HexaEditor

C# application

Atelier projet 2017

Serena SADEK – Michael RAMUSI

Table des matières

[1 Introduction 2](#_Toc482703967)

[2 Analyse fonctionnelle 3](#_Toc482703968)

[2.1 Fonctionnalités principales 3](#_Toc482703969)

[2.2 Fonctionnalités remarquables 3](#_Toc482703970)

[2.2.1 Gestion des pages et enregistrement du fichier 3](#_Toc482703971)

[2.2.2 Historique 3](#_Toc482703972)

[2.2.3 Affichage des octets du fichier 4](#_Toc482703973)

[2.3 Interface 5](#_Toc482703974)

[2.3.1 Fenêtre principale 5](#_Toc482703975)

[3 Analyse organique 6](#_Toc482703976)

[3.1 Diagramme de classe 6](#_Toc482703977)

[3.1.1 Classe Reader 6](#_Toc482703978)

[3.1.2 Classe ModelHexaEditor 7](#_Toc482703979)

[3.1.3 Classe ViewHexaEditor 8](#_Toc482703980)

[3.2 Plan de tests 9](#_Toc482703981)

# Introduction

Ce travail est réalisé dans le cadre de l’atelier projet 3ème année accélérée de l’année 2016-2017. **HexaEditor**, comme son nom l’indique, est une application d’édition en hexadécimal de fichiers. Elle permet l’édition octet par octet en hexadécimal mais également directement sous la représentation en ASCII. L’application se base sur une application déjà existante : **EditHexa** version 7.8. Cette application permet également d’avoir un aperçu des valeurs du fichier selon différents encodages.

# Analyse fonctionnelle

L’application est développée en .NET C# Windows Form.

## Fonctionnalités principales

* Ouverture d’un fichier
* Affichage de son contenu sous forme hexadécimal
* Affichage de son contenu sous forme ASCII
* Edition en hexadécimal
* Edition en ASCII
* Affichage de la valeur sous plusieurs formats (32 bits, décimal, octal, float etc.)
* Annulation de la dernière action d’édition (« ctrl+Z »)
* Sauvegarde des modifications

## Fonctionnalités remarquables

### Gestion des pages et enregistrement du fichier

La taille du tableau référençant toutes les valeurs du fichier lu pouvant être très grand, nous avons préféré créer un tableau de valeurs intermédiaires, correspondant à un ensemble de valeurs de taille fixe, dont le début se situe dans le tableau de référence, à un endroit donné. Pour faire court, un système de pagination.

Ainsi, toute la valeur de références se trouve dans la classe Reader. La fonction *GetValue*, permet d’obtenir la valeur décimale du bit à la position donnée. Lorsque la vue ordonne de charger une nouvelle page (lors d’un scroll ou du chargement d’un nouveau fichier), le modèle calcul la position des bits demandés en fonction du numéro de  la page et de sa taille (une constante).  Le programme va retourner une collection de valeurs hexadécimales par la fonction getValues ou de caractères par la fonction getASCIIValues.

Pour garder la version de référence à jour (dans la classe reader), les informations de la page sont réécrites par-dessus celle d’origine à chaque fois que le programme charge une nouvelle page, ou souhaite enregistrer son fichier.

### Historique

La fonctionnalité d'historique a été construite avec pour objectif de ne prendre que le stricte minimum en matière de place : sauvegarder le fichier entier, byte par byte, ou même la page pourrait vite prendre une place énorme. Ainsi, l’historique se présente sous forme de stack contenant des chaînes de caractère. Les deux premiers caractères de cette chaîne représentent la valeur hexadécimale qu’avait la case avant sa dernière modification. Le reste des caractères (de la troisième à la dernière position) représente, sous forme décimale, la position de la dernière modification.

Nous avons également dû adapter  le système d’historique à celui de pagination, auparavant mis en place. Le fichier possède ainsi deux historiques : un historique de la page en cours (dans le model), auquel chaque modification apportée par l’utilisateur est inscrit, et un autre historique général (dans le reader), auquel sont ajoutés toute les étapes d’une page en même temps que les valeurs de celle-ci sont réécrites par-dessus les valeurs de référence. A noter que les index des cases dans  l’historique d’une page prennent comme origine le début de la page, alors que ceux inscrits dans l’historique général prennent comme origine le début du fichier. Il convient donc le modifier ces identifiants lorsqu’on enregistre  les étapes de l’historique d’une page dans l’historique général.

Lorsque l’utilisateur commande le rétablissement du dernier stade, il y a deux cas de figure : dans le premier, l’historique de la page en cour n’est pas vide ; dans le second, celui-ci est vide. Si l’historique de la page n’est pas vite, le programme va simplement supprimer l’étape de l’historique et rétablir la valeur dans la page. Dans le cas contraire, le rétablissement va se faire dans le reader (si l’historique du reader n’est pas vide, bien sûr). Pour confirmer à l’utilisateur que l’annulation a bien été effectuée, le programme va chercher la page de la dernière modification et placer le curseur sur la case ou la valeur a été rétablie.

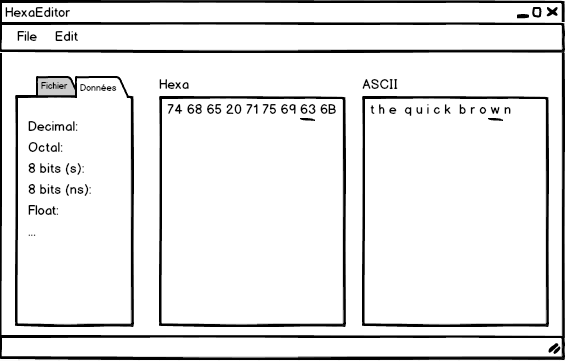
### Affichage des octets du fichier

Le contenu du fichier est tout d’abord stocké sous forme d’un tableau de byte par le reader qui est ensuite passé au ModelHexaEditor. Ce dernier va ensuite, avec la méthode GenerateDrawnValues, créer une image bitmap adéquate afin d’afficher le contenu du tableau de byte. Cette image représente un tableau à deux entrées. GenerateDrawnValuesAsAscii fait le même travail mais convertit au préalable la valeur au format Ascii. Pour gérer la sélection d’une case du tableau, lors de la génération des images, une List de Rectangle est également créée.

Une fois ces deux images créées, les deux PictureBox de la vue affichent ces dernières. Pour parcourir ces deux images-tableaux, la méthode Paint propre aux picturebox est utilisée. A l’appui d’une touche des flèches directionnelles, on incrémente/décrémente un index puis on colorie en bleu le rectangle de la List de rectangle correspondant à la position dans le tableau de valeurs.

## Interface

### Fenêtre principale

L’application possède une vue principale qui permet d’ouvrir un fichier puis d’afficher son contenu sous différents formats. L’affichage des octets en hexa et ASCII se fait grâce à une génération dynamique de PictureBox.

**Picturebox :** Contenu du fichier en hexadécimal généré dynamiquement

**TabControl Données :**

Affichage sous différents formats par rapport à la position actuelle

**MenuStrip :**

Menu pour ouvrir un fichier, sauvegarder et annuler la dernière modification

**Picturebox :** Contenu du fichier en ASCII généré dynamiquement

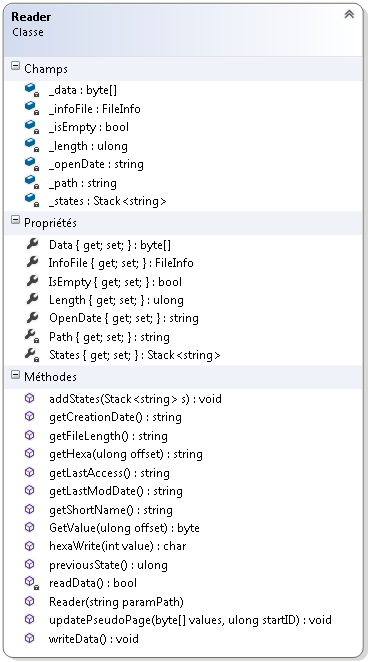
**TabControl Fichier :**

Affichage des informations du fichier (date, nom, dernière modification etc.)

# Analyse organique

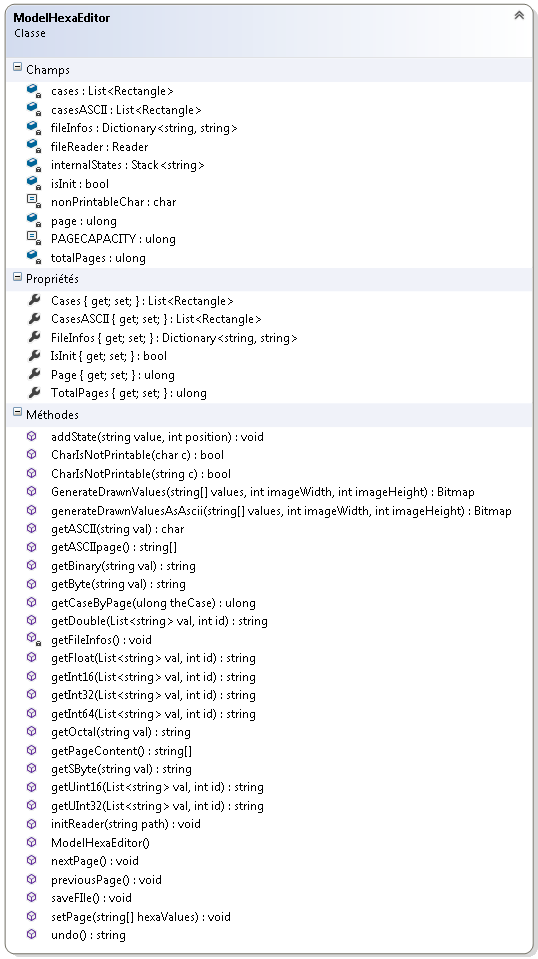
## Diagramme de classe

### Classe Reader



Reader est la classe qui s’occupe de gérer le fichier qui va être traité. Il stocke le contenu du fichier sous forme d’octets dans un tableau de byte (champs \_data). C’est également cette classe qui permet de retourner les informations du fichier comme sa date de dernière modification, son nom etc. Reader permet également de sauvegarder, écraser le fichier avec le contenu (tableau de byte[]) modifié par la classe ModelHexaEditor.

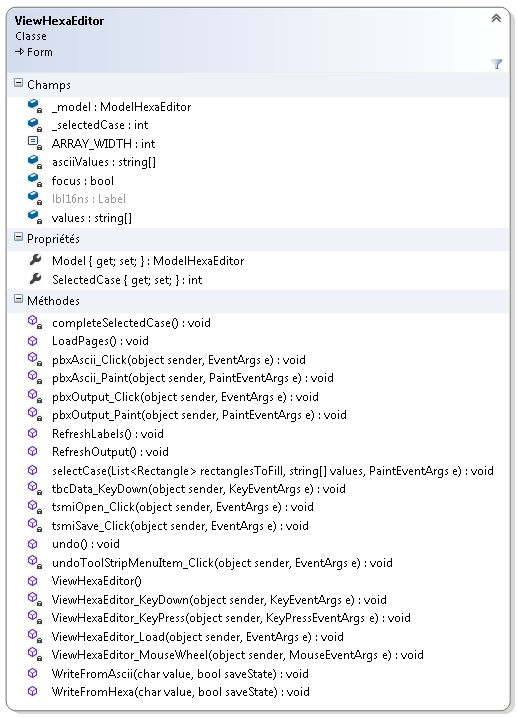
### Classe ModelHexaEditor



La classe ModelHexaEditor est la classe qui s’occupe du traitement principal.

Elle permet d’une part de générer les images Bitmap représentant le contenu du fichier sous forme hexadécimal et ASCII qui seront utilisés par les PictureBox de la vue (GenerateDrawnValues, GenerateDrawnValuesAsASCII). Et d’autre part, ModelHexaEditor contient les différentes méthodes de « conversions » sous différents formats (getInt, getOctal, getDecimal etc.). C’est également dans cette classe qu’est géré le système de pagination.

### Classe ViewHexaEditor



La classe ViewHexaEditor est la form qui gère les contrôleurs et les actions de l’utilisateur, comme le déplacement dans les cases des picturebox.

Elle possède également les rafraichissements de ces derniers avec les méthodes Paint. Cette classe gère aussi l’écriture de l’utilisateur dans les cases, que ce soit sur l’hexadécimal(WriteFromHexa) ou l’ascii (WriteFromAscii).

## Plan de tests

|  |  |  |
| --- | --- | --- |
| **N°** | **Description** | **Résultat attendu** |
| **1** | On ouvre un fichier quelconque, de taille non nulle | Le programme charge la première page du fichier et charge les valeurs correspondantes à la première case dans l’onglet Data |
| **2** | On ouvre un fichier quelconque, de taille nulle | Le programme indique à l’utilisateur par une MessageBox que le fichier est vide |
| **3** | L’utilisateur navigue avec le pavé directionnel | Le curseur bleu va se déplacer en fonction de la touche pressée sur la page. Lorsque l’utilisateur arrive aux limites de la page, il ne peut pas aller plus loin. |
| **4** | L’utilisateur navigue à l’aide de la roulette | Dans les limites du document, l’affichage passe d’une page à l’autre. |
| **5** | L’utilisateur, ayant son focus sur le tableau des valeurs hexadécimales, presse une touche ne correspondant pas à une valeur hexadécimale. | Rien ne se passe |
| **6** | L’utilisateur, ayant son focus sur le tableau des valeurs hexadécimales, presse une touche correspondant à une valeur hexadécimale. | Si la case contient déjà deux digits, il seront remplacés par le caractère de l’utilisateur. Sinon, ils seront ajoutés à la suite de la valeur initiale. Dans les deux cas, le tableau des ascii se met à jour. |
| **7** | L’utilisateur, ayant son focus sur le tableau des valeurs ascii, presse une touche correspondant à un caractère imprimable. | Le caractère remplace celui de la case et la valeur hexadécimale se met à jour |
| **8** | L’utilisateur effectue une modification, va à la page suivante puis revient sur la page ou il a effectué sa modification | Les modifications ont persisté |
| **9** | L’utilisateur se trouve à moins de 8 bit de la limite du fichier | Dans l’onglet Data, toutes les valeurs nécessitant plus de bit qu’il n’en reste dans le fichier, à partir du curseur, sont indiquées par un « - » |
| **10** | L’utilisateur se trouve à moins de 8 bit de la limite de la page | Dans l’onglet Data, toutes les valeurs nécessitant plus de bit qu’il n’en reste dans la page, à partir du curseur, sont indiquées correctement |
| **11** | L’utilisateur effectue des modifications, enregistre ses modifications, ferme le fichier puis l’ouvre à nouveau | Les modifications ont persisté |
| **12** | L’utilisateur, n’ayant effectué aucune modification, appuie sur CTRL+Z | Rien ne se passe |
| **13** | L’utilisateur, ayant effectué une modification sur la page ou il se trouve, appuie sur CTRL+Z | Le curseur se place sur l’emplacement de sa dernière modification, ou la dernière valeur a été rétablie |
| **14** | L’utilisateur, ayant effectué une modification sur une page ou il ne se trouve pas, appuie sur CTRL+Z | L’utilisateur est redirigé sur la page de sa dernière modification, le curseur se place à son emplacement, ou la dernière valeur a été rétablie |