# Teoretická část - Práce se soubory v jazyce CS

## Úvod

Práce se soubory je důležitou, ne-li klíčovou částí každého programu, který nějakým způsobem ukládá nebo načítá data. Ať už jde o nejrůznější textové, zvukové a grafické editory, které načítají data v souborech již dříve uložená a ukládají data nová, o hry, které načítají textury, dialogy nebo zvukové soubory a ukládají postup hráče, nebo jakýkoli program ukládající svá nastavení, všechny tyto programy musí nějak se soubory pracovat.

Ve své práci se pokusím popsat, jaké jsou možnosti práce se soubory obecně, na které věci se při ní musí dávat pozor a to vše aplikovat na jazyk C#.

### Soubory

Jako soubor[[1]](#footnote-2) [[2]](#footnote-3) označujeme sadu dat uloženou na datovém médiu (disku v počítači, USB disku, DVD), se kterou lze pracovat jako s jedním celkem a která má nějaké jméno.

Soubor může obsahovat jak pouze jeden typ dat (text, zvuk, obrázek), tak i více typů - multimediální soubory (kontejnery) zabalují obraz, zvuk a případně i titulky, kancelářské dokumenty mohou krom textu obsahovat například obrázek.

#### Typy souborů

Soubory můžeme rozdělit na textové a binární.[[3]](#footnote-4) Mezi textové se řadí takové soubory, které můžeme otevřít v jakémkoli textovém editoru a jejich obsah bude uživateli buď naprosto čitelný, nebo v něm alespoň rozezná strukturu souboru a některá klíčová slova (například *html* obsahující text a tagy). Patří sem textové dokumenty jako *txt, markdown* nebo *tex*, zdrojové kódy programů a webových standardů (*html, css, json*), vektorové obrázky formátu *svg*, titulky nebo konfigurační soubory. I když je většina spustitelných souborů binárních, najdou se i výjimky - například dávkový soubor *.bat* systému MS Windows nebo unixový *shell script* jsou textové povahy a jakýmkoli textovým editorem je lze otevřít.

Binární soubory oproti textovým vyžadují speciální program pro jejich otevření. Pokud bychom se jejich obsah pokusili zobrazit v textovém editoru, dostaneme pouze surová data, ze kterých nic nevyčteme. Binárními soubory jsou například (multi)mediální soubory obrázků (vyjma svg), videí a zvuku, kancelářské dokumenty (*PDF*, *.doc*, prezentace *.ppt*), archivy jako *zip, rar, ISO* obrazy nebo spustitelné soubory jako *.exe* a *.dll*.

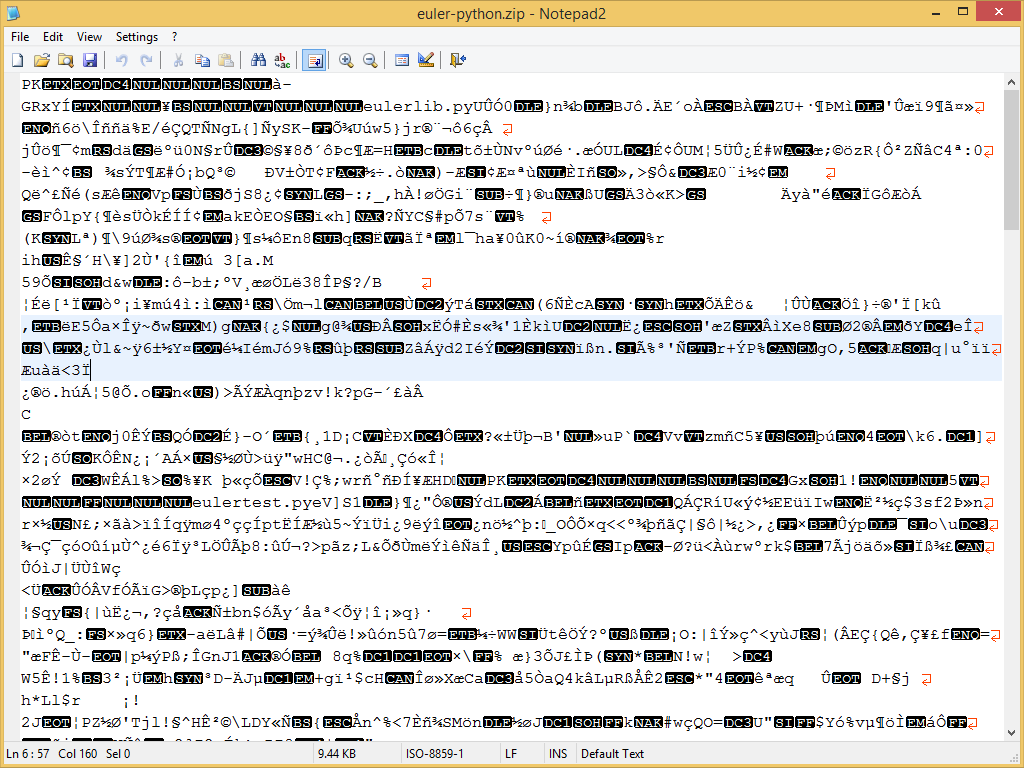
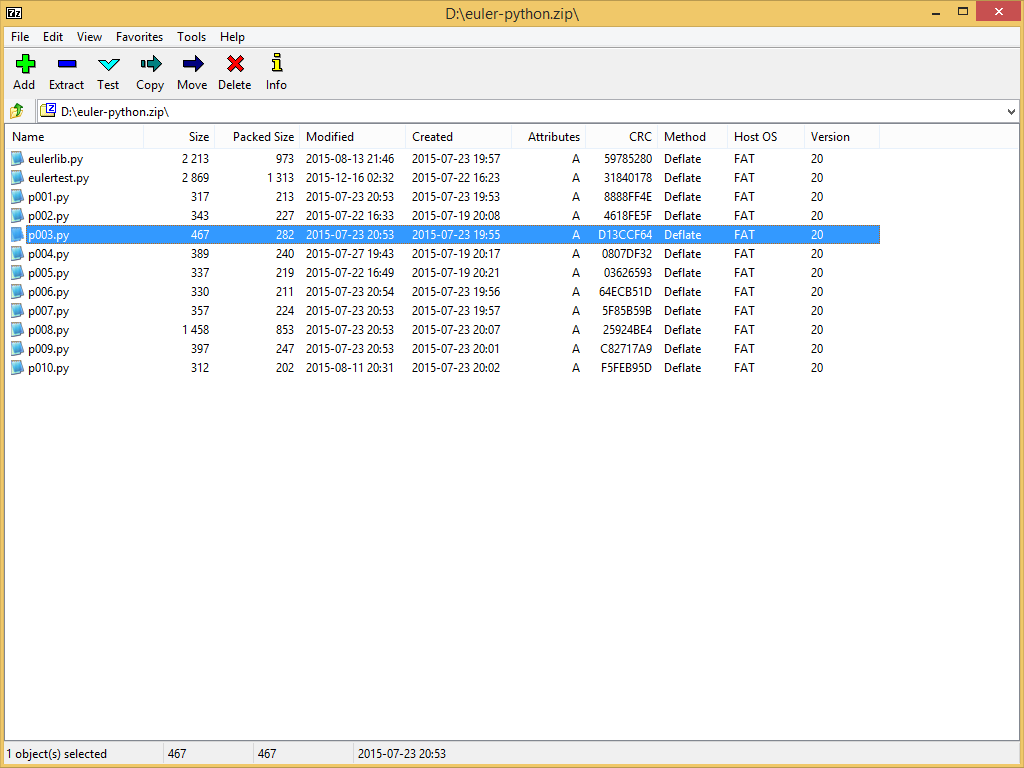


Figure 1: Zobrazení bitového souboru v aplikaci

Figure 2: Zobrazení bitového souboru v textovém editoru

#### Koncovky souborů

Formát souboru a to, zda je textový, nebo binární, můžeme zpravidla určit podle jeho koncovky. Toho můžeme využít kupříkladu tehdy, chceme-li načíst všechny obrázky v adresáři a zobrazit je v galerii - v tomto případě chceme vyloučit všechny dokumenty a jiné nepodporované soubory, jelikož nemáme kontrolu nad tím, co přesně se v adresáři nalézá. Pokud bychom předpokládali, že jsou v adresáři pouze obrázky, a ve skutečnosti by tomu tak nebylo, program by mohl skončit chybou.

##### Odlišnosti systémů v souvislosti s koncovkami

Zatímco software běžící na MS Windows koncovky souborů nutně potřebuje k úspěšnému zpracování souboru (tedy zobrazení jeho obsahu nebo jeho přehrání - výjimku tvoří systémové soubory), linuxové systémy formát souboru určují podle jeho hlavičky. Shell script například nemusí mít koncovku .sh, aby ho systém jako spustitelný script rozpoznal, ale musí pak na první řádce obsahovat sekvenci #!/bin/bash. Pokud ani jednu z těchto podmínek nesplňuje, stává se z něj obyčejný nespustitelný soubor.

Neplatí to však u všech typů souborů - pokud například smažeme koncovku certifikátu *.pem*, systém ho bude mít za soubor programu MATLAB, přestože jeho obsah zůstal nezměněný. (ověřeno lokálním experimentem).

Některé linuxové programy však navrch kontroly hlavičky provádí i kontrolu koncovky souboru, což může vést k neočekávaným chybám a některé soubory nemusí být rozpoznány, přestože jde o správný formát.[[4]](#footnote-5) Příkladem mohl být prohlížeč obrázků *Eye of Gnome*, který nedokázal obrázek otevřít, pokud neměl správnou koncovku. (V nejnovější verzi programu už tato chyba/funkcionalita není.) V některých případech koncovku souboru vyžaduje samotný systém - při zpracování seznamu softwarových zdrojů čte kernel pouze ze souborů s koncovkou .list, aby zamezil načtení nežádoucích dat.

## Práce se soubory obecně

Jakoukoli práci s obsahem souboru můžeme rozdělit podle 3 dílčích fází procesu - otevření, čtení nebo zápis a zavření souboru.

### Otevření

Způsob otevření souboru závisí na tom, jak s ním má být nakládáno. [[5]](#footnote-6) [[6]](#footnote-7) [[7]](#footnote-8)

Read-only mód se používá, chceme-li ze souboru pouze číst, protože se tím minimalizuje riziko poškození souboru. Program v tomto módu totiž nemá přístup k funkcím, jimiž by do souboru zapisoval.

Pokud bychom soubor otevřeli v read-write módu, v němž k těmto funkcím program přístup má, při chybě běhu programu by se do souboru mohla zapsat náhodná data a poškodit jej tak. Tento mód otevření se proto používá v případě, kdy v jedné funkci ze souboru čteme, načtená data zpracujeme a následně je zapisujeme nazpět. Typickým příkladem může být úprava protokolů nebo přepis dat na základě svých hodnot.

Od read-write módu je odvozený append mód, který zápis začíná na konci souboru a text pouze připojuje - předchozí obsah souboru tedy nechává netknutý.

Posledním způsobem otevření je write-only mód, používaný při zápisu do souboru. Vzhledem k tomu, že nenačítá soubor do RAM, ale rovnou zapisuje na určené místo na disku, je mnohem efektivnější než read-write mód, který soubor do paměti načítá. Pokud soubor, do něhož má zapisovat, neexistuje, sám ho vytvoří a začne zápis. Na rozdíl od append módu však začíná zapisovat na začátek souboru, což znamená, že přepíše veškerý jeho obsah.

### Čtení

Čtení probíhá díky datovému bytestreamu[[8]](#footnote-9) [[9]](#footnote-10), který načítá jednotlivé byty tvořící soubor. S tímto streamem můžeme dále pracovat a podle potřeby z něj číst jednotlivé znaky, řádky, bloky o určitém počtu znaků nebo celý obsah streamu - pro jazyk C# je to specificky třída StreamReader, kterou popíši v pozdější části své práce.

### Zápis

Zápis probíhá stejně jako čtení za pomoci bytestreamu, do kterého (v C# pomocí třídy StreamWriter) zapisujeme jednotlivé znaky nebo řádky a který je poté zapsán do paměti počítače na adresu souboru. Možnost zápisu bloku znaků do streamu již na rozdíl od jeho přečtení není na platformě .NET implementována.

### Zavření

Konec souboru se (při čtení) typicky pozná tak, že StreamReader (či jiná třída pro čtení ze streamu) vrátí hodnotu null - na dané pozici streamu už nejsou žádná čitelná data.

Soubor je po skončení práce dobré zavřít z bezpečnostních důvodů.[[10]](#footnote-11) Pokud by program skončil chybou a soubor by byl stále otevřený, mohl by dojít k jeho nechtěnému poškození ať už samotným programem, nebo systémem. Dalším důvodem je fakt, že systém dovoluje programu držet v paměti jen určité množství souborů a pokud by nebyly po skončení práce zavírány, mohlo by dojít k zahlcení a program by skončil chybou. Toto omezení existuje mimo jiné pro případ chyby v programu, který by mohl kupříkladu otevírat tisíce souborů a systém tak zpomalovat.

Různé jazyky implementují zavření souboru různě - konkrétně C# a .NET runtime zavře StreaReader nebo StreamWriter, s nímž se streamem pracoval, čímž následně .NET nad reader/writer objektem zavolá Dispose()[[11]](#footnote-12) a Flush()[[12]](#footnote-13) metody. Ty uvolní systémové prostředky a zajistí, že se do souboru zapíší jakákoli data, která by čekala na zápis ve vyrovnávací paměti[[13]](#footnote-14) (bufferu).

## Práce se soubory v jazyce C#

### Práce s obsahem souborů

#### Using

C# umožňuje použít using statement, jehož výhodou je zavolání Dispose nad instancí, kterou v using definujeme. V praxi to znamená, že po skončení using bloku řeší uvolnění paměti a deskriptorů souboru .NET runtime sám od sebe, kód je tak bezpečnější.[[14]](#footnote-15) Pokud by navíc v bloku došlo k chybě, díky použití using bude nad instancí rovněž Dispose zavoláno a zamezí se tak nežádoucímu chování programu.

Příkladem je užití při jednoduchém načítání dat ze souboru - pokud by při načítání došlo k chybě, instance (zde StreamReader) by nebyla ukončena a prostředky by nebyly uvolněny z paměti. Užitím using se tomu však zamezí.

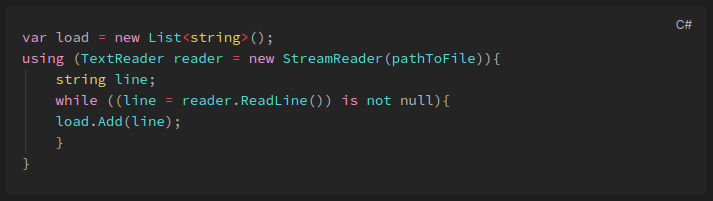


Figure 3: Příklad použití using bloku

Bez použití using bychom museli definovat try - finally blok a ve finally části vynutit ukončení instance. [[15]](#footnote-16)

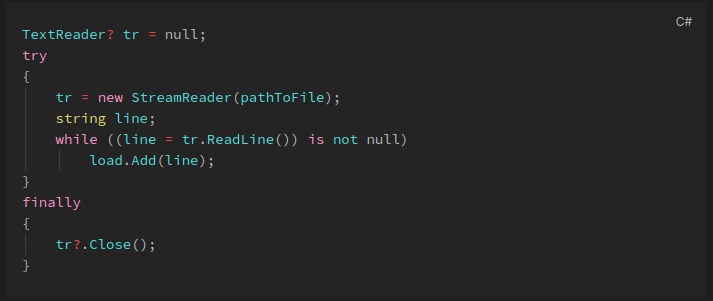


Figure 4: Příklad řešení bez using bloku

#### FileStream

Konstruktor třídy FileStream[[16]](#footnote-17) vrací bytový stream, který se dá použít jak pro čtení, tak pro zápis dat do souboru. Voláme ho s několika parametry: *FileStream(String, FileMode, FileAccess, FileShare)*, případně ještě s velikostí bufferu *Int32*[[17]](#footnote-18). Tím, že pracuje s daty v bytové formě, musí být data (nemá-li s nimi program takto pracovat) po přečtení převedena na text kvůli pozdějšímu zpracování, a to samé platí pro zápis.

##### String

Určuje cestu k otevíranému souboru.

##### FileAccess[[18]](#footnote-19)

Tento parametr určuje přístup streamu k souboru a tím pádem způsob, jak s ním pak můžeme pracovat. Může nabývat těchto hodnot:

- Read - Stream má přístup pouze pro čtení.

- ReadWrite - Stream má přístup pro čtení i zápis.

- Write - Stream má přístup pouze pro zápis.

##### FileMode[[19]](#footnote-20)

Parametr FileMode určuje, jakým způsobem bude soubor otevřen, popřípadě vytvořen, a může mít tyto hodnoty:

- CreateNew - Vytváří nový soubor, do kterého bude posléze zapisováno, a vyžaduje FileAccess.Write. Pokud již soubor existuje, program končí chybou.

- Create - Pokud soubor neexistuje, vytvoří nový, v opačném případě přepíše jeho obsah. Je tedy ekvivalentem funkce, která v případě existence souboru použije mód Truncate (viz níže) a v opačném případě použije CreateNew mód.

- Truncate - V tomto módu FileStream otevírá již existující soubor a veškerý jeho obsah maže. Pokud otevíraný soubor neexistuje, vrací chybu.

- Append - Otevírá soubor a zápis začíná na jeho konci.

- Open - Otevírá již existující soubor, přičemž možný způsob práce s ním dále závisí na atributu FileAccess.

- OpenOrCreate - Pokud soubor neexistuje, konstruktor ho vytvoří, jinak soubor otevře. Vyžaduje atribut FileAccess podle toho, jak má být se souborem nakládáno.

##### FileShare[[20]](#footnote-21)

FileShare parametr kontroluje to, zda budou mít jiné procesy k otevřenému souboru přístup a pokud ano, tak jaký. Metody jazyka C#, které tento konstruktor zapouzdřují a které popisuji níže, ho volají s parametrem FileShare.None, čímž jakémukoli jinému procesu přístup k souboru zamítají. Výjimku tvoří Asynchronní funkce pro čtení a zápis, které mají jako parametr FileShare.Read.

Dalšími možnými hodnotami tohoto parametru jsou Write, ReadWrite, Delete a Inheritable. Hodnota Inheritable uděluje dědičnému procesu stejná oprávnění, jaká má jeho mateřský proces.

##### Zapouzdřující metody

C# obsahuje několik metod, které zapouzdřují volání konstruktoru a předávají mu parametry, které bychom jinak museli při volání konstruktoru ve zdrojovém kódu předávat sami. Vrací nám pak nový objekt typu FileStream.

- File.Create(string)[[21]](#footnote-22) - Tato metoda volá konstruktor s parametry FileMode.Create a odpovídajícími parametry FileAccess. Je třeba jí předat argument cesty k souboru.

- File.Open(string, FileMode)[[22]](#footnote-23) - Metoda volá konstruktor s argumenty předanými programátorem, ale navíc určuje hodnotu parametru FileAccess podle zadané hodnoty FileMode. Je přetížitelná,

- File.OpenRead(string)[[23]](#footnote-24) - Jediná metoda, která krom toho, že volá konstruktor s argumenty umožňujícími pouze čtení souboru, také oproti ostatním metodám argument FileShare nastavuje na Read, což znamená, že více procesů může v jednu chvíli číst ze stejného souboru.

- File.OpenWrite(string)[[24]](#footnote-25) - Tato metoda volá konstruktor s argumentem FileMode.OpenOrCreate a tudíž s ní můžeme jak otevřít již existující soubor, tak vytvořit soubor nový. Obsah souboru je smazán a přepsán obsahem novým.

###### Metody

Většina metod třídy FileStream se používá pouze v případě, kdy stream není začleněn do using statementu. Jak jsem již popsal výše, užití using je v programu bezpečnější, a to mimo jiné i proto, že eliminuje faktor lidské chyby (př. vynechání dispozivní metody) při psaní kódu. V této části se budu věnovat pouze vybraným metodám jednak z důvodu zbytečnosti zbytku metod za předpokladu použití streamu v using statementu a jednak proto, že jsem za celé své studium s třídou FileStream nepracoval a rád bych se více věnoval třídám, se kterými tomu tak bylo.

- Read(Byte[], Int32, Int32)[[25]](#footnote-26) - Tato metoda ze streamu přečte data v rozmezí daném dvěma číselnými hodnotami a zapíše je do pole předaného funkci v argumentu (tedy do bufferu). První číselná hodnota označuje pozici ve streamu, na které má čtení začít, a druhá počet bytů, které mají být přečteny.

- Write(Byte[], Int32, Int32)[[26]](#footnote-27) - Metoda zapisuje do streamu data z předaného bufferu, číselné parametry učujíc počáteční pozici zápisu a maximální zapsatelný počet bytů.

- (Un)Lock(Int64, Int64)[[27]](#footnote-28) [[28]](#footnote-29) - Zpřístupňuje a znepřístupňuje ostatním procesům danou část streamu pro čtení nebo zápis.

Takto může vypadat kód, který nejdříve vytvoří soubor a zapíše do něj text. Text poté přečte a vypíše do konzole.[[29]](#footnote-30)



Figure 5: Příklad užití třídy FileStream

#### StreamReader[[30]](#footnote-31)

Třída implementuje objekt abstraktní třídy TextReader a umožňuje nám číst text z bytového streamu, kterým otevře soubor.

1. Soubor. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2023-04-02]. Dostupné z: https://cs.wikipedia.org/wiki/Soubor [↑](#footnote-ref-2)
2. Potáček, Jiří. soubor. In: \_KTD: Česká terminologická databáze knihovnictví a informační vědy (TDKIV)\_ [online]. Praha : Národní knihovna ČR, 2003- [cit. 2023-03-29]. Dostupné z: https://aleph.nkp.cz/F/?func=direct&doc\_number=000000018&local\_base=KTD. [↑](#footnote-ref-3)
3. What are binary and text files?. *Project Nayuki* [online]. [cit. 2023-04-02]. Dostupné z: https://www.nayuki.io/page/what-are-binary-and-text-files [↑](#footnote-ref-4)
4. Do file extensions have any purpose in Linux?. *AskUbuntu* [online]. [cit. 2023-04-02]. Dostupné z: https://askubuntu.com/a/803451 [↑](#footnote-ref-5)
5. C File Handling. *Programiz* [online]. [cit. 2023-04-02]. Dostupné z: https://www.programiz.com/c-programming/c-file-input-output [↑](#footnote-ref-6)
6. Basics of File Handling in C. *GeeksforGeeks* [online]. [cit. 2023-04-02]. Dostupné z: https://www.geeksforgeeks.org/basics-file-handling-c/ [↑](#footnote-ref-7)
7. Computer Programming - File I/O. *Tutorialspoint* [online]. [cit. 2023-04-02]. Dostupné z: https://www.tutorialspoint.com/computer\_programming/computer\_programming\_file\_io.htm [↑](#footnote-ref-8)
8. What is a byte stream actually?. *StackExchange: Software Engineering* [online]. [cit. 2023-04-02]. Dostupné z: https://softwareengineering.stackexchange.com/a/216600 [↑](#footnote-ref-9)
9. What is the difference between BufferedStream and MemoryStream in terms of application?. *StackOverflow* [online]. [cit. 2023-04-02]. Dostupné z: https://stackoverflow.com/a/1439808 [↑](#footnote-ref-10)
10. Why Is It Important to Close Files in Python?. *RealPython* [online]. [cit. 2023-04-02]. Dostupné z: https://realpython.com/why-close-file-python/ [↑](#footnote-ref-11)
11. MICROSOFT. Implement a Dispose method. *Microsoft Learn* [online]. [cit. 2023-04-02]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/standard/garbage-collection/implementing-dispose [↑](#footnote-ref-12)
12. MICROSOFT. FileStream.Flush Method. *Microsoft Learn* [online]. [cit. 2023-04-02]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.io.filestream.flush?view=net-7.0 [↑](#footnote-ref-13)
13. Vyrovnávací paměť. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2023-04-02]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Vyrovn%C3%A1vac%C3%AD\_pam%C4%9B%C5%A5&oldid=22452896 [↑](#footnote-ref-14)
14. MICROSOFT. Using statement - ensure the correct use of disposable objects. *Microsoft Learn* [online]. [cit. 2023-04-02]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/statements/using [↑](#footnote-ref-15)
15. C# using. *ZetCode* [online]. 6. 2. 2022 [cit. 2023-04-02]. Dostupné z: https://zetcode.com/csharp/using/ [↑](#footnote-ref-16)
16. MICROSOFT. FileStream Class. *Microsoft Learn* [online]. [cit. 2023-04-02]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.io.filestream?view=net-8.0 [↑](#footnote-ref-17)
17. MICROSOFT. FileStream(String, FileMode, FileAccess, FileShare, Int32). *Microsoft Learn* [online]. [cit. 2023-04-02]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.io.filestream.-ctor?view=net-8.0#system-io-filestream-ctor(system-string-system-io-filemode-system-io-fileaccess-system-io-fileshare-system-int32) [↑](#footnote-ref-18)
18. MICROSOFT. FileAccess Enum. *Microsoft Learn* [online]. [cit. 2023-04-02]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.io.fileaccess?view=net-8.0 [↑](#footnote-ref-19)
19. MICROSOFT. FileMode Enum. *Microsoft Learn* [online]. [cit. 2023-04-02]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.io.filemode?view=net-7.0 [↑](#footnote-ref-20)
20. MICROSOFT. FileShare Enum. *Microsoft Learn* [online]. [cit. 2023-04-02]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.io.fileshare?view=net-7.0 [↑](#footnote-ref-21)
21. MICROSOFT. File.Create Method. *Microsoft Learn* [online]. [cit. 2023-04-02]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.io.file.create?view=net-8.0 [↑](#footnote-ref-22)
22. MICROSOFT. File.Open Method. *Microsoft Learn* [online]. [cit. 2023-04-02]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.io.file.open?view=net-8.0 [↑](#footnote-ref-23)
23. MICROSOFT. File.OpenRead(String) Method. *Microsoft Learn* [online]. [cit. 2023-04-02]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.io.file.openread?view=net-8.0 [↑](#footnote-ref-24)
24. MICROSOFT. File.OpenWrite(String) Method. *Microsoft Learn* [online]. [cit. 2023-04-02]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.io.file.openwrite?view=net-8.0 [↑](#footnote-ref-25)
25. MICROSOFT. FileStream.Read Method. *Microsoft Learn* [online]. [cit. 2023-04-02]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.io.filestream.read?view=net-8.0 [↑](#footnote-ref-26)
26. MICROSOFT. FileStream.Write Method. *Microsoft Learn* [online]. [cit. 2023-04-02]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.io.filestream.write?view=net-8.0 [↑](#footnote-ref-27)
27. MICROSOFT. FileStream.Lock(Int64, Int64) Method. *Microsoft Learn* [online]. [cit. 2023-04-02]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.io.filestream.lock?view=net-8.0 [↑](#footnote-ref-28)
28. MICROSOFT. FileStream.Unlock(Int64, Int64) Method. *Microsoft Learn* [online]. [cit. 2023-04-02]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.io.filestream.unlock?view=net-8.0 [↑](#footnote-ref-29)
29. MICROSOFT. FileStream Examples. *Microsoft Learn* [online]. [cit. 2023-04-02]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.io.filestream?view=net-8.0#examples [↑](#footnote-ref-30)
30. MICROSOFT. StreamReader Class. *Microsoft Learn* [online]. [cit. 2023-04-02]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.io.streamreader?view=net-8.0 [↑](#footnote-ref-31)