|  |  |
| --- | --- |
| Symbol projektu:  Model OpenXml Lib | |
| Nazwa projektu:  **Model obiektowy OpenXml Lib** | |
| Tytuł dokumentu: | **Dokumentacja modelu** |
| Nr wersji: | **1.0** |
| Data utworzenia: | **7.07.2024** |
| Data ostatniej aktualizacji: | **07.07.2024** |
| Osoba odpowiedzialna: | **Jarosław Kuchta (JK)** |
| Autorzy: | **Jarosław Kuchta (JK)** |

Historia dokumentu

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Wersja | Opis modyfikacji | Autor | Data otwarcia | Data zamknięcia |
| 1.0 | Na podstawie ModelXmlSchemaDocumentation | JK | 7.07.2024 | 7.07.2024 |

**Spis treści**

[1. Wprowadzenie 2](#_Toc171267456)

[2. Model danych aplikacji 2](#_Toc171267457)

[2.1. Pliki i przestrzenie nazw 2](#_Toc171267458)

[2.2. Typy 3](#_Toc171267459)

[2.3. Wartości wyliczane 3](#_Toc171267460)

[2.4. Właściwości 4](#_Toc171267461)

[2.5. Relacje związane z typami 4](#_Toc171267462)

[3. Kontekst danych 4](#_Toc171267463)

[4. Parsowanie modelu schematu 6](#_Toc171267464)

[5. Wyniki i uwagi końcowe 6](#_Toc171267465)

# Wprowadzenie

Projekt Model OpenXml Lib reprezentuje model obiektowy OpenXml pobrany z plików DLL i zapisany w bazie danych MS Access.

Projekt składa się z dwóch modułów:

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

Projekt czyta i parsuje pliki DLL zebrane w katalogu d:\VS\Docs\OpenXml\Libraries i zapisuje model schematu w bazie danych MS Access o nazwie OpenXmlLib.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 klasy i właściwości z plików DLL 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 i przestrzenie nazw

Informacje o plikach bibliotecznych są przechowywane w tabeli Files. Każdą encję reprezentuje klasa LibFile o właściwościach:

* Id: int {autonum, PK} – identyfikator encji,
* FileName: string – nazwa pliku (bez ścieżki i rozszerzenia),

Przestrzenie nazw są przechowywane w tabeli Namespaces. Każdą encję reprezentuje klasa Namespace o właściwościach podstawowych:

* Id: int {autonum, PK} – identyfikator encji,
* Name: string – nazwa przestrzeni nazw.

W każdym pliku DLL mogą być definiowane różne przestrzenie nazw. Każda przestrzeń nazw może być definiowana w różnych plikach DLL. Do rejestracji relacji między plikami i przestrzeniami nazw służy tabela FileNamespaces. Encja FileNamespace reprezentuje relację wiele-wiele pomiędzy plikami schematów i przestrzeniami nazw i ma następujące właściwości:

* FileId: int {FK „Files”} – identyfikator pliku DLL,
* NamespaceId: int {FK „Namespaces”} – identyfikator przestrzeni nazw,
* File: LibFile – właściwość nawigacyjna do pliku bibliotecznego,
* Namespace: Namespace – właściwość nawigacyjna do przestrzeni nazw.

Relacje między tabelami Files, Namespaces i FileNamespaces przedstawiono na rys. 2.

A screenshot of a computer

Description automatically generated

Rys. 2. Relacje między plikami a przestrzeniami nazw

## Typy

W modelu danych ModelOpenXmlLib typy są przechowywane we wspólnej tablicy Types. Encja TypeDef reprezentuje wszystkie typy.

Encja TypeDef ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* Name: string {required} – nazwa typu,
* Kind: TypeKind {dbType „byte”} – określa rodzaj typu (Value, Enum, Struct, Class, Interface),
* OwnerNamespaceId: int {FK „Namespaces”} – identyfikator przestrzeni nazw, w której jest zdefiniowany dany typ,
* BaseTypeId: int? {FK „Types”} – identyfikator typu bazowego (opcjonalny),
* OwnerNamespace: Namespace – właściwość nawigacyjna do przestrzeni nazw, w której jest zdefiniowany dany typ,
* BaseType: TypeDef? – właściwość nawigacyjna do typu bazowego (opcjonalna),
* DerivedTypes: ICollection<TypeDef> – właściwość nawigacyjna do typów pochodnych od danego typu,
* EnumValues: ICollection<EnumValue> – właściwość nawigacyjna do wartości wyliczeniowych (dotyczy typów wyliczeniowych),
* EnumValuesDictionary: Dictionary<string, EnumValue> – słownik wartości wyliczeniowych,
* Properties: ICollection<Property> – właściwość nawigacyjna do właściwości,
* PropertiesDictionary: Dictionary<string, Property> – słownik wartości właściwości.

## Wartości wyliczane

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

* Id: int {autonum, PK} – identyfikator encji,
* Name: string – nazwa wartości wyliczanej,
* OrdNum: int? – numer kolejny wartości (od zera),
* OwnerTypeId: int {FK „Types”} – identyfikator typu wyliczeniowego, dla którego zdefiniowano tę wartość,
* OwnerType: TypeDef – właściwość nawigacyjna do typu wyliczeniowego, dla którego podano tę wartość.

## Właściwości

Właściwości są zapisywane w tabeli Properties. Encja Property ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* Name: string – nazwa właściwości,
* OrdNum: int? – numer kolejny wartości (od zera),
* OwnerTypeId: int {FK „Types”} – identyfikator typu, dla którego zdefiniowano tę właściwość,
* ValueTypeId: int {FK „Types”} – identyfikator typu wartości,
* OwnerType: TypeDef – właściwość nawigacyjna do typu, dla którego zdefiniowano tę właściwość,
* ValueType: TypeDef – właściwość nawigacyjna do typu wartości.

## Relacje związane z typami

Relacje wartości wyliczanych, właściwości, typów i przestrzeni nazw są przedstawione na poniższym rysunku.

A screenshot of a computer

Description automatically generated

Rys. 4. Relacje związane z typami

# Kontekst danych

Kontekst danych reprezentuje klasa LibDbContext. Klasa ta:

1. Deklaruje zbiory danych (DbSet) reprezentujące tabele.
2. Deklaruje słowniki plików i przestrzeni nazw (FilesDictionary i NamespacesDictionary). Słowniki te umożliwiają 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 słowniki plików FilesDictionary i przestrzeni nazw NamespacesDictionary. Do zbiorów danych SchemaFiles i Namespaces przypisuje procedury obsługi zdarzenia dodania encji, które dodają encje do odpowiedniego słownika.
7. W metodzie LoadNamespaces inicjuje słowniki typów, atrybutów 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 modelu schematu

Za parsowanie odpowiada klasa publiczna OpenXmlLibParser zadeklarowana w module ModelOpenXmlLibApp.

Klasa OpenXmlLibParser ma główną metodą publiczną ParseLibraries, dla której podaje się:

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

Metoda ta:

1. Ładuje wszystkie pliki DLL z podanego katalogu do zbioru MetadataLoadContext.
2. Tworzy wspólny dla wszystkich innych metod obiekt dbContext klasy LibDbContext.
3. Przegląda pliki ze zbioru MetadataLoadContext tworząc encje File, Namespace i FileNamespace i wypełniając odpowiednie tabele w bazie danych.
4. Wczytuje obiekty ze zbioru MetadataLoadContext.

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 LibDbContext w metodzie OnModelCreating oraz w metodach LoadFiles i LoadNamespaces. 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;
* NamespacesTotal, NamespacesAdded;
* FileNamespacesTotal, FileNamespacesAdded;
* TypesTotal, TypesAdded, TypesUpdated;
* PropertiesTotal, PropertiesAdded, PropertiesUpdated;
* EnumValuesTotal, EnumValuesAdded;

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 2 pliki DLL:

* DocumentFormat.OpenXml.dll,
* DocumentFormat.OpenXml.Framework.dll.

W katalogu Libraries umieszczono jeszcze kilka innych bibliotek DLL, które są potrzebne do wczytania powyższych bibliotek. Typy z innych bibliotek są wczytywane tylko wtedy, gdy są wykorzystywane w pierwszych dwóch, a wówczas nie są rozwijane.

Wyniki

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

Tab. 1. Podsumowanie wyników

|  |  |
| --- | --- |
| Obiekty | Liczba |
| Pliki | 9 |
| Przestrzenie nazw | 222 |
| Relacje plików z przestrzeniami nazw | 232 |
| Typy | 5166 |
| Właściwości | 12407 |
| Wartości wyliczane | 4309 |

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 prawie 8 min (przy pustej bazie danych) do 4 sekund (przy braku konieczności zmian w bazie danych).