|  |  |
| --- | --- |
| Symbol projektu:  Model Xml Schema | |
| Nazwa projektu:  **Model obiektowy Xml Schema** | |
| Tytuł dokumentu: | **Dokumentacja modelu** |
| Nr wersji: | **1.0** |
| Data utworzenia: | **22.06.2024** |
| Data ostatniej aktualizacji: | **23.06.2024** |
| Osoba odpowiedzialna: | **Jarosław Kuchta (JK)** |
| Autorzy: | **Jarosław Kuchta (JK)** |

Historia dokumentu

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Wersja | Opis modyfikacji | Autor | Data otwarcia | Data zamknięcia |
| 1.0 | Ekstrakcja z ModelDocumentation | JK | 22.06.24 |  |

# Wprowadzenie

Projekt Model Xml Schema reprezentuje model obiektowy schematu OpenXml pobrany z plików XSD i zapisany w bazie danych MS Access.

Projekt składa się z dwóch modułów:

* ModelXmlSchema – moduł biblioteczny definiujący model obiektowy i parser plików XSD.
* ModelXmlSchema App – aplikacji konsolowej do parsowania plików i zapisywania modelu w bazie danych.

Projekt czyta i parsuje pliki XSD zebrane w katalogu d:\VS\Docs\OpenXml\Schemas i zapisuje model schematu w bazie danych MS Access o nazwie XmlSchema.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 schematu XSD

Dokumenty pakietu Office (a w szczególności dokumenty programu Word) są w swojej istocie pakietami ZIP składającymi się z plików XML.

Schemat standardu OpenXml zapisany w plikach z rozszerzeniem XSD zawiera definicje elementów i atrybutów plików XML. Elementy są definiowane głównie jako typy złożone (o nazwach zaczynających się od „CT\_”), a atrybuty jako typy proste (o nazwach zaczynających się od „ST\_”).

Przykład

<xsd:complexType name="CT\_Color">

<xsd:attribute name="val" type="ST\_HexColor" use="required"/>

<xsd:attribute name="themeColor" type="ST\_ThemeColor" use="optional"/>

<xsd:attribute name="themeTint" type="ST\_UcharHexNumber" use="optional"/>

<xsd:attribute name="themeShade" type="ST\_UcharHexNumber" use="optional"/>

</xsd:complexType>

W aplikacji ModelDocApp zebrano pliki XSD w jednym katalogu (tutaj: „D:\VS\Docs\OpenXml\Schemas”) i je sparsowano do bazy danych programu Access (plik Models.accdb). Zapisano je w tabelach o nazwach zaczynających się od „Schema”.

## Pliki i przestrzenie nazw

Schematy XSD zapisane są w kilkudziesięciu plikach z rozszerzeniem XSD. Każdy z tych plików definiuje elementy w określonej przestrzeni nazw schematu. Pliki schematu są przechowywane w tabeli SchemaFiles. Każdą encję reprezentuje klasa SchemaFile o właściwościach:

* Id: int {autonum, PK} – identyfikator encji,
* FileName: string – nazwa pliku (bez ścieżki i rozszerzenia),
* TargetNamespaceId: int {FK „SchemaNamespaces”} – identyfikator przestrzeni nazw definiowanej w tym pliku,
* SchemaNamespace? TargetNamespace – właściwość nawigacyjna do przestrzeni nazw definiowanej w tym pliku.

Przestrzenie nazw są przechowywane w tabeli SchemaNamespaces. Każdą encję reprezentuje klasa SchemaNamespace o właściwościach:

* Id: int {autonum, PK} – identyfikator encji,
* Url: string – URL przestrzeni nazw (długa nazwa),
* Prefix: string[10] – prefiks przestrzeni nazw powszechnie używany w schematach.

Przestrzeni nazw jest więcej niż plików XSD, bowiem wiele plików odwołuje się do standardowych przestrzeni nazw XML i XSD. Są trzy takie przestrzenie:

* xml http://www.w3.org/XML/1998/namespace
* xsd http://www.w3.org/2001/XMLSchema
* inkml http://www.w3.org/2003/InkML

Długie nazwy przestrzeni nazw (URL) zaczynają się od:

* http://purl.oclc.org/ooxml,
* http://schemas.microsoft.com,
* http://schemas.openxmlformats.org,
* urn:schemas-microsoft-com:office.

Nie ma jednego, ustalonego standardu dla przestrzeni nazw OpenXml. Przy wczytywaniu plików XSD z różnych źródeł trzeba czasami zmieniać URL przestrzeni nazw dla dopasowania się do innych plików. W przeciwnym wypadku mogą się pojawić podwójne definicje tych samych elementów.

W każdym pliku schematu mogą być wykorzystywane inne przestrzenie nazw. Do rejestracji używanych przestrzeni nazw służy tabela SchemaUsedNamespaces. Encja SchemaUsedNamespace reprezentuje relację wiele-wiele pomiędzy plikami schematów i przestrzeniami nazw i ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* SchemaFileId: int {FK „SchemaFiles”} – identyfikator pliku schematu,
* SchemaNamespaceId: int {FK „SchemaNamespaces”} – identyfikator przestrzeni nazw,
* Prefix: string[10] – prefiks przestrzeni nazw używany w danym pliku.

Prefiksy używane do odwołania się do tej samej przestrzeni nazw mogą być różne w różnych plikach. Zostały one ujednolicone w zebranych plikach XSD tak, aby mogły zostać jednoznacznie wpisane do tabeli SchemaNamespaces. Posłużyła to tego celu klasa pomocnicza SchemaNamespacePrefix, która została wykorzystana do tymczasowego przechowywania wyników kwerendy na tabelach SchemaFiles i SchemaNamespaces.

Ponieważ nie wszystkie przestrzenie nazw zdefiniowane w plikach XSD są wykorzystywane jawnie w innych plikach, więc niektóre prefiksy zostały arbitralnie ustalone.

## Model obiektowy XmlSchema

Wszystkie klasy w modelu XmlSchema opierają się na jednej, podstawowej klasie XmlSchemaObject (rys. 2). W hierarchii dziedziczenia najbardziej rozbudowana jest gałąź XmlSchemaAnnotated. Ta klasa abstrakcyjna reprezentuje wszystkie obiekty XmlSchema, które mogą mieć adnotacje. Jednocześnie obiekty tych klas tworzą główną strukturę schematu.

Obiekt XmlSchema może zawierać:

* XmlSchemaSimpleType – typy proste,
* XmlSchemaComplexType – typy złożone,
* XmlSchemaGroup – grupy elementów, które mogą być wykorzystywane w całości przy definicji typów złożonych,
* XmlSchemaAttributeGroup – grupy atrybutów, które mogą być wykorzystywane w całości przy definicji typów złożonych,
* XmlSchemaElement – elementy globalne, które przypisują typy złożone do elementów XML,
* XmlSchemaAttribute – atrybuty globalne, które przypisują typy proste do atrybutów XML.

Rys. 2.Hierarchia dziedziczenia klas XmlSchema

### Typy proste

Typy proste określają sposób interpretacji łańcuchów tekstowych – wartości atrybutów i zawartości tekstowej elementów. Reprezentowane są przez obiekty klasy XmlSimpleType. Ich znaczenie precyzuje obiekt typu XsdSchemaSimpleTypeRestriction. Obiekt taki zazwyczaj odwołuje się do typu bazowego i może na niego nakładać ograniczenia, np. co do długości:

<xsd:simpleType name="ST\_LongHexNumber">

<xsd:restriction base="xsd:hexBinary">

<xsd:length value="4"/>

</xsd:restriction>

</xsd:simpleType>

Ograniczenia typu prostego mają fasetki (ang. Facets), które mogą być różnych typów (zależnie od typu bazowego):

* Dla typu xsd:hexBinary:
* XmlSchemaLengthFacet – określa długość ciągu w bajtach,
* Dla typu xsd:string i xsd:NCName:
* XmlSchemaMaxLengthFacet – określa maksymalną długość w znakach,
* XmlSchemaMinLengthFacet – określa minimalną długość w znakach,
* Dla typu xsd:string:
* XmlSchemaPatternFacet – określa wzorzec (wyrażenie regularne) łańcucha,
* XmlSchemaEnumerationFacet – zazwyczaj jest ich kilka, określają dopuszczalne wartości łańcucha,
* XmlSchemaWhiteSpaceFacet – określa sposób traktowania białych znaków (spacji). Dopuszczalne wartości to preserve (zachowuj), replace (zamieniaj na spacje), collapse (znaki nowego wiersza, tabulatory, spacje, powroty karetki są zastępowane spacjami, spacje wiodące i końcowe są usuwane, a wiele spacji jest redukowanych do jednej spacji). Nie używane w przetwarzanych plikach XSD,
* Dla typów liczbowych:
* XmlSchemaMaxInclusiveFacet – określa maksymalną dozwoloną wartość,
* XmlSchemaMaxExclusiveFacet -– określa maksymalną niedozwoloną wartość
* XmlSchemaMinInclusiveFacet – określa minimalną dozwoloną wartość,
* XmlSchemaMinExclusiveFacet – określa minimalną niedozwoloną wartość,
* XmlSchemaFractionDigitsFacet – określa liczbę cyfr dziesiętnych (nie używane),
* XmlSchemaTotalDigitsFacet – określa całkowitą liczbę cyfr (nie używane).

Przykład ograniczenia długości

<xsd:simpleType name="ST\_XmlName">

<xsd:restriction base="xsd:NCName">

<xsd:minLength value="1"/>

<xsd:maxLength value="255"/>

</xsd:restriction>

</xsd:simpleType>

Przykład wzorca

<xsd:simpleType name="ST\_TextScalePercent">

<xsd:restriction base="xsd:string">

<xsd:pattern value="0\*(600|([0-5]?[0-9]?[0-9]))%"/>

</xsd:restriction>

</xsd:simpleType>

Przykład wyliczenia wartości

<xsd:simpleType name="ST\_Em">

<xsd:restriction base="xsd:string">

<xsd:enumeration value="none"/>

<xsd:enumeration value="dot"/>

<xsd:enumeration value="comma"/>

<xsd:enumeration value="circle"/>

<xsd:enumeration value="underDot"/>

</xsd:restriction>

</xsd:simpleType>

Przykład wartości minimalnej i maksymalnej

<xsd:simpleType name="ST\_Integer255">

<xsd:restriction base="xsd:integer">

<xsd:minInclusive value="1"/>

<xsd:maxInclusive value="255"/>

</xsd:restriction>

</xsd:simpleType>

Przykład wartości wykluczonych

<xsd:simpleType name="ST\_FixedAngle">

<xsd:restriction base="ST\_Angle">

<xsd:minExclusive value="-5400000"/>

<xsd:maxExclusive value="5400000"/>

</xsd:restriction>

</xsd:simpleType>

#### Rejestrowanie typów prostych

Typy proste są rejestrowane w bazie danych w tabeli SchemaSimpleTypes. Encja SchemaSimpleType ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* SchemaNamespaceId: int {FK „SchemaNamespaces”} – identyfikator przestrzeni nazw, w której jest zdefiniowany dany typ,
* TypeName: string – nazwa typu,
* BaseNamespaceId: int {FK „SchemaNamespacess”} – identyfikator przestrzeni nazw typu bazowego (pomijany, gdy typ bazowy jest zdefiniowany w tej samej przestrzeni nazw),
* BaseTypeName: string– nazwa typu bazowego,
* Length: int – określenie długości dla typu bazowego hexBinary (w bajtach),
* MinLength: int – ograniczenie minimalnej długości łańcucha lub nazwy (w znakach),
* MaxLength: int – ograniczenie maksymalnej długości łańcucha lub nazwy (w znakach),
* MinInclusive: decimal – minimalna dozwolona wartość dla typu liczbowego,
* MaxInclusive: decimal – maksymalna dozwolona wartość dla typu liczbowego,
* MaxExclusive: decimal – maksymalna niedozwolona wartość dla typu liczbowego,
* MinExclusive: decimal – minimalna niedozwolona wartość dla typu liczbowego,

Wzorce są zapisywane w osobnej tabeli SchemaPatterns. Chociaż w analizowanych plikach XSD nie stwierdzono różnych wzorców przypisanych do tego samego typu prostego, to teoretycznie istnieje taka możliwość. Encja SchemaPattern ma następujące właściwości.

* Id: int {autonum, PK} – identyfikator encji,
* SimpleTypeID: int {FK „SchemaSimpleTypes”} – identyfikator typu prostego, do którego jest przypisany wzorzec,
* Pattern: string – tekst wzorca (wyrażenie regularne).

#### Typy wyliczeniowe

Typy wyliczeniowe w schemacie to są typy proste, które mają określone wartości do wyboru. Wartości te są zapisywane w tabeli SchemaEnumValues. Encja SchemaEnumValue ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* SimpleTypeId: int {FK „SimpleTypes”} – identyfikator typu prostego, dla którego podano tę wartość,
* EnumValueStr: string – łańcuch znaków, który jest rozpoznawalny jako dana wartość,
* EnumValueNum: int? – numer kolejny wartości (od zera).

Właściwość EnumValueStr nie musi być nazwą. Może być łańcuchem pustym. Przykładowe rozpoznawalne łańcuchy są następujące:

* -+
* 00
* 1
* 1024,768
* 1024x768
* 1pic
* 35mm
* Aa

Typ wyliczeniowy najczęściej jako bazowy ma typ string, ale też często występuje token, a w pojedynczych przypadkach ST\_String i byte.

#### Typy unijne

Typy unijne to są typy proste, które mogą być interpretowane alternatywnie, a jako zawartość w schemacie mają obiekty typu XsdSchemaSimpleTypeUnion, np.:

<xsd:simpleType name="ST\_Coordinate">

<xsd:union memberTypes="ST\_CoordinateUnqualified s:ST\_UniversalMeasure"/>

</xsd:simpleType>

Typy proste będące typami unijnymi mają jako właściwość BaseTypeName wpisane słowo kluczowe „union”.

Składowe unii są zapisywane w tabeli SchemaUnionMembers. Encja SchemaUnionMember ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* SimpleTypeId: int {FK „SimpleTypes”} – identyfikator typu prostego, do którego należy ta składowa,
* MemberNamespaceId: int? {FK „SchemaNamespaces”} – identyfikator przestrzeni nazw, do której należy typ składowej (pomijany, gdy typ składowej jest zdefiniowany w tej samej przestrzeni nazw),
* MemberTypeName: string – nazwa typu składowej.

#### Anonimowe składowe unii

W niektórych uniach jako składowe występują typy proste bez nazwy. Przykład:

<xsd:simpleType name="ST\_DoubleOrAutomatic">

<xsd:union memberTypes="xsd:double">

<xsd:simpleType>

<xsd:restriction base="xsd:string">

<xsd:enumeration value="auto"/>

</xsd:restriction>

</xsd:simpleType>

</xsd:union>

</xsd:simpleType>

Ponieważ odwołanie do składowych w bazie danych następuje przez nazwę, to takim typom w czasie zapisu do bazy danych nadawana jest nazwa „\_anon\_*N*”, gdzie *N* jest kolejnym numerem typu anonimowego.

#### Typy listowe

Typy listowe to typy proste, które określają taki sposób interpretacji łańcucha znaków jakby to miały być listy wartości. W schemacie te typy mają zawartość typu XsdSchemaSimpleTypeList, która określa typ elementów, np.:

<xsd:simpleType name="ST\_Sqref">

<xsd:list itemType="ST\_Ref"/>

</xsd:simpleType>

Typy proste będące typami listowymi mają jako właściwość BaseTypeName wpisane słowo kluczowe „list”.

Elementy składowe listy są zapisywane w tabeli SchemaListItems. Encja SchemaListItem ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* SimpleTypeId: int {FK „SimpleTypes”} – identyfikator typu prostego, do którego należy ta składowa,
* ItemNamespaceId: int? {FK „SchemaNamespaces”} – identyfikator przestrzeni nazw, do której należy typ składowej (pomijany, gdy typ składowej jest zdefiniowany w tej samej przestrzeni nazw),
* ItemTypeName: string – nazwa typu składowej.

### Typy złożone

Typy złożone określają sposób interpretacji elementów Xml. Reprezentowane są przez obiekty klasy XmlComplexType. Szczegóły są zapisane w atrybutach i elementach definicji.

Typy złożone są rejestrowane w tabeli SchemaComplexTypes. Encja SchemaComplexType ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* SchemaNamespaceId: int {FK „SchemaNamespaces”} – identyfikator przestrzeni nazw, w której jest zdefiniowany dany typ,
* TypeName: string – nazwa typu,
* BaseNamespaceId: int {FK „SchemaNamespacess”} – identyfikator przestrzeni nazw typu bazowego (pomijany, gdy typ bazowy jest zdefiniowany w tej samej przestrzeni nazw),
* BaseTypeName: string– nazwa typu bazowego, może to być zarówno typ złożony, jak i typ prosty.
* ContentType: ContentType – typ zawartości (TextOnly, Empty, ElementOnly, Mixed).

Definicje typów złożonych zawierają w sobie deklaracje atrybutów oraz ewentualnie elementów składowych.

#### Atrybuty

Deklaracje atrybutów są reprezentowane przez obiekty typu XmlSchemaAttribute, które określają nazwę atrybutu XML, nazwę typu prostego, do którego atrybut się odnosi i sposób użycia atrybutu.

Przykład

<xsd:complexType name="CT\_Boolean">

<xsd:attribute name="val" type="xsd:boolean" use="optional" default="true"/>

</xsd:complexType>

Encja SchemaAttribute ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* ComplexTypeId: int {FK „SchemaComplexTypes”} – identyfikator typu złożonego, w którym jest zadeklarowany dany atrybut,
* AttributeGroupId: int {FK „SchemaAttributeGroups”} – alternatywnie stosowany identyfikator grupy atrybutów, w którym jest zadeklarowany dany atrybut,
* SchemaNamespaceId: int {FK „SchemaNamespaces”} – alternatywnie stosowany identyfikator przestrzeni nazw, w której jest zdefiniowany dany atrybut (dotyczy atrybutów globalnych),
* RefNamespaceId: int {FK „SchemaNamespaces”} – identyfikator przestrzeni nazw atrybutu redefiniowanego,
* AttributeName: string – nazwa deklarowanego atrybutu,
* TypeNamespaceId: int {FK „SchemaNamespaces”} – identyfikator przestrzeni nazw typu atrybutu (pomijany, gdy typ atrybutu jest zdefiniowany w tej samej przestrzeni nazw),
* TypeName: string – nazwa typu atrybutu,
* Use: AttributeUse – sposób użycia atrybutu (Optional, Prohibited, Required),
* DefaultValue: string – domyślna wartość atrybutu (w zapisie tekstowym),
* IsFixed: bool – określa, czy wartość domyślna atrybutu jest wartością ustaloną (nie używane w analizowanych plikach XSD).

Jeśli SchemaAttribute ma określoną właściwość RefNamespaceId, to znaczy, że redefiniuje atrybut zdefiniowany w innej przestrzeni nazw.

Przykład redefinicji atrybutu

Poniższa deklaracja redefiniuje atrybut globalny o nazwie id zdefiniowany w przestrzeni nazw http://purl.oclc.org/ooxml/ officeDocument/relationships.

<xsd:complexType name="CT\_RelId">

<xsd:attribute ref="r:id" use="required"/>

</xsd:complexType>

Deklaracje atrybutów występują nie tylko w typach złożonych, ale też w grupach atrybutów definiowanych na poziomie głównym schematu oraz na poziomie globalnym schematu.

#### Odwołania do grup atrybutów

W definicji typu złożonego obok atrybutów mogą występować referencje do grup atrybutów definiowanych na poziomie głównym schematu. Reprezentowane są przez encję SchemaAttributeGroupRef, która ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* ComplexTypeId: int {FK „SchemaComplexTypes”} – identyfikator typu złożonego, w którym jest zadeklarowana dana referencja,
* AttributeGroupId: int {FK „SchemaAttributeGroups”} – alternatywnie stosowany identyfikator grupy atrybutów, w którym jest zadeklarowana dana referencja,
* RefNamespaceId: int {FK „SchemaNamespaces”} – identyfikator przestrzeni nazw przywoływanej grupy atrybutów (pomijany, gdy grupa jest zdefiniowana w tej samej przestrzeni nazw),
* RefName: string – nazwa przywoływanej grupy atrybutów,

Przykład odwołania do grupy atrybutów

<xsd:complexType name="CT\_SVGBlip">

<xsd:attributeGroup ref="a:AG\_Blob"/>

</xsd:complexType>

#### Elementy i inne partykuły

Elementy XML występujące w typach złożonych są zorganizowane w hierarchiczną strukturę *partykuł* (ang. particles). Jest kilka typów partykuł. Wszystkie one są pochodne od abstrakcyjnego typu SchemaParticle. Element XML jest reprezentowany przez encję SchemaElement, która jest podtypem SchemaParticle.

Oprócz tego wyróżnia się następujące partykuły:

* SchemaAny – reprezentuje dowolny element lub inną zawartość,
* SchemaGroupRef – reprezentuje odwołanie do zdefiniowanej grupy elementów.
* SchemaAll – reprezentuje grupę elementów, które mogą pojawiać się w dowolnej kolejności. Każdy element może wystąpić raz, ale nie musi.
* SchemaSequence – reprezentuje grupę elementów, które muszą pojawić się w określonej sekwencji.
* SchemaChoice – reprezentuje grupę elementów do wyboru. Tylko jeden z podanych elementów może się pojawić.

Przykład deklaracji typu złożonego z partykułami

<xsd:complexType name="CT\_QuickTimeFile">

<xsd:sequence>

<xsd:element name="extLst" type="CT\_OfficeArtExtensionList" minOccurs="0" maxOccurs="1"/>

</xsd:sequence>

<xsd:attribute ref="r:link" use="required"/>

</xsd:complexType>

Wszystkie partykuły są rejestrowane we wspólnej tabeli SchemaParticles. Typ abstrakcyjny SchemaParticle deklaruje właściwości wspólne dla wszystkich partykuł:

* Id: int {autonum, PK} – identyfikator encji,
* ParticleType: ParticleType – konkretny typ partykuły (Any, Element, GroupRef, All, Choice, Sequence),
* SchemaNamespaceId: int? {FK „SchemaNamespaces”} – identyfikator przestrzeni nazw, w której jest zadeklarowana dana partykuła (dotyczy partykuł zdefiniowanych na poziomie schematu – grup elementów i elementów globalnych),
* GroupId: int? {FK „SchemaGroups”} – identyfikator grupy, w której jest zadeklarowana dana partykuła (dotyczy partykuł zdefiniowanych w grupie elementów),
* ComplexTypeId: int? {FK „SchemaComplexTypes”} – identyfikator typu złożonego, w którym jest zadeklarowana dana partykuła (dotyczy partykuł zdefiniowanych w typie złożonym),
* ParentParticleId: int? {FK „SchemaParticles”} – identyfikator partykuły nadrzędnej, w której jest zadeklarowana dana partykuła (dotyczy partykuł zagnieżdżonych),
* OrdNum: int? – numer porządkowy partykuły zadeklarowanej w ramach innej partykuły (liczony od 1),
* MinOccurs: int? – minimalna liczba wystąpień danej partykuły,
* MaxOccurs: int? – maksymalna liczba wystąpień danej partykuły (wartość MaxInt oznacza nieograniczoną liczbę).

Każda encja ma określoną właściwość GroupId albo ComplexTypeId (w zależności od tego, gdzie jest zadeklarowana).

Encje SchemaAll, SchemaChoice i SchemaSequence pochodzą od wspólnego typu abstrakcyjnego SchemaGroupBase. Zawierają one kolekcję zagnieżdżonych partykuł.

Przykład sekwencji

<xsd:complexType name="CT\_FontCollection">

<xsd:sequence>

<xsd:element name="latin" type="CT\_TextFont" minOccurs="1" maxOccurs="1"/>

<xsd:element name="ea" type="CT\_TextFont" minOccurs="1" maxOccurs="1"/>

<xsd:element name="cs" type="CT\_TextFont" minOccurs="1" maxOccurs="1"/>

<xsd:element name="font" type="CT\_SupplementalFont" minOccurs="0" maxOccurs="unbounded"/>

<xsd:element name="extLst" type="CT\_OfficeArtExtensionList" minOccurs="0" maxOccurs="1"/>

</xsd:sequence>

</xsd:complexType>

Partykuła sekwencji występuje nawet, gdy typ złożony zawiera tylko jeden element.

Przykład partykuły Choice

<xsd:complexType name="CT\_AnimationElementChoice">

<xsd:choice minOccurs="1" maxOccurs="1">

<xsd:element name="dgm" type="CT\_AnimationDgmElement"/>

<xsd:element name="chart" type="CT\_AnimationChartElement"/>

</xsd:choice>

</xsd:complexType>

W partykule wyboru powinno być zadeklarowanych wiele elementów (lub innych partykuł) składowych.

Przykład partykuły All

<xsd:complexType name="CT\_DocPartPr">

<xsd:all>

<xsd:element name="name" type="CT\_DocPartName" minOccurs="1"/>

<xsd:element name="style" type="CT\_String" minOccurs="0"/>

<xsd:element name="category" type="CT\_DocPartCategory" minOccurs="0"/>

<xsd:element name="types" type="CT\_DocPartTypes" minOccurs="0"/>

<xsd:element name="behaviors" type="CT\_DocPartBehaviors" minOccurs="0"/>

<xsd:element name="description" type="CT\_String" minOccurs="0"/>

<xsd:element name="guid" type="CT\_Guid" minOccurs="0"/>

</xsd:all>

</xsd:complexType>

Partykuła Any występuje w ramach partykuły grupowej, najczęściej sekwencji.

Przykład partykuły Any

<xsd:complexType name="CT\_GraphicalObjectData">

<xsd:sequence>

<xsd:any minOccurs="0" maxOccurs="unbounded" processContents="strict"/>

</xsd:sequence>

<xsd:attribute name="uri" type="xsd:token" use="required"/>

</xsd:complexType>

Encja SchemaAny ma dwie właściwości:

* Namespace: string? – określa przestrzenie nazw zawierające elementy, które mogą być używane. Możliwości są następujące:
* ##any – dozwolone są elementy z dowolnej przestrzeni nazw (jest to ustawienie domyślne),
* ##other – mogą być obecne elementy z dowolnej przestrzeni nazw, która nie jest przestrzenią nazw elementu nadrzędnego,
* ##local - elementy muszą pochodzić z lokalnej przestrzeni nazw,
* ##targetNamespace - mogą być obecne elementy z przestrzeni nazw elementu nadrzędnego,
* lista odwołań do przestrzeni nazw poprzez ich URI, — mogą być obecne elementy z rozdzielanej spacjami listy przestrzeni nazw. Mogą też wystąpić słowa kluczowe ##targetNamespace i ##local.
* ProcessContents: ContentProcessing? – określa sposób walidacji elementów przez procesor XML. Możliwości są następujące:
* strict – procesor XML musi uzyskać schemat dla wymaganych przestrzeni nazw i zweryfikować elementy (jest to ustawienie domyślne),
* lax – tak samo jak strict, ale jeśli nie można uzyskać schematu, nie są zgłaszane żadne błędy,
* skip – procesor XML nie próbuje sprawdzać poprawności żadnych elementów z określonych przestrzeni nazw.

#### Elementy

Elementy są reprezentowane przez encje SchemaElement, która pochodzi od typu SchemaParticle i sama ma następujące właściwości:

* RefNamespaceId: int? {FK „SchemaNamespaces”} – identyfikator przestrzeni nazw, do której odwołuje się dany element. Dotyczy to elementów, które odwołują się do innych elementów,
* Name: string – nazwa elementu definiowanego lub nazwa elementu, do którego dany element się odwołuje,
* TypeNamespaceId: int? {FK „SchemaNamespaces”} – identyfikator przestrzeni nazw typu elementu (może być pominięty, gdy typ należy do tej samej przestrzeni nazw),
* TypeName: string – nazwa typu elementu.

Przykład

W poniższym przykładzie trzy elementy odwołują się do typów, które są zadeklarowane w tej samej przestrzeni nazw, a dwa do innej.

<xsd:complexType name="CT\_TrendlineLbl">

<xsd:sequence>

<xsd:element name="layout" type="CT\_Layout" minOccurs="0" maxOccurs="1"/>

<xsd:element name="tx" type="CT\_Tx" minOccurs="0" maxOccurs="1"/>

<xsd:element name="numFmt" type="CT\_NumFmt" minOccurs="0" maxOccurs="1"/>

<xsd:element name="spPr" type="a:CT\_ShapeProperties" minOccurs="0" maxOccurs="1"/>

<xsd:element name="txPr" type="a:CT\_TextBody" minOccurs="0" maxOccurs="1"/>

<xsd:element name="extLst" type="CT\_ExtensionList" minOccurs="0" maxOccurs="1"/>

</xsd:sequence>

</xsd:complexType>

Przykład elementu odwołującego się do innego elementu

<xsd:complexType name="CT\_TextMath">

<xsd:sequence>

<xsd:element ref="m:oMath"/>

</xsd:sequence>

</xsd:complexType>

#### Odwołania do grup elementów

W schemacie mogą być grupy elementów, do których można się odwoływać w różnych miejscach. Encja odwołania do grupy SchemaGroupRef jest pochodna od SchemaParticle i ma dwie właściwości:

* RefNamespaceId: int? {FK „SchemaNamespaces”} – identyfikator przestrzeni nazw, w której jest zadeklarowana grupa (pomijany, gdy grupa jest zdefiniowana w tej samej przestrzeni nazw co referencja),
* RefName: string – nazwa grupy, do której odwołuje się dana referencja.

Przykład

<xsd:complexType name="CT\_Drawing">

<xsd:sequence>

<xsd:group ref="EG\_Anchor" minOccurs="0" maxOccurs="unbounded"/>

</xsd:sequence>

</xsd:complexType>

### Grupy elementów

W schemacie można definiować grupy elementów, do których można się odwoływać w typach złożonych lub innych grupach elementów.

Przykład

<xsd:group name="EG\_DLblShared">

<xsd:sequence>

<xsd:element name="numFmt" type="CT\_NumFmt" minOccurs="0" maxOccurs="1"/>

<xsd:element name="spPr" type="a:CT\_ShapeProperties" minOccurs="0" maxOccurs="1"/>

<xsd:element name="txPr" type="a:CT\_TextBody" minOccurs="0" maxOccurs="1"/>

<xsd:element name="dLblPos" type="CT\_DLblPos" minOccurs="0" maxOccurs="1"/>

<xsd:element name="showLegendKey" type="CT\_Boolean" minOccurs="0" maxOccurs="1"/>

<xsd:element name="showVal" type="CT\_Boolean" minOccurs="0" maxOccurs="1"/>

<xsd:element name="showCatName" type="CT\_Boolean" minOccurs="0" maxOccurs="1"/>

<xsd:element name="showSerName" type="CT\_Boolean" minOccurs="0" maxOccurs="1"/>

<xsd:element name="showPercent" type="CT\_Boolean" minOccurs="0" maxOccurs="1"/>

<xsd:element name="showBubbleSize" type="CT\_Boolean" minOccurs="0" maxOccurs="1"/>

<xsd:element name="separator" type="xsd:string" minOccurs="0" maxOccurs="1"/>

</xsd:sequence>

</xsd:group>

Grupy są reprezentowane przez encje SchemaGroup. Encja ta ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* SchemaNamespaceId: int {FK „SchemaNamespaces”} – identyfikator przestrzeni nazw, w której jest zdefiniowana dana grupa,
* GroupName: string – nazwa grupy.

Do zdefiniowanej grupy można się odwołać przez element SchemaGroupRef.

Przykład

<xsd:group name="Group\_DLbl">

<xsd:sequence>

<xsd:element name="layout" type="CT\_Layout" minOccurs="0" maxOccurs="1"/>

<xsd:element name="tx" type="CT\_Tx" minOccurs="0" maxOccurs="1"/>

<xsd:group ref="EG\_DLblShared" minOccurs="1" maxOccurs="1"/>

</xsd:sequence>

</xsd:group>

### Grupy atrybutów

Podobnie s schemacie można definiować grupy atrybutów, do których można się odwoływać w typach złożonych lub innych grupach atrybutów.

Przykład

<xsd:attributeGroup name="AG\_Locking">

<xsd:attribute name="noGrp" type="xsd:boolean" use="optional" default="false"/>

<xsd:attribute name="noSelect" type="xsd:boolean" use="optional" default="false"/>

<xsd:attribute name="noRot" type="xsd:boolean" use="optional" default="false"/>

<xsd:attribute name="noChangeAspect" type="xsd:boolean" use="optional" default="false"/>

<xsd:attribute name="noMove" type="xsd:boolean" use="optional" default="false"/>

<xsd:attribute name="noResize" type="xsd:boolean" use="optional" default="false"/>

<xsd:attribute name="noEditPoints" type="xsd:boolean" use="optional" default="false"/>

<xsd:attribute name="noAdjustHandles" type="xsd:boolean" use="optional" default="false"/>

<xsd:attribute name="noChangeArrowheads" type="xsd:boolean" use="optional" default="false"/>

<xsd:attribute name="noChangeShapeType" type="xsd:boolean" use="optional" default="false"/>

</xsd:attributeGroup>

Grupy atrybutów są reprezentowane przez encje SchemaAttributeGroup. Encja ta ma następujące właściwości:

* Id: int {autonum, PK} – identyfikator encji,
* SchemaNamespaceId: int {FK „SchemaNamespaces”} – identyfikator przestrzeni nazw, w której jest zdefiniowana dana grupa,
* GroupName: string – nazwa grupy.

Do zdefiniowanej grupy można się odwołać przez element SchemaAttributeGroupRef.

Przykład

<xsd:complexType name="CT\_PictureLocking">

<xsd:sequence>

<xsd:element name="extLst" type="CT\_OfficeArtExtensionList" minOccurs="0" maxOccurs="1"/>

</xsd:sequence>

<xsd:attributeGroup ref="AG\_Locking"/>

<xsd:attribute name="noCrop" type="xsd:boolean" use="optional" default="false"/>

</xsd:complexType>

### Elementy globalne

Elementy globalne są definiowane na poziomie głównym schematu. Odwołują się one do zdefiniowanych typów złożonych.

Przykład

<xsd:element name="footnotes" type="CT\_Footnotes"/>

Do elementów globalnych można, ale nie trzeba się odwoływać w schemacie.

### Atrybuty globalne

Podobnie atrybuty globalne są definiowane na poziomie głównym schematu. Odwołują się one do zdefiniowanych typów prostych.

Przykłady

<xsd:attribute name="id" type="ST\_RelationshipId"/>

<xsd:attribute name="embed" type="ST\_RelationshipId"/>

<xsd:attribute name="link" type="ST\_RelationshipId"/>

<xsd:attribute name="dm" type="ST\_RelationshipId" default=""/>

<xsd:attribute name="lo" type="ST\_RelationshipId" default=""/>

<xsd:attribute name="qs" type="ST\_RelationshipId" default=""/>

<xsd:attribute name="cs" type="ST\_RelationshipId" default=""/>

<xsd:attribute name="blip" type="ST\_RelationshipId" default=""/>

<xsd:attribute name="pict" type="ST\_RelationshipId"/>

<xsd:attribute name="href" type="ST\_RelationshipId"/>

<xsd:attribute name="topLeft" type="ST\_RelationshipId"/>

<xsd:attribute name="topRight" type="ST\_RelationshipId"/>

<xsd:attribute name="bottomLeft" type="ST\_RelationshipId"/>

<xsd:attribute name="bottomRight" type="ST\_RelationshipId"/>

W odróżnieniu od elementów globalnych, do atrybutów globalnych trzeba się odwoływać w definicji typów złożonych (lub grup atrybutów), aby mogły być wykorzystane.

Przykład

<xsd:attributeGroup name="AG\_Blob">

<xsd:attribute ref="r:embed" use="optional" default=""/>

<xsd:attribute ref="r:link" use="optional" default=""/>

</xsd:attributeGroup>

...

<xsd:complexType name="CT\_Blip">

…

<xsd:attributeGroup ref="AG\_Blob"/>

<xsd:attribute name="cstate" type="ST\_BlipCompression" use="optional" default="none"/>

</xsd:complexType>

# Parsowanie modelu schematu

Za parsowanie odpowiada klasa publiczna XmlSchemaParser zadeklarowana w module ModelXmlSchemaApp.

Klasa XmlSchemaParser ma główną metodą publiczną ParseSchemaFiles, dla której podaje się:

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

Metoda ta:

1. Ładuje wszystkie pliki XSD z podanego katalogu do zbioru XmlSchemaSet i kompiluje ten zbiór.
2. Tworzy wspólny dla wszystkich innych metod obiekt dbContext klasy XmlSchemaDbContext.
3. Przegląda pliki ze zbioru XmlSchemaSet tworząc encje SchemaFile, SchemaNamespace i SchemaUsedNamespace i wypełniając odpowiednie tabele w bazie danych.
4. Ustala prefiksy przestrzeni nazw w tabeli SchemaNamespaces wykonując kwerendę grupującą na tabeli SchemaUsedNamespaces.
5. Parsuje schematy ze zbioru XmlSchemaSet.

Jeśli baza danych podana jako parametr nie istnieje, to jest tworzona.

W czasie parsowania elementów schematu program sprawdza, czy istnieją odpowiednie encje w bazie danych wyszukując je po właściwościach wchodzących w skład kluczy drugoplanowych. Jeśli nie, to tworzy brakujące encje i dodaje je do kontekstu danych. Właściwości niekluczowe są ustalane na encjach dodanych lub istniejących wcześniej.