Як це виглядає

Інструкції в схемах складають набір правил, використовуючи який, програма-клієнт буде робити висновок про те, коректний документ або ні.

Схема даних, наприклад, може виглядати таким чином:

```
<schema id="OurSchema">
<elementType id="#title">
<string/>
</elementType>
<elementType id="photo">
<element type="#title">
<attribute name="src"/>
</elementType>
<elementType id="gallery">
<element type="#photo">
</elementType>
</schema>
```

Якщо ми включимо приведені правила всередину XML- документа, програма-клієнт зможе використовувати їх для перевірки. Тобто, вона тепер зможе

визначити, що правильним буде бути такий фрагмент:

<gallery>
<photo id="1"><title>My computer</title></photo>
<photo id="2"><title>My family</title></photo>
<photo id="3"><title>My dog</title></photo>
</gallery>

, а некоректним цей:

<gallery>
<photo id="1"/>
<photo index="2"><title>My family</title></photo>
<photo index="3"><title> My dog
</title><dogname>Sharik</dogname></photo>
</gallery>

Всі конструкції мови схем описуються правилами "XML DTD for XML-Data-Schema".

Область схеми даних

Створюючи схеми даних, ми визначаємо в документі спеціальний елемент, **<schema>**;, усередині якого містяться описи правил:

<schema id="OurSchema">
<!-- послідовність інструкцій -->
</schema>

Якщо використовувати окремий простір імен, то повний XML-документ, що містить у собі схему даних, буде виглядати в такий спосіб:

<?XML version='1.0' ?>
<?xml:namespace href="http://www.mrcpk.nstu.ru/schemas/" as="s"/?>
<s:schema id="OurSchema">
<!-- послідовність інструкцій -->

</s:schema>

Опис елементів

Для визначення класу елемента, до якого надалі будуть застосовуватися інструкції, що описують його вміст і структуру, призначений спеціальний елемент схеми **elementType**,

<elementType id="issue">

<descript>Елемент містить інформацію про черговий випуск часопису</descript>

</elementType>

Назва елемента задається атрибутом id. Всі подальші інструкції, що ставляться до описуваного класу, визначають його внутрішню структуру і набір припустимих даних, містяться всередині блока, заданого тегами <elementType> і </elementType>.

Як очевидно з приклада, при визначенні класу елемента, можна також використовувати коментар до нього, що заключають у тэги <**descript></descript>**

Атрибути елемента

Для того, щоб в описі елемента визначити його атрибути й описати властивості цих атрибутів ми повинні використовувати елемент **attribute**:

<elementType id="photo">

<attribute name="src"/>

<empty/>

</elementType>

У даному прикладі елементу <*photo*> визначається атрибут *src*, значенням якого може бути будь-яка послідовність дозволених символів:

<photo src="0"/>

<photo src="some text">

Подібно DTD, схеми даних дозволяють встановлювати обмеження на

значення і засіб використання атрибутів. Для цього в дескрипторі <attribute> необхідно використовувати параметр **atttype**.

Наприклад, якщо ми хочемо зазначити, що значення атрибута повинно використовуватися програмою-аналізатором як унікальний ідентифікатор, то нам необхідно створити таке правило:

<elementType id="bouquet">
<attribute name="id" atttype="ID">
</elementType>

Якщо ж потрібно задати список можливих значень атрибута, то приклад будет виглядати в такий спосіб:

<attribute name="flower" atttype="ENUMERATION" values="red green blue" default="red">

Модель вмісту елемента

Під моделлю вмісту в схемі даних розуміють опис усіх припустимих об'єктів XML- документа, використання котрих усередині даного елемента є коректним. Модель вмісту визначається інструкціями, розташованими всередині блока <elementType>.

<elementType id="article">
<attribute name="id" atttype="ID">
<element type="#title">
<string/>
</elementType>

Для цього правила коректним буде бути такий фрагмент документа:

<article id="0">
<title>Психи і маніяки в Інтернет</title>
</article>

Вкладені елементи описуються за допомогою інструкції **element**, у якій

параметром type указується клас об'єкта - посилання на його визначення:

<elementType id="article">

<element type="#title"/>

<element type="#author"/>

</elementType>

Якщо потрібно зазначити режим використання вкладеного елемента, то треба визначити параметр **occurs**:

<elementType id="article">

<element type="#title" occurs="REQUIRED"/>

<element type="#author" occurs="OPTIONAL"/>

<element type="#subject" occurs="ONEORMORE"/>

</elementType>

Можливі значення цього параметра такі:

REQUIRED - елемент повинний бути обов'язково визначений

OPTIONAL - використання елемента не є обов’язковим

ZEROORMORE - вкладений елемент може зустрічатися декілька разів або жодного разу

ONEORMORE - елемент повинний зустрічатися хоча б один раз

Приклади правильних XML-документів, що використовують приведену вище схему:

<article>

<title>Навіщо він потрібний, XML?</title>

<author>Іван Петров</author>

<subject>Що таке XML</subject>

<subject>потрібний чи він нам</subject>

</article>

або

<article>

<title>Навіщо він потрібний, XML?</title>

<subject>Що таке XML</subject>

</article>

Крім елементів, вмістом XML-документа можуть також бути звичайним текстом і областями CDATA. Для позначення типів вмісту поточного елемента в схемах використовуються такі інструкції:

<string/> - вказує на те, що вмістом елемента є тільки вільна текстова інформація (секція PCDATA) :

<elementType id="flower">

<string/>

</elementType>

<any/> - вказує на те, що вмістом елемента повинні бути тільки елементи, без тексту, незаключенного ні в один елемент:

<elementType id="issue">

<any/>

</elementType>

<mixed> - будь-яке сполучення елементів і тексту

<elementType id="contacts">

<mixed/>

</elementType>

<empty> - порожній елемент

Приклад:

<elementType id="title">

<string/>

</elementType>

<elementType id="chapter">

<string/>
</elementType>
<elementType id="chapters-list">
<any/>
</elementType>
<elementType id="content">
<element type="#chapters-list" occurs="OPTIONAL">
</elementType>
<elementType id="article">
<mixed><element type="#title"></mixed>
<element type="#content" occurs="OPTIONAL">
</elementType>

1.1.6 Простір імен

Розширювана мова розмітки (Extensible Markup Language, XML) дозволяє вам створювати свої власні теги, документувати їх за допомогою визначень типів документів (Document Type Definition, DTD) або схеми XML і потім без проблем обмінюватися даними з іншими джерелами. Все це добре, але може виявитися, що інші використовують ті ж самі, що і ви, імена для елементів і атрибутів, але при цьому спираються на інші DTD. Це прямий шлях до проблем.

Щоб уникнути подібних конфліктів W3C розробив концепцію просторів імен і ключового слова xmlns. Завдяки їм в одному документі можуть використовуватися імена елементів і атрибутів, що інакше вступили б у конфлікт один з одним. Тепер же вони різняться різними префіксами простору імен і визначаються по різноманітним DTD або схемах.

От, наприклад, фрагмент коду XML із використанням просторів імен:

<inventory xmlns:storea=

```
«http://www.knowknew.com/
books.dtd" xmlns:storeb=
«http://www.amazon.com/schema">
<storea:magazine>
<storea:title>Network
Magazine</storea:title>
</storea:magazine>
<storeb:magazine>
<storeb:magazine storeb:title=
«Data Communications">
</storeb:magazine>
</inventory>
```

У визначенні DTD магазина А назва книги є піделементом часопису. У схемі магазина Б назва є атрибутом часопису.

Завдяки розрізненню імен за допомогою різних префіксів просторів імен вони можуть застосовуватися разом. Місцезнаходження DTD і схеми вказується в даному прикладі за допомогою URL, але воно може також визначатися за допомогою Uniform Resource Name (URN, див. RFC 2141) або Uniform Resource Identifier (URI, див. RFC 2396).

Використання для опису даних (Intelligent Enterprise, August 03, 1999, Volume 2, Number 11)

Однією з особливостей XML, що привертає увагу промисловості, є можливість опису структур даних і даних, що зберігаються. З використанням XML можна визначити нові теги спеціально для опису еквівалента таблиць і стовпчиків (або сутностей і атрибутів) у структурі реляційної бази даних. Ще більш істотно те, що теги для набору стовпчиків або атрибутів можуть зв'язуватися з тегами для їхньої батьківської таблиці або сутності. Хоча теговая структура здається гарним механізмом для опису і розуміння структури бази

даних, спосіб організації даних потребує як ніколи раніше суворої дисципліни. XML не забороняє мати повторювані групи, жахливі структури даних і т.д.

OMG сформувала набір тегов, названий *XML Metadata Interchange* (XMI), із метою надання можливості опису в стандартних термінах структури даних про дані ("метаданих"). Цей стандарт буде корисний для обміну метаданими між CASE-засобами і для опису "репозиторія метаданих" у проектах сховищ даних. Рухаючись у тому ж напрямку, група компаній (щовключає, зокрема, IBM і Oracle) знаходиться в процесі визначення *Common Warehouse Metadata Interchange (CWMI)*, підмножини XMI для підтримки сховищ даних.

Це означає, що є два підходи до опису структури бази даних на XML:

По-перше, прикладну базу даних може описувати DTD XML-документа. У цьому випадку операційні дані бази даних можуть бути розміщені між наборами описаних тегів. Таке DTD може, наприклад, генеруватися одним CASE-засобом, а читатися іншим, забезпечуючи засіб передачі структури даних.

По-друге, можна розмістити самі визначення таблиці і стовпчиків між тегами XMI, визначеними на більш високому рівні абстракції. Цей підхід трохи більш хитрий, оскільки метамодель XMI *дуже* абстрактна, але використання метамоделі XMI дозволяє описувати набагато більше, чим таблиці і стовпчики.

Проте зауважимо, що проблема визначення репозиторія метаданих або обміну метаданими між CASE-засобами не пов'язаний із використанням XML або якогось іншої мови. Проблемою є структура і семантика бази даних. Важливе питання полягає не в тому, *як* буде представлятися універсальний репозиторій метаданих. (Можна легко уявити репозиторий у виді набору реляционных таблиць або діаграм сутність/зв'язок.) Питання полягає в тому, *що знаходиться в репозиторії* і *що це означає*? Які об'єкти є істотними і повинні бути описані? Це набагато складіша тема, і вона усе ще знаходиться в стадії обговорення. Наявність нової мови не вносить істотний внесок у це обговорення [4;320].

Насправді при наявності розуміння, що XML є гарним засобом для опису структури бази даних, найбільше очевидним висновком є те, що використання цієї мови накладає велику відповідальність на адміністраторів даних із приводу коректності визначення даних. XML не забезпечує таку коректність; XML усього лише реєструє будь-який проект даних, що надходить від розробника.

Поява XML підвищує важливість моделювання і проектування даних.

1.2 Існуючі бібліотеки парсингу XML

libxml2 - це Сі-бібліотека для обробки XML документів, розроблена для проекту Gnome. Код бібліотеки є переносимим (Linux, Unix, Windows, встроєні системи та ін.) і модульним; більшу частину модулів можна відкомпілювати. В Libxml2 реалізований ряд існуючих стандартів, що відносяться для мов розмітки, включаючи стандарт XML, простори імен XML, XML Base, Relax NG, RFC 2396, XPath, XPointer, HTML4, XInclude, CGML Catalogs і XML Catalogs. В більшості випадків специфікації в libxml реалізовані на відносно суворому рівні. В деякій мірі забезпечується підтримка слідуючих специфікацій (але не повна їх реалізація): DOM, FTP-клієнт, HTTP-клієнт і SAX2. В стадії розробки знаходиться підтримка W3C XML. Бібліотека включає xmllint - валідатор XML з інтерфейсом командного рядка [21].

Дана бібліотека характеризується високою якістю реалізації і підтримкою великої кількості стандартів що є частиною XML. До не доліків даної бібліотеки можна віднести лише її великий розмір що часто ставить під сумнів доцільність її використання для статичного лінкування з невеликими программами.

Expat - вільна потокоорієнтована бібліотека парсингу XML, написана на мові C. Як одна з найбільш доступних XML-парсерів, широко використовується в відкритому програмному забезпеченні. Серед проектів що її використовують є:

Apache, Mozilla Firefox, GNU Debugger, Perl, Python і PHP [22].

Дана бібліотека є менш потужною ніж libxml2 що стосується підтримки стандартів XML, але при цьому вона має набагато менший розмір і більшу швидкість роботи.

Розділ ІІ. Огляд технології регулярних виразів

2.1 Мова регулярних виразів (regular expressions)

Словосполучення “регулярні вирази", прямий переказ англійського “Regular expressions", звучить досить незграбно. Проте воно вже настільки прижилося, що потрапило в словники, тому доведеться використовувати саме його – за відсутністю кращого.

Регулярні вирази – це один із способів пошуку підрядків (відповідностей) в рядках. Здійснюється це за допомогою проглядання рядка у пошуках деякого шаблону. Загальновідомим прикладом можуть бути символи “*” і “?”, використовувані в командному рядку DOS. Перший з них замінює нуль або більш довільних символів, другий же – один довільний символ. Так, використання шаблону пошуку типу "text?.*" знайде файли textf.txt, text1.asp і інші аналогічні, але не знайде text.txt або text.htm. Якщо в DOS використання регулярних виразів було украй обмежене, то в інших місцях (тобто операційних системах і мовах програмування) вони майже досягли рівня високого мистецтва. «Майже" тому, що предмети високого мистецтва практично неможливо вживати в повсякденному житті. Складнішим прикладом застосування регулярних виразів може бути видалення сміття, внесеного Microsoft Word при збереженні документа у форматі HTML. Розробники Word вмудрилися все зробити по-своєму, в результаті чого HTML-документ деколи стає більше початкового DOC-файла за рахунок величезної кількості зрозумілих тільки IE5 тегів, вичистити які вручну немає ніякої можливості [6;73].

Особливо корисні регулярні вирази в програмах, написаних на скриптових (що інтерпретуються) мовах, наприклад, VBScript, JavaScript і Perl. Через те, що весь їх код інтерпретується, розбір текстових рядків і виразів виконується неприйнятно повільно. Застосування регулярних виразів дає значне збільшення продуктивності, оскільки бібліотеки, що інтерпретують регулярні вирази, зазвичай пишуться на низькорівневих високопродуктивних мовах (C, C++, Assembler).

Зазвичай за допомогою регулярних виразів виконуються три дії:

- Перевірка наявності відповідною шаблону підрядка.
- Пошук і видача користувачеві відповідних шаблону підрядків.
- Заміна відповідних шаблону підрядків.

Найбільший розвиток регулярні вирази отримали в Perl, де їх підтримка вбудована безпосередньо в інтерпретатор. У інших мовах, як правило, використовуються реалізовуючі регулярні вирази доповнення і модулі сторонніх розробників. У Vbscript і Jscript використовується об'єкт Regexp, в C/C++ можна використовувати бібліотеки Regex++ і PCRE (Perl Compatible Regular Expression), а також ряд менш відомих бібліотек, для Java існує цілий набір розширень – ORO, Regexp, Rex і gnu.regexp [6;42].

Осібно стоїть Microsoft Visual Studio.Net, але що вже удостоїлася маси публікацій і розмов. Реалізація регулярних виразів в .Net (Regex) повністю сумісна з Perl, і навіть декілька розширена. Все, що мовиться про Perl, цілком застосовно до .Net.

У складі ATL 7, що також входить в VC.Net, є шаблон XXX, який дозволяє вбудовувати регулярні вирази в C++-программи (незалежно від CLR). Він доступний в початкових текстах, тому його можна досить просто пристосувати до своїх потреб, тобто вбудувати в нього підтримку потрібної мови або додати необхідну функціональність. Цей шаблон видно, повинен виявитися найшвидшою реалізацією регулярних виразів, оскільки весь код підставляється компілятором як inline і, відповідно, компілятор може якісно оптимізувати код. Пряма робота з будь-якими видами рядків (вид рядка задається як параметр шаблону) також підвищує продуктивність.

Реалізації регулярних виразів розрізняються, проте в цілому вони дуже схожі один на одного, і, як правило, програміст, що одного дня освоїв використання регулярних виразів, надалі практично не зустрічає утруднень.

Синтаксис регулярних виразів до цих пір не повністю стандартизований. Існує POSIX-версия регулярних виразів, що повністю описує стандарт синтаксису для POSIX. Але версія Perl ширша і більш гнучка, чим і пояснюється її широка поширеність. Більшість бібліотек по синтаксису і використовуваним метасимволам сумісні з Perl, тому має сенс почати розбиратися з використанням регулярних виразів на прикладі саме цієї мови.

Регулярні вирази зародилися в 1940-х роках. «Батьками" регулярних виразів були двоє нейрофізіологів, Уоррен Мак-каллох і Уолтер Піттс. Вони займалися моделюванням роботи нервової системи на нейронному рівні. Але в реальність регулярні вирази утілилися через декілька років, коли математик Стівен Клін формально описав ці моделі за допомогою алгебри, яку він називав регулярними множинами (regular sets). Він розробив для регулярних множин простий математичний запис, який і називав регулярними виразами.

Лише у 1950-х і 60-х роках регулярні вирази стали предметом серйозного вивчення в кругах теоретичної математики. В цей час з'явилися декілька статей, присвячених регулярним виразам, написані Робертом Констейблом і Кеном Томпсоном. Робота Томпсона описує компілятор регулярних виразів генеруючий об'єктний код для IBM 7094. Це послужило відправною крапкою для створення qed – редактора, який був покладений в основу відомого редактора Unix ed. Регулярні вирази ed поступалися по своїх можливостях виразам qed, та зате вони вперше набули широкого поширення серед користувачів. Одна з команда ed виводила рядки редагованого файлу, в яких знаходився збіг для заданого регулярного вираження. Ця команда, «g/regular expression/p" (глобальний вивід по регулярному вираженню). Функція виявилася настільки корисною, що була перетворена в окрему утиліту. Так з'явилася програма grep [18;13].

На практиці застосовуються три типи машин регулярних виразів.

1. DFA (Deterministic Finite-state Automaton – детерміновані кінцеві автомати) машини працюють лінійно за часом, оскільки не потребують відкотів (і ніколи не

перевіряють один символ двічі). Вони можуть гарантовано знайти щонайдовший рядок з можливих. Проте, оскільки DFA містить тільки кінцевий стан, він не може знайти зразок із зворотним посиланням і, через відсутність конструкцій з явним розширенням, не ловить підвиразів. Вони використовуються, наприклад, в awk, egrep або lex.

2. Традиційні NFA-машины (Nondeterministic Finite-state Automaton – недетерміновані кінцеві автомати) використовують "жадібний" алгоритм відкоту, перевіряючи всі можливі розширення регулярного вираження в певному порядку і вибираючи перше відповідне значення. Оскільки традиційний NFA конструює певні розширення регулярного вираження для пошуку відповідностей, він може шукати підвирази і backreferences. Але із-за відкотів традиційний NFA може перевіряти одне і те ж місце декілька раз. В результаті працює він повільніше. Оскільки традиційний NFA приймає першу знайдену відповідність, він може і не знайти щонайдовше з входжень. Саме такі механізми регулярних виразів використовуються в Perl, Python, Emacs, Tcl і .Net.

3. POSIX NFA - машини схожі на традиційні NFA-машины, за винятком "терплячості" – вони продовжують пошук, поки не знайдуть щонайдовшу відповідність. Тому POSIX NFA-машины повільніше традиційних, і тому ж не можна змусити POSIX NFA віддати перевагу коротшій відповідності довгому. Одна з головних переваг POSIX NFA-машины – наявність стандартної реалізації [17].

Найчастіше програмісти використовують традиційні NFA-машины, оскільки вони точніші, ніж DFA або POSIX NFA. Хоча в якнайгіршому випадку час їх роботи росте по експоненті, використання зразків, що знижують рівень неоднозначності і що обмежують глибину пошуку з поверненням (backtracking), дозволяє управляти їх поведінкою, зменшуючи час пошуку до прийнятних значень.

Регулярні вирази, як вже було сказано вище, є рядком. Рядок завжди

починається з символу роздільника, за яким слідує безпосередньо регулярне вираження, потім ще один символ роздільника і потім необязятельний список модифікаторів. Як символ роздільника зазвичай використовується слеш ('/'). Таким чином в наступному регулярному вираженні: /\d{3}-\d{2}/m, символ '/' є роздільником, рядок '\d{3}-\d{2}' - безпосередньо регулярним вираженням, а символ 'm', розташований після другого роздільника, - це модифікатор.

Основою синтаксису регулярних виразів є той факт, що деякі символи, що зустрічаються в рядку розглядуються не як звичайні символи, а як що мають спеціальне значення (т.з. метасимволи). Саме це рішення дозволяє працювати всьому механізму регулярних виразів. Кожен метасимвол має свою власну роль в синтаксисі регулярних виразів. Далі ми розгледимо всі ці метасимволи.

Одним з найважливіших метасимволів є символ зворотного слеша ('\'). Якщо в рядку зустрічається цей символ, то парсер розглядує символ, безпосередньо наступний за ним двояко:

* якщо наступний символ в звичайному режимі має яке-небудь спеціальне значення, то він втрачає це своє спеціальне значення і розглядується як звичайний символ. Це абсолютно необхідно для того, щоб мати можливість вставляти в рядок спеціальні символи, як звичайні. Наприклад метасимвол '.', в звичайному режимі означає "будь-який одиничний символ", а '\.' означає просто крапку. Також можна позбавити спеціального значення і сам цей символ: '\\'.
* якщо наступний символ в звичайному режимі не має ніякого спеціального значення, то він може набути такого значення, будучи сполученим з символом '\'. Наприклад символ 'd' в звичайному режимі сприймається просто як буква, проте, будучи сполученою із зворотним слешем ('\d') стає метасимволом, що означає "будь-яка цифра".

Існує безліч символів, які утворюють метасимволи в парі із зворотним слешем. Як правило подібні пари використовуються для того, щоб показати, що на цьому місці в рядку повинен знаходитися символ, з кодом, який не має

відповідного йому зображення або ж символ, що належить якійсь певній групі символів. Нижче приведені деякі найбільш споживані:

Метасимволи для завдання символів, що не мають зображення

\n Символ переказу рядка (код 0x0a)

\r Символ повернення каретки (код 0x0d)

\t Символ табуляції (код 0x09)

\xhh Вставка символу з шістнадцятковим кодом 0xhh, наприклад \x41 вставить латинську букву 'a'

Метасимволи для завдання груп символів

\d Цифра (0-9)

\D Не цифра (будь-який символ окрім символів 0-9)

\s Порожній символ (зазвичай пропуск і символ табуляції)

\S Непорожній символ (все, окрім символів, визначуваних метасимволом \s)

\w "Словесний" символ (символ, який використовується в словах. Зазвичай всі букви, всі цифри і знак підкреслення ('_'))

\W Все, окрім символів, визначуваних метасимволом \w

З рештою наявних метасимволів, побудованих за цим принципом ви за бажання зможете ознайомитися в оригінальному описі.

Наведу декілька простих прикладів для того, щоб ви розуміли, про що йде мова. Відразу обмовлюся, що приклади декілька громіздкі і неприваблівіі, але лише тому, що я не почав використовувати в них метасимволи, про які ще не розповів і які зробили б їх набагато простіше.

/\d\d\d/ Будь-яке тризначне число ('123', '719', '001')

/\w\s\d\d/ Буква, пропуск (або табуляція) і двозначне число ('A 01', 'z 45', 'S 18')

/\d and \d/ Будь-який з наступних рядків: '1 and 2', '9 and 5', '3 and 4'.

Синтаксис регулярних виразів має засоби для визначення власних підмножин символів. Наприклад вам може знадобитися задати умову, що в цьому

місці рядка повинна знаходитися шістнадцяткова цифра або ще щось подібне. Для опису таких підмножин застосовуються символи квадратних дужок '[]'. Квадратні дужки, зустрінуті усередині регулярного вираження вважаються за один символ, який може набувати значень, перерахованих усередині цих дужок.

Є невелика тонкість в тому, як працюють метасимволи усередині квадратних дужок. Річ у тому, що в синтаксисі регулярних виразів існує ще безліч метасимволів, але практично всі вони працюють тільки поза секціями описів підмножин. Єдині метасимволи, які працюють усередині цих секцій це:
* Зворотний слеш ('\'). Тобто всі метасимволи з приведеної раніше таблиці працюватимуть.
* Мінус ('-'). Використовується для завдання набору символів з одного проміжку (наприклад всі цифри можуть бути задані як '0-9')
* Символ '^'. Якщо цей символ стоїть першим в секції завдання підмножини символів (і лише в цьому випадку!) він розглядуватиметься як символ заперечення. Т.ч. можна задати все сиволи, які не описані в даній секції.

Декілька прикладів, щоб було зрозуміле, як це працює:
[0-9a-fa-f] Цифра в шістнадцятковій системі числення
[\da-fa-f] Те ж саме, але з використанням метасимвола
[02468] Парна цифра
[^\d] Все, окрім цифр (аналог метасимвола \D)
[a^b] Будь-який з символів 'a', 'b', '^'. Відмітьте, що тут символ '^' не має якого-небудь спеціального значення, тому що стоїть не на першій позиції усередині квадратних дужок.

Тепер необхідно розгледіти ще декілька метасимволів. Як вже було сказано раніше, всі вони працюють тільки поза секціями опісчаній підмножин символів (поза квадратними дужками).

Символи '^' і '$'. Вони іспользутся для того, що того, щоб вказати парсеру регулярних виразів на те, щоб він звернув увагу на положення шуканого тексту в

рядку. Символ '^' указує, що шуканий текст повинен знаходитися на початку рядка, символ '$' навпаки, указує, що шуканий текст повинен знаходитися в кінці рядка. Подивимося, як це працює на прикладі:

Допустимо, у нас є текст:

12 aaa bbb

aaa 27 ccc

aaa aaa 45

І регулярне вираження для пошуку чисел в цьому тексті: /\d\d/m (не обертайте поки уваги на модифікатор). Пошук по цьому регулярному виразу поверне нам 3 значення: '12', '27', '45'. Тепер обмежимо пошук, вказавши, де саме усередині рядка повинен розташовуватися текст: /^\d\d/m . Тут результат буде тільки один - '12', тому що тільки це число розташовується на початку рядка. Аналогічно, регулярне вираження /\d\d$/m поверне результат '45'.

Символ точки '.'. Цей метасимвол указує, що на даному місці в рядку може знаходитися будь-який символ (за винятком символу переказу рядка). Дуже зручно використовувати його, якщо вам потрібно "пропустити" яку-небудь букву в слові при перевірці. Наприклад регулярне вираження /.bc/ знайде в тексті і 'abc' і 'Abc' і 'Zbc' і '5bc' [15;230].

Символ вертикальної межі '|'. Використовується для завдання списку альтернатив. Наприклад регулярне вираження:

/(червоне зелене)|яблоко/

Знайде в тексті всі словосполуки 'червоне яблуко' і 'зелене яблуко'. Про значення круглих дужок в цьому виразі див. далі.

Символи круглих дужок '(' і ')'. Ці символи дозволяють отримати з шуканого рядка додаткову інформацію. Зазвичай, якщо парсер регулярних виразів шукає в тексті інформацію по заданому вираженню і знаходить її - він просто повертає знайдений рядок. Проте, якщо він зустрічає усередині регулярного

вираження круглі дужки, то він розглядує вміст цих дужок як ще одне регулярне вираження, по якому необхідно провести пошук. Парсер рекурсивно викликає сам себе для пошуку по новому регулярному вираженню і використовує результати пошуку для подальшої обробки основного регулярного вираження. При цьому, якщо пошук хоч би по одному з внутрішніх регулярних виразів не увінчався успіхом - пошук по всьому регулярному вираженню вважається за безуспішний.

Щоб було зрозуміліше, розгледимо як приклад те, як працює парсер регулярних виразів в разі приведеного вище регулярного вираження про яблука: /(червоне зелене)|яблоко/ .

1. Парсер починає розбір регулярного вираження і зустрічає вираження в дужках: (червоне зелене)
2. Парсер викликає себе для пошуку по знайденому регулярному вираженню
3. Отримавши результати пошуку парсер підставляє по черзі кожен з отриманих результатів на місце вираження в дужках і дивиться, чи задовольняє знайдений результат всім умовам основного регулярного вираження (в даному випадку дивиться, чи є після знайденого слова слово "яблуко").
4. Якщо все гаразд - результати пошуку по кожному з наявних регулярних виразів для цього випадку повертаються, якщо немає - парсер просто переходить до наступного знайденого фрагмента. Результат пошуку внутрішнього регулярного вираження для цього фрагмента при цьому втрачається.

Тепер приклад, коли отримання результатів внутрішніх регулярних виразів може бути корисним. Припустимо, нам необхідно перевірити, чи є рядок семизначним телефонним номером з вказівкою коди міста і отримати з неї код міста і номер телефону:

/\((\d{3,5})\)\s+(\d{3}-\d{2}-\d{2})/

Деякі із застосованих тут метасимволів вам ще невідомі і розгледять трохи пізніше. Давайте рассотрім цей regexp докладніше.

Перша кругла дужка тут втрачає своє спеціальне значення і

розглядуватиметься як звичайний символ:

/\((\d{3,5})\)\s+(\d{3}-\d{2}-\d{2})/

Далі йде регулярне вираження в дужках (перевірка коди міста):

/\((\d{3,5})\)\s+(\d{3}-\d{2}-\d{2})/

Після цього йде закриваюча кругла дужка, яка також позбавлена свого спеціального значення із-за символу зворотного слеша, що стоїть перед нею:

/\((\d{3,5})\)\s+(\d{3}-\d{2}-\d{2})/

Потім йде пропуск порожнього місця:

/\((\d{3,5})\)\s+(\d{3}-\d{2}-\d{2})/

І ще одне регулярне вираження в дужках, яке перевіряє номер телефону:

/\((\d{3,5})\)\s+(\d{3}-\d{2}-\d{2})/

Як бачите, тут є 3 регулярних вирази - основне і два внутрішніх. При цьому основне вираження дозволяє нам перевірити, чи має рядок необхідний нам формат, а два внутрішніх - отримати відповідно код міста і номер телефону. Тобто одним регулярним вираженням ми можемо вирішити відразу декілька завдань!

Подивимося, як працює цей регулярний вираз. Хай у нас є рядок: "My phone is (095) 123-45-67". Результатами пошуку будуть 3 рядки: '(095) 123-45-67', '095' і '123-45-67'.

Нам залишилося розгледіти ще одну групу метасимволів, що визначають кількісні показники (т.з. quantifiers). Як ви вже могли відмітити раніше - дуже часто буває необхідно вказати, що якийсь символ повинен повторюватися певна кількість разів. Звичайно, можна просто вказати його необхідну кількість разів безпосередньо в рядку, але це, природно не вихід. Тим паче, що дуже часто зустрічаються ситуації, коли точна кількість символів невідома. Тому синтаксис регулярних виразів містить набір метасимволів, призначених саме для вирішення подібних завдань. Кожен з описаних нижче метасимволів визначає кількісну характеристику символу який знаходиться безпосередньо перед ним.

Зірочка '*'. Указує, що символ має бути повторений 0 або більше разів (тобто

символ може бути відсутнім або бути присутнім в будь-яких кількостях). Приклад: вираження /ab*c/ знайде рядки 'ac', 'abc', 'abbc' і так далі

Плюс '+'. Указує, що символ має бути повторений 1 або більше разів (тобто символ зобов'язаний бути присутнім і може бути присутнім в будь-яких кількостях). Приклад: вираження /ab+c/ знайде рядки 'abc', 'abbc', 'abbbc' і так далі, але не знайде рядок 'ac'.

Знак питання '?'. Указує, що символ миє як бути присутнім, так і немає, але при цьому не може повторюватися більше одного разу. Приклад: вираження /ab?c/ знайде рядки 'ac' і 'abc', але не знайде рядок 'abbc'.

Фігурні дужки '{' і '}'. Визначають кількісну характеристику символу. Усередині дужок через кому перераховуються мінімальна і максимальна кількість повторень символу. При цьому будь-який з параметрів може бути опущений, а крім того можна задати точну кількість повторень, вказавши тільки одне число.

Приклади:

* {2,4} - символ долен повторитися мінімум 2 рази, але не більше 4.
* {,5} - символ може бути відсутнім (оскільки не задана мінімальна кількість повторень), але якщо присутній, то не повинен повторюватися більше 5 разів.
* {3} - символ повинен повторюватися мінімум 3 рази, але може бути і більше.
* {4} - символ повинен повторюватися рівно 4 рази

Є ще одна тонкість у використанні метасимвола '?'. Подивитеся на таке вираження: /.+a/ . Очікується, що воно поверне нам частку тексту до першого входження символу 'a' в цей текст. Насправді воно працюватиме декілька не так, як очікується і результатом пошуку буде весь текст до останнього входження символу 'a'. Річ у тому, що за умовчанням кількісні метасимволи "скупилися" і намагаються захопити як можна більший шматок тексту. Якщо це не потрібно (як а нашому випадку), то необхідно "відучити" їх від жадності, вказавши знак '?' після кількісного метасимвола: /.+?a/ . Після цього вираження працюватиме оскільки треба [15;247].

Модифікатори регулярних виразів

Як вже було сказано раніше - механізм регулярних виразів дозволяє додавати модифікатори, що впливають на обробку регулярного вираження. Нижче розгледіли найбільш споживані, про останніх ви можете прочитати в оригінальному описі.

i - включення режиму case-insensitive, тобто великі і маленькі букви у вираженні не розрізняються.
m Указує на те, що текст, по якому ведеться пошук, повинен розглядуватися як що складається з декількох рядків. За умовчанням механізм регулярних виразів розглядує текст як один рядок незалежно від того, чим вона є насправді. Відповідно метасимволи '^' і '$' указують на початок і кінець всього тексту. Якщо ж цей модифікатор вказаний, то вони указуватимуть відповідно на початок і кінець кожного рядка тексту.

s - за умовчанням метасимвол '.' не включає в своє визначення символ переказу рядка. Тобто для багаторядкового тексту вираження /.+/ поверне тільки перший рядок, а не весь текст, як очікується. Вказівку цього модифікатора знімає це обмеження.

U - робить всі кількісні метасимволи "не жадібними" за умовчанням (про "жадність" кількісних метасимволів див. вище).

2.2 Існуючі бібліотеки парсингу регулярних виразів

Найбільш популярною для мови C бібліотекою регулярних виразів є PCRE - Perl Compatible Regular Expressions. Назва говорить про те що дана бібліотека підтримує інтерпритацію синтаксису регулярних виразів що існує в мові Perl. Оскільки мова Perl є найбільш популярною мовою для обробки текстової інформації і відповідно вона розвинула синтаксис регулярних виразів до більш потужної реалізації, тому логічним є використовувати синтаксис і

граматику регулярних виразів саме з цієї мови.

Синтаксис регулярних виразів PCRE є значно потужнішим і гнучкішим ніж стандартних регулярних виразів POSIX. Бібліотека сумісна з великою кількістю C компіляторів і операційних систем. Багато людей розробили бібліотеки на основі PCRE щоб зробити її сумісною з іншими мовами програмування.

Використання PCRE є дуже очевидним. Перед тим як використати регулярний вираз він повинен бути конвертований в двійковий формат для підвищення ефективності роботи. Для цього просто викликається функція pcre_compile() якій передається регулярний вираз як текстова строка. Функція вертає вказівник на двійковий формат. Отриманий двійковий регулярний вираз передаємо функції pcre_exec() разом з вказівником на масив символів в якому ми будемо проводити пошук і розміром масиву [23].

Бібліотека PCRE підтримує лише пошук по регулярному виразу і цю роботу вона виконує досить добре. З іншої сторони вона не реалізує ніяких функцій типу знайти-і-замінити, розділення строк та інші.

Розділ III. Програмна реалізація модуля парсингу XML

3.1 Особливості реалізації

При реалізації модуля парсингу XML були поставлені завдання досягти слідуючі характеристики кінцевої реалізації: швидка робота, толерантне використання пам'яті. Щоб отримати такі характеристики була застосована модель — одне джерело, багато користувачів. Дана модель в контексті парсингу XML представляє слідуючий алгоритм роботи. На початку роботи модуль зчитує в буфер весь файл, що містить XML. Після цього запускається процедура парсингу, якій передається даний буфер. Парсинг представляє собою процес пошуку і аналізу XML-тегів в буфері. Для кожного знайденого тега створюється структура в якій зберігається вказівник на місце в буфері де даний тег має початок. Дане рішення має ключове значення. Зберігання вказівника на дані, а не копіювання, робить алгоритм парсингу швидким і економічним в використанні пам'яті. Необхідно відзначити, що процедура виділення пам'яті, крім того що з'їдається сама пам'ять, створює додаткові затримки в роботі. Тобто маючи один буфер, що представлає собою одне джерело, ми створюємо багато структур, користувачі, що містять вказівники на місця в буфері де беруть початок відповідні теги.

3.2 Програмна модель

Основою алгоритму парсингу XML є рекурентна функція dexml(). Дана функція працює за слідуючим принципом. На вході вона отримує вказівник на буфер з даними XML. Запускається цикл який послідовно шукає теги на одному рівні ієрархії. На початку циклу перевіряється чи міститься на початку тексту тег XML. Якщо так, то даний текст обробляється як тег, якщо ж ні, то даний текст обробляється як кінцеві дані що містяться в тегу що знаходиться на рівень вище. В першому випадку з тексту виділяється самий зовнішній тег і його вміст. Інформація про виділений тег і його атрибути розміщується в новоствореній структурі DTag:

```
typedef struct _DAttr DAttr;
struct _DAttr {
    wchar_t* name;
        int nlen;
    wchar_t* value;
        int vlen;
        int pos;
    DAttr* next;
};


typedef struct _DTag DTag;
struct _DTag {
        int type;
        unsigned sid;
        wchar_t *name;
        int nlen;
        wchar_t *value;
        int vlen;
        DAttr* attributes;
        uint32_t pos;
        uint32_t len;

        DTag *parent;
        DTag *body;
        DTag *next;

        wchar_t *source_xml;
};

typedef enum _DTagType {
    DT_Tag,
    DT_Value,
    DT_Attribute,
        DT_Cdata
} DTagType;
```

Тип даних позначається як DT_Tag.

Дана структура відноситься до списочних структур. Це означає що вона вказує на слідуючу структуру яка є такого самого типу [11;266].

Після запису інформації про тег в дану структуру, функція dexml() викликає сама себе (рекурсія), передаючи вміст останнього тегу, а результат її роботи записується в елемент структури body [10;310]. Після того як рекурентний виклик вернув дані, функція dexml() шукає кінець виділеного тега і зміщує вказівник початку буфера на дані що починаються відразу за кінцевим символом

виділеного тега. Контроль вертається на початок цикла.

В кінці функція dexml() вертає вказівник на вже заповнену структуру Dtag.

Якщо на початку чикла функції dexml() виявлені прості дані, а не тег, то ці дані записуються в структуру DTag і тип даних позначається як DT_Value.

Крім самої функції парснгу XML даний модуль предоставляє зручні користувацькі функції перебору і пошуку тегів по структурі DTag, яка містить всі розпарсені дані XML. До таких функцій відносяться: tag_seek() і tag_rnext(). Також в модулі є зручна функція перевірки тега is_tag_name().

3.3 Комп’ютерна реалізація

Модуль парсингу XML був реалізований на мові програмування С в середовищі Linux з використанням компілятора GNU GCC, дебагера GDB, текстового редактора VIM. Модуль був протестований на XML-файлах розміром 300 MB. Тести показали коректність роботи модулі з вхідними даними різної складності. Крім того модуль визначається високою швидкість роботи і малим об'ємом пам'яті що споживається під час парсингу.

Комп’ютерна реалізація складається із слідуючих файлів:

- xml_utf8.c - вихідний файл програми на мові С що реалізує парсер XML. Версія з підтримкою Юнікод (Додаток А).
- xml_utf8.h - вихідний заголовочний файл мови С що містить структури XML і об’явлення функцій модуля XML. Версія з підтримкою Юнікод (Додаток А).
- Makefile - вхідний файл програми make для автоматичного компілювання і збирання модуля програми (Додаток А).
- config.mk - файл що включається в Makefile. Містить загальні параметри для компілятора (Додаток А).
- parse.c - програма для тестування XML парсера

Розділ IV. Програмна реалізація модуля регулярних виразів

4.1 Програмна модель

Алгоритм парсингу регулярного виразу є рекурсивним. Рекурсивність алгоритму походить від рекурсивної природи самих регулярних виразів, як і більшості інших коп'ютерних мов. Основою реалізації є функція regTest_r. Дана функція отримує 2 параметра: текст, в якому здійснюється пошук, і регулярний вираз, по якому текст сканується. Кожен символ регулярного виразу перевіряється на спеціальний симівол. Якщо зустрічається спеціальний символ що вказує на множину можливих послідовностей символів, функція рекурсивно викликає саму себе, зменшуючи діапазон тексту пошуку, доти поки якийсь з викликів не верне позитивний результат, що означає що дана частина тексту збігається з даним регулярним виразом, або текст пошуку не співпадатиме з даним піддіапазоном регулярного виразу. В першому випадку функція закінчує роботу і вертає позитивний результат, інакше вертає результат False.

Дана функція є дуже швидкою і її легко розширити для підтримки різних синтаксичних конструкцій регулярних виразів мови Perl.

4.2 Комп'ютерна реалізація

Модуль парсингу регулярних виразів був реалізований на мові програмування C в середовищі Linux з використанням компілятора GNU GCC, дебагера GDB, текстового редактора VIM.

Комп'ютерна реалізація складається із слідуючих файлів:

- regexp_utf8.c - вихідний файл програми на мові C що реалізує парсер регулярних виразів. Версія з підтримкою Юнікод (Додаток Б).
- regexp_utf8.h - вихідний заголовочний файл мови C що містить об'явлення функцій модуля регулярних виразів. Версія з підтримкою Юнікод (Додаток Б).
- Makefile - вхідний файл програми make для автоматичного компілювання і

збирання модуля програми (Додаток Б).

- config.mk - файл що включається в Makefile. Містить загальні параметри для компілятора (Додаток Б).
- demo.c - програма для тестування парсера регулярних виразів

Розділ V. Реалізація повнофункціональної прикладної програми з використанням розроблених модулів парсингу XML і регулярних виразів

5.1 Опис програмного продукту

Для демонстрації практичного застосування модулів парсингу XML і регулярних виразів, а також з метою прикладного застосування, був розроблений програмний продукт — клієнт баз даних XDClient. Дана програма розроблена для реалізації слідуючих задач: пошук інформації в базах даних що зберігаються в форматі XML.

Одним із форматів баз даних є XDXF. Це спеціалізований формат вищого рівня що базується на форматі XML. Формат XDXF розроблений спеціально для зберігання баз даних словників та енциклопедій.

Для XDClient версії 1.0 передбачена повна підтримка баз даних в формат XDXF, тобто програма зможе здійснювати повноцінний пошук і виведення інформації любих баз даних що зберігаються в форматі XDXF. На момент написання даної дисертації XDClient мав версію 0.8. Дана версія повністю вміє читати бази даних XDXF, здійснювати пошук і виведення інформації.

Особливістю даної програми є її здатність виконувати пошук потрібних даних дуже швидко. Для прикладу, при наявності 123 баз даних загальним об'ємом 570 MB, пошук даних триває в середньому 1с. Така безпрецедентна швидкість пошуку була досягнута завдяки двум ключовим технологіям: індексація і бінарний пошук [11;210]. Опис технологій індексації і бінарного пошуку, а також опис реалізації даних технологій в XDClient, виходить за рамки теми даної дисертації. Опис програмної реалізації XDClient також виходить за рамки теми даної дисертації (дивіться Додаток В).

5.2 Комп’ютерна реалізація

Програма XDClient була реалізована на мові програмування C в середовищі Linux з використанням компілятора GNU GCC, дебагера GDB, текстового редактора VIM. В подальшому вона була портована на Windows.

Комп’ютерна реалізація складається із слідуючих файлів:

- xdc.c - вихідний файл програми на мові C що містить точку входу і головні контрольні процедури (Додаток В).
- xdc.h - вихідний заголовочний файл мови C що містить об’явлення внутрішніх функцій і типів XDClient (Додаток В).
- xdxf.c - вихідний файл програми на мові C що містить модуль парсингу баз даних XDXF (Додаток В).
- xdxf.h - вихідний заголовочний файл мови C що містить об’явлення функцій і типів модуля парсингу XDXF (Додаток В).
- index.c - вихідний файл програми на мові C що містить модуль індексації даних (Додаток В).
- index.h - вихідний заголовочний файл мови C що містить об’явлення функцій і типів модуля індексації даних (Додаток В).
- utf8.c - вихідний файл програми на мові C що містить спеціалізовані функції для роботи з даними UNICODE в форматах UTF-8 і Wide Char (Додаток В).
- utf8.h - вихідний заголовочний файл мови C що містить об’явлення спеціалізованих функцій і типів (Додаток В).
- file.c - вихідний файл програми на мові C що містить спеціалізовані функції роботи з файлами і каталогами файлової системи (Додаток В).
- file.h - вихідний заголовочний файл мови C що містить об’явлення спеціалізованих функцій і типів (Додаток В).
- config.h – вихідний заголовочний файл мови C що містить об'явлення

глобальних константних даних і конфігураційних параметрів (Додаток В).

- Makefile - вхідний файл програми make для автоматичного компілювання і збирання модуля програми (Додаток В).
- config.mk - файл що включається в Makefile. Містить загальні параметри для компілятора (Додаток В).

Висновки

На даний час існують стабільні реалізації парсерів XML і регулярних виразів, сере яких виділяються libxml2 і Expa, PCRE. В Libxml2 реалізований ряд існуючих стандартів, що відносяться для мов розмітки. Дана бібліотека характеризується високою якістю реалізації і підтримкою великої кількості стандартів що є частиною XML. До не доліків даної бібліотеки можна віднести лише її великий розмір що часто ставить під сумнів доцільність її використання для статичного лінкування з невеликими программами. Expat є менш потужною ніж libxml2 що стосується підтримки стандартів XML, але при цьому вона має набагато менший розмір і більшу швидкість роботи.

Найбільш популярною для мови С бібліотекою регулярних виразів є PCRE - Perl Compatible Regular Expressions. Синтаксис регулярних виразів PCRE є значно потужнішим і гнучкішим ніж стандартних регулярних виразів POSIX. Бібліотека сумісна з великою кількістю С компіляторів і операційних систем. Багато людей розробили бібліотеки на основі PCRE щоб зробити її сумісною з іншими мовами програмування. Бібліотека PCRE підтримує лише пошук по регулярному виразу і цю роботу вона виконує досить добре.

Загальними перевагами даних бібліотек є їхня портабельність, тобто можливість скомпілювати для різних операційних систем, і відкритість коду.

XML корисний для автоматизованих програмних засобів, що шукають у Web. Недосконаліть HTML призвела до того, що мережа перетворилася в мішанину тексту, повну різноманітних елементів і тегів, часто використовуваних, що називається Pro Forma і нічого не значущих.

XML має величезний потенціал для удосконалення гіпертекста. Наприклад у HTML для створення зв'язку використовується елемент A, XML же дозволяє створити не просто посилання, а наприклад, двонаправлений зв'язок.

Перспектива XML полягає в тому, що він буде використовуватися для

опису інших мов розмітки, наприклад, JavaScript, що використовується в HTML-документах.

XML розроблений для того, щоб спростити і полегшити використання SGML, при цьому зберігши його великі можливості по створенню, поширенню і публікації Web-документів мережі. XML – підмножина SGML, причому любий дійсний документ XML є дійсним документом SGML. І, як і SGML, XML - це метамова, що визначає інші мови розмітки для специфічних цілей. Наприклад, мова синхронізованої інтеграції мультимедіа (Synchronized Multimedia Integration Language, SMIL) базується на XML.

XML дозволяє визначити формальний синтаксис мови, наприклад правила вкладення елементів. Семантику можна, звичайно, описувати на звичайній англійській мові.

XML використовується для розмітки стандартних документів багато в чому так само, як HTML. Проте XML перевершує його при роботі зі структурованими даними, такими, як результати запиту, метаінформація про вузол Web або елементи і типи схеми.

Регулярні вирази це основна мова пошуку текстової інформації в WEB. Особливо корисні регулярні вирази в програмах, написаних на скриптових (що інтерпретуються) мовах, наприклад, VBScript, JavaScript і Perl.

Найбільший розвиток регулярні вирази отримали в Perl, де їх підтримка вбудована безпосередньо в інтерпретатор. У інших мовах, як правило, використовуються реалізовуючі регулярні вирази доповнення і модулі сторонніх розробників.

Реалізації регулярних виразів розрізняються, проте в цілому вони дуже схожі один на одного, і, як правило, програміст, що одного дня освоїв використання регулярних виразів, надалі практично не зустрічає утруднень.

Синтаксис регулярних виразів до цих пір не повністю стандартизований.

Існує POSIX-версия регулярних виразів, що повністю описує стандарт синтаксису для POSIX. Але версія Perl ширша і більш гнучка, чим і пояснюється її широка поширеність.

Бібліотека парсингу XML і регулярних виразів була написана на мові C в середовищі Linux. Дана бібліотека складається з двох модулів. Перший модуль реалізує рекурсивну функцію парсингу XML. Дана функція вертає просту програмну структуру DTag що містить XML дані. В дальшому все що залишається програмісту зробити так це виконати обхід даної структури для розбору XML документа. Операція обходу структури DTag є набагато простішою в порівнянні з парсингом оригінального XML документу.

Модуль регулярного виразу також побудований з використанням рекурсивного алгоритму. Даний модуль реалізує функцію regTest_r, яка отримує два параметри: текст, в якому здійснюється пошук, і регулярний вираз, що також є текстовою стрічкою. Даний алгоритм вирізняється високою гнучкістю в сенсі можливості його розширення для підтримки ширшого набору мета-символів регулярних виразів Perl. Даний модуль був протестований з різними вхідними даними різної складності і довів правильність і ефективність свої роботи.

Перелік використаної літератури та джерел

1. Хантер Д., Рафтер Д. и др. XML. Базовый курс. - М.: Вильямс, 2009. - 1344 с. - ISBN 978-5-8459-1533-7

2. Сергеев А.П. HTML и XML. Профессиональная работа. - М.: «Диалектика", 2004.

3. Технічна рекомендація XML 1.0. Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation 04 February 2004

4. Erik T.R. Learning XML, 2nd Edition. O'Reilly. 2003. Pages: 416. ISBN: 0-596-00420-6.

5. Benoît M. XML by Example. 2000. Que. ISBN: 0-7897-2242-9

6. Фридл Д. Регулярные выражения. - СПб.: “Питер”, 2001. - 352 с. ISBN 5-318-00056-8

7. Смит Б. Методы и алгоритмы вычислений на строках. - М.: «Вильямс", 2006. - 496 с. ISBN 0-201-39839-7

8. Форта Б. Освой самостоятельно регулярные выражения. 10 минут на урок. - М.: «Вильямс", 2005. - 184 с. - ISBN 5-8459-0713-6

9. Керниган Б., Пайк Р. Практика программирования.: Пер. С англ. - М.: Вильямс, 2004 г. 288 с. ISBN 5-8459-0679-2 (рус.)

10. Керниган Б., Ритчи Д. Язык программирования Си. Пер. с англ., 3-е изд., испр. - СПб.: «Невский Диалект», 2001. - 352 с. ISBN 5-7940-0045-7

11. Хэзфилд Р., Кирби Л. И др. Искусство программирования на С. Фундаментальные алгоритмы, структуры данных и примеры приложений. Издательство “ДиаСофт”, 2001. ISBN: 966-7393-82-8.

12. Ахо А.В., Хопкрофт Д.Э., Ульман Д.Д. Структуры данных и алгоритмы.: Пер. с англ. : Уч. пос. - М.: Издательский дом «Вильямс», 2000. - 384 с. ISBN 5-8459-0122-7 (рус.)

13. Митчелл М., Оулдем Д., Самьюэл А. Программирование для Linux. Профессиональный подход.: Пер. с англ. - М.: Издательский дом «Вильямс», 2003. - 288 с. ISBN 5-8459-0243-6 (рус.)

14. Реймонд Э.С. Искусство программирования для UNIX.: Пер. с англ. - М.: Издательский дом «Вильямс», 2005. - 544 с. ISBN 5-8459-0791-8 (рус.)

15. Уолл Р., Кристиансен Т., Орвант Д. Программирование на Perl. - Пер. с англ. - СПб: Символ-Плюс, 2002. - 1152 с. ISBN 5-93286-020-0

16. "Regular Expressions", The Single UNIX Specification, Version 2, The Open Group, 1997

17. Cox R. Implementing Regular Expressions. www.swtch.com/~rsc/regexp.

18. Friedl J. Mastering Regular Expressions. O'Reilly. ISBN 0-596-00289-0.

19. Hopcroft J. E., Motwani R., Ullman J. D. (2000). Introduction to Automata Theory, Languages, and Computation (2nd ed.). Addison-Wesley. ISBN-10: 0201441241

20. Документація по XML на сайті IBM. http://www-128.ibm.com/developerworks/ru/xml/.

21. Офіційна документація по libxml2 на стайті проекту. http://xmlsoft.org/.

22. Офіційна документація по Expat на сайті проекту. http://expat.sourceforge.net/.

23. Офіційна документація по PCRE на сайті проекту. http://www.pcre.org/ .

24. Офіційна документація по GNU C Library на сайті проекту. http://www.gnu.org/software/libc.

Додатки

Додаток А. Програмна реалізація модуля парсингу XML

```
//xml_utf8.h
#ifndef XML_UTF8_H
#define XML_UTF8_H

#ifdef TARGET_FREEBSD
#include <sys/types.h>
#else
#include <stdint.h>
#endif

typedef struct _DAttr DAttr;
struct _DAttr {
    wchar_t* name;
      int nlen;
    wchar_t* value;
      int vlen;
      int pos;
    DAttr* next;
};


typedef struct _DTag DTag;
struct _DTag {
      int type;
      unsigned sid;
      wchar_t *name;
      int nlen;
      wchar_t *value;
      int vlen;
      DAttr* attributes;
      uint32_t pos;
      uint32_t len;

      DTag *parent;
      DTag *body;
      DTag *next;

      wchar_t *source_xml;
};
```

```
typedef enum _DTagType {
    DT_Tag,
    DT_Value,
    DT_Attribute,
      DT_Cdata
} DTagType;


DTag* dexml(wchar_t* ascii);

#define S_DEEP (1 << 0)
#define S_DOWN (1 << 1)
#define S_UP   (1 << 2)
#define S_ALL  (S_DEEP | S_DOWN | S_UP)
#define S_SKIP (1 << 3)


DTag* tag_seek(DTag *tag_vroot, DTag *tag_root, wchar_t *tag_name, unsigned
flags);
DTag* _tag_seek(DTag* tag_root, wchar_t *tag_name, unsigned flags);
DTag* tag_rnext(DTag *tag_vroot, DTag *tag_root, unsigned flags);
int is_tag_name(DTag *tag, wchar_t *str);
wchar_t* xml_wcsndup(wchar_t *wcs_xml, int len);
#endif
```

```
//xml_utf8.c
#define _GNU_SOURCE
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <sys/time.h>
#include "xml_utf8.h"
#include "utf8.h"
#include "config.h"

typedef unsigned long long timestamp_t;
static timestamp_t get_timestamp();

#ifndef BUFSZ
#define    BUFSZ 1024
```

```
#endif
DAttr* atexpose(wchar_t *wcs_xml, int len);
void attr_free(DAttr* attr);
void tag_free(DTag* tag);
int _isalnum(wchar_t c);
int islvalue(wchar_t c);
int _isalnum_full(wchar_t c);
int _isspace(wchar_t c);
int str_skip_space(wchar_t* str, int len);
wchar_t* space_clean(wchar_t* text);
wchar_t* str_skip_xml_header(wchar_t* wcs_xml);
wchar_t* str_skip_doctype(wchar_t* wcs_xml);
wchar_t* find_open_tag(wchar_t *wcs_xml, int len, wchar_t *tag);
DTag* _dexml(wchar_t* wcs_xml, int len, int *err_ret, wchar_t *xml_start, DTag
*parent, long *wc_mb_addr);
DTag* create_tag(void);
int get_comment_tag(wchar_t *wcs_xml, int i, int len);
int get_name_len(wchar_t *wcs_xml, int i, int len);
int _isdigit(wchar_t wc);
int str_skip_space2(wchar_t *wcs_xml, int i, int len);
int get_cdata(wchar_t *wcs_xml, int i, int len);

int ncall = 0;

#define CODE_NO_ERROR 0
#define CODE_INVALID_XML 1

DTag* dexml(wchar_t* wcs_xml)
{
  wchar_t *str;
  int i;
  int _err_ret;
  DTag *tag_root;
  wchar_t *xml_start;
  long *wc_mb_addr;
  long len;
  long mb_addr;
  int j;

  PRDEBUG("enter")

  xml_start = wcs_xml;

  len = wcslen(wcs_xml);
```

```
  wc_mb_addr = (long*) malloc(sizeof(long) * len);

  for (j=0, mb_addr=0; j < len; j++) {
      wc_mb_addr[j] = mb_addr;
      mb_addr += wc_mb_len(wcs_xml[j]);
  }

  if ((i = str_skip_space(wcs_xml, -1)) == -1) {
      free(wc_mb_addr);
      return NULL;
  }
  str = &wcs_xml[i];
  wcs_xml = str_skip_xml_header(str);
  if (!wcs_xml) {
      free(wc_mb_addr);
      return NULL;
  }
  if ((i = str_skip_space(wcs_xml, -1)) == -1) {
      free(wc_mb_addr);
      return NULL;
  }
  str = &wcs_xml[i];
  wcs_xml = str_skip_doctype(str);
  if (!wcs_xml) {
      free(wc_mb_addr);
      return NULL;
  }

  _err_ret = CODE_NO_ERROR;
  tag_root = _dexml(wcs_xml, wcslen(wcs_xml), &_err_ret, xml_start, NULL,
wc_mb_addr);

  if (tag_root)
      tag_root->source_xml = xml_start;

  free(wc_mb_addr);

  PRDEBUG("exit")
  return tag_root;

}

wchar_t buf[BUFSZ];
wchar_t buf2[BUFSZ];
```

```
wchar_t tag_name[BUFSZ];

DTag* _dexml(wchar_t* wcs_xml, int len, int *err_ret, wchar_t *xml_start, DTag
*parent, long *wc_mb_addr)
{
  int i, i1, i2;
  int _l, _dq, _sq;
  wchar_t *topen, *tclose;
  wchar_t *attribs;
  wchar_t *body;
  DTag *tag, *tag_root, *_tag;
  wchar_t *value;
  int bi;
  int j;
  int buf2_len;
  int nloop = 0;
  int ret;
  int _err_ret;
  int len2;
  int i_start;
  int i_tag_start;
  long mb_pos;
  wchar_t *_wcs_xml;
  long _wc_pos;
  timestamp_t t0;
  timestamp_t t1;

  ncall++;

  tag_root = NULL;
  tag = NULL;
  i=0;
  i_start = 0;
  while (1) {
      nloop++;
      i1 = str_skip_space2(wcs_xml, i_start, len);
      if (i1 == -1) {
        fwprintf(stderr, L"xml parser: error: %s:%d: invalid xml¥n", __FUNCTION__,
__LINE__);
        tag_free(tag_root);
        *err_ret = CODE_INVALID_XML;
        return NULL;
      } else if (i1 >= len) {
        return tag_root;
```

```
      }

      if (wcs_xml[i1] != L'<') {
        _tag = create_tag();
        if (tag)
            tag->next = _tag;
        tag = _tag;

        if (! tag_root)
            tag_root = tag;

        tag->parent = parent;

        for (i2=i1+1; i2<len && wcs_xml[i2] != L'<' && wcs_xml[i2] != L'¥0';
i2++);

        tag->value = &wcs_xml[i_start];
        tag->vlen = i2 - i_start;
        tag->type = DT_Value;
        i_start = i2;
      } else {
        i = i1+1;
        i1 = str_skip_space2(wcs_xml, i, len);
        if (i1 == -1 || i1 >= len-1) {
            fwprintf(stderr, L"xml parser: error: %s:%d: invalid xml¥n",
__FUNCTION__, __LINE__);
            tag_free(tag_root);
            *err_ret = CODE_INVALID_XML;
            return NULL;
        }

        len2 = get_name_len(wcs_xml, i1, len);
        if((len2 && wcs_xml[len2] == L'¥0') || i1+len2 >= len) {
            fwprintf(stderr, L"xml parser: error: %s:%d: invalid xml¥n",
__FUNCTION__, __LINE__);
            tag_free(tag_root);
            *err_ret = CODE_INVALID_XML;
            return NULL;
        } else if (!len2) {
            /* parse comment or CDATA */

            i1--;
            i2 = get_comment_tag(wcs_xml, i1, len);
            if (i2 != -1) {
```

```
            i_start = i2;
          } else {
            i2 = get_cdata(wcs_xml, i1, len);
            if (i2 != -1) {
                _tag = create_tag();
                if (tag)
                  tag->next = _tag;
                tag = _tag;

                if (! tag_root)
                  tag_root = tag;

                tag->parent = parent;

                tag->value = &wcs_xml[i1+9];
                tag->vlen = i2 - i1 - 12;
                tag->type = DT_Cdata;
                i_start = i2;
            }else {
                fwprintf(stderr, L"xml parser: error: %s:%d: invalid xml¥n",
__FUNCTION__, __LINE__);
                tag_free(tag_root);
                *err_ret = CODE_INVALID_XML;
                return NULL;
            }
          }

        } else {
            /* parse tag */

            i_tag_start = i-1;
            _tag = create_tag();
            if (tag)
              tag->next = _tag;
            tag = _tag;

            if (! tag_root)
              tag_root = tag;

            tag->parent = parent;

            tag->name = &wcs_xml[i1];
            tag->nlen = len2;
            tag->type = DT_Tag;
```

```
            _wc_pos = &wcs_xml[i_tag_start] - xml_start;
            tag->pos = wc_mb_addr[_wc_pos];

            i2 = i1 + len2;
            _sq = _dq = 0;
            _l = 1;
            i = i2;
            while (_l) {
              switch (wcs_xml[i]) {
                  case L'>':
                    _l = 0;
                    break;
                  case L'"':
                    if (!_sq) _dq = !_dq;
                    i++;
                    break;
                  case L'¥'':
                    if (!_dq) _sq = !_sq;
                    i++;
                    break;
                  case L'¥0':
                    fwprintf(stderr, L"xml parser: error: %s:%d: invalid xml¥n",
__FUNCTION__, __LINE__);
                    tag_free(tag_root);
                    *err_ret = CODE_INVALID_XML;
                    return NULL;
                    break;
                  default:
                    i++;
                    break;
              }
              if (i>= len) {
                  fwprintf(stderr, L"xml parser: error: %s:%d: invalid xml¥n",
__FUNCTION__, __LINE__);
                  tag_free(tag_root);
                  *err_ret = CODE_INVALID_XML;
                  return NULL;
              }
            }

            if (wcs_xml[i-1] != L'/') {
              attribs = &wcs_xml[i2];
              tag->attributes = atexpose(attribs, i-i2);
```

```
                i++;
                bi=0;
                buf2[bi++] = L'<';
                buf2[bi++] = L'/';
                for(j=0; j<tag->nlen && bi<BUFSZ-2; j++, bi++)
                    buf2[bi] = tag->name[j];
                buf2[bi++] = L'>';
                buf2[bi] = L'¥0';
                buf2_len = bi;

                for(j=0; j<tag->nlen && j<BUFSZ-1; j++)
                    tag_name[j] = tag->name[j];
                tag_name[j] = L'¥0';

                topen = find_open_tag(&wcs_xml[i], len - i, tag_name);
/*              tclose = wcsstr(&wcs_xml[i], buf2); */
                tclose = wcsstr_fast(&wcs_xml[i], buf2);
                while (tclose && topen && topen < tclose) {
                    topen = find_open_tag(topen+1, len - ((topen+1) - wcs_xml),
tag_name);
/*                  tclose = wcsstr(tclose+1, buf2); */
                    tclose = wcsstr_fast(tclose+1, buf2);
                    if (tclose && tclose + buf2_len > wcs_xml + len)
                      tclose = NULL;
                }
                if (!tclose) {
                    fwprintf(stderr, L"xml parser: error: %s:%d: invalid xml: tclose
not found¥n", __FUNCTION__, __LINE__);
                    tag_free(tag_root);
                    *err_ret = CODE_INVALID_XML;
                    return NULL;
                }
                _err_ret = CODE_NO_ERROR;
                tag->body = _dexml(&wcs_xml[i], tclose - &wcs_xml[i], &_err_ret,
xml_start, tag, wc_mb_addr);
                if (!tag->body && _err_ret == CODE_INVALID_XML) {
                    *err_ret = _err_ret;
                    tag_free(tag_root);
                    return NULL;
                }
                ncall--;
                j = 0;
                while(tclose[j] != L'>' && j<len) j++;
                j++;
```

```
            i_start = i_start + (&tclose[j] - &wcs_xml[i_start]);
            _wc_pos = &wcs_xml[i_start] - xml_start;
            tag->len = wc_mb_addr[_wc_pos] - tag->pos;

          } else {
            i--;
            attribs = &wcs_xml[i2];
            tag->attributes = atexpose(attribs, i-i2);
            i+=2;
            i_start = i;
            /*tag->len = i_start - i_tag_start; */
            _wc_pos = &wcs_xml[i_start] - xml_start;
            tag->len = wc_mb_addr[_wc_pos] - tag->pos;
          }
        }
      }
  }
  return tag_root;
}

wchar_t* space_clean(wchar_t* text)
{
  int len;
  wchar_t *ret, *_buf;
  int i, j, n;

  len = wcslen(text);
  _buf = (wchar_t*) malloc(sizeof(wchar_t) * (len+1));
  for (i=0, j=0; i<len; j++, i++) {
      if (_isspace(text[i])) {
        _buf[j] = L' ';
        while (_isspace(text[i])) i++;
        if (text[i] != L'¥0' && text[i] != L'¥n')
            _buf[++j] = text[i];
      } else if (text[i] == L'¥n') {
        i++;
        while(_isspace(text[i])) i++;
        if (text[i] != L'¥n' && text[i] != L'¥0') {
            if (j>0) {
              _buf[j] = L'¥n';
              j++;
            }
            _buf[j] = text[i];
        }
```

```
      } else {
        _buf[j] = text[i];
      }
  }
  _buf[j] = L'¥0';
  ret = wcsdup(_buf);
  free(_buf);
  return ret;
}

/* Attributes Expose */
DAttr* atexpose(wchar_t *wcs_xml, int len)
{
  int _l;
  int i, i1;
  int name_i, name_len;
  int value_i, value_len;
  DAttr* attr;

  if ((i = str_skip_space(wcs_xml, len)) == -1) return NULL;
  i1 = i;
  while (islvalue(wcs_xml[i]) && i < len) i++;
  if (wcs_xml[i] != L'=') {
      fwprintf(stderr, L"xml parser: error: %s:%d: invalid xml¥n", __FUNCTION__,
__LINE__);
      return NULL;
  }
  name_i = i1;
  name_len = i-i1;
  i++;
  if (i < len && wcs_xml[i] == L'"') {
      i1 = ++i;
      while (i < len && !(wcs_xml[i] == L'"' && wcs_xml[i-1] != L'¥¥') &&
wcs_xml[i] != L'¥0') i++;
      if (wcs_xml[i] == L'¥0' || i >= len) {
        fwprintf(stderr, L"xml parser: error: %s:%d: invalid xml¥n", __FUNCTION__,
__LINE__);
        return NULL;
      }
      value_i = i1;
      value_len = i-i1;
      i++;
  } else if (i < len && ! _isspace(wcs_xml[i])) {
      i1 = i;
```

```
      while (_isalnum_full(wcs_xml[i]) && i < len) i++;
      value_i = i1;
      value_len = i-i1;
  }
  attr = (DAttr*) malloc(sizeof(DAttr));
  attr->name = &wcs_xml[name_i];
  attr->nlen = name_len;
  attr->value = &wcs_xml[value_i];
  attr->vlen = value_len;
  attr->next = atexpose(&wcs_xml[i], len - i);
  return attr;
}

void tag_free(DTag* tag)
{
  DTag *_tag;
  DTag *tag_next;

  tag_next = tag;
  while(_tag = tag_next) {
      tag_next = _tag->next;
      if (_tag->source_xml) {
        free(_tag->source_xml);
      }
      attr_free(_tag->attributes);
      tag_free(_tag->body);
      free(_tag);
  }
}

void attr_free(DAttr* attr)
{
  DAttr *_attr;
  DAttr *attr_next;

  attr_next = attr;
  while(_attr = attr_next) {
      attr_next = _attr->next;
      free(_attr);
  }
}

int _isalnum(wchar_t c)
{
```

```
  if ((c >= L'0' && c <= L'9') || (c >= L'A' && c <= L'Z') || (c >= L'a' && c <=
L'z'))
      return 1;
  else
      return 0;
}

int get_name_len(wchar_t *wcs_xml, int i, int len)
{
  int name_len
 = 0;

  if (!wcs_xml || len <= 0 || i < 0 || i >= len) {
      fwprintf(stderr, L"xml parser: error: %s:%d: invalid arguments¥n",
__FUNCTION__, __LINE__);
      return 0;
  }

  if ((islvalue(wcs_xml[i]) || wcs_xml[i] == L':') && !_isdigit(wcs_xml[i])) {
      i++;
      name_len++;
      while ((islvalue(wcs_xml[i]) || wcs_xml[i] == L':' || wcs_xml[i] == L'.' ||
wcs_xml[i] == L'-') && i<len) {
        name_len++;
        i++;
      }
      return name_len;
  }
  return 0;
}

int _isdigit(wchar_t wc)
{
  if (wc >= L'0' && wc <= L'9')
      return 1;

  return 0;
}

int islvalue(wchar_t c)
{
  if ((c >= L'0' && c <= L'9') || (c >= L'A' && c <= L'Z') || (c >= L'a' && c <=
L'z') || c == L'_')
      return 1;
```

```
  else
      return 0;
}

int _isalnum_full(wchar_t c)
{
  if (c >= L'!' && c <= L'~' && c != L'<' && c != L'>')
      return 1;
  else
      return 0;
}

int _isspace(wchar_t c)
{
  if (c == L' ' || c == L'¥t')
      return 1;
  else
      return 0;
}

int str_skip_space2(wchar_t *wcs_xml, int i, int len)
{
  int _l;

  if (!wcs_xml|| i < 0) {
      fwprintf(stderr, L"xml parser: error: %s:%d: invalid arguments¥n",
__FUNCTION__, __LINE__);
      return -1;
  }

  if (len <= 0 || i >= len)
      return i;

  _l = 1;
  while(_l) {
      if (i == len)
        return i;
      switch (wcs_xml[i]) {
        case L' ':
        case L'¥t':
        case L'¥n':
            break;
        case L'¥0':
            return i;
```

```
            break;
        default:
            return i;
            break;
      }
      i++;
  }
}


int str_skip_space(wchar_t *wcs_xml, int len)
{
  int i;
  int _l;

  i = 0;
  _l = 1;
  while(_l) {
      if (i == len)
        return -1;
      switch (wcs_xml[i]) {
        case L' ':
        case L'¥t':
        case L'¥n':
            break;
        case L'¥0':
            return -1;
            break;
        default:
            return i;
            break;
      }
      i++;
  }
}

wchar_t* str_skip_xml_header(wchar_t *wcs_xml)
{
  int i, _q, _l;

  if (! (wcsncmp(wcs_xml, L"<?xml", 5) == 0))
      return wcs_xml;

  i = 5;
```

```
  _q = 0;
  _l = 1;
  while (_l) {
      switch (wcs_xml[i]) {
        case L'>':
            if (!_q) {
              if (wcs_xml[i-1] != L'?') {
                  fwprintf(stderr, L"xml parser: error: %s:%d: invalid xml
header¥n", __FUNCTION__, __LINE__);
                  return NULL;
              }
              _l = 0;
            }
            break;
        case L'"':
        case L'¥'':
            if (wcs_xml[i-1] != L'¥¥')
              _q = !_q;
            i++;
            break;
        case L'¥0':
            fwprintf(stderr, L"xml parser: error: %s:%d: invalid xml¥n",
__FUNCTION__, __LINE__);
            return NULL;
            break;
        default:
            i++;
            break;
      }
  }
  return &wcs_xml[i+1];

}

wchar_t* str_skip_doctype(wchar_t *wcs_xml)
{

  int i, _q, _l;

  if (! (wcsncmp(wcs_xml, L"<!DOCTYPE", 9) == 0))
      return wcs_xml;

  i = 5;
  _q = 0;
```

```
  _l = 1;
  while (_l) {
      switch (wcs_xml[i]) {
        case L'>':
            if (!_q) _l = 0;
            break;
        case L'"':
        case L'¥'':
            if (wcs_xml[i-1] != L'¥¥')
              _q = !_q;
            i++;
            break;
        case L'¥0':
            fwprintf(stderr, L"xml parser: error: %s:%d: invalid xml¥n",
__FUNCTION__, __LINE__);
            return NULL;
            break;
        default:
            i++;
            break;
      }
  }
  return &wcs_xml[i+1];

}

wchar_t* xml_wcsndup(wchar_t *wcs_xml, int len)
{
  wchar_t *wstr;
  int _len;

  if (!wcs_xml)
      return NULL;

  _len = wcslen(wcs_xml);
  if (_len < len)
      len = _len;

  wstr = (wchar_t*) malloc(sizeof(wchar_t) * len + 1);
  wcsncpy(wstr, wcs_xml, len);
  wstr[len] = L'¥0';

  return wstr;
}
```

```
wchar_t* find_open_tag(wchar_t *wcs_xml, int len, wchar_t *tag)
{
  wchar_t *topen;
  int patlen;
  int bi;
  int j;

  if (!wcs_xml|| !tag || len <= 0) {
      fwprintf(stderr, L"xml parser: error: %s:%d: invalid arguments¥n",
__FUNCTION__, __LINE__);
      return NULL;
  }

  bi=0;
  buf[bi++] = L'<';
  j=0;
  while (tag[j] != L'¥0' && bi<BUFSZ-1) {
      buf[bi] = tag[j];
      j++;
      bi++;
  }
  buf[bi] = L'¥0';

  patlen = bi;
  topen = wcs_xml;
  while (topen) {
/*    topen = wcsstr(topen, buf); */
      topen = wcsstr_fast(topen, buf);
      if (!topen) return NULL;
      if (topen + patlen > wcs_xml + len) return NULL;
      if (topen[patlen] == L' ' || topen[patlen] == L'>') {
        for (j=patlen; topen[j] != L'>'; j++) {
            if (topen + j >= wcs_xml + len)
              return NULL;
        }

        if (topen[j] == L'>' && topen[j-1] != L'/')
            return topen;
      }
      topen++;
  }
  return NULL;
}
```

```
DTag* create_tag(void)
{
  DTag *_tag;

  _tag = (DTag*) malloc(sizeof(DTag));
  if (!_tag) {
      perror("WTF");
      exit(1);
  }
  _tag->type = 0;
  _tag->sid = 0;
  _tag->name = NULL;
  _tag->nlen = 0;
  _tag->value = NULL;
  _tag->vlen = 0;
  _tag->attributes = NULL;
  _tag->pos = -1;
  _tag->len = -1;

  _tag->parent = NULL;
  _tag->body = NULL;
  _tag->next = NULL;

  _tag->source_xml = NULL;

  return _tag;
}

/* returns index of next char after comment closing '>' character */
int get_comment_tag(wchar_t *wcs_xml, int i, int len)
{
  int j;

  if (!wcs_xml || len <= 0 || i < 0 || i >= len) {
      fwprintf(stderr, L"xml parser: error: %s:%d: invalid arguments¥n",
__FUNCTION__, __LINE__);
      return -1;
  } else if (len-i < 7) {
      return -1;
  }

  if (wcs_xml[i] == L'<' && wcs_xml[i+1] == L'!' && wcs_xml[i+2] == L'-' &&
wcs_xml[i+3] == L'-') {
```

```
      for (j=i+6; j<len; j++) {
        if (wcs_xml[j-2] == L'-' && wcs_xml[j-1] == L'-' && wcs_xml[j] == L'>')
            return j+1;
      }
      fwprintf(stderr, L"xml parser: error: %s:%d: invalid xml¥n", __FUNCTION__,
__LINE__);
      return -1;
  }

  return -1;
}

/* returns index of next char after cdata closing ']]>' string */
int get_cdata(wchar_t *wcs_xml, int i, int len)
{
  int j;

  if (!wcs_xml || len <= 0 || i < 0 || i >= len) {
      fwprintf(stderr, L"xml parser: error: %s:%d: invalid arguments¥n",
__FUNCTION__, __LINE__);
      return -1;
  } else if (len-i < 12) {
      return -1;
  }

  if (wcs_xml[i] == L'<' && wcs_xml[i+1] == L'!' && wcs_xml[i+2] == L'[' &&
wcs_xml[i+3] == L'C' ¥
        && wcs_xml[i+4] == L'D' && wcs_xml[i+5] == L'A' && wcs_xml[i+6] == L'T' &&
wcs_xml[i+7] == L'A' ¥
        && wcs_xml[i+8] == L'[') {
      for (j=i+11; j<len; j++) {
        if (wcs_xml[j-2] == L']' && wcs_xml[j-1] == L']' && wcs_xml[j] == L'>')
            return j+1;
      }
      fwprintf(stderr, L"xml parser: error: %s:%d: invalid xml¥n", __FUNCTION__,
__LINE__);
      return -1;
  }

  return -1;
}

DTag* tag_rnext(DTag *tag_vroot, DTag *tag_root, unsigned flags)
{
```

```
  if (!tag_root)
      return NULL;

  if (flags & S_DEEP && tag_root->body)
      return tag_root->body;

  if (tag_root == tag_vroot)
      return NULL;

  if (flags & S_DOWN && tag_root->next)
      return tag_root->next;

  if (flags & S_UP) {
      while (tag_root = tag_root->parent) {
        if (tag_root == tag_vroot)
            return NULL;
        if (tag_root->next) {
            tag_root = tag_root->next;
            break;
        }
      }
  } else {
      return NULL;
  }
  return tag_root;
}

DTag* tag_seek(DTag *tag_vroot, DTag *tag_root, wchar_t *tag_name, unsigned flags)
{
  DTag *tag_ret;

  if (!tag_root)
      return NULL;

  if (flags & S_SKIP)
      tag_root = tag_rnext(tag_vroot, tag_root, flags);

  if (tag_root == tag_vroot)
      flags &= ~S_DOWN;

  while (1) {
      tag_ret = _tag_seek(tag_root, tag_name, flags);
      if (tag_ret || !(flags & S_UP))
        return tag_ret;
```

```
      if (tag_root == tag_vroot)
        return NULL;

      while (tag_root = tag_root->parent) {
        if (tag_root == tag_vroot)
            return NULL;
        if (tag_root->next) {
            tag_root = tag_root->next;
            break;
        }
      }
      if (!tag_root)
        return NULL;
  }

  return tag_ret;
}

DTag* _tag_seek(DTag* tag_root, wchar_t *tag_name, unsigned flags)
{
  DTag *tag;
  DTag *_tag;

  tag = tag_root;
  while (tag) {
      if (tag->type == DT_Tag) {
        if (wcsncmp(tag->name, tag_name, tag->nlen) == 0)
            break;

        if (flags & S_DEEP) {
            _tag = _tag_seek(tag->body, tag_name, flags | S_DOWN);
            if (_tag) {
              tag = _tag;
              break;
            }
        }
      }
      if (!(flags & S_DOWN)) {
        tag = NULL;
        break;
      }

      tag = tag->next;
```

```
  }
  return tag;
}

int is_tag_name(DTag *tag, wchar_t *str)
{
  if (!tag || !str)
      return 0;

  if (tag->type != DT_Tag)
      return 0;

  if (wcsncmp(tag->name, str, tag->nlen) == 0)
      return 1;

  return 0;
}

static timestamp_t get_timestamp()
{
  struct timeval now;
  gettimeofday(&now, NULL);
  return now.tv_usec + (timestamp_t)now.tv_sec * 1000000;
}
```

Додаток Б

Програмна реалізація модуля парсингу регулярних виразів

```
//regexp_utf8.h
#ifndef REGEXP_UTF8_H
#define REGEXP_UTF8_H

#include <wchar.h>

int regTest(wchar_t *, wchar_t *);
int regTest_r(wchar_t *, wchar_t *);

int is_regexp_start(wchar_t *str);

#endif
```

```
//regexp_utf8.c
/*
 *  Copyright (C) Oleksiy Chernyavskyy
 */




#include <stdio.h>
#include "regexp_utf8.h"



#define CHARS_NUM 95
#define true 1


/*
 * WRAPPER FUNCTION
 */
int regTest (wchar_t *data, wchar_t *re_text)
{

  if (*re_text == L'^') return regTest_r(data, re_text+1) ;
```

```
  while(true)
  {
      if (regTest_r(data, re_text)) return 1 ;
      if (*data && *data != L'¥n') data++; else return 0;
  }

}


/*
 * MAIN RECURSIVE FUNCTION
 *
 */
int regTest_r (wchar_t *data, wchar_t *re_text)
{


  wchar_t enum_chars[CHARS_NUM];
  int i, j;
  wchar_t c;


  if (*re_text == L'*' || *re_text == L'+' || *re_text == L'?') return 0 ;

  while (true)
  {
      if (! *re_text) return 1;


      /* ############################################################
       * CHECK . (ANY SYMBOL)
       */
      if (*re_text == L'.')
      {
        re_text++;
        if (*re_text == L'*')
        {
            re_text++;
            while(true)
            {
              if (regTest_r(data, re_text)) return 1;
              if (*data && *data != L'¥n') data++; else return 0;
            }
```

```
  }
  else if (*re_text == L'+')
  {
      re_text++;
      while(true)
      {
        if (*data && *data != L'¥n') data++ ; else return 0;
        if (regTest_r(data, re_text)) return 1 ;
      }
  }
  else if (*re_text == L'?')
  {
      re_text++;
      if (regTest_r(data, re_text)) return 1;
      if (*data && *data != L'¥n') data++ ; else return 0;
  }
  else
  {
      if (*data && *data != L'¥n') data++ ; else return 0;
  }
} //  if (*re_text == L'.')



/* ######################################################################
 *  CHECK [] BLOCK
 */
else if (*re_text == L'[')
{
  re_text++;

  /* Read [ ] block and init array */
  for (i = 0; i < CHARS_NUM; i++)
      enum_chars[i] = 0;

  while(*re_text != L']')
  {
      if(*(re_text+1) == L'-')
      {
        if (*re_text <= *(re_text+2))
        {
            i = *re_text;
            j = *(re_text + 2);
        }
```

```
      else {
          i = *(re_text + 2);
          j = *re_text;
      }
      for (; i <= j; i++)
      {
          enum_chars[i - 32] = 1;
      }
      re_text += 3;
    }
    else {
      enum_chars[*re_text - 32] = 1;
      re_text++;
    }
}
// [ ] array init END
// ----------------------------------

re_text++;

if (*re_text == L'*')
{
    re_text++;
    while(true)
    {
      if (regTest_r(data, re_text)) return 1 ;
      if ( *data && *data != L'¥n' && enum_chars[*data - 32]) data++ ;
      else return 0 ;
    }
}
else if (*re_text == L'+')
{
    re_text++;
    while(true)
    {
      if ( *data && *data != L'¥n' && enum_chars[*data - 32]) data++ ;
      else return 0 ;
      if (regTest_r(data, re_text)) return 1 ;
    }
}
else if (*re_text == L'?')
{
    re_text++;
    if (regTest_r(data, re_text)) return 1 ;
```

```
            if (*data && *data != L'¥n' && enum_chars[*data - 32]) data++ ;
            else return 0 ;
          }
          else
          {
            if (*data && *data != L'¥n' && enum_chars[*data - 32]) data++ ;
            else return 0 ;
          }
          // ------------------------------------------------------

        } /* if *re_text == L'[' */


        /*
###############################################################################
         * CHECK END OF LINE $
         */
        else if (*re_text == L'$')
        {
          if (! *data || *data == L'¥n') return 1 ; else return 0;
        }


        /*
###############################################################################
         * CHECK ORDINARY SYMBOL
         */
        else
        {
          if (*re_text == L'¥¥') re_text++ ;

          c = *re_text;
          re_text++;
          if (*re_text == L'*')
          {
            re_text++;
            while(true)
            {
              if (regTest_r(data, re_text)) return 1 ;
              if (c == *data) data++ ; else return 0 ;
            }
          }
          else if (*re_text == L'+')
          {
```

```
            re_text++;
            while(true)
            {
              if (c == *data) data++ ; else return 0 ;
              if (regTest_r(data, re_text)) return 1 ;
            }
        }
        else if (*re_text == L'?')
        {
            re_text++;
            if (regTest_r(data, re_text)) return 1 ;
            if (c == *data) data++ ; else return 0 ;
        }
        else
            if (c == *data) data++ ; else return 0 ;

      } // else



      // GUESS I dnt need this, I think so
      //    if (! *data || *data == L'¥n') return 0 ;

  } // while(true)

} // regTest_r()

int is_regexp_start(wchar_t *str)
{
  if (!str)
      return 0;
  if (str[0] == L'.' || str[0] == L'[')
      return 1;

  return 0;
}
```

Додаток В

Програмна реалізація прикладної програми з Розділу V

```
//dict.h
#ifndef WTF_H
#define WTF_H

#include "xdxf.h"
#include "index.h"

typedef struct _wtfdb_t wtfdb_t;
struct _wtfdb_t {
  char *path;
  wtfdb_t *next;
};

#define CMP_IGNORE_CASE     (0x1 << 0)
#define CMP_CHOP_SPACES     (0x1 << 1)
#define CMP_LEN_ARG1        (0x1 << 2)
#define CMP_IGNORE_NONALPHA (0x1 << 3)
#define CMP_CROP_SPC        (0x1 << 4)

int lookup_article(char* idx_file, wchar_t *key, wchar_t *dic_name, wchar_t *from,
wchar_t *to, int is_regexp);
long get_istart_pos(idxfile_t *idxfile, wchar_t *key);
void key_to_art(wtfidx_t *idx, wchar_t *key, wchar_t *dic_name, wchar_t *from,
wchar_t *to, int is_regexp);
void print_usage(void);
char *get_cfg_dir(void);
void print_line(int len);
void chomp_path(char* path);
void print_wtfdbs(wtfdb_t *db_ll);
void wtfdb_free_all(wtfdb_t *db_ll);
size_t nchars(const char *s);
int get_xdxf(char *path_in, wtfdb_t **db_ret);
int is_xdxf(char *path);
int is_idx(char *path);
int cmpwcs(wchar_t *user_key, wchar_t *dict_key, unsigned flags);
void print_tag_article(DTag* tag);
void print_tag_key(DTag *tag);
void print_tag_tr(DTag *tag);
void print_tag_opt(DTag *tag);
void print_tag_abr(DTag *tag);
```

```
void shrink_spaces(wchar_t *str);
void crop_non_alnum(wchar_t *str);
void convert_xml_escape(wchar_t *str);

#define ESC_RESET      "¥033[0m"

#define ESC_NBLACK     "¥033[0;30m"
#define ESC_NRED       "¥033[0;31m"
#define ESC_NGREEN     "¥033[0;32m"
#define ESC_NYELLOW    "¥033[0;33m"
#define ESC_NBLUE      "¥033[0;34m"
#define ESC_NMAGENTA   "¥033[0;35m"
#define ESC_NCYAN      "¥033[0;36m"
#define ESC_NWHITE     "¥033[0;37m"

#define ESC_BBLACK     "¥033[1;30m"
#define ESC_BRED       "¥033[1;31m"
#define ESC_BGREEN     "¥033[1;32m"
#define ESC_BYELLOW    "¥033[1;33m"
#define ESC_BBLUE      "¥033[1;34m"
#define ESC_BMAGENTA   "¥033[1;35m"
#define ESC_BCYAN      "¥033[1;36m"
#define ESC_BWHITE     "¥033[1;37m"

#define ESC_UBLACK     "¥033[4;30m"
#define ESC_URED       "¥033[4;31m"
#define ESC_UGREEN     "¥033[4;32m"
#define ESC_UYELLOW    "¥033[4;33m"
#define ESC_UBLUE      "¥033[4;34m"
#define ESC_UMAGENTA   "¥033[4;35m"
#define ESC_UCYAN      "¥033[4;36m"
#define ESC_UWHITE     "¥033[4;37m"

typedef enum {
  XDTAG_NULL,
  XDTAG_UNDEF,
  XDTAG_VALUE,

  XDTAG_XDXF,
  XDTAG_FULL_NAME,
  XDTAG_DESCRIPTION,
  XDTAG_ABBREVIATIONS,
  XDTAG_ABR_DEF,
  XDTAG_V,
```

```
  XDTAG_AR,
  XDTAG_I,
  XDTAG_B,
  XDTAG_K,
  XDTAG_OPT,
  XDTAG_DEF,
  XDTAG_POS,
  XDTAG_TENSE,
  XDTAG_TR,
  XDTAG_DTRN,
  XDTAG_KREF,
  XDTAG_RREF,
  XDTAG_IREF,
  XDTAG_ABR,
  XDTAG_C,
  XDTAG_EX,
  XDTAG_CO,
  XDTAG_SUP,
  XDTAG_SUB,

  XDTAG_NU,
  XDTAG_SU,
  XDTAG_TT,
  XDTAG_BIG,
  XDTAG_SMALL,
  XDTAG_BLOCKQUOTE,
} xdtag_id_t;


#endif
```

```
//dict.c
#include <stdio.h>
#include <stdlib.h>
#ifdef TARGET_FREEBSD
#include <sys/types.h>
#else
#include <stdint.h>
#include <wctype.h>
#endif
#include <sys/stat.h>
#include <dirent.h>
```

```
#include <string.h>
#include <unistd.h>
#include <wchar.h>
#include <locale.h>
#include "dict.h"
#include "xdxf.h"
#include "index.h"
#include "xml_utf8.h"
#include "regexp.h"
#include "file.h"
#include "config.h"

wchar_t sbuf[BUFSZ];
char scbuf[BUFSZ];

int main(int argc, char *argv[])
{
  int ret;
  wchar_t *from;
  wchar_t *to;
  int update_index;
  int i, j, h;
  wchar_t *key;
  wchar_t _key[BUFSZ];
  wtfidx_t *idx, *_idx;
  const char *arg_key;
  int arg_key_id;
  wchar_t **w_argv;
  int mb_len;
  wchar_t *dic_name;
  int is_regexp;

  setlocale(LC_CTYPE, "en_US.UTF-8");

  if (argc < 2) {
      print_usage();
      return 1;
  }

  w_argv = (wchar_t **) malloc(sizeof(wchar_t*) * argc);
  for (i=0; i<argc; i++) {
      mb_len = nchars(argv[i]);
      w_argv[i] = (wchar_t*) malloc(sizeof(wchar_t) * (mb_len+1));
      if (mbsrtowcs(w_argv[i], (const char**) &argv[i], mb_len, NULL) != -1) {
```

```
        w_argv[i][mb_len] = L'¥0';
      } else {
        fprintf(stderr, "wtf: error: convert mb string to wchar_t
string: %s:%d¥n", __FUNCTION__, __LINE__);
        for (j=0; j<=i; j++)
            free(w_argv[j]);
        free(w_argv);
        return 1;
      }
  }

  dic_name = NULL;
  ret = 0;
  from = NULL;
  to = NULL;
  is_regexp = 0;
  update_index = 0;
  key = NULL;
  arg_key_id = 0;

  for (i=1; i<argc; i++) {
      if (wcscmp(w_argv[i], L"-g") == 0) {
        update_index = 1;
      } else if (wcscmp(w_argv[i], L"-f") == 0) {
        i++;
        if (i < argc) {
            for (j=0; w_argv[i][j] != L'¥0'; j++);
            if (j>=2) {
              from = (wchar_t*) malloc(sizeof(wchar_t) *3);
              for (j=0; j<2; j++)
                  from[j] = towlower(w_argv[i][j]);
              from[j] = L'¥0';
            } else {
              fprintf(stderr, "wtf: error: invalid lang-from string¥n");
              return 1;
            }
        } else {
            print_usage();
            return 1;
        }
      } else if (wcscmp(w_argv[i], L"-t") == 0) {
        i++;
        if (i < argc) {
            for (j=0; w_argv[i][j] != L'¥0'; j++);
```

```
            if (j>=2) {
              to = (wchar_t*) malloc(sizeof(wchar_t) *3);
              for (j=0; j<2; j++)
                  to[j] = towlower(w_argv[i][j]);
              to[j] = L'¥0';
            } else {
              fprintf(stderr, "wtf: error: invalid lang-to string¥n");
              return 1;
            }
          } else {
              print_usage();
              return 1;
          }
        } else if (wcscmp(w_argv[i], L"-d") == 0) {
          i++;
          if (i < argc) {
              for (j=0; w_argv[i][j] != L'¥0'; j++);
              if (j>0) {
                dic_name = (wchar_t*) malloc(sizeof(wchar_t) *(j+1));
                for (h=0; h<j; h++)
                    dic_name[h] = towlower(w_argv[i][h]);
                dic_name[h] = L'¥0';
              } else {
                print_usage();
                return 1;
              }
          } else {
              print_usage();
              return 1;
          }
        } else if (wcscmp(w_argv[i], L"-k") == 0) {
          i++;
          if (i < argc) {
              arg_key_id = i;
          } else {
              print_usage();
              return 1;
          }
        } else {
          arg_key_id = i;
        }
    }

    if (update_index)
```

```
      ret = gen_index();

  if (arg_key_id) {
      if (idx = get_idx()) {
        key_to_art(idx, w_argv[arg_key_id], dic_name, from, to, is_regexp);
        idx_free_all(idx);
      }
  }

  return ret;
}

void key_to_art(wtfidx_t *idx, wchar_t *key, wchar_t *dic_name, wchar_t *from,
wchar_t *to, int is_regexp)
{
  wkey_t *key_cnt;
  wtfidx_t *_idx;
  int nhit = 0;

  if (!key || !idx)
      return;

  _idx = idx;
  while(_idx) {
      nhit += lookup_article(_idx->path, key, dic_name, from, to, is_regexp);
      _idx = _idx->next;
  }
}

int lookup_article(char* idx_file, wchar_t *key, wchar_t *dic_name, wchar_t *from,
wchar_t *to, int is_regexp)
{
  idxfile_t *idxfile;
  long fsize;
  long istart_pos;
  const int ret_error = -1;
  FILE *ifp;
  FILE *dfp;
  long _pos;
  int j;
  xdxf_idx_t xdxf_idx;
  DTag *tag_ar;
  int cmpret;
  wchar_t *_key;
```

```
  int match_from;
  int match_to;
  int match_dic;
  static int nhit_all = 0;
  int nhit = 0;

  if (!idx_file || ! key)
      return ret_error;

  idxfile = (idxfile_t*) malloc(sizeof(idxfile_t));
  init_idx_file(idxfile);

  idxfile->idx_path = strdup(idx_file);
  if (! read_idx_header(idxfile)) {
      free_idx_file(idxfile);
      return ret_error;
  }

  if (dic_name) {
      if (idxfile->db_name) {
        mbstowcs(sbuf, idxfile->db_name, BUFSZ);
        sbuf[BUFSZ-1] = L'¥0';
        if (cmpwcs(dic_name, sbuf, CMP_IGNORE_CASE | CMP_CHOP_SPACES |
CMP_LEN_ARG1) == 0) {
            match_dic = 1;
        } else {
            match_dic = 0;
        }
      } else {
        match_dic = 0;
      }
  } else {
      match_dic = 1;
  }

  if (from) {
      if (idxfile->from) {
        mbstowcs(sbuf, idxfile->from, BUFSZ);
        sbuf[BUFSZ-1] = L'¥0';
        if (cmpwcs(from, sbuf, CMP_IGNORE_CASE | CMP_CHOP_SPACES) == 0) {
            match_from = 1;
        } else {
            match_from = 0;
        }
```

```
      } else {
        match_from = 0;
      }
  } else {
      match_from = 1;
  }

  if (to) {
      if (idxfile->to) {
        mbstowcs(sbuf, idxfile->to, BUFSZ);
        sbuf[BUFSZ-1] = L'¥0';
        if (cmpwcs(to, sbuf, CMP_IGNORE_CASE | CMP_CHOP_SPACES) == 0) {
            match_to = 1;
        } else {
            match_to = 0;
        }
      } else {
        match_to = 0;
      }
  } else {
      match_to = 1;
  }


  if (!is_regexp || !is_regexp_start(key)) {
      if ((istart_pos = get_istart_pos(idxfile, key)) != -1 && match_dic &&
match_from && match_to) {

        if (! (ifp = fopen(idxfile->idx_path, "r"))) {
            free_idx_file(idxfile);
            return ret_error;
        }

        if (! (dfp = fopen(idxfile->db_path, "r"))) {
            free_idx_file(idxfile);
            fclose (ifp);
            return ret_error;
        }

        _pos = istart_pos;
        xdxf_idx.idxfile = idxfile;

        fseek(ifp, _pos, SEEK_SET);
```

```
        fread(&xdxf_idx.shift, sizeof(uint32_t), 1, ifp);
        fread(&xdxf_idx.size, sizeof(uint32_t), 1, ifp);
        fread(&xdxf_idx.ktag_index, ktag_index_len(), 1, ifp);
        fread(&xdxf_idx.mask, opt_mask_len(), 1, ifp);

        cmpret = 0;
        j = 1;
        while(cmpret == 0) {
            nhit_all++;
            nhit++;
            tag_ar = read_xdxf(xdxf_idx.idxfile->db_path, xdxf_idx.shift,
xdxf_idx.size, dfp);
            if (tag_ar) {
              if (nhit_all > 1)
                  fwprintf(stdout, L"¥n----------------------------------------
------¥n");
              fwprintf(stdout, L"%sDICT NAME: %s¥n%s", ESC_BWHITE, idxfile-
>db_name, ESC_RESET);
              fwprintf(stdout, L"%sFROM     : %s¥n%s", ESC_BWHITE, idxfile->from,
ESC_RESET);
              fwprintf(stdout, L"%sTO       : %s¥n¥n%s", ESC_BWHITE, idxfile->to,
ESC_RESET);
              print_tag_article(tag_ar);
              tag_free(tag_ar);
            } else {
              fprintf(stderr, "wtf: %s:%d: could not get article¥n", __FUNCTION__,
__LINE__);
            }

            _pos = istart_pos + j * (8 + idxfile->ktag_index_len + idxfile-
>mask_len);
            j++;

            if (_pos >= idxfile->fsize)
              break;

            fseek(ifp, _pos, SEEK_SET);
            fread(&xdxf_idx.shift, sizeof(uint32_t), 1, ifp);
            fread(&xdxf_idx.size, sizeof(uint32_t), 1, ifp);
            fread(&xdxf_idx.ktag_index, ktag_index_len(), 1, ifp);
            fread(&xdxf_idx.mask, opt_mask_len(), 1, ifp);

            _key = read_key_by_idx(&xdxf_idx, dfp);
            if (_key) {
```

```
              cmpret = cmpwcs(key, _key, CMP_IGNORE_CASE | CMP_CHOP_SPACES |
CMP_CROP_SPC | CMP_IGNORE_NONALPHA);
              free(_key);
            } else {
              break;
            }
        }

        fwprintf(stdout, L"¥n");
        fflush(stdout);
        fclose(ifp);
        fclose(dfp);
        free_idx_file(idxfile);
      } else {
        free_idx_file(idxfile);
        return 0;
      }
  } else {
      free_idx_file(idxfile);
  }
  return nhit;
}

void print_tag_article(DTag* tag)
{
  int vlen;
  DTag *_tag;
  xdtag_id_t prev_tag = XDTAG_NULL;

  _tag = tag;
  while (_tag) {
      if (_tag->type == DT_Tag) {
        if (! is_tag_name(_tag, L"abr") && prev_tag == XDTAG_ABR)
            fwprintf(stdout, L"¥n");
        else if (prev_tag == XDTAG_DTRN)
            fwprintf(stdout, L"¥n¥n");
        else if (prev_tag == XDTAG_EX)
            fwprintf(stdout, L"¥n");

        if (is_tag_name(_tag, L"ar")) {
            print_tag_article(_tag->body);
            prev_tag = XDTAG_AR;
        } else if (is_tag_name(_tag, L"k")) {
            print_tag_key(_tag->body);
```

```
            fwprintf(stdout, L"¥n¥n");
            prev_tag = XDTAG_K;
        } else if (is_tag_name(_tag, L"tr")) {
            print_tag_tr(_tag->body);
            fwprintf(stdout, L"¥n¥n");
            prev_tag = XDTAG_TR;
        } else if (is_tag_name(_tag, L"abr")) {
            print_tag_abr(_tag->body);
            prev_tag = XDTAG_ABR;
        } else if (is_tag_name(_tag, L"c")) {
            print_tag_article(_tag->body);
            prev_tag = XDTAG_C;
        } else if (is_tag_name(_tag, L"co")) {
            print_tag_article(_tag->body);
            prev_tag = XDTAG_CO;
        } else if (is_tag_name(_tag, L"dtrn")) {
            print_tag_article(_tag->body);
            prev_tag = XDTAG_DTRN;
        } else if (is_tag_name(_tag, L"ex")) {
            print_tag_article(_tag->body);
            prev_tag = XDTAG_EX;
        } else {
            print_tag_article(_tag->body);
            prev_tag = XDTAG_UNDEF;
        }
      } else if (_tag->type == DT_Value) {
        vlen =  BUFSZ > _tag->vlen ? _tag->vlen : BUFSZ-1;
        wcsncpy(sbuf, _tag->value, vlen);
        sbuf[vlen] = L'¥0';
        convert_xml_escape(sbuf);
        fwprintf(stdout, L"%S", sbuf);
        prev_tag = XDTAG_VALUE;
      }
      _tag = _tag->next;
  }
}

void print_tag_abr(DTag *tag)
{
  int vlen;
  DTag *_tag;

  _tag = tag;
  while (_tag) {
```

```
      fwprintf(stdout, L"%s", ESC_NGREEN);
      if (_tag->type == DT_Tag) {
        if (is_tag_name(_tag, L"c")) {
            print_tag_abr(_tag->body);
        } else if (is_tag_name(_tag, L"i")) {
            print_tag_abr(_tag->body);
        } else if (is_tag_name(_tag, L"co")) {
            print_tag_abr(_tag->body);
        } else {
            print_tag_abr(_tag->body);
        }
      } else if (_tag->type == DT_Value) {
        vlen =  BUFSZ > _tag->vlen ? _tag->vlen : BUFSZ-1;
        wcsncpy(sbuf, _tag->value, vlen);
        sbuf[vlen] = L'¥0';
        convert_xml_escape(sbuf);
        fwprintf(stdout, L"%S", sbuf);
      }
      _tag = _tag->next;
  }
  fwprintf(stdout, L"%s", ESC_RESET);
}

void print_tag_key(DTag *tag)
{
  int vlen;
  DTag *_tag;
  xdtag_id_t prev_tag = XDTAG_NULL;

  _tag = tag;
  while (_tag) {
      fwprintf(stdout, L"%s", ESC_BYELLOW);
      if (_tag->type == DT_Tag) {
        if (is_tag_name(_tag, L"opt")) {
            print_tag_opt(_tag->body);
            prev_tag = XDTAG_OPT;
        } else if (is_tag_name(_tag, L"nu")) {
            print_tag_key(_tag->body);
            prev_tag = XDTAG_NU;
        } else {
            print_tag_key(_tag->body);
            prev_tag = XDTAG_UNDEF;
        }
      } else if (_tag->type == DT_Value) {
```

```
        if (! (prev_tag == XDTAG_NU && _tag->vlen > 0 && _tag->value[0] == L'['))
{
            vlen =  BUFSZ > _tag->vlen ? _tag->vlen : BUFSZ-1;
            wcsncpy(sbuf, _tag->value,
 vlen);
            sbuf[vlen] = L'¥0';
            convert_xml_escape(sbuf);
            fwprintf(stdout, L"%S", sbuf);
        }
        prev_tag = XDTAG_VALUE;
      }
      _tag = _tag->next;
  }
  fwprintf(stdout, L"%s", ESC_RESET);
}

void print_tag_opt(DTag *tag)
{
  int vlen;
  DTag *_tag;

  _tag = tag;
  while (_tag) {
      if (_tag->type == DT_Tag) {
        print_tag_opt(_tag->body);
      } else if (_tag->type == DT_Value) {
        vlen =  BUFSZ > _tag->vlen ? _tag->vlen : BUFSZ-1;
        wcsncpy(sbuf, _tag->value, vlen);
        sbuf[vlen] = L'¥0';
        convert_xml_escape(sbuf);
        fwprintf(stdout, L"%S", sbuf);
      }
      _tag = _tag->next;
  }
}

void print_tag_tr(DTag *tag)
{
  int vlen;
  DTag *_tag;

  _tag = tag;
  fwprintf(stdout, L"%s", ESC_NCYAN);
  while (_tag) {
```

```
      if (_tag->type == DT_Tag) {
        print_tag_tr(_tag->body);
      } else if (_tag->type == DT_Value) {
        vlen =  BUFSZ > _tag->vlen ? _tag->vlen : BUFSZ-1;
        wcsncpy(sbuf, _tag->value, vlen);
        sbuf[vlen] = L'¥0';
        convert_xml_escape(sbuf);
        fwprintf(stdout, L"[%S]", sbuf);
      }
      _tag = _tag->next;
  }
  fwprintf(stdout, L"%s", ESC_RESET);
}

long get_istart_pos(idxfile_t *idxfile, wchar_t *key)
{
  long start, end;
  int cmpret;
  int j;
  FILE *ifp;
  FILE *dfp;
  long _pos, _ipos;
  uint32_t ar_shift;
  uint32_t ar_size;
  unsigned ktag_index;
  unsigned opt_mask;
  wchar_t *_key;
  xdxf_idx_t xdxf_idx;
  int i;
  const long ret_error = -1;

  if (!idxfile || !key)
      return ret_error;

  if (!idxfile->db_path || !idxfile->idx_path)
      return ret_error;

  if (! is_rfile(idxfile->db_path) || ! is_rfile(idxfile->idx_path))
      return ret_error;

  if (! (ifp = fopen(idxfile->idx_path, "r")))
      return ret_error;

  if (! (dfp = fopen(idxfile->db_path, "r"))) {
```

```
      fclose (ifp);
      return ret_error;
  }

  for (i=0; key[i] != L'¥0'; i++)
      key[i] = towlower(key[i]);

  xdxf_idx.idxfile = idxfile;
  xdxf_idx.shift = 0;
  xdxf_idx.size = 0;
  xdxf_idx.ktag_index = 0;
  xdxf_idx.mask = 0;
  _ipos = -1;
  start = 0;
  end = idxfile->idx_size / (8 + idxfile->ktag_index_len + idxfile->mask_len);

  for (j = (end - start)/2; ;) {
      _pos = idxfile->idx_start_pos + (start+j) * (8 + idxfile->ktag_index_len +
idxfile->mask_len);
      fseek(ifp, _pos, SEEK_SET);

      fread(&xdxf_idx.shift, sizeof(uint32_t), 1, ifp);
      fread(&xdxf_idx.size, sizeof(uint32_t), 1, ifp);
      fread(&xdxf_idx.ktag_index, ktag_index_len(), 1, ifp);
      fread(&xdxf_idx.mask, opt_mask_len(), 1, ifp);

      _key = read_key_by_idx(&xdxf_idx, dfp);

      if (_key) {
        cmpret = cmpwcs(key, _key, CMP_IGNORE_CASE | CMP_CHOP_SPACES |
CMP_CROP_SPC | CMP_IGNORE_NONALPHA);

        if (cmpret != 0 && j == 0)
            break;

        if (cmpret > 0) {
            if (_ipos != -1)
              break;
            start += j;
            j = (end - start)/2;
        } else if (cmpret < 0) {
            if (_ipos != -1)
              break;
            end -= j;
```

```
            j = (end - start)/2;
        }  else {
            _ipos = _pos;
            start+= j;
            j=0;
            if (start > 0)
              start--;
            else
              break;
        }
      }
      if (_key) {
        free(_key);
        _key = NULL;
      }
  }
  if (_key) {
      free(_key);
      _key = NULL;
  }

  fclose(ifp);
  fclose(dfp);

  return _ipos;
}

wchar_t sbuf1[BUFSZ];
wchar_t sbuf2[BUFSZ];
int cmpwcs(wchar_t *user_key, wchar_t *dict_key, unsigned flags)
{
  int cmpret;
  wchar_t wc;
  int i, h;

  if (!user_key || !dict_key)
      return -1;

  for (i=0; user_key[i] != L'¥0' && i < BUFSZ-1; i++) {
      wc = user_key[i];
      if (flags & CMP_IGNORE_CASE)
        wc = towlower(wc);
      sbuf1[i] = wc;
  }
```

```
  sbuf1[i] = L'¥0';

  for (i=0; dict_key[i] != L'¥0' && i < BUFSZ-1; i++) {
      wc = dict_key[i];
      if (flags & CMP_IGNORE_CASE)
        wc = towlower(wc);
      sbuf2[i] = wc;
  }
  sbuf2[i] = L'¥0';

  if (flags & CMP_CHOP_SPACES) {
      for (i=0; iswspace(sbuf1[i]) && sbuf1[i] != L'¥0'; i++) ;

      for (h=0; sbuf1[i] != L'¥0'; i++, h++)
        sbuf1[h] = sbuf1[i];
      sbuf1[h] = L'¥0';

      for(h--; iswspace(sbuf1[h]) && h >= 0; h--);
      sbuf1[++h] = L'¥0';


      for (i=0; iswspace(sbuf2[i]) && sbuf2[i] != L'¥0'; i++) ;

      for (h=0; sbuf2[i] != L'¥0'; i++, h++)
        sbuf2[h] = sbuf2[i];
      sbuf2[h] = L'¥0';

      for(h--; iswspace(sbuf2[h]) && h >= 0; h--);
      sbuf2[++h] = L'¥0';

  }

  if (flags & CMP_IGNORE_NONALPHA) {
      crop_non_alnum(sbuf1);
      crop_non_alnum(sbuf2);
  }

  if (flags & CMP_CROP_SPC) {
      shrink_spaces(sbuf1);
      shrink_spaces(sbuf2);
  }

/*  fwprintf(stdout, L"wtf: %s: user key = %S, dict key = %S¥n", __FUNCTION__,
sbuf1, sbuf2); */
```

```
  if (! (flags & CMP_LEN_ARG1)) {
      cmpret = wcscmp(sbuf1, sbuf2);
  } else {
      cmpret = wcsncmp(sbuf1, sbuf2, wcslen(sbuf1));
  }
  return cmpret;
}

void shrink_spaces(wchar_t *str)
{
  int i, h;

  for (i=0, h=0; str[h] != L'¥0'; h++) {
      if (iswspace(str[h])) {
        if (i>0)
            str[i++] = L' ';
        while(iswspace(str[++h]));
        h--;
      } else {
        str[i++] = str[h];
      }
  }
  str[i] = L'¥0';
}

void crop_non_alnum(wchar_t *str)
{
  int i, h;

  if (!str)
      return;

  for (i=0, h=0; str[h] != L'¥0'; h++) {
      if (iswalnum(str[h]) || iswspace(str[h])) {
        if (i<h)
            str[i] = str[h];
        i++;
      }
  }
  str[i] = L'¥0';
}

void print_usage(void)
```

```
{
  fwprintf(stderr, L"wtf: usage: wtf [-g] [[-d dic_name] [-f from_lang] [-t
to_lang] [-k] key_phrase]¥n");
}

char *get_cfg_dir(void)
{
  static char *cfg_dir = NULL;

  if (!cfg_dir) {
      if (getenv("HOME")) {
        snprintf(scbuf, BUFSZ, "%s/.wtf", getenv("HOME"));
        if (strlen(scbuf)) {
            cfg_dir = strdup(scbuf);
        }
      }
  }
  return cfg_dir;
}

void wtfdb_free_all(wtfdb_t *db)
{
  wtfdb_t *_db;

  while (db) {
      _db = db->next;
      free(db->path);
      free(db);
      db = _db;
  }
}

void print_wtfdbs(wtfdb_t *db_ll)
{
  int dbnum;
  wtfdb_t *db;
  char printstr[PSTRMAX];

  dbnum = 0;
  db = db_ll;
  while (db) {
      dbnum++;
      db = db->next;
  }
```

```
  snprintf(printstr, PSTRMAX, "wtf: found %d databases", dbnum);
  fprintf(stdout, "%s¥n", printstr);
  print_line(100);
  fprintf(stdout, "¥n");

  db = db_ll;
  while (db) {
      fprintf(stdout, "%s¥n", db->path);
      db = db->next;
  }
}

void print_line(int len)
{
  int i;
  for (i=0; i<len; i++)
      fprintf(stdout, "-");
  fprintf(stdout, "¥n");
}


int get_xdxf(char *path_in, wtfdb_t **db_ret)
{
  int ret;
  DIR *dirp;
  struct dirent *de;
  char path[MAXPATH];
  char fname[MAXPATH];
  wtfdb_t *db_last = NULL;
  wtfdb_t *_db_ret;
  wtfdb_t *_db;

  chomp_path(path_in);

  if (ret = is_xdxf(path_in))
      return ret;

  if (! is_dir(path_in)) {
      *db_ret = NULL;
      return 0;
  }

  dirp = opendir(path_in);
```

```
  if (! dirp) {
      fprintf(stderr, "wtf: warning: could not open %s¥n", path_in);
      *db_ret = NULL;
      return 0;
  }

  while((de = readdir(dirp)) != NULL) {
      if (de->d_name[0] == '.') {
#ifdef _DIRENT_HAVE_D_NAMLEN
        if (de->d_namlen == 1)
            continue;
        if (de->d_name[1] == '.' && de->d_namlen == 2)
            continue;
#else
        if (strlen(de->d_name) == 1)
            continue;
        if (de->d_name[1] == '.' && strlen(de->d_name) == 2)
            continue;
#endif
      }
#ifdef _DIRENT_HAVE_D_NAMLEN
      strncpy(fname, de->d_name, de->d_namlen);
      fname[de->d_namlen] = '¥0';
#else
      strcpy(fname, de->d_name);
#endif
      snprintf(path, MAXPATH, "%s/%s", path_in, fname);
      ret = get_xdxf(path, &_db_ret);
      switch(ret) {
        case 1:
        case 2:
            _db = (wtfdb_t*) malloc(sizeof(wtfdb_t));
            _db->path = strdup(path);
            _db->next = db_last;
            db_last = _db;
            break;
        case 3:
            if (db_last) {
              _db = db_last;
              while (_db->next)
                  _db = _db->next;
              _db->next = _db_ret;
            } else {
              db_last = _db_ret;
```

```
            }
            break;
        default:
            break;
      }
  }

  closedir(dirp);

  if (! db_last) {
      *db_ret = NULL;
      return 0;
  }

  *db_ret = db_last;
  return 3;
}

int is_xdxf(char *path)
{

  if (! is_rfile(path))
      return 0;

  if (regTest(path, ".*¥¥.xdxf$")) {
      if (regTest(path, "^dict¥¥.xdxf$"))
        return 1;
      else
        return 2;
  }
  return 0;
}

int is_idx(char *path)
{

  if (! is_rfile(path))
      return 0;

  if (regTest(path, ".*¥¥.idx$"))
      return 1;
  return 0;
}
```

```
void chomp_path(char* path)
{
  int len;
  int i;

  len = strlen(path);
  for (i=len-1; i>0; i--) {
      if (path[i] == '/')
        path[i] = '¥0';
      else
        break;
  }
}

size_t nchars(const char *s)
{
  size_t charlen, chars;
  mbstate_t mbs;

  chars = 0;
  memset(&mbs, 0, sizeof(mbs));
  while ((charlen = mbrlen(s, MB_CUR_MAX, &mbs)) != 0 &&
        charlen != (size_t)-1 && charlen != (size_t)-2) {
      s += charlen;
      chars++;
  }

  return (chars);
}

void convert_xml_escape(wchar_t *str)
{
  int i, h;

  if (!str)
      return;

  for (i=0, h=0; str[i] != L'¥0' && h <BUFSZ-1; i++, h++) {
      if (str[i] == L'&')
        if (wcsncmp(&str[i], L"&lt;", 4) == 0) {
            str[h] = L'<';
            i+=3;
        } else if (wcsncmp(&str[i], L"&gt;", 4) == 0) {
            str[h] = L'>';
```

```
            i+=3;
        } else if (wcsncmp(&str[i], L"&amp;", 5) == 0) {
            str[h] = L'&';
            i+=4;
        } else if (wcsncmp(&str[i], L"&apos;", 6) == 0) {
            str[h] = L'¥'';
            i+=5;
        } else if (wcsncmp(&str[i], L"&quot;", 6) == 0) {
            str[h] = L'"';
            i+=5;
        } else {
            fwprintf(stderr, L"wtf: warning: invalid xml: undefined escape
code¥n");
            str[h] = str[i];
        }
      } else {
        str[h] = str[i];
      }
  }
  str[h] = L'¥0';
}
```

```
//file.h
#ifndef FILE_H
#define FILE_H

int check_cfg(char *cfg_dir);
int can_rw(char *path);
int is_rfile(char *path);
int is_dir(char *path);
long file_size(char* path);

#endif
```

```
//file.c
#include <sys/types.h>
#include <sys/stat.h>
#include "file.h"
#include "config.h"

int check_cfg(char *cfg_dir)
{
  if (is_dir(cfg_dir) && can_rw(cfg_dir))
      return 1;

  return 0;
}

int can_rw(char *path)
{
  struct stat statbuf;

  if (stat(path, &statbuf) == -1)
      return 0;

  if (statbuf.st_mode & S_IRUSR && ¥
        statbuf.st_mode & S_IWUSR && ¥
        statbuf.st_mode & S_IXUSR) {
      if (statbuf.st_uid == getuid())
        return 1;
  }

  if (statbuf.st_mode & S_IRGRP && ¥
        statbuf.st_mode & S_IWGRP && ¥
        statbuf.st_mode & S_IXGRP) {
      if (statbuf.st_gid == getgid())
        return 1;
  }

  if (statbuf.st_mode & S_IROTH && ¥
        statbuf.st_mode & S_IWOTH && ¥
        statbuf.st_mode & S_IXOTH) {
        return 1;
  }

}

long file_size(char* path)
```

```
{
  struct stat statbuf;

  if (stat(path, &statbuf) == -1)
      return -1;
  return statbuf.st_size;
}

int is_rfile(char *path)
{
  struct stat statbuf;

  if (stat(path, &statbuf) == -1)
      return 0;
  if (S_ISREG(statbuf.st_mode))
      return 1;

  return 0;
}

int is_dir(char *path)
{
  struct stat statbuf;

  if (stat(path, &statbuf) == -1)
      return 0;

  if (S_ISDIR(statbuf.st_mode))
      return 1;

  return 0;
}
```

```
//index.h
#ifndef INDEX_H
#define INDEX_H

#include "xdxf.h"

#define KPMAX 32

typedef struct _idxfile_t idxfile_t;
```

```
struct _idxfile_t {
  char *idx_path;
  char *db_path;
  char *db_name;
  char *from;
  char *to;
  long fsize;
  long idx_size;
  long idx_start_pos;
  unsigned ktag_index_len;
  unsigned mask_len;
};

typedef struct _wtfidx_t wtfidx_t;
struct _wtfidx_t {
  char *path;
  wtfidx_t *next;
};

typedef struct _xdxf_idx_t xdxf_idx_t;
struct _xdxf_idx_t {
  idxfile_t *idxfile;
  uint32_t shift;
  uint32_t size;
  unsigned ktag_index;
  unsigned mask;
};

void print_idx_header(idxfile_t *idxfile);
void init_idx_file(idxfile_t *idxfile);
void free_idx_file(idxfile_t *idxfile);
int read_idx_header(idxfile_t *idxfile);
int fetch_db_langs(char *path, char **lang_from, char **lang_to);
char* fetch_db_name(char *path);
int idx_clean(char *cfg_dir);
int idx_file_create(char *db_file, key_ll_t **keys, int klen);
wtfidx_t* get_idx(void);
int gen_index(void);
void idx_free_all(wtfidx_t *idx);
key_ll_t* base_key_fetch(keypart_t *kp, int kp_len);
void free_base_keys(key_ll_t **keys, int nkeys);
void free_opt_keys(key_ll_t *optkeys);
int keyparts_fetch(DTag *tag_key, keypart_t *key_parts, int kpsize);
int count_key_tags(DTag *tag_xdxf);
```

```
void free_key_parts(keypart_t *key_parts, int kp_len);
wchar_t* read_key_by_idx(xdxf_idx_t *idx, FILE *dfp);
void free_keys(key_ll_t **keys, int nkeys);


#endif
-----------------------------------------------------------------
//index.c
#include <stdio.h>
#include <stdlib.h>
#define _GNU_SOURCE
#include <wchar.h>
#include <string.h>
#ifdef TARGET_FREEBSD
#include <sys/types.h>
#else
#include <stdint.h>
#endif
#include <dirent.h>
#include "index.h"
#include "file.h"
#include "xml_utf8.h"
#include "config.h"
#include "dict.h"

char scbuf[BUFSZ];
wchar_t sbuf[BUFSZ];
char scfname[MAXPATH];
char scpath[MAXPATH];

void print_idx_header(idxfile_t *idxfile)
{
  if (idxfile->idx_path)
      fwprintf(stdout, L"idx path: %s¥n", idxfile->idx_path);
  if (idxfile->db_path)
      fwprintf(stdout, L"db path: %s¥n", idxfile->db_path);
  if (idxfile->db_name)
      fwprintf(stdout, L"db name: %s¥n", idxfile->db_name);
  if (idxfile->from)
      fwprintf(stdout, L"from: %s¥n", idxfile->from);
  if (idxfile->to)
      fwprintf(stdout, L"to: %s¥n", idxfile->to);

  fwprintf(stdout, L"ktag index len: %d¥n", idxfile->ktag_index_len);
```

```
  fwprintf(stdout, L"mask len: %d¥n", idxfile->mask_len);
  fwprintf(stdout, L"idx size: %d¥n", idxfile->idx_size);
  fwprintf(stdout, L"¥n");
}

void init_idx_file(idxfile_t *idxfile)
{
  if (! idxfile)
      return;

  idxfile->idx_path = NULL;
  idxfile->db_path = NULL;
  idxfile->db_name = NULL;
  idxfile->from = NULL;
  idxfile->to = NULL;

  idxfile->fsize = -1;
  idxfile->idx_size = -1;
  idxfile->idx_start_pos = -1;
  idxfile->ktag_index_len = 0;
  idxfile->mask_len = 0;
}

void free_idx_file(idxfile_t *idxfile)
{
  PRDEBUG("enter")
  if (!idxfile)
      return;

  if (idxfile->idx_path)
      free(idxfile->idx_path);

  if (idxfile->db_path)
      free(idxfile->db_path);

  if (idxfile->db_name)
      free(idxfile->db_name);

  if (idxfile->from)
      free(idxfile->from);

  if (idxfile->to)
      free(idxfile->to);
```

```
  free(idxfile);
  PRDEBUG("exit")
}

int read_idx_header(idxfile_t *idxfile)
{
  int i, j;
  int _l;
  FILE *fp;
  char c, _c;
  long debug_pos;

  if (!idxfile)
      return 0;

  if (! idxfile->idx_path)
      return 0;

  if ((idxfile->fsize = file_size(idxfile->idx_path)) == 0)
      return 0;

  fp = fopen(idxfile->idx_path, "r");
  if (!fp)
      return 0;

  debug_pos = ftell(fp);

  i=0;
  _l = 1;
  c = fgetc(fp);
  debug_pos = ftell(fp);
  while(_l) {
      if (c == EOF) {
        _l = 0;
      } else if (c != '¥n') {
        if (i<BUFSZ-1)
            scbuf[i++] = c;
      } else if (c == '¥n') {
        if (i>0) {
            scbuf[i] = '¥0';
            if (strncmp(scbuf, "path:", 5) == 0) {
              for(j=5; scbuf[j] != '¥0' && isspace(scbuf[j]); j++);
              idxfile->db_path = strdup(&scbuf[j]);
            } else if (strncmp(scbuf, "name:", 5) == 0) {
```

```
            for(j=5; scbuf[j] != '¥0' && isspace(scbuf[j]); j++);
            idxfile->db_name = strdup(&scbuf[j]);
          } else if (strncmp(scbuf, "lang from:", 10) == 0) {
            for(j=10; scbuf[j] != '¥0' && isspace(scbuf[j]); j++);
            idxfile->from = strdup(&scbuf[j]);
          } else if (strncmp(scbuf, "lang to:", 8) == 0) {
            for(j=8; scbuf[j] != '¥0' && isspace(scbuf[j]); j++);
            idxfile->to = strdup(&scbuf[j]);
          } else if (strncmp(scbuf, "ktag index len:", 15) == 0) {
            for(j=15; scbuf[j] != '¥0' && isspace(scbuf[j]); j++);
            idxfile->ktag_index_len = atoi(&scbuf[j]);
          } else if (strncmp(scbuf, "mask len:", 9) == 0) {
            for(j=9; scbuf[j] != '¥0' && isspace(scbuf[j]); j++);
            idxfile->mask_len = atoi(&scbuf[j]);
          }
          i=0;
        }

      }
      _c = fgetc(fp);
      if (c == '¥n' && _c == '¥n') {
        c = _c;
        _c = fgetc(fp);
        if (_c == '¥n') {
            _l = 0;
            debug_pos = ftell(fp);
            idxfile->idx_start_pos = ftell(fp);
        }
      }
      c = _c;
  }
  fclose(fp);

  idxfile->idx_size = idxfile->fsize - idxfile->idx_start_pos;
  if (idxfile->idx_size % (8 + idxfile->ktag_index_len + idxfile->mask_len) != 0)
{
      fprintf(stderr, "wtf: error: invalid index block size¥n");
      fprintf(stderr, "     file: %s¥n", idxfile->idx_path);
      fprintf(stderr, "     idx_size: %d¥n", idxfile->idx_size);
      fprintf(stderr, "     ktag_index_len: %d¥n", idxfile->ktag_index_len);
      fprintf(stderr, "     mask_len: %d¥n", idxfile->mask_len);
      return 0;
  }
```

```
  return 1;
}

int idx_clean(char *cfg_dir)
{
  DIR *dirp;
  struct dirent *de;
  int nfiles;

  PRDEBUG("enter")

  if (cfg_dir && check_cfg(cfg_dir) && can_rw(cfg_dir)) {
      dirp = opendir(cfg_dir);
      if (! dirp) {
        fprintf(stderr, "wtf: warning: could not open %s¥n", cfg_dir);
        return 0;
      }

      fprintf(stdout, "cleaning:¥n");
      nfiles = 0;
      while((de = readdir(dirp)) != NULL) {
        if (de->d_name[0] == '.') {
#ifdef _DIRENT_HAVE_D_NAMLEN
            if (de->d_namlen == 1)
              continue;
            if (de->d_name[1] == '.' && de->d_namlen == 2)
              continue;
#else
            if (strlen(de->d_name) == 1)
                continue;
            if (de->d_name[1] == '.' && strlen(de->d_name) == 2)
                continue;
#endif
        }
#ifdef _DIRENT_HAVE_D_NAMLEN
        strncpy(scfname, de->d_name, de->d_namlen);
        scfname[de->d_namlen] = '¥0';
#else
        strcpy(scfname, de->d_name);
#endif
        snprintf(scpath, MAXPATH, "%s/%s", cfg_dir, scfname);
        if (is_idx(scpath)) {
            fprintf(stdout, "  %s¥n", scfname);
            remove(scpath);
```

```
            nfiles++;
          }
        }
        closedir(dirp);
        fprintf(stdout, "total removed: %d¥n", nfiles);
    }
    PRDEBUG("exit")
    return 1;
}

char* fetch_db_name(char *path)
{
    int i, j, h;
    char *name_ret;

    if (!path)
        return NULL;

    name_ret = NULL;
    for (i=0; path[i] != '¥0'; i++);
    for (; i >= 0 && path[i] != '/'; i--);
    for (j=i-1; j >= 0 && path[j] != '/'; j--);
    j++;
    if (j<i) {
        name_ret = (char*) malloc(sizeof(char) * (i-j+1));
        for (h=0; j<i; j++, h++)
          name_ret[h] = path[j];
        name_ret[h] = '¥0';
    }
    return name_ret;
}

int fetch_db_langs(char *path, char **lang_from, char **lang_to)
{
    int i, j, h;
    char *lfrom, *lto;

    if (!path || !lang_from || !lang_to)
        return 0;

    lfrom = lto = NULL;
    for (i=0; path[i] != '¥0'; i++);
    for (; i >= 0 && path[i] != '/'; i--);
    for (i--; i >= 0 && path[i] != '/'; i--);
```

```
  for (j=i-1; j >=0 && path[j] != '/'; j--);
  j++;
  if (i-j == 4) {
      lfrom = (char*) malloc(sizeof(char) *3);
      lto = (char*) malloc(sizeof(char) *3);
      for (h=0; h<2; h++) {
        lfrom[h] = tolower(path[j+h]);
        lto[h] = tolower(path[j+h+2]);
      }
      lfrom[2] = lto[2] = '¥0';
  }
  *lang_from = lfrom;
  *lang_to = lto;
  return 1;
}

int idx_file_create(char *db_file, key_ll_t **keys, int klen)
{
  char *cfg_dir;
  int fnum;
  FILE *fp;
  int k;
  key_ll_t *_key;
  char *db_name;
  char *lang_from;
  char *lang_to;

  PRDEBUG("enter")

  if (!db_file || !keys || klen == 0)
      return 0;

  cfg_dir = get_cfg_dir();
  if (cfg_dir && ! check_cfg(cfg_dir)) {
      mkdir(cfg_dir, 0777);
  }

  if (cfg_dir && check_cfg(cfg_dir)) {

      fnum = 0;
      do {
        snprintf(scpath, MAXPATH, "%s/%d.idx", cfg_dir, fnum);
        fnum++;
      } while (is_idx(scpath));
```

```
      fnum--;

      fp = fopen(scpath, "w");
      if (!fp) {
        fwprintf(stderr, L"wtf: error: could not create index file: %s¥n",
scpath);
        return 0;
      }

      fprintf(stdout, "creating index file: %d.idx¥n", fnum);

      db_name = fetch_db_name(db_file);
      fetch_db_langs(db_file, &lang_from, &lang_to);

      fprintf(fp, "path: %s¥n", db_file);
      if (db_name) {
        fprintf(fp, "name: %s¥n", db_name);
        free(db_name);
      }

      if (lang_from) {
        fprintf(fp, "lang from: %s¥n", lang_from);
        free(lang_from);
      }

      if (lang_to) {
        fprintf(fp, "lang to: %s¥n", lang_to);
        free(lang_to);
      }

      fprintf(fp, "ktag index len: %d¥n", ktag_index_len());
      fprintf(fp, "mask len: %d¥n", opt_mask_len());

      fprintf(fp, "¥n¥n");
      for (k=0; k<klen; k++) {
        _key = keys[k];
        if (!_key) {
            fwprintf(stderr, L"wtf: error: %s:%d: _key == NULL¥n", __FUNCTION__,
__LINE__);
            fclose (fp);
            return 0;
        }
        while (_key->prev)
            _key = _key->prev;
```

```
        while(_key) {
            fwrite(&_key->tag->pos, sizeof(uint32_t), 1, fp);
            fwrite(&_key->tag->len, sizeof(uint32_t), 1, fp);
            fwrite(&_key->ktag_index, ktag_index_len(), 1, fp);
            fwrite(&_key->kmask, opt_mask_len(), 1, fp);
            _key = _key->next;
        }
      }
  } else {
      fwprintf(stdout, L"wtf: error: could not open config directory¥n");
      return 0;
  }

  fclose(fp);
  PRDEBUG("exit")
  return 1;
}

keypart_t key_parts_p1[KPMAX];
wchar_t* read_key_by_idx(xdxf_idx_t *idx, FILE *dfp)
{
  DTag *tag_ar;
  DTag *tag_key;
  int ktag_index;
  wchar_t *key_str;

  if (!idx)
      return NULL;

  tag_ar = read_xdxf(idx->idxfile->db_path, idx->shift, idx->size, dfp);
  if (!tag_ar)
      return NULL;

  if (! is_tag_name(tag_ar, L"ar")) {
      tag_free(tag_ar);
      return NULL;
  }

  key_str = NULL;
  ktag_index = 0;
  tag_key = tag_seek(tag_ar, tag_ar->body, L"k", S_ALL);

  while(ktag_index != idx->ktag_index && tag_key) {
```

```
        ktag_index++;
        tag_key = tag_seek(tag_ar, tag_key, L"k", S_ALL | S_SKIP);
    }

    if (tag_key) {
        if (keyparts_fetch(tag_key, key_parts_p1, KPMAX)) {
          key_str = get_key_by_mask(key_parts_p1, KPMAX, idx->mask);
          free_key_parts(key_parts_p1, KPMAX);
        }
    }

    tag_free(tag_ar);

    return key_str;
}

keypart_t key_parts_p2[KPMAX];
int gen_index(void)
{
    char *wtf_path;
    wtfdb_t *db_ll;
    wtfdb_t *db;
    int ret;
    DTag *tag_root;
    int ar;
    int i, j;
    key_ll_t **keys;
    int nkeys;
    DTag *tag_xdxf;
    DTag *tag_k;
    DTag *tag_ar;
    DTag *tag_key;
    key_ll_t *optkeys;
    int ktag_index;

    PRDEBUG("enter")

    if (! (wtf_path = getenv("WTFPATH"))) {
        fprintf(stderr, "wtf: error: no WTFPATH set¥n");
        return 0;
    }

    if (! is_dir(wtf_path)) {
        fprintf(stderr, "wtf: error: invalid WTFPATH¥n");
```

```
    return 0;
}

ret = get_xdxf(wtf_path, &db_ll);
print_wtfdbs(db_ll);

db = db_ll;
idx_clean(get_cfg_dir());
while (db) {
    fprintf(stdout, "¥n");
    fprintf(stdout, "file: %s¥n", db->path);
    tag_root = read_xdxf(db->path, 0, 0, NULL);
    if (tag_root) {
      fprintf(stdout, "parsed: ok¥n");
      tag_xdxf = tag_seek(NULL, tag_root, L"xdxf", S_ALL);

      nkeys = count_key_tags(tag_xdxf);
      fwprintf(stdout, L"number of key words: %d¥n", nkeys);

      tag_ar = tag_seek(tag_xdxf, tag_xdxf->body, L"ar", S_ALL);

      optkeys = NULL;
      if (tag_ar && nkeys) {
          keys = (key_ll_t**) malloc(sizeof(key_ll_t*) * nkeys);
          if (!keys) {
            fwprintf(stderr, L"wtf: error: malloc¥n");
            tag_free(tag_root);
            wtfdb_free_all(db_ll);
            return 0;
          }
          bzero(keys, sizeof(key_ll_t*)
* nkeys);
          j = 0;
          while (tag_ar) {
            tag_key = tag_seek(tag_ar, tag_ar->body, L"k", S_ALL);

            ktag_index = 0;
            while(tag_key) {
                if (keyparts_fetch(tag_key, key_parts_p2, KPMAX)) {
                  keys[j] = base_key_fetch(key_parts_p2, KPMAX);
                  keys[j]->ktag_index = ktag_index;
                  keys[j]->tag = tag_ar;
                  j++;
```

```
                    optkeys = fill_opts(key_parts_p2, KPMAX, tag_ar, optkeys,
ktag_index);

                    free_key_parts(key_parts_p2, KPMAX);
                  }
                  ktag_index++;
                  tag_key = tag_seek(tag_ar, tag_key, L"k", S_ALL | S_SKIP);
                }
                tag_ar = tag_seek(tag_xdxf, tag_ar, L"ar", S_ALL | S_SKIP);
              }

              /* double sort operation to exclude case when two articles exist
having same key phrases
               * with different cases, like Postal (game) and postal (man working in
post office)
               */
              qsort(keys, nkeys, sizeof(key_ll_t*), cmpkeys);
              dup_remove(keys, &nkeys);
              crop_nonalpha(keys, nkeys);
              crop_spaces(keys, nkeys);
              start_spaces_remove(keys, nkeys, 1);
              keys_tolower(keys, nkeys);
              qsort(keys, nkeys, sizeof(key_ll_t*), cmpkeys);
              optkeys = sort_keys(optkeys);
              dup_remove2(optkeys);
              crop_nonalpha2(optkeys);
              crop_spaces2(optkeys);
              start_spaces_remove2(optkeys, 1);
              keys_tolower2(optkeys);
              optkeys = sort_keys(optkeys);
              kl_base_opt_merge(keys, nkeys, optkeys);

              idx_file_create(db->path, keys, nkeys);

              free_keys(keys, nkeys);
              /*
              free_opt_keys(optkeys);
              free_base_keys(keys, nkeys);
              */

            }
            tag_free(tag_root);
          } else {
            fprintf(stdout, "parsed: no¥n");
```

```
        }
        db = db->next;
    }

    wtfdb_free_all(db_ll);

    PRDEBUG("exit")
    return 1;
}

int count_key_tags(DTag *tag_xdxf)
{
    DTag *tag_k;
    int nkeys;

    nkeys = 0;
    if (tag_xdxf) {
        tag_k = tag_seek(tag_xdxf, tag_xdxf->body, L"k", S_ALL);
        if (tag_k)
          nkeys++;
        while(tag_k = tag_seek(tag_xdxf, tag_k, L"k", S_ALL | S_SKIP))
          nkeys++;
    }

    return nkeys;
}


void free_key_parts(keypart_t *key_parts, int kp_len)
{
    int h;
    int npart;

    PRDEBUG("enter")

    npart = count_keys(key_parts, kp_len);

    for (h=0; h<npart; h++) {
        if (key_parts[h].str) {
              free(key_parts[h].str);
              key_parts[h].str = NULL;
        }
    }
    PRDEBUG("exit")
```

```
}

void free_keys(key_ll_t **keys, int nkeys)
{
  int i;
  key_ll_t *_key;
  key_ll_t *_key_next;

  if (!keys || nkeys == 0)
      return;

  for (i=0; i<nkeys; i++) {
      if (keys[i]) {
        _key = keys[i];
        while(_key->prev)
            _key = _key->prev;

        while(_key) {
            _key_next = _key->next;
            if (_key->key)
              free(_key->key);
            free(_key);
            _key = _key_next;
        }
      }
  }
  free(keys);
}

void free_base_keys(key_ll_t **keys, int nkeys)
{
  int i;

  PRDEBUG("enter")
  if (!keys || nkeys == 0)
      return;

  for (i=0; i<nkeys; i++) {
      if (keys[i]) {
        if (keys[i]->key)
            free(keys[i]->key);
        free(keys[i]);
      }
  }
```

```
  free(keys);
  PRDEBUG("exit")

}

void free_opt_keys(key_ll_t *optkeys)
{
  key_ll_t *_key;
  key_ll_t *_key_next;

  PRDEBUG("enter")

  _key = optkeys;
  while(_key->prev)
      _key = _key->prev;

  while(_key) {
      _key_next = _key->next;
      if (_key->key)
        free(_key->key);
      free(_key);
      _key = _key_next;
  }

  PRDEBUG("exit")

}

int keyparts_fetch(DTag *tag_key, keypart_t *key_parts, int kpsize)
{
  DTag *_tag;
  int i, h;
  DTag *tag_opt;
  int npart;
  int nopt;
  DTag *tag_vroot;
  wchar_t *_buf;

  if (!tag_key || !key_parts || kpsize == 0)
      return 0;

  tag_opt = NULL;
  _tag = tag_key->body;
  h=0;
```

```
  nopt = 0;
  npart = 0;
  tag_vroot = tag_key;
  while(_tag) {
      if (_tag->type == DT_Value) {
        for (i=0; i<_tag->vlen && h <BUFSZ-1; i++, h++) {
            if (_tag->value[i] == L'&') {
              if (wcsncmp(&_tag->value[i], L"&lt;", 4) == 0) {
                  sbuf[h] = L'<';
                  i+=3;
              } else if (wcsncmp(&_tag->value[i], L"&gt;", 4) == 0) {
                  sbuf[h] = L'>';
                  i+=3;
              } else if (wcsncmp(&_tag->value[i], L"&amp;", 5) == 0) {
                  sbuf[h] = L'&';
                  i+=4;
              } else if (wcsncmp(&_tag->value[i], L"&apos;", 6) == 0) {
                  sbuf[h] = L'¥'';
                  i+=5;
              } else if (wcsncmp(&_tag->value[i], L"&quot;", 6) == 0) {
                  sbuf[h] = L'"';
                  i+=5;
              } else {
                  fwprintf(stderr, L"wtf: warning: invalid xml: undefined escape
code¥n");
                  sbuf[h] = _tag->value[i];
              }
            } else {
              sbuf[h] = _tag->value[i];
            }
        }
        _tag = tag_rnext(tag_vroot, _tag, S_ALL);
      } else if (_tag->type == DT_Tag) {
        if (wcsncmp(_tag->name, L"nu", _tag->nlen) == 0) {
            _tag = tag_rnext(tag_vroot, _tag, S_DOWN | S_UP);
        } else if (wcsncmp(_tag->name, L"opt", _tag->nlen) == 0) {
            if (nopt < MAXOPTS) {
              if (h>0) {
                  sbuf[h] = L'¥0';
                  _buf = (wchar_t*) malloc(sizeof(wchar_t) * (h+1));
                  wcscpy(_buf, sbuf);
      /*          key_parts[npart].str = wcsdup(sbuf); */
                  key_parts[npart].str = _buf;
                  key_parts[npart].is_opt = 0;
```

```
                npart++;
                h=0;
              }
              tag_opt = _tag;
              tag_vroot = tag_opt;
              _tag = tag_rnext(tag_vroot, tag_opt, S_DEEP);
            } else {
              fwprintf(stderr, L"wtf: warning: %s:%d: nopt >= MAXOPTS¥n",
__FUNCTION__, __LINE__);
              _tag = tag_rnext(tag_vroot, _tag, S_DOWN | S_UP);
            }
        } else {
            _tag = tag_rnext(tag_vroot, _tag, S_ALL);
        }
      } else {
        _tag = tag_rnext(tag_vroot, _tag, S_DOWN | S_UP);
      }

      if (!_tag && tag_opt) {
        if (h>0) {
            sbuf[h] = L'¥0';
            _buf = (wchar_t*) malloc(sizeof(wchar_t) * (h+1));
            wcscpy(_buf, sbuf);
/*          key_parts[npart].str = wcsdup(sbuf); */
            key_parts[npart].str = _buf;
            key_parts[npart].is_opt = 1;
            nopt++;
            npart++;
            h=0;
        }
        tag_vroot = tag_key;
        _tag = tag_rnext(tag_vroot, tag_opt, S_DOWN | S_UP);
        tag_opt = NULL;
      }
  }

  if (h>0) {
      sbuf[h] = L'¥0';
      _buf = (wchar_t*) malloc(sizeof(wchar_t) * (h+1));
      wcscpy(_buf, sbuf);
/*    key_parts[npart].str = wcsdup(sbuf); */
      key_parts[npart].str = _buf;
      key_parts[npart].is_opt = 0;
      npart++;
```

```
      h=0;
  }
  key_parts[npart].str = NULL;

  return 1;
}

key_ll_t* base_key_fetch(keypart_t *kp, int kp_len)
{
  int h, i, k;
  key_ll_t *kl;
  int npart;
  wchar_t *_buf;

  if (!kp || kp_len == 0)
      return NULL;

  npart = count_keys(kp, kp_len);

  for (h=0, k=0; h<npart; h++) {
      if (! kp[h].is_opt) {
        for (i=0; k <= BUFSZ && kp[h].str[i] != L'¥0'; i++, k++)
            sbuf[k] = kp[h].str[i];
      }
  }
  sbuf[k] = L'¥0';
  kl = (key_ll_t*) malloc(sizeof(key_ll_t));
  _buf = (wchar_t*) malloc(sizeof(wchar_t) * (k+1));
  wcscpy(_buf, sbuf);
/*  kl->key = wcsdup(sbuf); */
  kl->key = _buf;
  kl->ktag_index = 0;
  kl->kmask = 0;
  kl->tag = NULL;
  kl->next = NULL;
  kl->prev = NULL;

  return kl;
}

wtfidx_t* get_idx(void)
{
  int ret;
  DIR *dirp;
```

```
  struct dirent *de;
  wtfidx_t *idx_last = NULL;
  wtfidx_t *_idx;
  char *cfg_dir;

  cfg_dir = get_cfg_dir();

  if (! is_dir(cfg_dir))
      return NULL;

  dirp = opendir(cfg_dir);
  if (! dirp) {
      fprintf(stderr, "wtf: error: could not open %s¥n", cfg_dir);
      return NULL;
  }

  while((de = readdir(dirp)) != NULL) {
      if (de->d_name[0] == '.') {
#ifdef _DIRENT_HAVE_D_NAMLEN
        if (de->d_namlen == 1)
            continue;
        if (de->d_name[1] == '.' && de->d_namlen == 2)
            continue;
#else
        if (strlen(de->d_name) == 1)
            continue;
        if (de->d_name[1] == '.' && strlen(de->d_name) == 2)
            continue;
#endif
      }
#ifdef _DIRENT_HAVE_D_NAMLEN
      strncpy(scfname, de->d_name, de->d_namlen);
      scfname[de->d_namlen] = '¥0';
#else
      strcpy(scfname, de->d_name);
#endif
      snprintf(scpath, MAXPATH, "%s/%s", cfg_dir, scfname);
      if (is_idx(scpath)) {
            _idx = (wtfidx_t*) malloc(sizeof(wtfidx_t));
            _idx->path = strdup(scpath);
            _idx->next = idx_last;
            idx_last = _idx;
      }
  }
```

```
  closedir(dirp);

  return idx_last;
}

void idx_free_all(wtfidx_t *idx)
{
  wtfidx_t *_idx;

  while (idx) {
      _idx = idx->next;
      if (idx->path)
        free(idx->path);
      free(idx);
      idx = _idx;
  }
}
```

```
//utf8.h
#ifndef UTF8_H
#define UTF8_h

#include <wchar.h>

int wc_mb_len(wchar_t wc);
size_t wcslen_fast(const wchar_t *s);
int wcsncmp_fast(const wchar_t *s1, const wchar_t *s2, size_t n);
wchar_t* wcsstr_fast(wchar_t *s1, const wchar_t *s2);

#endif
```

```
//utf8.c
#include "utf8.h"

int wc_mb_len(wchar_t wc)
{
  if (wc <= 0x7f)
      return 1;
  else if (wc <= 0x7ff)
```

```
      return 2;
  else if (wc <= 0xffff)
      return 3;
  else if (wc <= 0x1fffff)
      return 4;
  else if (wc <= 0x3ffffff)
      return 5;
  else if (wc <= 0x7fffffff)
      return 6;
}

size_t wcslen_fast(const wchar_t *s)
{
  unsigned long j;

  if (!s)
      return 0;

  for (j=0; s[j] != L'¥0'; j++);
  return j;
}

int wcsncmp_fast(const wchar_t *s1, const wchar_t *s2, size_t n)
{
  unsigned long j;

  if (!s1) {
      if (!s2) {
        return 0;
      } else {
        if (wcslen_fast(s2))
            return -1;
        else
            return 0;
      }
  } else if (!s2) {
      if (wcslen_fast(s1))
        return 1;
      else
        return 0;
  }

  if (n == 0)
      return 0;
```

```
  for (j=0; j<n && s1[j] == s2[j] && s1[j] != L'¥0' && s2[j] != L'¥0'; j++);
  if (j == n) {
      return 0;
  } else if (s2[j] == L'¥0') {
      if (j>0) {
        return 0;
      } else if (s1[j] == L'¥0') {
        return 0;
      } else {
        return 1;
      }
  } else if (s1[j] == L'¥0') {
      return -1;
  } else if (s1[j] > s2[j]) {
      return 1;
  } else {
      return -1;
  }
}

wchar_t* wcsstr_fast(wchar_t *s1, const wchar_t *s2)
{
  unsigned long j;
  size_t l2;

  l2 = wcslen_fast(s2);

  for(j=0; s1[j] != L'¥0'; j++) {
      if (wcsncmp_fast(&s1[j], s2, l2) == 0)
        return &s1[j];
  }
  return NULL;
}
```

```
//xdxf.h
#ifndef XDXF_H
#define XDXF_H

#ifdef TARGET_FREEBSD
#include <sys/types.h>
#else
```

```
#include <stdint.h>
#endif
#include "xml_utf8.h"

#define OPTS_8 1

#if defined(OPTS_8)
#define MAXOPTS 8
#elif defined(OPTS_16)
#define MAXOPTS 16
#elif defined(OPTS_32)
#define MAXOPTS 32
#else
#define OPTS_16 1
#define MAXOPTS 16
#endif

#define KTAG_INDEX_16 1

typedef struct _article_t article_t;
struct _article_t {
  wchar_t *key;
  wchar_t *article;
  long ar_start_pos;
  long ar_size;
/*  idxfile_t *idx; */
};

typedef struct _wkey_t wkey_t;
struct _wkey_t {
  wchar_t *key;
  article_t **arts;
  int narts;
};

typedef struct _key_ll_t key_ll_t;
struct _key_ll_t {
  wchar_t *key;
#if defined (KTAG_INDEX_16)
  uint16_t ktag_index;
#else
  uint32_t ktag_index;
#endif
```

```
#if defined(OPTS_8)
  uint8_t kmask;
#elif defined(OPTS_16)
  uint16_t kmask;
#elif defined(OPTS_32)
  uint32_t kmask;
#endif

  DTag *tag;
  key_ll_t *prev;
  key_ll_t *next;
};

typedef struct _keypart_t {
  wchar_t *str;
  int is_opt;
} keypart_t;

#define OPT1 (1 << 0)
#define OPT2 (1 << 1)
#define OPT3 (1 << 2)
#define OPT4 (1 << 3)
#define OPT5 (1 << 4)
#define OPT6 (1 << 5)
#define OPT7 (1 << 6)
#define OPT8 (1 << 7)
#define OPT9 (1 << 8)
#define OPT10 (1 << 9)
#define OPT11 (1 << 10)
#define OPT12 (1 << 11)
#define OPT13 (1 << 12)
#define OPT14 (1 << 13)
#define OPT15 (1 << 14)
#define OPT16 (1 << 15)
#define OPT17 (1 << 16)
#define OPT18 (1 << 17)
#define OPT19 (1 << 18)
#define OPT20 (1 << 19)
#define OPT21 (1 << 20)
#define OPT22 (1 << 21)
#define OPT23 (1 << 22)
#define OPT24 (1 << 23)
#define OPT25 (1 << 24)
#define OPT26 (1 << 25)
```

```
#define OPT27 (1 << 26)
#define OPT28 (1 << 27)
#define OPT29 (1 << 28)
#define OPT30 (1 << 29)
#define OPT31 (1 << 30)
#define OPT32 (1 << 31)

#if defined(OPTS_8)
#define MAXOPT OPT8
#elif defined(OPTS_16)
#define MAXOPT OPT16
#elif defined(OPTS_32)
#define MAXOPT OPT32
#else
#define MAXOPT 0
#endif


DTag* read_xdxf(char *path, long shift, long siz, FILE *dfp);
key_ll_t* fill_opts(keypart_t *key_parts, int kp_len, DTag *tag_ar, key_ll_t
*optkeys, int ktag_index);
key_ll_t* sort_keys(key_ll_t *key_start);
void kl_base_opt_merge(key_ll_t **keys, int nkeys, key_ll_t *optkeys);
int cmpkeys(const void *p1, const void *p2);
int dup_remove(key_ll_t **keys, int *klen);
int dup_remove2(key_ll_t *key_start);
int keys_tolower(key_ll_t **keys, int nkeys);
int keys_tolower2(key_ll_t *key_start);
int start_spaces_remove(key_ll_t **keys, int nkeys, int sorted);
int start_spaces_remove2(key_ll_t *key_start, int sorted);
wchar_t* get_key_by_mask(keypart_t *key_parts, int kp_len, unsigned opt_mask);
int count_keys(keypart_t *key_parts, int kp_len);
int count_opt_keys(keypart_t *key_parts, int kp_len);
int ktag_index_len(void);
int crop_nonalpha(key_ll_t **keys, int klen);
int crop_nonalpha2(key_ll_t *key_start);
int crop_spaces(key_ll_t **keys, int klen);
int crop_spaces2(key_ll_t *key_start);

#endif
```

//xdxf.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define _GNU_SOURCE
#include <wchar.h>
#ifdef TARGET_FREEBSD
#include <sys/types.h>
#else
#include <stdint.h>
#include <wctype.h>
#endif
#include "xdxf.h"
#include "xml_utf8.h"
#include "config.h"
#include "file.h"

wchar_t sbuf[BUFSZ];

int cmpkeys(const void *p1, const void *p2)
{
  key_ll_t **key1, **key2;

  key1 = (key_ll_t**) p1;
  key2 = (key_ll_t**) p2;
  return wcscmp((*key1)->key, (*key2)->key);
}

void kl_base_opt_merge(key_ll_t **keys, int nkeys, key_ll_t *optkeys)
{
  key_ll_t *_key;
  int start, end, j, cmpret;
  key_ll_t *_key_next;

  if (!optkeys || !keys || nkeys <= 0)
      return;

  _key = optkeys;
  while(_key) {
      _key_next = _key->next;
      start = 0;
      end = nkeys;

      for (j = (end - start)/2; ; j = (end - start)/2) {
        cmpret = wcscmp(_key->key, keys[start+j]->key);
```

```
        if (cmpret > 0) {
            start += j;
        } else if (cmpret < 0) {
            end -= j;
        }  else {
            break;
        }
        if (j == 0)
            break;
      }
      if (cmpret < 0 && start > 0) {
        start--;
      } else if (cmpret == 0 && start+j < nkeys-1) {
        do {
            start++;
            cmpret = wcscmp(_key->key, keys[start+j]->key);
        } while (cmpret == 0 && start+j < nkeys-1);

        if (cmpret < 0)
            start--;
      }

      _key->next = keys[start+j]->next;
      if (_key->next)
        _key->next->prev = _key;
      _key->prev = keys[start+j];
      keys[start+j]->next = _key;

      sort_keys(keys[start+j]);
      _key = _key_next;
    }
  }

  int count_opt_keys(keypart_t *key_parts, int kp_len)
  {
    int i, nopt;

    if (!key_parts)
        return 0;

    for (i=0, nopt=0; i<kp_len && key_parts[i].str; i++)
        if (key_parts[i].is_opt) nopt++;

    return nopt;
```

```
}

int count_keys(keypart_t *key_parts, int kp_len)
{
  int i, npart;

  for (i=0; i<kp_len && key_parts[i].str; i++);

  return i;
}


key_ll_t* fill_opts(keypart_t *key_parts, int kp_len, DTag *tag_ar, key_ll_t
*optkeys, int ktag_index)
{
  unsigned opt_key_num;
  unsigned opt_mask;
  key_ll_t *optkey_first;
  key_ll_t *optkey_last;
  key_ll_t *_key;
  wchar_t *key_str;
  int nopt;
  int i;

  PRDEBUG("enter")

  nopt = count_opt_keys(key_parts, kp_len);

  if (nopt>0) {
      opt_key_num = 1;
      for(i=0; i<nopt && i<=MAXOPTS; i++) {
        opt_key_num = opt_key_num * 2;
      }
  } else {
      opt_key_num = 0;
  }

  optkey_first = optkey_last = NULL;
  for (opt_mask=1; opt_mask < opt_key_num; opt_mask++) {
      key_str = get_key_by_mask(key_parts, kp_len, opt_mask);
      _key = (key_ll_t*) malloc(sizeof(key_ll_t));
      _key->key = key_str;
      _key->ktag_index = ktag_index;
      _key->kmask = opt_mask;
```

```
        _key->tag = tag_ar;
        _key->next = NULL;
        if (optkey_last) {
          optkey_last->next = _key;
          _key->prev = optkey_last;
          optkey_last = _key;
        } else {
          _key->prev = NULL;
          optkey_first = optkey_last = _key;
        }
    }

    if (optkeys) {
        if (optkey_first) {
          _key = optkeys;
          while (_key->next)
              _key = _key->next;
          _key->next = optkey_first;
          optkey_first->prev = _key;
        }
    } else {
        optkeys = optkey_first;
    }

    PRDEBUG("exit")

    return optkeys;
}

wchar_t* get_key_by_mask(keypart_t *key_parts, int kp_len, unsigned opt_mask)
{
    int h, k, i;
    unsigned oflag;
    int npart;
    int nopt;
    unsigned opt_key_num;
    wchar_t *str_ret;

    if (!key_parts || kp_len == 0)
        return NULL;

    nopt = count_opt_keys(key_parts, kp_len);
    npart = count_keys(key_parts, kp_len);
```

```
  if (nopt>0) {
      opt_key_num = 1;
      for(i=0; i<nopt && i<=MAXOPTS; i++)
        opt_key_num = opt_key_num * 2;
  } else {
      opt_key_num = 0;
  }

  if (opt_mask > 0 && opt_mask >= opt_key_num)
      return NULL;

  for (h=0, k=0, oflag = 1; oflag <= MAXOPT && k < npart && h < BUFSZ-1; oflag =
oflag << 1, k++) {
      while (! key_parts[k].is_opt && k < npart) {
        for(i=0; key_parts[k].str[i] != L'¥0' && h < BUFSZ-1; i++, h++)
            sbuf[h] = key_parts[k].str[i];
        k++;
      }
      if (opt_mask & oflag) {
        if (k<npart) {
            for(i=0; key_parts[k].str[i] != L'¥0' && h < BUFSZ-1; i++, h++)
              sbuf[h] = key_parts[k].str[i];
        } else {
            fwprintf(stderr, L"wtf: error: %s:%d: key_part index >= npart¥n",
__FUNCTION__, __LINE__);
        }
      }
  }
  sbuf[h] = L'¥0';

  //  str_ret = wcsdup(sbuf);
  str_ret = (wchar_t*) malloc(sizeof(wchar_t) * (h+1));
  wcscpy(str_ret, sbuf);

  return str_ret;
}

key_ll_t* sort_keys(key_ll_t *key_start)
{
  key_ll_t *sorted_start;
  key_ll_t *sorted_end;
  key_ll_t *_key, *_key2, *_key_next;

  if (!key_start)
```

```
        return NULL;

    while (key_start->prev)
        key_start = key_start->prev;

    _key = key_start->next;
    sorted_start = key_start;
    sorted_start->next = NULL;
    sorted_start->prev = NULL;
    sorted_end = sorted_start;

    while(_key) {
        _key_next = _key->next;
        _key->prev = NULL;
        _key->next = NULL;
        _key2 = sorted_end;
        while (1) {
          if (wcscmp(_key->key, _key2->key) > 0) {
              _key->next = _key2->next;
              _key->prev = _key2;
              _key2->next = _key;
              if (_key->next)
                _key->next->prev = _key;
              else
                sorted_end = _key;
              break;
          }
          if (! _key2->prev) {
              _key2->prev = _key;
              _key->next = _key2;
              break;
          }
          _key2 = _key2->prev;
        }
        _key = _key_next;
    }
    sorted_start = sorted_end;
    while (sorted_start->prev)
        sorted_start = sorted_start->prev;

    return sorted_start;
}

int crop_nonalpha(key_ll_t **keys, int klen)
```

```
{
  int j, i, h;
  wchar_t *key;

  if (!keys || klen <=0)
      return 0;

  for (j=0; j<klen && keys[j]; j++) {
      key = keys[j]->key;
      if (key) {
        for (i=0, h=0; key[h] != L'¥0'; h++) {
/*          if (key[h] != L'¥'') { */
            if (iswalnum(key[h]) || iswspace(key[h])) {
              if (i<h)
                  key[i] = key[h];
              i++;
            }
        }
        key[i] = L'¥0';
      } else {
        fwprintf(stderr, L"wtf: error: %s:%d¥n", __FUNCTION__, __LINE__);
      }
  }
  return 1;
}

int crop_nonalpha2(key_ll_t *key_start)
{
  key_ll_t *_key;
  int i, h;
  wchar_t *key_str;

  if (!key_start)
      return 0;

  _key = key_start;
  while (_key->prev)
      _key = _key->prev;

  while(_key = _key->next) {
      key_str = _key->key;
      if (key_str) {
        for (i=0, h=0; key_str[h] != L'¥0'; h++) {
/*          if (key_str[h] != L'¥'') { */
```

```
            if (iswalnum(key_str[h]) || iswspace(key_str[h])) {
              if (i<h)
                  key_str[i] = key_str[h];
              i++;
            }
        }
        key_str[i] = L'¥0';
      } else {
        fwprintf(stderr, L"wtf: error: %s:%d¥n", __FUNCTION__, __LINE__);
      }
  }
  return 1;
}


int crop_spaces(key_ll_t **keys, int klen)
{
  int j, i, h;
  wchar_t *key;

  if (!keys || klen <= 0)
      return 0;

  for (j=0; j<klen && keys[j]; j++) {
      key = keys[j]->key;
      if (key) {
        for (i=0, h=0; key[h] != L'¥0'; h++) {
            if (iswspace(key[h])) {
              if (i>0)
                  key[i++] = L' ';
              while(iswspace(key[++h]));
              h--;
            } else {
              key[i++] = key[h];
            }
        }
        key[i] = L'¥0';
      } else {
        fwprintf(stderr, L"wtf: error: %s:%d¥n", __FUNCTION__, __LINE__);
      }
  }
  return 1;
}
```

```
int crop_spaces2(key_ll_t *key_start)
{
  key_ll_t *_key;
  int i, h;
  wchar_t *key_str;

  if (!key_start)
      return 0;

  _key = key_start;
  while (_key->prev)
      _key = _key->prev;

  while(_key = _key->next) {
      key_str = _key->key;
      if (key_str) {
        for (i=0, h=0; key_str[h] != L'¥0'; h++) {
            if (iswspace(key_str[h])) {
              if (i>0)
                  key_str[i++] = L' ';
              while(iswspace(key_str[++h]));
              h--;
            } else {
              key_str[i++] = key_str[h];
            }
        }
        key_str[i] = L'¥0';
      } else {
        fwprintf(stderr, L"wtf: error: %s:%d¥n", __FUNCTION__, __LINE__);
      }
  }
  return 1;
}

int dup_remove(key_ll_t **keys, int *klen)
{
  int i, j;

  if (!keys || !klen || *klen <=0)
      return 0;

  for (j=0, i=1; i<*klen && keys[i]; i++) {
      if (wcscmp(keys[j]->key, keys[i]->key) != 0) {
        j++;
```

```
        if (j<i)
            keys[j] = keys[i];
      } else {
        if (keys[i]->key)
            free(keys[i]->key);
        free(keys[i]);
      }
  }
  *klen = j+1;
  return 1;
}

int dup_remove2(key_ll_t *key_start)
{
  key_ll_t *_key;
  key_ll_t *_key2;

  if (!key_start)
      return 0;

  _key = key_start;
  while (_key->prev)
      _key = _key->prev;

  while(_key2 = _key->next) {
      if (wcscmp(_key->key, _key2->key) != 0) {
        _key = _key2;
      } else {
        _key->next = _key2->next;
        if (_key->next)
            _key->next->prev = _key;
        if (_key2->key)
            free(_key2->key);
        free(_key2);
      }
  }
  return 1;
}

int keys_tolower(key_ll_t **keys, int nkeys)
{
  int i, j;

  if (!keys || nkeys <= 0)
```

```
      return 0;

  for (i=0; i<nkeys; i++) {
      for (j=0; keys[i]->key[j] != L'¥0'; j++)
        keys[i]->key[j] = towlower(keys[i]->key[j]);
  }
  return 1;
}

int keys_tolower2(key_ll_t *key_start)
{

  key_ll_t *_key;
  int j;

  if (!key_start)
      return 0;

  _key = key_start;
  while (_key->prev)
      _key = _key->prev;

  while (_key) {
      for (j=0; _key->key[j] != L'¥0'; j++)
        _key->key[j] = towlower(_key->key[j]);
      _key = _key->next;
  }
  return 1;
}

int start_spaces_remove(key_ll_t **keys, int nkeys, int sorted)
{
  int i, j, h;

  if (!keys || nkeys <= 0)
      return 0;

  for (i=0; i<nkeys; i++) {
      for (j=0; iswspace(keys[i]->key[j]) && keys[i]->key[j] != L'¥0'; j++) ;

      if (!j && sorted) {
        break;
      } else if (j) {
        for (h=0; keys[i]->key[j] != L'¥0'; j++, h++)
```

```
            keys[i]->key[h] = keys[i]->key[j];
        keys[i]->key[h] = L'¥0';
      }
  }
  return 1;
}

int start_spaces_remove2(key_ll_t *key_start, int sorted)
{
  key_ll_t *_key;
  int h, j;

  if (!key_start)
      return 0;

  _key = key_start;
  while (_key->prev)
      _key = _key->prev;

  while (_key) {
      for (j=0; iswspace(_key->key[j]) && _key->key[j] != L'¥0'; j++) ;

      if (!j && sorted) {
        break;
      } else if (j) {
        for (h=0; _key->key[j] != L'¥0'; j++, h++)
            _key->key[h] = _key->key[j];
        _key->key[h] = L'¥0';
      }
      _key = _key->next;
  }
  return 1;
}

DTag* read_xdxf(char *path, long shift, long size, FILE *dfp)
{
  FILE *fp;
  int i;
  wchar_t *buf;
  DTag *tag;
  wint_t wc;
  long fsize;
  long mb_len;
  long nbytes;
```

```
long wc_num;
long debug_fpos;

PRDEBUG("enter")

dfp = NULL;
if (dfp)
    fp = dfp;
else
    fp = fopen(path, "r");

if (fp) {
    if (size == 0) {
      wc_num = 0;
      while((wc = fgetwc(fp)) != WEOF) wc_num++;
      nbytes = file_size(path);
      rewind(fp);
    } else {
      fsize = file_size(path);
      if (fsize > shift) {
          if (shift + size > fsize)

size = fsize - shift;
          fseek(fp, shift, SEEK_SET);
          debug_fpos = ftell(fp);
          wc_num = size;
          nbytes = size;
      } else {
          fwprintf(stderr, L"wtf: error: read_xdxf: file size <= shift¥n");
          fwprintf(stderr, L"     file: %s¥n", path);
          if (!dfp)
            fclose(fp);
          return NULL;
      }
    }
    buf = (wchar_t*) malloc(sizeof(wchar_t) * (wc_num+1));
    debug_fpos = ftell(fp);
    for (i=0, mb_len=0; i < wc_num && mb_len < nbytes; i++) {
      wc = fgetwc(fp);
      debug_fpos = ftell(fp);
      buf[i] = wc;
      mb_len += wc_mb_len(wc);
    }
    buf[i] = '¥0';
```

```
      tag = dexml(buf);
      if (!tag)
        free(buf);
      if (!dfp)
        fclose(fp);
      PRDEBUG("exit")
      return tag;
  } else {
      fwprintf(stderr, L"wtf: error: could not open file %s¥n", path);
      return NULL;
  }

  return NULL;
}

int opt_mask_len(void)
{
#if defined(OPTS_8)
  return sizeof(uint8_t);
#elif defined(OPTS_16)
  return sizeof(uint16_t);
#elif defined(OPTS_32)
  return sizeof(uint32_t);
#else
  return 0;
#endif
}

int ktag_index_len(void)
{
#if defined(KTAG_INDEX_16)
  return sizeof(uint16_t);
#else
  return sizeof(uint32_t);
#endif
}
```

ТОВАРИСТВО З ОБМЕЖЕНОЮ ВІДПОВІДАЛЬНІСТЮ «ДА ТЕХ РІВНЕ»

Україна 33013 Рівненська обл. м. Рівне вул. Бахарєва,22 ***тел./ факс /0362/ 436178,***
Код ЄДРПОУ 36597842 ***р/р 26004000051270 у ПуАТ «СЕБ Банк» м. Київ, МФО 300175***

м. Рівне 26 березня 2012 року

Довідка про впровадження програмного продукту

Декану факультету кібернетики
Міжнародний економіко-гуманітарний
університет імені академіка Степана Дем'янчука
Янчуку Петру Степановичу
ТзОВ "ДА ТЕХ РІВНЕ"
м.Рівне, вул. Бахарєва 22
Патріюк Юрій Ростиславович

Програмний продукт "XDClient" призначений для пошуку інформації в базах даних, автором якого є студент Міжнародного економіко-гуманітарного університету імені академіка Степана Дем'янчука, Чернявський Олексій Ігорович, був представлений на тестування його ефективності і коректності роботи.

Робота даного програмного продукту була проаналізована провідними спеціалістами підприємства та протестована у середовищі Windows XP. Після детальних тестів ПЗ "XDClient" отримав оцінку — відмінно. Від 14.02.2012 року за певною домовленістю із автором, програма була впроваджена на підприємстві і широко використовується працівниками при роботі з технічною документацією на автомобільні вузли, а також при спілкуванні з іноземними інвесторами та постачальниками. Збереження всіх авторських прав підприємство гарантує.

Директор *Місце печатки*_____ *Патріюк Ю. Р.*

Відгук

наукового керівника, доцента, кандидата технічних наук *Літнаровича Р.М.*
на магістерську дисертацію студента *Чернявського Олексія Ігоровича*
(прізвище, ім`я, по батькові)
групи *01КІН-М* спеціальності *прикладна математика*
на тему *Розробка модуля функцій парсингу XML і регулярних виразів як важливої технологічної складової АРМ в автосервісі (на прикладі ТОВ "ДА ТЕХ Рівне").*
Актуальність теми *Мова XML привабила достатньо уваги серед розробників і користувачів середовища Інтернет, щоб дослідити питання використання її в якості основного інструменту для створення Web-програм, зберігання даних, тощо. Регулярні вирази в свою чергу є найбільш популярною мовою пошуку текстової інформації. Це робить актуальним розробку стандарних швидких алгоритмів парсингу XML і регулярних виразів.*
Мета дослідження *Метою даної роботи є розробка алгоритму швидкого парсингу XML і регулярних виразів.*
Об`єкт дослідження *Досліджується алгоритм реалізації парсингу XML і регулярних виразів*
Коротка характеристика розділів роботи *Теоретичні основи першого розділу визначають важливість і актуальність мови XML для сучасних систем зберігання, передачі, пошуку і обробки інформації. Другий розділ лаконічно на прикладах розкриває мову регулярних виразів і показує важливість даної мови для пошуку даних в сучасному світі інформаційних технологій. Алгоритми описані в третьому і четвертону розділах мають чітку і зрозумілу структуру, гарні динамічні характеристики, оптимально використовують пам'ять комп'ютера і їх легко розширяти додатковими функціями. П'ятий розділ демонструє практичне використання розробленого модуля на прикладі повнофункціональної програми.*
Практичне значення роботи *Розроблений модуль може бути повторно використаний в зовнішніх програмних проектах або взятий за основу для розробки стандартної бібліотеки парсингу текстової інформації. Модуль характеризується невеликим розміром що робить його привабливим для статичного лінкування*
Реалізація результатів дослідження *Результати досліджень реалізовані у форматі ефективних алгоритмів та відповідних програмних модулів на мові С.*
Зауваження та недоліки *Спостерігаються деякі диспропорції в змістовому наповненні розділів роботи.*
Висновки та оцінка *Робота виконана на належному науково-практичному рівні згідно стандартів та вимог до кваліфікаційних робіт і заслуговує на оцінку «Відмінно»*
Науковий керівник Підпис Р.М.Літнарович (к.т.н., доцент)

"3" квітня 20 12 р.

Рецензія

на магістерську дисертацію студента *Чернявського Олексія Ігоровича*
групи *01КІН-М* **факультету** *кібернетики*
ПВНЗ "Міжнародний економіко-гуманітарний університет імені академіка Степана Дем'янчука"
Тема роботи *Розробка модуля функцій парсингу XML і регулярних виразів як важливої технологічної складової АРМ в автосервісі (на прикладі ТОВ "ДА ТЕХ Рівне").*

Стисла характеристика розділів роботи *Розділ I. Огляд технології XML*
Розділ II. Огляд технології регулярних виразів
Розділ III. Програмна реалізація модуля парсингу XML
Розділ IV. Програмна реалізація модуля регулярних виразів
Розділ V. Реалізація повнофункціональної прикладної програми з використанням розроблених модулів парсингу XML і регулярних виразів

Пропозиції, внесені студентом, рівень їх наукового обгрунтування та ефективність
В реалізації алгоритмів парсингу було застосовано лаконічний рекурсивний алгоритм і застосовані ефективні структури даних. Особливо виділяється малий розмір розробленого модуля що дає можливість його статичного компонування в зовнішніх програмних проектах.

Практичне значення роботи *Програмний модуль може бути використаний в зовнішніх програмних проектах для зчитування і розбору даних в форматі XML і пошуку інформації з використанням мови регулярних виразів. Алгоритм парсингу регулярних виразів легко розширити для підтримки всіх синтаксичних і лексичних конструкцій регулярних виразів мови програмування Perl.*

Якість оформлення роботи *Дисертація виконана на професійному рівні.*

Недоліки в роботі *Незначні орфографічні помилки тексту.*

Загальний висновок *Студент Чернявський О.І. професійно підготовлений і заслуговує на присвоєння кваліфікації "магістр інформатики".*

Оцінка дипломної роботи *Відмінно (93 бали –А за шкалою ECST).*

Рецензент Панченко Ігор Михайлович,
кандидат фізико-математичних наук, доцент,
завідувач кафедрою математичних дисциплін
та інформаційних систем Рівненської філії
Європейського університету
"_3_" _квітня__ 2012р. (Підпис рецензента)

Місце печатки

Підпис засвідчую
Начальник відділу кадрів

(Підпис)