**Week 11 File and Stream I/O**

**Namespace : System.IO**

ใน .NET Framework นั้นจะมี namespaces ที่ชื่อ System.IO ซึ่งข้างในจะมีเมธอดที่ช่วยให้อ่านและเขียนข้อมูลบนไฟล์ได้ทั้งแบบ synchronous และ asynchronous นอกจากนั้น ใน namespaces นี้ยังประกอบด้วย ฟังก์ชันที่ทำการบีบอัดและขยายข้อมูล (compression and decompression) ที่อยู่บนแฟ้มข้อมูล ฟังก์ชันที่ช่วยให้การสื่อสารผ่านท่อ (pipe) เช่นไฟล์บนระบบเครือข่าย และพอร์ตอนุกรม เป็นต้น

ไฟล์เป็นที่เก็บข้อมูลระดับไบต์ที่เรียงต่อกัน ซึ่งสามารถเก็บข้อมูลได้แม้จะปิดเครื่องคอมพิวเตอร์ไปแล้วก็ตาม เมื่อเราทำงานกับไฟล์ จะมีสิ่งที่ต้องทำควบคู่กันเสมอคือ directory path, disk storage, file name และ directory name ในการอ่านและเขีนไบต์ข้อมูล จะมีการเรียงลำดับของข้อมูลที่ไหลเข้า-ออกจากไฟล์เหมือนสายน้ำ เราเรียกลักษณะดังกล่าวว่า stream ซึ่งปลายทางของ stream สามารถเป็นได้ทั้ง memory, disk storage, serial port, network files หรือ pipe stream

**Files and Directories**

เราสามารถใช้คลาสต่าง ๆ ใน System.IO namespace เพื่อทำงานกับไฟล์และไดเร็กทอรี ตัวอย่างเช่นสามารถอ่านและตั้งค่าคุณสมบัติ (properties) สำหรับไฟล์และไดเร็กทอรี และสืบค้นรายการของไฟล์และไดเร็กทอรีตามเกณฑ์การค้นหา

**คลาสทั่วไปสำหรับทำงานกับไฟล์และไดเร็กทอรี**

|  |  |
| --- | --- |
| **ชื่อคลาส** | **รายละเอียด** |
| File | จัดเตรียม static method ที่ใช้สร้าง (creating) คัดลอก (copying) ลบ (deleting) ย้าย (moving) และเปิดไฟล์ (opening files) และช่วยสร้างออบเจกต์ FileStream |
| FileInfo | จัดเตรียม instance method ที่ใช้สร้าง คัดลอก ลบ ย้าย และเปิดไฟล์ และช่วยสร้างออบเจกต์ FileStream |
| Directory | จัดเตรียม static method ที่ใช้สร้าง ย้าย และแจกแจงผ่านไดเร็กทอรีและไดเรกทอรีย่อย |
| DirectoryInfo | จัดเตรียม instance method ที่ใช้สร้าง ย้ายและแจกแจงผ่านไดเร็กทอรีและไดเรกทอรีย่อย |
| Path | จัดเตรียม methods และ properties สำหรับการประมวลผลสตริงไดเรกทอรีในลักษณะข้ามแพลตฟอร์ม |

ในการดำเนินการต่าง ๆ เกี่ยวกับระบบไฟล์อาจมีข้อผิดพลาดขณะทำงาน (exception) เกิดขึ้นได้เสมอ ดังนั้นเราควรจัดเตรียม exception handling ที่มีประสิทธิภาพไว้รอบรับเสมอ

**Streams**

Streams เป็น abstract base class รองรับการอ่านและเขียนข้อมูลชนิดไบต์ คลาสทั้งหมดในภาษา C# ที่มีความสามารถด้าน stream ล้วนแล้วแต่สืบทอดจากคลาส Streams โดยจะมีการปกปิดรายละเอียดบางส่วนและเปิดเผยรายละเอียดบางส่วนตามหลัก OOP ซึ่งจะช่วยให้นักพัฒนาโปรแกรมสนใจและมุ่งเน้นไปที่การทำงานกับไฟล์ โดยไม่ต้องสนใจรายละเอียดเฉพาะของระบบปฏิบัติการหรืออุปกรณ์ที่เกี่ยวข้อง (นอกจากไฟล์ธรรมดาแล้ว stream ยังสามารถทำงานกับไฟล์บนระบบ network หรือ serial port ได้อีกด้วย) โดยstream เกี่ยวข้องกับการดำเนินงานพื้นฐาน 3 อย่าง ได้แก่

|  |  |
| --- | --- |
| การอ่าน (reading) | การโอนย้ายข้อมูลจาก stream ไปยังโครงสร้างข้อมูล เช่น อาร์เรย์ของข้อมูลชนิดไบต์ |
| การเขียน (writing) | การถ่ายโอนข้อมูลจากแหล่งข้อมูลไปยังสตรีม |
| การค้นหา (seeking) | การสอบถามตำแหน่งและแก้ไขข้อมูลที่ตำแหน่งปัจจุบันภายในสตรีม |

**Readers and Writers**

เนมสเปซ System.IO มีความสามารถในการอ่านอักขระที่เข้ารหัส (encoded charactors) จาก stream และเขียน (writing) ลงใน stream โดยปกติ stream ได้รับการออกแบบมาสำหรับการเขียนและอ่านข้อมูลชนิด byte ซึ่งมีตัวเขียน (reader) และตัวอ่าน (writer) คอยดำเนินการแปลงอักขระที่เข้ารหัสมาจากต้นทางเพื่อให้ stream สามารถดำเนินการต่อได้ เราสามารถดึงค่า properties ของ reader และ writer ของสตรีมผ่านทาง property ที่ชื่อว่า BaseStream

ตัวเขียนและตัวอ่าน ที่นิยมใช้งาน ได้แก่

|  |  |
| --- | --- |
| BinaryReader และ BinaryWriter | สำหรับการอ่านและเขียนข้อมูลชนิดดั้งเดิมเป็นค่าไบนารี |
| StreamReader และ StreamWriter | สำหรับการอ่านและเขียน charactor โดยใช้การเข้ารหัสเพื่อแปลงระหว่าง charactor และ byte |
| StringReader และ StringWriter | สำหรับการอ่านและการเขียน charactor จาก string |
| TextReader และ TextWriter | ทำหน้าที่เป็น abstract base class สำหรับ reader และ writer ที่อ่านและเขียน charactor และ string ที่ไม่ใช่ข้อมูลไบนารี |

**Asynchronous I/O Operations** การอ่านหรือเขียนข้อมูลเป็นจำนวนมากจะใช้ทรัพยากรมากตามไปด้วย เพื่อป้องกันระบบตกอยู่ในสภาวะวิกฤติเนื่องจากทรัพยากรเหลือในจำนวนที่จำกัด เราควรดำเนินการดังกล่าวแบบอะซิงโครนัส (asynchronous) โดยเฉพาะในกรณีที่ application ต้องการตอบสนองต่อผู้ใช้เป็นหลัก

ในการดำเนินการ I/O แบบอะซิงโครนัสตามปกตินั้น เธรด UI มีโอกาสที่จะถูกบล็อคจนกว่าการดำเนินการใช้ทรัพยากรจะเสร็จสมบูรณ์ (ในที่นี้คือการอ่านเขียนไฟล์) เมธอดแบบอะซิงโครนัสจะมีคำว่า Async ในชื่อของมันเองเช่น CopyToAsync, FlushAsync, ReadAsync และ WriteAsync โดยปกติเราต้องใช้เมธอดเหล่านี้ร่วมกับคีย์เวิร์ด async และ await

**Compression** การบีบอัด (data compression) หมายถึงกระบวนการในการลดขนาดของไฟล์ที่เก็บบนสื่อเก็บข้อมูล การขยายหรือคลายข้อมูล (decompression) เป็นกระบวนการในการขยาย (extracting) เนื้อหาที่บรรจุในไฟล์ เพื่อให้อยู่ในรูปแบบที่ใช้งานได้ โดยที่เนมสเปซ System.IO.Compression จะบรรจุความสามารถในการบีบอัดและคลายข้อมูลในการทำงานร่วมกับไฟล์  
 คลาสที่ใช้งานบ่อยสำหรับการบีบอัดและคลายข้อมูล ได้แก่

|  |  |
| --- | --- |
| ZipArchive | สำหรับการสร้างและอ่านข้อมูลทั้งหมดที่เก็บใน zip ไฟล์ |
| ZipArchiveEntry | สำหรับการนำเสนอไฟล์บีบอัด |
| ZipFile | สำหรับการสร้าง การแตกไฟล์ และการเปิด compressed package |
| ZipFileExtensions | สำหรับการสร้างและการแตกไฟล์ทั้งหมดใน compressed package |
| DeflateStream | สำหรับการ compressing และ decompressing streams โดยใช้อัลกอริทึมแบบ Deflate |
| GZipStream | สำหรับการ compressing และ decompressing streams ใน gzip data format |

**การทดลอง 11.1 การใช้งานไฟล์เบื้องต้นด้วยคลาส File**

การทดลองนี้จะทำการสร้างไฟล์ขึ้นมาหนึ่งไฟล์บน harddisk จากนั้นจะทำการเขียนข้ความบางอย่างลงไปในไฟล์ และอ่านข้อความออกมาแสดงบนหน้าจอ console

**ลำดับการทดลอง**

1. สร้างโปรเจคแบบ Console
2. แก้ไขโปรแกรมให้เป็นดังต่อไปนี้

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.IO;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace FileStream1  {  class Program  {  static void Main(string[] args)  {  string path = @"c:\complab1\MyTest.txt";  if (!File.Exists(path))  {  // Create a file to write to.  using (StreamWriter sw = File.CreateText(path))  {  sw.WriteLine("Hello");  sw.WriteLine("And");  sw.WriteLine("Welcome");  }  }  }  }  } |

1. ทดลองรันโปรแกรม สังเกตุผลที่เกิดขึ้น บันทึกผลที่ได้
2. จากตัวอย่างโปรแกรม พบว่าจะเกิด Exception System.IO.DirectoryNotFoundException: 'Could not find a part of the path 'c:\complab1\MyTest.txt'.' เนื่องจากในเครื่องคอมพิวเตอร์ยังไม่มี path ที่ชื่อว่า 'c:\complab1\' ดังนั้น ให้ไปสร้าง folder ชื่อ “complab1” ในไดรว์ C:
3. หลังจากสร้าง folder แล้วให้กดรันอีกครั้ง โดยการรันแบบ Start Without Debugging หรือกด Ctrl+F5 บันทึกผลการทดลอง
4. โปรแกรมจะสร้าง text file ที่ชื่อ MyText.txt ขึ้นในโฟลเดอร์ 'c:\complab1\' ให้ใช้ notepad หรือโปรแกรม text editor เปิดดูเนื้อหาในไฟล์ดังกล่าว บันทึกผล
5. เพื่อให้สามารถดูเนื้อหาภายในไฟล์ ให้แก้ไขโปรแกรมให้เป็นดังต่อไปนี้

|  |
| --- |
| using System; using System.Collections.Generic; using System.IO; using System.Linq; using System.Text; using System.Threading.Tasks;  namespace FileStream1 {  class Program  {  static void Main(string[] args)  {  string path = @"c:\complab1\MyTest.txt";   // Open the file to read from.  using (StreamReader sr = File.OpenText(path))  {  string s;  while ((s = sr.ReadLine()) != null)  {  Console.WriteLine(s);  }  }  }  } } |

1. ทดลองรันโปรแกรม สังเกตุผลที่เกิดขึ้น บันทึกผลที่ได้

**การทดลอง 11.2 FileAttributes**

ในการใช้งานไฟล์ เราอาจจะต้องรู้ attibute ของไฟล์นั้น ๆ เพื่อให้เกิดความมั่นใจว่าการกระทำใด ๆ ต่อไฟล์นั้นจะไม่เกิดผลที่ไม่คาดคิด เช่นการเขียนข้อมูลลงในไฟล์ที่เป็น readonly เป็นต้น

Fields ต่าง ๆ ที่เป็น attibute ของไฟล์

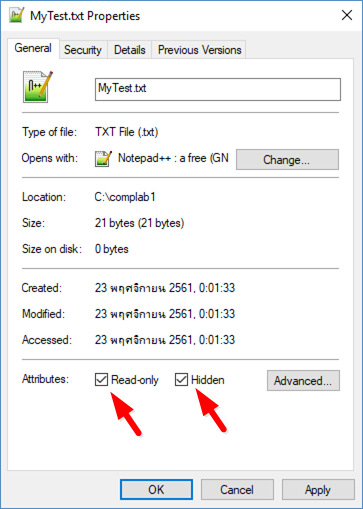
|  |  |  |
| --- | --- | --- |
| attibute | รหัส | ความหมาย |
| Archive | 32 | The file is a candidate for backup or removal. |
| Compressed | 2048 | The file is compressed. |
| Device | 64 | Reserved for future use. |
| Directory | 16 | The file is a directory. |
| Encrypted | 16384 | The file or directory is encrypted. For a file, this means that all data in the file is encrypted. For a directory, this means that encryption is the default for newly created files and directories. |
| Hidden | 2 | The file is hidden, and thus is not included in an ordinary directory listing. |
| IntegrityStream | 32768 | The file or directory includes data integrity support. When this value is applied to a file, all data streams in the file have integrity support. When this value is applied to a directory, all new files and subdirectories within that directory, by default, include integrity support. |
| Normal | 128 | The file is a standard file that has no special attributes. This attribute is valid only if it is used alone. |
| NoScrubData | 131072 | The file or directory is excluded from the data integrity scan. When this value is applied to a directory, by default, all new files and subdirectories within that directory are excluded from data integrity. |
| NotContentIndexed | 8192 | The file will not be indexed by the operating system's content indexing service. |
| Offline | 4096 | The file is offline. The data of the file is not immediately available. |
| ReadOnly | 1 | The file is read-only. |
| ReparsePoint | 1024 | The file contains a reparse point, which is a block of user-defined data associated with a file or a directory. |
| SparseFile | 512 | The file is a sparse file. Sparse files are typically large files whose data consists of mostly zeros. |
| System | 4 | The file is a system file. That is, the file is part of the operating system or is used exclusively by the operating system. |
| Temporary | 256 | The file is temporary. A temporary file contains data that is needed while an application is executing but is not needed after the application is finished. File systems try to keep all the data in memory for quicker access rather than flushing the data back to mass storage. A temporary file should be deleted by the application as soon as it is no longer needed. |

**การทดลอง 11.2.1 การอ่านค่า Attributes ของไฟล์**

1. เขียนโปรแกรมตามตัวอย่างต่อไปนี้

|  |
| --- |
| using System; using System.IO;  namespace ConsoleApplication {  class Program  {  static void Main(string[] args)  {  string path = "c:/complab1/MyTest.txt";  FileAttributes attributes = File.GetAttributes(path);  Console.WriteLine("Attribute of " + path + " is " + attributes.ToString());   }  } } |

1. ทดลองรันโปรแกรม สังเกตุผลที่เกิดขึ้น บันทึกผลที่ได้
2. เปิด file explorer แล้วทดลองเปลี่ยน attribute ของไฟล์ ตามตัวเลือกที่ชี้ด้วยลูกศรสีแดงดังตัวอย่าง



3.1 [ ] Read-only = checked, [ ] Hidden = unchecked กดปุ่ม OK แล้วรันโปรแกรมอีกครั้ง บันทึกผล

3.2 [ ] Read-only = unchecked, [ ] Hidden = checked

3.3 [ ] Read-only = checked, [ ] Hidden = checked

**หมายเหตุ** เมื่อเรากำหนด attribute ของไฟล์เป็น hidden จะไม่สามารถมองเห็นไฟล์ได้จาก browser ตามปกติ ให้ตั้งค่า browser ให้แสดงไฟล์ที่ซ่อนไว้

**การทดลอง 11.2.2 การกำหนดค่า Attributes ของไฟล์**

การกำหนดค่า attribute ของไฟล์จากโปรแกรมช่วยให้เราสามารถกำหนด attribute ได้จากโปรแกรม ซึ่งจะมีความสะดวกกว่ากำหนดจาก file browser

1. เขียนโปรแกรมต่อไปนี้

|  |
| --- |
| using System;  using System.IO;  namespace ConsoleApplication  {  class Program  {  static void Main(string[] args)  {  string path = "c:/complab1/MyTest.txt";  FileAttributes attributes = File.GetAttributes(path);  if ((attributes & FileAttributes.Hidden) == FileAttributes.Hidden)  {  // Show the file.  attributes &= ~FileAttributes.Hidden; // remove hidden attribute  File.SetAttributes(path, attributes);  Console.WriteLine("The {0} file is no longer hidden.", path);  }  else  {  // Hide the file.  File.SetAttributes(path, File.GetAttributes(path) | FileAttributes.Hidden);  Console.WriteLine("The {0} file is now hidden.", path);  }  }  }  } |

1. ทดลองรันโปรแกรม อย่างน้อย 3 ครั้ง สังเกตุผลที่เกิดขึ้น บันทึกผลที่ได้จากการรันแต่ละครั้ง โดยสังเกตุทั้งผลการรันและคุณลักษณะของไฟล์ใน file browser

<http://faculty.otterbein.edu/psanderson/comp3400/notes/lecture10.html>

<http://students.cypresscollege.edu/cis218/lc12.pdf>