|  |  |
| --- | --- |
| Symbol projektu:  Model OpenXml Lib | |
| Nazwa projektu:  **Model obiektowy OpenXml Doc** | |
| Tytuł dokumentu: | **Dokumentacja modelu** |
| Nr wersji: | **1.0** |
| Data utworzenia: | **7.07.2024** |
| Data ostatniej aktualizacji: | **10.07.2024** |
| Osoba odpowiedzialna: | **Jarosław Kuchta (JK)** |
| Autorzy: | **Jarosław Kuchta (JK)** |

Historia dokumentu

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Wersja | Opis modyfikacji | Autor | Data otwarcia | Data zamknięcia |
| 1.0 | Na wzór dokumentacji ModelOpenXmlLib | JK | 7.07.2024 | 10.07.2024 |

**Spis treści**

[1. Wprowadzenie 2](#_Toc171538845)

[2. Model danych aplikacji 2](#_Toc171538846)

[2.1. Pliki, rozdziały i podrozdziały 2](#_Toc171538847)

[2.2. Typy proste – wartości wyliczeniowe 3](#_Toc171538848)

[2.3. Elementy i atrybuty 5](#_Toc171538849)

[2.4. Relacje związane z elementami i typami prostymi 6](#_Toc171538850)

[3. Kontekst danych 6](#_Toc171538851)

[4. Parsowanie dokumentów 7](#_Toc171538852)

[5. Wyniki i uwagi końcowe 8](#_Toc171538853)

# Wprowadzenie

Projekt Model OpenXml Doc reprezentuje model obiektowy OpenXml pobrany z wybranych plików DOCX i zapisany w bazie danych MS Access.

Projekt składa się z dwóch modułów:

* ModelOpenXmlDoc – moduł biblioteczny definiujący model obiektowy klas.
* ModelOpenXmlDocApp – aplikacji konsolowej do parsowania plików i zapisywania modelu w bazie danych.

Projekt czyta i parsuje pliki DOCX zebrane w katalogu d:\VS\Docs\OpenXml\Documents. Pliki zostały wybrane z dokumentacji Ecma Office Open Xml. Oryginalna dokumentacja została ściągnięta ze strony <https://ecma-international.org/publications-and-standards/standards/ecma-376/> w postaci plików PDF. Dokumenty zostały przetransformowane na format DOCX a ich treść dodatkowo sformatowana tak, aby nadawały się do parsowania.

Aplikacja zapisuje model schematu w bazie danych MS Access o nazwie OpenXmlDoc.accdb położonej w ścieżce „d:\VS\Docs\OpenXml”. Ten sposób magazynowania danych został wybrany ze względu na dużą łatwość edycji rekordów bazy danych. Do odwzorowania modelu klas encyjnych w model relacyjnej bazy danych jest wykorzystywany framework Microsoft.­EntityFramework.Core.

# Model danych aplikacji

Aplikacja odczytuje wybrane informacje z plików DOCX i zapisuje je w relacyjnej bazie danych. W bibliotece ModelOpenXmlLib zdefiniowano klasy encyjne (z wykorzystaniem frameworku EntityFramework.Core), które reprezentują dane dopasowane do relacyjnej bazy danych.

## Pliki, rozdziały i podrozdziały

Informacje o plikach DOCX są przechowywane w tabeli Files. Każdą encję reprezentuje klasa DocFile o właściwościach:

* Id: int {autonum, PK} – identyfikator encji,
* FileName: string – nazwa pliku (bez ścieżki i rozszerzenia),
* Chapters: List<Chapter> – właściwość nawigacyjna do kolekcji rozdziałów,
* ChaptersDictionary: Dictionary<string, Chapter> {notMapped} – słownik rozdziałów (kluczem jest numer rozdziału),
* Document: WordprocessingDocument? {notMapped} – tymczasowo przechowywana referencja do pakietu dokumentu.

Rozdziały i podrozdziały są przechowywane w tabeli Chapters. Encje reprezentuje klasa Chapter o właściwościach:

* Id: int {autonum, PK} – identyfikator encji,
* OwnerFileId: int {FK „Files”} – identyfikator pliku, do którego należy ten rozdział (lub podrozdział),
* OwnerChapterId: int {FK „Chapters”} – identyfikator rozdziału, do którego należy dany podrozdział,
* OrdNum: int – numer kolejny rozdziału (podrozdziału) w ramach dokumentu (rozdziału nadrzędnego) – liczony od 1,
* NumStr: string – pełny numer rozdziału wyświetlany przed nagłówkiem,
* Heading: string – tekst nagłówka,
* ParagraphId: string[8] – identyfikator akapitu z nagłówkiem rozdziału,
* HasSubchapters: bool – czy rozdział posiada podrozdziały,
* HasSimpleTypes: bool – czy rozdział deklaruje typy proste (w podrozdziałach),
* HasElements: bool – czy rozdział deklaruje elementy (w podrozdziałach),
* OwnerFile: DocFile – właściwość nawigacyjna do dokumentu,
* ParentChapter: Chapter? – właściwość nawigacyjna do rozdziału nadrzędnego,
* SubChapters: List<Chapter> – właściwość nawigacyjna do kolekcji podrozdziałów,
* SubChaptersDictionary: Dictionary<string, Chapter> {notMapped} – słownik kolekcji podrozdziałów,
* SimpleTypes: List< SimpleType > – właściwość nawigacyjna do kolekcji typów prostych deklarowanych w tym rozdziale,
* SimpleTypesDictionary: dictionary<string, SimpleType> {notMapped} – słownik deklarowanych typów prostych,
* Elements: List< Element > – właściwość nawigacyjna do kolekcji elementów deklarowanych w tym rozdziale,
* ElementsDictionary: dictionary<string, Element> {notMapped} – słownik deklarowanych elementów.

Relacje między tabelami Files i Chapters przedstawiono na rys. 1.

Obraz zawierający tekst, zrzut ekranu, diagram, linia

Opis wygenerowany automatycznie

Rys. 1. Relacje między plikami a rozdziałami i podrozdziałami

## Typy proste – wartości wyliczeniowe

W dokumentach występują rozdziały pt. „Simple Types”, które zawierają podrozdziały deklarujące typy proste. Część spośród typów prostych to typy wyliczeniowe, które deklarują wartości wyliczane w tabeli, której pierwsza komórka ma treść „Enumeration Value”. Przykład podano na rys. 2.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Rys. 2. Deklaracja typu wyliczeniowego

W modelu danych ModelOpenXmlDoc typy proste są przechowywane w tablicy SimpleTypes. Encja SimpleType ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* ShortName: string {required} – krótka nazwa typu (na początku nagłówka, przed długą nazwą w nawiasach),
* LongName: string {required} – długa nazwa typu (w nagłówku w nawiasach),
* OwnerChapterId: int {FK „Chapters”} – identyfikator rozdziału, w którym jest zdefiniowany dany typ,
* IsEnum: bool – czy typ prosty jest typem wyliczeniowym,
* DescriptionText: text? – tekst opisowy typu prostego (z pierwszego akapitu),
* OwnerChapter: Chapter – właściwość nawigacyjna rozdziału, w którym jest zdefiniowany dany typ,
* BaseType: TypeDef? – właściwość nawigacyjna do typu bazowego (opcjonalna),
* EnumValues: ICollection<EnumValue> – właściwość nawigacyjna do wartości wyliczeniowych zdefiniowanych w tym typie,
* EnumValuesDictionary: Dictionary<string, EnumValue> – słownik wartości wyliczeniowych.

Wartości typu wyliczeniowego są zapisywane w tabeli EnumValues. Encja EnumValue ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* OwnerTypeId: int {FK „SimpleTypes”} – identyfikator typu wyliczeniowego, który deklaruje tę wartość,
* OrdNum: int? – numer kolejny wartości (od zera),
* ShortName: string {required} – krótka nazwa wartości (przed długą nazwą w nawiasach),
* LongName: text {required} – długa nazwa wartości (w nawiasach),
* DescriptionText: string? – tekst opisowy wartości (z drugiej kolumny),
* OwnerType: SimpleType – właściwość nawigacyjna do typu wyliczeniowego, dla którego podano tę wartość.

## Elementy i atrybuty

Podobnie w dokumentach występują rozdziały które zawierają podrozdziały deklarujące elementy. W części z tych plików te rozdziały mają nagłówki „Elements”, a część ma inne nagłówki. W części elementów deklarowane są atrybuty w tabeli, której pierwsza komórka ma treść „Attributes”. Przykład podano na rys. 3.

Obraz zawierający tekst, zrzut ekranu, dokument, Czcionka

Opis wygenerowany automatycznie

Rys. 3.Elementy i atrybuty

Elementy są zapisywane w tabeli Elements. Encja Element ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* ShortName: string {required} – krótka nazwa elementu (na początku nagłówka, przed długą nazwą w nawiasach),
* LongName: string {required} – długa nazwa elementu (w nagłówku w nawiasach),
* OwnerChapterId: int {FK „Chapters”} – identyfikator rozdziału, w którym jest zdefiniowany dany element,
* HasAttributes: bool – czy element deklaruje atrybuty,
* DescriptionText: text? – tekst opisowy elementu (z pierwszego akapitu),
* Attributes: ICollection<Attribute> – właściwość nawigacyjna do atrybutów zdefiniowanych w tym elemencie,
* AttributesDictionary: Dictionary<string, Attribute > – słownik elementów.

Atrybuty są zapisywane w tabeli Attributes. Encja Attribute ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* OwnerElementId: int {FK „Elements”} – identyfikator elementu, który deklaruje ten atrybut,
* OrdNum: int? – numer kolejny atrybutu (od zera),
* ShortName: string {required} – krótka nazwa atrybutu (przed długą nazwą w nawiasach),
* LongName: text {required} – długa nazwa atrybutu (w nawiasach),
* Namespace: string? – przestrzeń nazw, w której jest zdefiniowany atrybut,
* DescriptionText: string? – tekst opisowy atrybutu (z drugiej kolumny),
* OwnerElement: Element – właściwość nawigacyjna do elementu, dla którego zadeklarowano ten atrybut.

## Relacje związane z elementami i typami prostymi

Relacje związane z elementami i atrybutami oraz typami prostymi i atrybutami są przedstawione na poniższym rysunku.

Obraz zawierający tekst, zrzut ekranu, diagram, linia

Opis wygenerowany automatycznie

Rys. 4. Relacje związane z elementami i typami prostymi

# Kontekst danych

Kontekst danych reprezentuje klasa DocDbContext. Klasa ta:

1. Deklaruje zbiory danych (DbSet) reprezentujące tabele.
2. Deklaruje słowniki i indeksy plików i rozdziałów (FilesDictionary, ChaptersDictionary, FilesIndex i ChaptersIndex). Umożliwiają one wyszukiwanie odpowiednich encji bez odwoływania się do bazy danych.
3. W konstruktorze:
   1. Ustala nazwę pliku bazy danych przekazaną przez parametr.
   2. Upewnia się, że baza danych została utworzona,
   3. Ustala niektóre właściwości bazy danych programu MS Access korzystając z mechanizmu automatyzacji COM w metodzie SetupAccessDatabase.
4. W metodzie OnConfiguring włącza użycie mechanizmu Jet MS Access.
5. W metodzie OnModelCreating tworzy relacje między encjami.
6. W metodzie LoadFiles inicjuje słownik i indeks plików FilesDictionary i FilesIndex. Do zbioru danych Files przypisuje procedury obsługi zdarzenia dodania encji, które dodają encje do odpowiedniego słownika i indeksu.
7. W metodzie LoadChapters inicjuje słowniki rozdziałów, typów prostych, elementów i inne, które są zdefiniowane w poszczególnych encjach.

W tym modelu korzysta się z ładowania chętnego (eagerly loading) zamiast domyślnej metody ładowania leniwego (lazy loading). Encje są indeksowane lokalnie (w pamięci aplikacji) przy użyciu słowników. Minimalizuje to odwołania do bazy danych.

Metoda SetupAccessDatabase ułatwia przeglądanie utworzonej i wypełnionej bazy danych w programie MS Access. Tworzy kwerendy danych i tabele reprezentujące typy wyliczeniowe. Ponadto ustawia właściwości związane z kolumnami danych tak, że MS Access wyświetla odpowiednie nazwy zamiast wartości liczbowych.

Ponieważ EntityFramework.Core nie zapewnia takich możliwości, więc metoda SetupAccessDatabase wykorzystuje interfejs COM (Component Object Model) do automatyzacji programu MS Access. Mechanizm ten jest zaimplementowany w pakietach MicrosoftOffice.Interop.Access i MicrosoftOffice.Interop.Access.Dao, które korzystają z pakietu MicrosoftOfficeCore15. Ten ostatni pakiet jest ściągnięty w wersji dla .NetFramework (a nie dla .Net8.0, dla którego jest skompilowany cały projekt). Przy kompilacji jest wyświetlane ostrzeżenie, ale automatyzacja działa.

Jedyny problem następuje z zamknięciem interfejsu użytkownika programu MS Access. Jest to znany problem przy wykorzystywaniu mechanizmu COM zaimplementowanych w kodzie natywnym (C++) w programie w kodzie zarządzanym (C#). Dla rozwiązania tego programu na koniec metody SetupAccessDatabase jest wywoływana metoda KillMsAccess, która „zabija” wszystkie procesy o nazwie „MSACCESS”. Skutkiem ubocznym jest zamykanie tej instancji programu MS Access, która została wcześniej otwarta przez użytkownika.

# Parsowanie dokumentów

Za parsowanie odpowiada klasa publiczna OpenXmlDocParser zadeklarowana w module ModelOpenXmlDocApp.

Klasa OpenXmlDocParser ma główną metodą publiczną ParseDocuments, dla której podaje się:

* sourceDocPath: string – ścieżkę do katalogu zawierającego pliki DOCX,
* dbFilename: string – pełną nazwę pliku z bazą danych MS Access.

Metoda ta:

1. Tworzy wspólny dla wszystkich innych metod obiekt dbContext klasy DocDbContext.
2. Ładuje nazwy wszystkich plików DOCX z podanego katalogu do bazy danych.
3. Otwiera i parsuje wszystkie pliki DOCX. Jeśli otwarcie jest niemożliwe, to pyta użytkownika o decyzję, czy ma powtórzyć próbę, pominąć plik, czy też przerwać aplikację.

W czasie wczytywania elementów schematu program sprawdza, czy istnieją odpowiednie encje w słownikach wyszukując je po odpowiednich kluczach drugoplanowych. Jeśli encja nie istnieje, to ją tworzy i dodaje je do kontekstu danych.

Poprawne działanie programu jest zapewnione dzięki odpowiedniej aktualizacji właściwości nawigacyjnych i słowników, co jest zapewniane przez zdefiniowanie odpowiednich relacji w klasie DocDbContext w metodzie OnModelCreating oraz w metodach LoadFiles i LoadChapters. Brak odpowiedniej aktualizacji danych w tych metodach powoduje, że program parsujący nie znajduje encji w pamięci operacyjnej i próbuje je dodawać w bazie danych, co kończy się wyjątkiem programu.

Na koniec program parsujący wyświetla statystykę:

* FilesTotal, FilesAdded;
* ChaptersTotal, ChaptersAdded;
* SimpleTypesTotal, SimpleTypesAdded, SimpleTypesUpdated;
* EnumValuesTotal, EnumValuesAdded, EnumValuesUpdated;
* ElementsTotal, ElementsAdded, ElementsUpdated;
* AttributesTotal, AttributesAdded, AttributesUpdated;

Nazwy liczników mówią same za siebie. Przy dodawaniu encji mogą być one czasem dodatkowo aktualizowane po dodaniu. Stąd pojawiają się też niezerowe liczniki Updated nawet, jeśli początkowo baza danych jest pusta.

# Wyniki i uwagi końcowe

W aplikacji przeanalizowano i wczytano 4 pliki DOCX:

* 17.WordprocessingML Reference Material.docx
* 20.DrawingML - Framework Reference Material.docx
* 21.DrawingML - Components Reference Material.docx
* 22.Shared MLs Reference Material.docx

Wyniki

Liczby obiektów rozpoznanych i zapisanych w bazie danych przedstawiono w poniższej tabeli.

Tab. 1. Podsumowanie wyników

|  |  |
| --- | --- |
| Obiekty | Liczba |
| Pliki | 4 |
| Rozdziały | 2309 |
| Typy proste | 373 |
| Wartości wyliczane | 2189 |
| Elementy | 1712 |
| Atrybuty | 2016 |

Uwaga dotycząca wydajności programu

Dzięki zastosowaniu słowników encji zależnych, przy wyszukiwaniu encji po nazwie jest przeszukiwany lokalny słownik bez odwoływania się do bazy danych. Dało to skrócenie czasu całkowitego przetwarzania z ponad 1 min (przy pustej bazie danych) do 16 sekund (przy braku konieczności zmian w bazie danych).