|  |  |
| --- | --- |
| Symbol projektu:  DocxDocument. ModelGen | |
| Nazwa projektu:  **Generator modelu obiektowego dokumentu** | |
| Tytuł dokumentu: | **ModelDocumentation** |
| Nr wersji: | **2.0** |
| Data utworzenia: | **30.05.2023** |
| Data ostatniej aktualizacji: | **22.06.2024** |
| Osoba odpowiedzialna: | **Jarosław Kuchta (JK)** |
| Autorzy: | **Jarosław Kuchta (JK)** |

Historia dokumentu

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Wersja | Opis modyfikacji | Autor | Data otwarcia | Data zamknięcia |
| 1.0 | Pierwsza wersja | JK | 30.05.2023 | 27.10.2023 |
| 2.0 | Wersja z bazą danych | JK | 2.06.2024 |  |

# Wprowadzenie

Projekt ModelDocumentation służy do pobierania dokumentacji modelu obiektowego w standardzie OpenXml i przygotowania jej do dalszego wykorzystania. Są trzy źródła dokumentacji:

* deklaracje schematu OpenXml zapisane w plikach z rozszerzeniem XSD,
* specyfikacja standardu ECMA-376 opublikowana w kilku plikach PDF i przekonwertowana do formatu DOCX,
* pliki źródłowe pakietu bibliotek DocumentFormat.OpenXml.

Model obiektowy z bibliotek DocumentFormat.OpenXml jest słabo zdokumentowany. Większość komentarzy dokumentacyjnych zawartych w kodzie źródłowym jest trywialna. Najbardziej bogata dokumentacja jest w zawarta w standardzie ECMA, zwłaszcza w pliku „Ecma Office Open XML Part 1 - Fundamentals And Markup Language Reference.pdf”. Celem projektu jest powiązanie modelu obiektowego DocumentFormat.OpenXml z dokumentacją pobraną z pliku ECMA.

Projekt składa się z dwóch modułów:

* ModelDocApp – aplikacji konsolowej do parsowania plików i tworzenia bazy danych dokumentacji modelu OpenXml,
* ModelDoc – modułu dostępowego do bazy danych, który jest wykorzystywany przez aplikację ModelDocApp i może być wykorzystywany w innych aplikacjach.

Projekt działa w oparciu o bazę danych MS Access o nazwie Models.accdb położonej w ścieżce „d:\VS\Docs\DocumentModel”. 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 encji

Model encji dokumentacji składa się z następujących klas encyjnych:

* Namespace – przestrzeń nazw,
* SchemaElement – element ze schematu OpenXml,
* SchemaAttribute – atrybut ze schematu OpenXml,
* ComplexType – typ złożony ze schematu OpenXml,
* SimpleType – typ prosty ze schematu OpenXml,
* StandardElement – element ze standardu ECMA,
* StandardAttribute – atrybut ze standardu ECMA,
* Type – typ (klasa, typ wyliczeniowy) z biblioteki DocumentFormat.OpenXml,
* Property – właściwość klasy z biblioteki DocumentFormat.OpenXml.
* EnumValue – wartość typu wyliczeniowego z biblioteki DocumentFormat.OpenXml

Każda encja jest przechowywana w bazie danych w odpowiedniej tabeli, której nazwa jest nazwą encji w liczbie mnogiej (np. encja Namespace jest przechowywana w tabeli Namespaces).

## Przestrzenie nazw

*Przestrzeń nazw* (Namespace) organizuje nazwy typów. W jednej przestrzeni nazw wszystkie typy muszą mieć unikatowe nazwy. Przestrzenie zostały zaimportowane z kodu źródłowego biblioteki DocumentFormat.OpenXML. Każda przestrzeń nazw ma *nazwę złożoną* z kilku *nazw prostych* oddzielonych kropkami. Nazwa prosta jest *identyfikatorem* składającym się z ciągu liter (łacińskich) i cyfr i rozpoczynającym się od litery. Wszystkie przestrzenie nazw mają nazwy złożone rozpoczynające się od „DocumentFormat.OpenXml”.

Oprócz nazwy przestrzeń nazw ma *prefiks*, który jest krótkim (maks. 10 znaków) ciągiem liter i cyfr, pod którym przestrzeń nazw jest deklarowana w pliku XML dyrektywą xmlns. Ten prefiks jest dodawany przed nazwą elementu XML (oddzielony dwukropkiem). Nie wszystkie przestrzenie nazw mają prefiksy.

Przykład

<w:document   
...  
xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main" xmlns:w14="http://schemas.microsoft.com/office/word/2010/wordml"   
...>

<w:body>

<w:p w14:paraId="1C1F454E" w14:textId="107F22F8" w:rsidR="007D6C6C" w:rsidRPr="00C0491F" w:rsidRDefault="00732D28" w:rsidP="00F043F7">

<w:bookmarkStart w:id="0" w:name="\_GoBack"/>

<w:bookmarkEnd w:id="0"/>

<w:r>

<w:rPr>

<w:noProof/>

<w14:ligatures w14:val="standardContextual"/>

</w:rPr>

Przestrzeni nazw jest aktualnie 131. Najważniejsze z nich (w kontekście dokumentów programu Word) to:

* XmlSchema – deklaruje podstawowe typy danych atrybutów,
* DocumentFormat.OpenXml – deklaruje podstawowe typy danych modelu OpenXml,
* DocumentFormat.OpenXml.Wordprocessing (w) – deklaruje elementy główne programu Word (w wersji 2007),
* DocumentFormat.OpenXml.Office2010.Word (w14) – deklaruje elementy programu Word dodane w wersji 2010,
* DocumentFormat.OpenXml.Math (m) – deklaruje elementy równań matematycznych,
* DocumentFormat.OpenXml.Drawing (a) – deklaruje elementy rysunkowe.

Encja Namespace ma następujące właściwości:

* ID: int – automatycznie nadawany identyfikator wewnętrzny,
* FullName: string(255) – złożona nazwa przestrzeni nazw,
* Prefix: string(10)? – opcjonalny prefiks,
* Description: text? – opcjonalny opis.

## Elementy 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. 1). 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.

Rys. 2.Hierarchia dziedziczenia klas XmlSchema

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.

### 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, np.:

<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, np. 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,

#### 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.

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

Partykuła 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.

Partykuły SchemaAll, SchemaChoice i SchemaSequence pochodzą od wspólnego typu abstrakcyjnego SchemaGroupBase.

#### Elementy

#### Odwołania do grup elementów

### Grupy elementów

### Grupy atrybutów

### Elementy globalne

### Atrybuty globalne

W kolejnych rozdziałach i podrozdziałach głównej części specyfikacji opisano elementy OpenXml. Dla każdego z nich podano jaki znacznik XML reprezentuje ten element, jakie inne elementy ten element może zawierać i jakie może mieć atrybuty XML. Atrybuty mają określone typy wartości. Mogą to być typy podstawowe XML Schema (jak np. xsd:string) lub typy proste OpenXml. Każdy z typów prostych jest opisany w osobnym podrozdziale specyfikacji ECMA.

Encja Element ma następujące właściwości:

* ID: int – automatycznie nadawany identyfikator wewnętrzny,
* SpecNum: varchar(255) – numer podrozdziału ze specyfikacji ECMA,
* Ns: int – identyfikator przestrzeni nazw,
* Tag: varchar(255) – znacznik elementu OpenXml,
* FullName: varchar(255) – złożona nazwa przestrzeni nazw,
* Description: varchar(MAX)? – opcjonalny opis elementu.
* Attributes: Collection<Attribute> – kolekcja atrybutów OpenXml,
* ChildElement: Collection<ChildElement> – kolekcja relacji wiążących element OpenXml z podrzędnymi elementami OpenXml.

Encja Attribute ma następujące właściwości:

* ID: int – automatycznie nadawany identyfikator wewnętrzny,
* ElementID: int – identyfikator encji Element, do której należy
* Ns: int – identyfikator przestrzeni nazw,
* Tag: varchar(255) – znacznik atrybutu OpenXml,
* FullName: varchar(255) – złożona nazwa przestrzeni nazw,
* Description: varchar(MAX)? – opcjonalny opis elementu.
* SimpleTypeID: int – identyfikator typu prostego wartości atrybutu.

Encja SimpleType ma następujące właściwości:

* ID: int – automatycznie nadawany identyfikator wewnętrzny,
* SpecNum: varchar(255)? – opcjonalny numer podrozdziału ze specyfikacji ECMA,
* Ns: int – identyfikator przestrzeni nazw,
* TypeName: varchar(255) – nazwa typu prostego OpenXml,
* FullName: varchar(255) – złożona nazwa przestrzeni nazw,
* Description: varchar(MAX)? – opcjonalny opis elementu.
* Values: Collection<SimpleTypeValue> – kolekcja wartości typu prostego.