**Конструювання програмного забезпечення**

**Лабораторна робота №12**

Конструювання програм для роботи з XML та файловою системою в середовищі програмування Visual C# 2010 Express Edition.

**Мета роботи:** засвоїти методику та виробити практичні навички роботи з файловою системою та в середовищі програмування Visual C# 2010 Express Edition.

**Теоретичні відомості**

**Файлова система**

Простір імен *System.IO* включає набір класів, призначених для навігації і управління файлами, папками і дисками. Класи файлової системи поділяються на два типи: інформаційні та допоміжні. Більшість інформаційних класів є похідними від базового класу FileSystemInfo. Такі класи дозволяють працювати з будь-якою системною інформацією об'єктів файлової системи, включаючи відомості про файли, папки і диски. Ці класи називаються FileInfo і DirectoryInfo. На додаток до цього клас DriveInfo представляє дискові пристрої; цей клас також інформаційний, але він не є похідним від класу FileSystemInfo, оскільки має дещо іншу функціональність (наприклад, файли і папки можна видаляти, а диски - ні). Допоміжні класи підтримують статичні методи для роботи з такими об'єктами файлової системи, як файли, папки і шляхи. До допоміжних класів відносяться класи File, Directory і Path.

**Клас FileSystemІnfo**

Клас *FileSystemІnfo* надає базову функціональність для всіх інформаційних класів.

Властивості класу *FileSystemlnfo*:

*Attributes* - повертає або встановлює *FileAttributes* для поточного файлу або папки.

*CreationTime -* повертає або встановлює час створення поточного файлу або папки.

*Exists* - визначає, чи існує файл або папка.

*Extension* - повертає рядок, що представляє розширення файлу або папки.

*FullName* - повертає повний шлях до файлу або папки.

*LastAccessTime* - повертає або встановлює час останнього звернення до файлу або папки.

*LastWriteTime* - повертає або встановлює час останнього запису у файл або папку.

Методи класу FileSystemlnfo:

*Delete -* видаляє файл або папку.

*Refresh* - оновлює екземпляр класу останніми даними про файлову систему.

**Клас Filelnfo**

Клас Filelnfo надає базову функціональність для доступу до окремих файлів і управління ними.

Властивості класу:

*Name* - ім'я.

*Directory* - повертає об'єкт DirectoryInfo, що представляє папку, в якій розташований файл.

*DirectoryName* - повертає ім'я папки, в якій розташований файл.

*IsReadOnly* - повертає або встановлює прапор, який визначає можливість зміни чи видалення файлу.

*Length* - повертає розмір файлу.

Найбільш важливі методи класу *Filelnfo*:

*AppendText* - створює новий об'єкт *StreamWriter*, що дозволяє додавати текст у файл.

*CopyTo* – копіювання файлу в новий файл.

*Create -* створює файл з параметрами поточного файлу.

*CreateText* - створює новий об'єкт *StreamWriter* і новий файл для запису тексту.

*Decrypt -* розшифровує файл, зашифрований поточним користувачем.

*Encrypt* - зашифровує файл, роблячи його вміст доступним тільки поточному користувачеві.

*MoveTo* - переміщає файл в інший каталог.

*Open* - відкриває файл із заданими дозволами (на читання, читання і запис і т. п.)

*OpenRead* - відкриває файл тільки для читання.

*OpenText* - відкриває файл і повертає об'єкт *StreamReader* для читання тексту з файлу.

*OpenWrite* - відкриває файл тільки для запису.

*Replace* - замінює файл файлом, створеним за параметрами поточного об'єкта *FileІnfo*.

Щоб отримати інформацію про вибраний файл використовують наступний код:

*Filelnfo ourFile = new FileInfo (@ с \ boot.ini ");  
if (ourFile.Exists) {  
Console.WriteLine ("Filename: {0}", ourFile Name); Console.WriteLine ("Path: {0}", ourFile.FullName);  
}*

Об'єкт *FileІnfo* дозволяє не тільки отримувати інформацію про файл, але і виконувати над файлом різні операції. Розглянемо копіювання файлу:

Filelnfo ourFile = new Filelnfо (@ "с: \ boot.ini");

ourFile.CopyTo (@ "c: \ boot.bak");

Як тільки допустимий об'єкт Filelnfo створений, можна звертатися до всіх його властивостей і викликати будь-які його методи.

**Клас DirectoryІnfo**

Клас DirectoryІnfo надає базову функціональність для доступу до окремих каталогів і управління ними.

Найбільш важливі властивості DirectoryІnfo:

Parent - повертає об'єкт DirectoryІnfo для батьківської папки.

Root - повертає рядок, що представляє кореневий елемент шляху.

**Найбільш важливі методи DirectoryІnfo:**

Create - створює папку, описану в поточному об'єкті DirectoryІnfo.

CreateSubdirectory - створює нову папку, вкладену в поточну папку в ієрархії каталогів.

GetDirectories - повертає масив об'єктів Directorylnfo, що представляють вкладені  
папки поточного каталогу

GetFiles - повертає масив об'єктів FileІnfo, які представляють усі файли  
в поточному каталозі

GetFileSystemІnfos - повертає масив об'єктів FileSystemІnfo, що представляють файли  
і папки в поточному каталозі

MoveTo - переміщає поточну папку в інший каталог.

Отримання інформації про файли в папці багато в чому схожі на отримання інформації про файл. Нижче наведений приклад для перерахунку файлів у папці:

*Directorylnfo ourDir = new DirectoryInfo (@ "c: \ windows"); Console.WriteLinef 'Directory: {0} ", ourDir.FullName);*

*foreach (Filelnfo file in ourDir.GetFilesO)*

*{*

*Console.WriteLine ("File: {0}", file.Name);*

*}*

Метод GetFiles об'єкта Directorylnfo дозволяє перерахувати файли в окремій папці.

**Клас DriveІnfo**

Клас DriveІnfo надає базову функціональність для доступу до окремих дисків і управління ними.

Найбільш важливі властивості Drivelnfo:

TotalFreeSpace – повертає об’єм вільного місця на диску;

[DriveFormat](http://msdn.microsoft.com/ru-ru/library/system.io.driveinfo.driveformat.aspx) - повертає формат диска, наприклад NTFS або FAT32;

DriveType - повертає тип диска як перераховування DriveType.

**Найбільш важливі методи Drivelnfo:**

GetDrives – статичний метод, повертає всі диски поточної системи.

**Перелік DriveType**

Перелік DriveType визначає типи дисків, які може представляти об'єкт DriveІnfo.

Члени DriveType:

CDRom - оптичний диск, CD-ROM, DVD і т.п.

Fixed - фіксований жорсткий диск.

Network - підключений мережевий диск.

NoRootDirectory - диск, який не має кореневого каталогу.

Ram - віртуальний диск.

Removable - знімний диск.

Unknown - дисковий пристрій невизначеного типу.

Клас Path

Клас Path надає методи для управління шляхами файлової системи.

Методи:

ChangeExtension - приймає як параметр шлях і повертає шлях до файлу з іншим розширенням.

Combine - поєднує масив стрічок в шлях.

GetDirectoryName - повертає ім'я папки, що відповідає заданому шляху.

GetExtension - повертає розширення файлу, що відповідає заданому шляху.

GetFileName - повертає ім'я файлу, що відповідає заданому шляху

GetFullPath - повертає повний шлях.

GetPathRoot - повертає кореневий каталог.

GetRandomFileName – генерує випадкове ім'я файлу.

GetTempFileName - створює тимчасовий файл і повертає повний шлях до нього.

GetTempPath - повертає шлях до папки тимчасових файлів поточного користувача або системи.

HasExtension - визначає чи містить ім'я файлу розширення, що відповідає заданому шляху

IsPathRooted - визначає чи входить заданий шлях у кореневий каталог

Клас Path дозволяє робити запит і розбирати шляхи файлової системи. Клас Path звільняє програміста від написання власного коду для розбору рядків про шляхи файлової системи. Наприклад, можна скористатися класом Path, щоб отримати і змінити розширення файлу, як показано в наступному фрагменті коду:

*string ourPath = @ "c: \ boot.ini"; Console.WriteLine (ourPath);*

*Console.WriteLine ("Ext: {0}", Path.GetExtension (ourPath));*

*Console.Writel.ine ("Change Path: {0}", Path.ChangeExtension (ourPath, "bak"));*

Читання і запис файлів - одні з найбільш поширених завдань у програмуванні. .NET-розробник повинен вміти програмувати читання і запис у файли.

**Клас File**

Клас File надає базову функціональність з відкриття файлових потоків для читання і запису.

Статичні методи класу File:

AppendAllText - додає заданий рядок до існуючого файлу, а якщо файл не існує, то створює його.

AppendText - відкриває файл (якщо він не існує, то створює його) і повертає об'єкт StreamWriter, налаштований для додавання тексту в файл.

Copy - створює копію файлу.

Move - переміщує файл в інший каталог.

Open - відкриває існуючий файл і повертає об'єкт FileStream.

OpenRead - відкриває існуючий файл і повертає об'єкт FileStream, налаштований тільки для читання.

OpenText - відкриває існуючий файл і повертає об'єкт StreamReader.

OpenWrite - відкриває існуючий файл для запису і повертає об'єкт StreamWriter.

ReadAllBytes - відкриває файл, зчитує його вміст цілком в масив байтів і закриває файл.

ReadAllLines - відкриває файл, зчитує його вміст в масив рядків (один рядок масиву відповідає одному рядку файлу) і закриває файл.

ReadAllText - відкриває файл, зчитує його вміст в рядок і закриває файл.

WriteAllBytes - відкриває файл, записує в нього вміст масиву байтів (перезаписуючи існуючі дані) і закриває файл

WriteAllLines - відкриває файл, записує в нього вміст масиву рядків (перезаписуючи існуючі дані) і закриває файл.

WriteAllText - відкриває файл, записує в нього вміст рядка (перезаписуючи існуючі дані) і закриває файл.

**Члени переліку FileAccess**

Read - визначає доступ до файлу лише для читання.

Write - визначає доступ до файлу для запису. Файл стає недоступним для  
читання - тільки для запису

ReadWrite - визначає повний доступ до файлу для читання і запису (еквівалент  
комбінації Read і Write)

Клас FileStream

Клас FileStream надає базову функціональність, дозволяючи відкривати файлові потоки для читання і запису.

Найважливіші властивості:

CanRead - визначає чи підтримує потік читання;

CanSeek - визначає чи підтримує потік пошук;

CanTimeout - визначає чи можна для потоку задати час очікування;

CanWrite - визначає чи підтримує потік запис;

Length - повертає довжину потоку (у байтах);

Name - повертає ім'я файлу;

Position - повертає або встановлює віртуальний курсор, що визначає поточне становище в потоці;

ReadTimeout - повертає або встановлює час очікування потоку при читанні;

WriteTimeout - повертає або встановлює час очікування потоку при записі.

**Члени переліку FileMode**

Append - відкриває файл і переміщає покажчик FileStream в кінець файлу.

WriteCreate - створює новий файл. Якщо файл існує, то він перезаписується.

CreateNew - створює новий файл.

Open - відкриває існуючий файл.

OpenOrCreate - відкриває існуючий файл. Якщо файл не існує, то створюється новий файл.

Truncate - відкриває існуючий файл і обмежує його до нульового розміру.

**Клас StreamReader**

Клас StreamReader надає базову функціональність для читання даних з екземпляра класу похідного від Stream.

Властивості StreamReader:

BaseStream - повертає потік, пов'язаний з об'єктом читання;

CurrentEncoding - повертає кодування потоку, пов'язаного з об'єктом читання;

EndOfStream - визначає чи досягнуто при читанні кінець потоку.

Методи StreamReader:

Close - закриває об'єкт читання і відповідний потік;

Peek - повертає наступний символ з потоку, не змінюючи положення курсору;

Read - зчитує з потоку наступний набір символів;

ReadBlock - зчитує з потоку наступний блок символів;

ReadLine - зчитує з потоку наступний рядок символів;

ReadToEnd - зчитує всі символи до кінця потоку.

Відкрити файл  простіше за все, викликавши відповідний метод класу File і вказавши шлях до файлу в якості параметра. Якщо файл відкривається для читання, член переліку FileMode.Open вказує на те, що файл існує, а член File Access.Read використовується для отримання доступу до файлу лише для читання, як і показано в наступному прикладі:

FileStream theFile = File.Open (@ "C \ boot ini", FileMode.Open, FileAccess.Read);

Метод File.Open повертає об'єкт FileStream. Файловий потік є звичайним потоком, тому проглянути його вміст можна за допомогою методів Read або ReadByte класу Stream. Але класи StreamReader і StreamWriter підтримують механізми, що спрощують операції читання і запису. Як показано в наступному прикладі, для читання файлу достатньо створити об'єкт класу StreamReader.

StreamReader rdr = new StreamReader (theFile);

Console.Write (rd r.ReadToEnd ());

rdr.Close ();

theFile.CloseO;

Клас StreamReader призначений для читання потоку як рядків символів, а не послідовність байтів. Таким чином, всі методи StreamReader для читання повертаються рядки або масивами рядків.

Клас File підтримує додаткові методи, що спрощують відкриття файлу для читання. У попередньому прикладі спочатку був створений об'єкт FileStream, а потім StreamReader. Як показано в цьому фрагменті, клас File може безпосередньо створювати StreamReader викликом методу OpenText:

*StreamReader rdr - File.OpenText (@ "C: \ boot.ini");*

*Console. Write (rdr. ReadToEnd());*

*rdr.CloseO;*

Коли потрібно просто вивести весь файл цілком, можна обійтися єдиним викликом методу ReadAllText класу File. Цей метод приховує всі деталі потоку і реалізацію об'єкта читання:

Console.WriteLine (File.ReadAllText (@ "C; \ boot.ini"));

 Як показано в цьому прикладі, дані вибираються порядково, кожен рядок перевіряється на відповідність умові, це не вимагає завантажувати в пам'ять увесь файл:

StreamReader rdr = File.OpenText (@ "C: \ boot.ini");

/ / Виявивши слово «boot», повідомити користувача

while (! rdr.EndOfStream) {

string line = rdr.ReadLineO;

if (line.Contains ("boot")) {

/ / Виявивши слово «boot», повідомити користувача

/ / І припинити читання файлу.

Console.WriteLine ("Found boot:");

Console.WriteLine (line);

break;

}

rdr.CloseO;

Такий прийом особливо корисний при пошуку в дуже великих файлах.

**Клас StreamWriter**

Клас Stream Writer надає базову функціональність для запису даних у класи, похідні від Stream.

Властиості:

AutoFlush - повертає або встановлює властивість, яка визначає чи будуть модифікації копіюватися у відповідний потік при кожному виклику методу Write;

BaseStream - повертає потік, до якого прив'язаний об'єкт запису;

Encoding - повертає кодування потоку, прив'язаного до об'єкта запису;

NewLine - повертає або встановлює символ кінця рядка.

**Методи:**

Close - закриває об'єкт запису і відповідний потік;

Write - виконує запис у потік;

WriteLine - записує в потік дані та символи кінця рядка.

Щоб мати можливість запису у файл, його потрібно попередньо відкрити для запису. Ця операція аналогічна до відкриття файла для читання.

*FileStream theFile = File.Create (@ "c: \ somefile.txt");*

Навідміну від коду, що відкриває файл для читання, цей код створює новий файл, повертаючи об'єкт FileStream, готовий для запису. Отримавши об'єкт FileStream, можна писати дані прямо в потік. Однак для запису даних у новий файл часто використовується об'єкт StreamWriter:

*StreamWriter writer = new StreamWriter (theFile);*

*writer.WriteLine("Hello");*

*writer. Close();*

*theFile Close();*

Об'єкт StreamWriter дозволяє записувати текст прямо в новий файл. Така конструкція дуже схожа на таку, як для читання файлу. Клас File створює об'єкт StreamWriter при виклику методу CreateText:

*StreamWriter writer = File.CreateText (@ "c: \ somefile.txt");*

*writer.WriteLine ("Hello");*

*writer. Close();*

Клас File також підтримує метод WriteAllText, що записує рядок у новий файл:

*File.WriteAllText (@ "c: \ somefile.txt", "Hello");*

Ця операція дуже проста, але іноді потрібно записати дані в існуючий файл. Запис в існуючий файл виконується практично так само, але файл необхідно відповідним чином відкрити. Щоб відкрити файл для запису, можна скористатися методом Open класу File, вказавши в параметрах, що необхідно записати дані:

*FileStream theFile = null;*

*theFile = File.Open (@ "c: \ somefile.txt", FileMode.Open, FileAccess.Write);*

Метод OpenWrite класу File дозволяє обійтися меншою кількістю коду для тієї ж операції. Виклик методу Open класу File з параметром, що задає доступ для запису, можна замінити таким кодом:

*theFile = File.OpenWrite (@ "c: \ somefile txt");*

Цей код працює, тільки якщо файл існує. Але часто потрібно відкрити існуючий файл або створити новий. На жаль, метод OpenWrite тільки відкриває існуючі файли. Звичайно, можна написати код, який перевіряє існування файлу і створює файл у разі його відсутності, але, на щастя, метод Open класу File підтримує параметр, який вказує, що потрібно зробити - створити файл або відкрити його:

theFile = File.Open (@ "c: \ somefile txt", FileMode.OpenOrCreate, FileAccess Write);

XML файли

XML (*Extensible Markup Language) –* це розширювана мова розмітки, запропонований консорціумом World Wide Web ([W3C](http://uk.wikipedia.org/wiki/W3C)), стандарт побудови [мов розмітки](http://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D1%80%D0%BE%D0%B7%D0%BC%D1%96%D1%82%D0%BA%D0%B8_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85) ієрархічно структурованих [даних](http://uk.wikipedia.org/wiki/%D0%94%D0%B0%D0%BD%D1%96) для обміну між різними [застосунками](http://uk.wikipedia.org/wiki/%D0%97%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA), зокрема, через [Інтернет](http://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82). XML-документ складається із текстових знаків, і придатний для читання людиною.

Стандарт XML визначає набір базових [лексичних](http://uk.wikipedia.org/wiki/%D0%9B%D0%B5%D0%BA%D1%81%D0%B8%D0%BA%D0%B0) та [синтаксичних](http://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81) правил для побудови мови описання інформації шляхом застосування простих *тегів*. Цей формат достатньо гнучкий для того, щоб бути придатним для застосування в різних галузях. Іншими словами, запропонований стандарт визначає [метамову](http://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%B0%D0%BC%D0%BE%D0%B2%D0%B0), на основі якої, шляхом запровадження обмежень на структуру та зміст документів, визначаються специфічні, предметно-орієнтовані мови розмітки даних.

Приклад XML-документа:

<?xml version="1.0"?>

<note>

<to>**Tove**</to>

<from>**Jani**</from>

<heading>**Reminder**</heading>

<body>**Don't forget me this weekend!**</body>

</note>

Для роботи з XML застосовуються XML-парсери. Існує два основних типи парсерів: Simple API for XML (SAX) і Document Object Model (DOM). SAX заснований на курсор і події, що виникають при проході по вузлах XML-документа. SAX-прасеру не потрібно великої кількості пам'яті для розбору навіть великих документів (тому йому непотрібно завантажувати в пам'ять весь документ), але його істотним обмеженням є те, що можна переміщатися по документу тільки в одному напрямку. DOM повністю завантажує документ в пам'ять і представляє його у вигляді дерева, тому можна довільно переміщатися по XML-документу.

Технологія .NET нерозривно пов'язана із XML. А значить, XML добре підтримується з боку Framework Class Library. Класи для роботи з XML зібрані в просторі імен System.Xml. В .NET підтримуються наступні технології:

* XML 1.0
* простір імен XML
* XSD-схеми
* вираження XPath
* XSL-перетворення
* DOM Level 1 Core, DOM Level 2 Core

**Створення XML-документів**

Для створення нових XML-документів застосовується клас *XmlTextWriter*. Він забезпечує швидке небуферизоване створення XML-документів та їх запис у файли, потоки.  Щоб використовувати цей клас, потрібно створити новий об'єкт XmlTextWriter. Потім додати шматочки XML до об'єкта. Існують методи для додавання кожного типу елемента в XML-файл. Нижче наведені деякі з цих методів:

|  |  |
| --- | --- |
| **МЕТОД** | **ОПИС** |
| WriteStartDocument | Записує XML-декларацію з версією "1.0". |
| WriteEndDocument | Закриває всі відкриті елементи чи атрибути. |
| Close | Закриває потік. |
| WriteDocType | Записує DOCTYPE декларації із вказаним ім'ям і додатковими атрибутами. |
| WriteStartElement | Записує вказаний тег. |
| WriteEndElement | Закриває один елемент. |
| WriteFullEndElement | Закриває один елемент. |
| WriteElementString | Записує елемент, що містить строкове значення. |
| WriteStartAttribute | Записує початок атрибута. |
| WriteEndAttribute | Закриває попередній виклик WriteStartAttribute. |
| WriteRaw | Записує вихідну розмітку безпосередньо з рядка. |
| WriteString | Записує рядок. |
| WriteAttributes | Записує атрибут із заданим значенням. |
| WriteCData | Записує <! [CDATA [...]]> блок, що містить вказаний текст. |
| WriteComment | Записує коментар <!--...-->, містить вказаний текст. |
| WriteWhitespace | Записує пробіли. |
| WriteProcessingInstruction | Записує посібник обробки з пропуском між ім'ям і текст наступним чином: <ім'я тексті?>. |

Розглянемо функцію для збереження у файл даних.

String filename = "test.xml";

XmlTextWriter writer = null;

writer = new XmlTextWriter("test.xml", null);

//Використання відступів для зручності

writer.Formatting = Formatting.Indented;

//Початок запису XML

writer.WriteStartDocument();

//Записати ProcessingInstruction вузол.

String PItext = "type='text/xsl' href='book.xsl'";

writer.WriteProcessingInstruction("xml-stylesheet", PItext);

//Записати DocumentType вузла.

writer.WriteDocType("book", null, null, "<!ENTITY h 'hardcover'>");

//Записати коментарь.

writer.WriteComment("sample XML");

//Запис головного елемнта.

writer.WriteStartElement("book");

//Запис genre атрибута.

writer.WriteAttributeString("genre", "novel");

//Запис ISBN атрибута.

writer.WriteAttributeString("ISBN", "1-8630-014");

//Запис елемента title.

writer.WriteElementString("title", "The Handmaid's Tale");

//Запис елемента title.

writer.WriteStartElement("title2");

writer.WriteString("Pride And Prejudice");

writer.WriteEndElement();

//Запис style елемента.

writer.WriteStartElement("style");

writer.WriteEntityRef("h");

writer.WriteEndElement();

//Запис елемента price.

writer.WriteElementString("price", "19.95");

//Запис CDATA.

writer.WriteCData("Prices 15% off!!");

//Закриття тегів

writer.WriteEndElement();

writer.WriteEndDocument();

//Запис у файл і закриття.

writer.Flush();

writer.Close();

Створюючи об'єкт XmlTextWriter, в його конструктор передається ім'я файлу, в який будуть зберігатися XML-дані та кодування.  Виклик методу WriteStartDocument () записує рядок з оголошенням версії XML і типом кодування. У нашому випадку це буде <? XML-версія = "1.0" кодування = "UTF-16"?>.   Пари функцій WriteStartElement () і WriteEndElement () записують початок і кінець тега з вказаною назвою.  Для створення коректного XML-документа кожному виклику функції WriteStartElement () повинен відповідати виклик WriteEndElement (). Перевантажені версії функції WriteStartElement () дозволяють вказати для тега простір імен і префікс. Для вказівки атрибута тега застосовується функція WriteAttributeString (). За допомогою перевантажених версій функції також можна вказати префікс і простір імен для атрибута. Функція WriteEndDocument () застосовується для захисту від випадкових помилок створення XML-документа - вона закриває всі відкриті теги і атрибути. Результат виконання програми:

<?xml version="1.0"?>

<?xml-stylesheet type='text/xsl' href='book.xsl'?>

<!DOCTYPE book[<!ENTITY h 'hardcover'>]>

<!--sample XML-->

<book genre="novel" ISBN="1-8630-014">

<title>The Handmaid's Tale</title>

<title2>Pride And Prejudice</title2>

<style>&h;</style>

<price>19.95</price><![CDATA[Prices 15% off!!]]></book>

**Читання документа XML – документу.**

Клас XmlTextReader забезпечує швидке односпрямоване читання потоку XML-даних. Дані можуть бути отримані з файлу, об'єкта потоку Stream або об'єкта TextReader. XmlTextReader зазвичай застосовується, якщо потрібно зчитати XML-документ і отримати з нього дані. XmlTextReader не завантажує весь документ в пам'ять, він є найкращим вибором при обробці великих файлів. Приклад зчитування XML-файлу:

XmlTextReader xmlReader = new XmlTextReader(filename);

string sNode;

while (xmlReader.Read())

{

sNode = xmlReader.Name;

// Console.WriteLine(sNode);

if (sNode == "book")

{

if (xmlReader.HasAttributes)

{

Console.WriteLine(xmlReader.GetAttribute("genre"));

Console.WriteLine(xmlReader.GetAttribute("ISBN"));

}

}

if (sNode == "title")

{

Console.WriteLine(xmlReader.ReadElementContentAsString());

}

}

# У наступному прикладі показано читання XML-файла і відображення кожного з вузлів. XmlNodeType - вказує тип вузла.

XmlTextReader xmlReader = new XmlTextReader(filename);

while (xmlReader.Read())

{

switch (xmlReader.NodeType)

{

//т елемент

case XmlNodeType.Element:

Console.Write("<{0}>", xmlReader.Name);

break;

// текст вузла

case XmlNodeType.Text:

Console.Write(xmlReader.Value);

break;

//розділ CDATA

case XmlNodeType.CDATA:

Console.Write("<![CDATA[{0}]]>", xmlReader.Value);

break;

// інструкція по обробці (наприклад <? Текст ?>)

case XmlNodeType.ProcessingInstruction:

Console.Write("<?{0} {1}?>", xmlReader.Name, xmlReader.Value);

break;

// коментарь

case XmlNodeType.Comment:

Console.Write("<!--{0}-->", xmlReader.Value);

break;

//оголошення XML

case XmlNodeType.XmlDeclaration:

Console.Write("<?xml version='1.0'?>");

break;

//Обєкт документа – є вершиною дерева елементів. Представляє доступ до всього документа.

case XmlNodeType.Document:

break;

//Оголошення типу документа

case XmlNodeType.DocumentType:

Console.Write("<!DOCTYPE {0} [{1}]", xmlReader.Name, xmlReader.Value);

break;

//Силка на сутність ( наприклад &num;)

case XmlNodeType.EntityReference:

Console.Write(xmlReader.Name);

break;

//Тег кінцевого елементу

case XmlNodeType.EndElement:

Console.Write("</{0}>", xmlReader.Name);

break;

}

}

**Читання документа за допомогою XmlDocument.**

Клас XmlDocument реалізує W3C Document Object Model (DOM) Level 1 Core та основних DOM Level 2. XmlDocument найбільш корисний у тому випадку, якщо потрібно завантажити XML-документ в пам'ять для того, щоб змінити атрибути вузлів, додати або видалити нові елементи. DOM представляє XML-документ як дерево, що зберігається в пам'яті.

Клас XmlDocument  добре використовувати для XML-файлів такого типу:

<books>

<book publisher="Addison-Wesley" name="Design Patterns" />

<book publisher="Addison-Wesley" name="The Pragmatic Programmer" />

<book publisher="Addison-Wesley" name="Test Driven Development" />

<book publisher="Addison-Wesley" name="Refactoring to Patterns" />

<book publisher="O'Reilly Media" name="The Cathedral & the Bazaar" />

<book publisher="O'Reilly Media" name="Unit Test Frameworks" />

</books>

Приклад роботи з XmlDocument:

XmlDocument doc = new XmlDocument();

doc.PreserveWhitespace = true;

//Завантаження файлу.

doc.Load(filename);

XmlNodeList List = doc.GetElementsByTagName("book");

foreach (XmlNode node in List)

{

Console.WriteLine(node.Attributes["genre"].Value);

Console.WriteLine(node.Attributes["ISBN"].Value);

}

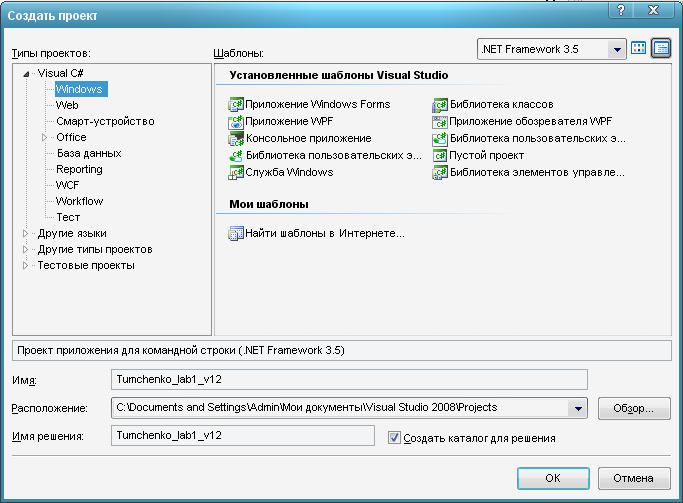
**Задача**

Розробити програму, а в ній клас, котрий:

1. записує у файл інформацію про диски (тип, формат вільний та заповнений об’єм пам’яті) комп’ютера та папки на дисках;
2. читає стрічку під’єднання до БД за технологією ADO.Net з файлу, за допомогою класу *StreamReader* всі записи з БД записати у файл, за допомогою класу *StreamWriter* всі записи з БД, що шукаються по фрагментах тексту, які виконуються по всіх полях таблиці, записати у окремі файли. За допомогою класу *StreamReader* зчитати всі створені файли і відобразити на консоль. Завдання вибирається за номером студента в журналі по лабораторних роботах.
3. Розробити програму, а в ній клас, котрий читав стрічку під’єднання до БД за технологією ADO.Net з xml-файлу, за допомогою класу *XmlTextWriter.* Всі записи з БД записати у xml-файл за допомогою класу *XmlTextWriter.* Всі записи з БД, що шукаються по фрагментах тексту, які виконуються по всіх полях таблиці, записати у окремі xml-файли. За допомогою класу *XmlTextReader* зчитати всі створені XML-файли і відобразити на консоль. Завдання вибирається за номером студента в журналі по лабораторних роботах.

**Хід роботи.**

1. Ознайомитися з теоретичними відомостями.
2. За номером у журналі вибрати власний варіант.
3. Вручну розробити блок схему програми чи UML діаграм у MS Visio.
4. Запустити Visual Studio 2003/2005/2008. Вибрати в меню **Файл/Создать/Проект,** у вікні, що відкриється, вибрати тип проекту, шаблон консольної програми і ввести назву, яка відповідає прізвищу студента, його варіанту і номеру лабораторної роботи.



1. Додати в проект новий клас, який буде працювати з файлами.
2. Спроектувати структуру файлів, які б містили достатню інформацію для розуміння файлу.
3. Реалізувати програму, клас та файл, який буде містити стрічку під’єднання із вказаним шляхом розміщення БД.
4. Розробити звіт. Звіт має включати номер і назву лабораторної роботи, мету, теоретичні відомості, задачу, розробку структури даних (опис змінних та їх тип), блок-схему чи UML-діаграми (діаграму класів, діаграму прецедентів, діаграма послідовності), код програми, результат виконання програми та висновок.
5. Захистити лабораторну роботу.

**Варіанти завдань до лабораторної роботи**

|  |  |  |
| --- | --- | --- |
| Варіант | Назва інформаційної БД | Поля |
| 1 | Автомобілі | Id, марка автомобіля, рік випуску, тип двигуна, об’єм двигуна, розхід палива на 100 км. |
| 2 | Комп’ютери | Id, материнська плата, ОП, тип ОП, об’єм вінчестера, відеокарта |
| 3 | Кондиціонери | Id, назва, площа кондиціонування, потужність охолодження, потужність обігріву, колір |
| 4 | Фільми | Id, назва, жанр, кіностудія, якість, тривалість |
| 5 | Музичні альбоми | Id, виконавець, назва альбому, жанр, кількість пісень, тривалість |
| 6 | Телевізори | Id, виробник, діагональ, тип телевізора, потужність |
| 7 | Носії інформації  (Flash) | Id, фірма, об’єм, тип, швидкість запису, швидкість читання |
| 8 | Комп’ютерні ігри | Id, назва, жанр, необхідна операційна система, мова інтерфейсу, розробник |
| 9 | Принтери | Id, фірма, модель, тип, швидкість друку, резерв сторінок |
| 10 | Мобільні телефони | Id, модель, дисплей, карта пам’яті, вага, колір |
| 11 | Підприємства | Id, назва, сфера діяльності, адреса, № телефону, кількість працівників |
| 12 | Планшетні ПК | Id, процесор, ОП, пам'ять, дисплей, вага, колір |
| 13 | Програмне забезпечення | Id, назва, версія, фірма виготовлення, призначення, платформа |
| 14 | Фотоапарати | Id, назва, матриця, зум, дисплей, вага |
| 15 | Велосипеди | Id, назва, тип, розмір рами, кількість швидкостей, діаметр коліс |
| 16 | Літаки | Id, назва, тип, швидкість, пасажироміскість, дальність польоту |
| 17 | Локомотиви | Id, назва, тип, швидкість, потужність, робоча маса |
| 18 | Готелі | Id, назва, кількість зірок, ціна за добу, адреса |
| 19 | Зброя | Id, назва, тип, калібр, вага, термін гарантії |
| 20 | Житло | Id, район, кількість кімнат, ціна за квадратний метр, площа |
| 21 | Ресторани | Id, назва, кухня, адреса, години роботи, кількість працівників |
| 22 | Футбол | Id, назва команди, країна, група, ліга, очки |
| 23 | Посуд | Id, тип, матеріал, об’єм, виробник, ціна |
| 24 | Канцтовари | Id, назва канцтоварів, група канцтоварів, ціна, виробник, |
| 25 | Косметика | Id, назва косметичного товару, ціна, фірма, вид косметики |
| 26 | Гелікоптери | Id, назва, тип, швидкість, пасажироміскість, дальність польоту |
| 27 | Відпочинкові бази | Id, назва бази, місто, ціна кімнати за добу, відстань до моря, |
| 28 | Навчальні заклади | Id, назва, рівень акредитації, кількість працівників, кількість учнів |
| 29 | Напої | Id, вид, назва, об’єм, ціна, інгредієнти |
| 30 | Взуття | Id, тип, назва, ціна, виробник, матеріал, колір |

**Вимоги до звітів.**

Варіант лабораторної роботи обирається згідно варіанту лабораторних робіт.

З усіх чотирьох боків аркуша повинні бути залишені поля розміром 20 мм, заповненість сторінки має становити 75% від її загального обсягу. Текст набирають на комп'ютері, шрифтом ґарнітурою Times New Roman. Заголовок розділу, підрозділу від попереднього тексту відокремлюють трьома інтервалами. Відступ від назви підрозділу (пункту, підпункту) до наступного тексту повинен складати півтора інтервали. Назви розділів та підрозділів для зручності читання потрібно виділяти напівжирним шрифтом, текст звіту вирівнювати по ширині, а визначення понять у тексті та термінів – курсивом.

Звіт має включати номер і назву лабораторної роботи, мету, теоретичні відомості, задачу, розробку структури даних (опис змінних та їх тип), блок-схему чи UML-діаграми (діаграму класів, діаграму прецедентів, діаграма послідовності), код програми, результат виконання програми та висновок.

Звіт здається скріпленим скріпкою у файлику.

**Оцінювання лабораторної роботи.**

Лабораторна робота оцінюється на **5 балів**, коли студент: повністю зробив програму згідно варіанту та оформив звіт згідно вимог; при захисті звіту дає повну, розгорнуту, вичерпну відповідь на питання; здатний здійснювати порівняльний аналіз різних теорій, концепцій, підходів та самостійно робить логічні висновки й узагальнення; здатний висловлювати та аргументувати власне ставлення до альтернативних поглядів на дану тему.

Лабораторна робота оцінюється на **4-3 бали**, коли студент: повністю зробив програму згідно варіанту та оформив звіт згідно вимог; при захисті звіту розкриває зміст питання в цілому правильно, але допускає помилки, посилається на конкретні історичні періоди та факти, неточний у формулюванні термінів і категорій, проте з допомогою викладача він швидко орієнтується і знаходить правильні відповіді

Лабораторна робота оцінюється на **2 бали**, коли студент: повністю зробив програму згідно варіанту та оформив звіт згідно вимог; при захисті звіту відповідь малообґрунтована, неповна; студент лише з допомогою викладача може зрозуміти та виправити свої помилки.

Лабораторна робота оцінюється на **1 бал**, коли студент: повністю зробив програму згідно варіанту та оформив звіт згідно вимог; при захисті звіту у відповіді відсутні належні докази та аргументи, зроблені висновки не відповідають загальноприйнятим, хибні; характер відповіді дає підставу стверджувати, що студент не знає правильної відповіді; допущені грубі помилки й студент не може їх виправити.

При здачі та захисту звіту знімається **по 1 балу** коли студент:

* Не оформив звіт відповідно до вимог.
* Не доробив програму згідно завдання чи варіанту.
* Немає UML-діаграм чи блок-схем, вони не відповідають реалізованій програмі або генеровані автоматизованими засобами проектування.
* В звіті присутній незрозумілий текст.