|  |  |
| --- | --- |
| Symbol projektu:  Model OpenXml Cmt | |
| Nazwa projektu:  **Model obiektowy OpenXml Cmt** | |
| Tytuł dokumentu: | **Dokumentacja modelu** |
| Nr wersji: | **1.0** |
| Data utworzenia: | **11.07.2024** |
| Data ostatniej aktualizacji: | **12.07.2024** |
| Osoba odpowiedzialna: | **Jarosław Kuchta (JK)** |
| Autorzy: | **Jarosław Kuchta (JK)** |

Historia dokumentu

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Wersja | Opis modyfikacji | Autor | Data otwarcia | Data zamknięcia |
| 1.0 | Na wzór dokumentacji ModelOpenXmlLDoc | JK | 1.07.2024 | 12.07.2024 |

**Spis treści**

[1. Wprowadzenie 2](#_Toc171687683)

[2. Struktura pliku z komentarzami 2](#_Toc171687684)

[3. Model danych aplikacji 3](#_Toc171687685)

[3.1. Pliki XML 3](#_Toc171687686)

[3.2. Elementy skomentowane 4](#_Toc171687687)

[4. Kontekst danych 4](#_Toc171687688)

[5. Parsowanie dokumentów 5](#_Toc171687689)

[6. Wyniki i uwagi końcowe 6](#_Toc171687690)

# Wprowadzenie

Projekt Model OpenXml Cmt reprezentuje komentarze dokumentacyjne do modelu OpenXml. Komentarze są pobierane z pliku XML (wygenerowanego przy kompilacji pakietu DocumentFormat.OpenXml i zapisane w bazie danych MS Access.

Projekt składa się z dwóch modułów:

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

Projekt czyta i parsuje pliki XML zawierające komentarze dokumentacyjne zebrane w katalogu d:\VS\Docs\OpenXml\Libraries. Pliki zostały przez Visual Studio uzyskane z plików PDB pakietów DocumentFormat.OpenXml.

Aplikacja zapisuje encje elementów skomentowanych w bazie danych MS Access o nazwie OpenXmlCmt.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.

W bazie danych zapisywane są tylko encje elementów skomentowanych, zaś treść komentarzy jest zapisywana w kolumnie DescriptionText jako tekst długi (bez ograniczenia).

# Struktura pliku z komentarzami

Komentarze dokumentacyjne są podczas kompilacji w systemie Windows ekstrahowane z kodu źródłowego i zapisywane w pliku XML. Plik ten ma strukturę podobną do poniższej:

<?xml version="1.0"?>

<doc>

<assembly>

<name>DocumentFormat.OpenXml</name>

</assembly>

<members>

<member name="T:DocumentFormat.OpenXml.AdditionalCharacteristics.AdditionalCharacteristicsInfo">

<summary>Defines AdditionalCharacteristics.</summary>

<summary>

<para>Set of Additional Characteristics.</para>

<para>This class is available in Office 2007 and above.</para>

<para>When the object is serialized out as xml, it's qualified name is ac:additionalCharacteristics.</para>

</summary>

<remark>

<para>The following table lists the possible child types:</para>

<list type="bullet">

<item><description><see cref="T:DocumentFormat.OpenXml.AdditionalCharacteristics.Characteristic" /> <c>&lt;ac:characteristic&gt;</c></description></item>

</list>

</remark>

</member>

Na początku, w elemencie assembly jest identyfikowana biblioteka (lub program), której elementy zostały skomentowane. Dalej, w elemencie members jest wyszczególniona lista skomentowanych elementów.

Każdy skomentowany element biblioteki jest opisany w elemencie XML o nazwie lokalnej member. W tym elemencie, w atrybucie name podana jest pełna identyfikacja skomentowanego elementu. Format identyfikacji składa się z następujących części:

* X: identyfikacja typu elementu.

Rozróżniane są

T: typ.

F: pole,

P: właściwość,

M: metoda,

E: zdarzenie.

* długa nazwa elementu w postaci

przestrzeń\_nazw.nazwa\_typu[.nazwa\_składowej]

* opcjonalna lista parametrów w nawiasach

W ten sposób każdy skomentowany element ma unikatową identyfikację.

Składowe typów nie są zagnieżdżane w typach, lecz umieszczane na jednej, wspólnej liście.

Wewnątrz elementu skomentowanego umieszczana jest zawartość komentarzy dokumentacyjnych

# Model danych aplikacji

Aplikacja parsuje pliki XML zapisuje je w relacyjnej bazie danych. W bibliotece ModelOpenXmlCmt zdefiniowano klasy encyjne (z wykorzystaniem frameworku EntityFramework.Core), które reprezentują dane dopasowane do relacyjnej bazy danych.

W modelu danych aplikacji są tylko dwie klasy encyjne:

* XmlFile – reprezentuje plik XML z dokumentacją,
* Member – reprezentuje element skomentowany.

Nie zdecydowano się na zamianę samych komentarzy na encje, gdyż okazało się to bardzo nieefektywne podczas parsowania.

## Pliki XML

Informacje o plikach XML są przechowywane w tabeli Files. Każdą encję reprezentuje klasa XmlFile o właściwościach:

* Id: int {autonum, PK} – identyfikator encji,
* FileName: string – nazwa pliku (bez ścieżki i rozszerzenia),
* Members: List<Members> – właściwość nawigacyjna do kolekcji elementów skomentowanych,
* MembersDictionary: Dictionary<string, Member> {notMapped} – słownik elementów skomentowanych (kluczem jest pełna nazwa elementu),
* Document: XDocument? {notMapped} – tymczasowo przechowywana referencja do dokumentu XML.

## Elementy skomentowane

Skomentowane elementy przechowywane w tabeli Members. Encje reprezentuje klasa Member o właściwościach:

* Id: int {autonum, PK} – identyfikator encji,
* OwnerFileId: int {FK „Files”} – identyfikator pliku, do którego należy ten element,
* ParentMemberId: int {FK „Members”} – identyfikator elementu, do którego należy dany element,
* FullName: string – pełna nazwa elementu (taka jak w pliku XML),
* ShortName: string – krótka nazwa elementu (ostatni element nazwy pełnej),
* Params: string? – opcjonalna lista parametrów (wyciągnięta z nawiasów w nazwie pełnej),
* Type: MemberType – typ elementu (Type, Field, Property, Method, Event),
* HasMembers: bool – czy element ma przypisane własne składowe,
* DescriptionText: text – długi opis elementu (zawiera jego komentarze),
* OwnerFile: XmlFile – właściwość nawigacyjna do pliku XML,
* ParentMember: Member? – właściwość nawigacyjna do elementu nadrzędnego,
* Members: Collection<Chapter> – właściwość nawigacyjna do kolekcji elementów składowych,
* MembersDictionary: Dictionary<string, Member> {notMapped} – słownik elementów składowych.

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

Obraz zawierający tekst, zrzut ekranu, diagram, linia

Opis wygenerowany automatycznie

Rys. 1. Relacje między plikami a rozdziałami i podrozdziałami

# Kontekst danych

Kontekst danych reprezentuje klasa CmtDbContext. Klasa ta:

1. Deklaruje zbiory danych (DbSet) reprezentujące tabele.
2. Deklaruje słownik plików (FilesDictionary). który umożliwia 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 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 LoadMembers inicjuje słowniki 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 OpenXmlCmtParser zadeklarowana w module ModelOpenXmlCmtApp.

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

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

Metoda ta:

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

Parsowane są tylko elementy typu Type, Field i Property. Elementy typu Method i Event są pomijane. Powodem tego jest fakt, że wynikowa baza danych ma współdziałać innymi bazami, w których metody i zdarzenia nie występują.

W czasie wczytywania elementów tworzy się relacje zawierania między polami i właściwościami, a odpowiednimi typami. Jeśli typ nie występował wcześniej na liście elementów w pliku XML, to jest tworzony jego element zastępczy o odpowiedniej nazwie pełnej.

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 CmtDbContext w metodzie OnModelCreating oraz w metodach LoadFiles i LoadMembers. 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;
* TypesTotal, TypesAdded, TypesUpdated;
* FieldsTotal, FieldsAdded, FieldsUpdated;
* PropsTotal, PropsAdded, PropsUpdated;

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 tylko jeden plik: DocumentFormat.OpenXml.xml, gdyż tylko taki plik był dostępny.

Wyniki

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

Tab. 1. Podsumowanie wyników

|  |  |
| --- | --- |
| Obiekty | Liczba |
| Pliki | 1 |
| Typy | 5144 |
| Pola | 4356 |
| Właściwości | 13720 |

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 5 min (przy pustej bazie danych) do 4 sekund (przy braku konieczności zmian w bazie danych).