|  |  |  |  |
| --- | --- | --- | --- |
| Logotyp  C:\Users\Jerry\Documents\doseLOGO.png | Nazwa organizacji  Politechnika Gdańska | | |
| Symbol projektu  DOSE | Nazwa projektu  Document Oriented System Engineering | | |
| Symbol zadania  IML | Nazwa zadania  Model obiektowy semantyki języka IML | | |
| Typ dokumentu  Opis implementacji | Tytuł dokumentu  Model IML | | |
| Data utworzenia dokumentu  2014-06-28 | Numer wersji  1.0.1 | Edycja | Data ostatniej aktualizacji  2014-07-04 |
| Osoba odpowiedzialna  Jarosław Kuchta | Autor tej wersji dokumentu  Jarosław Kuchta | | Status  Szkic |

Spis treści

[1. Wprowadzenie 2](#_Toc391841592)

[1.1. Poziomy „meta” modelu 2](#_Toc391841593)

[1.2. Przestrzenie nazw 2](#_Toc391841594)

[1.3. Klasy a interfejsy 3](#_Toc391841595)

[1.4. Pliki partial i shared 3](#_Toc391841596)

[2. Bazowe elementy infrastruktury języka IML 4](#_Toc391841597)

[2.1. Atrybuty 4](#_Toc391841598)

[2.1.1. CanBelongToAttribute 4](#_Toc391841599)

[2.1.2. CanContainAttribute 5](#_Toc391841600)

[2.1.3. CanReferToAttribute 6](#_Toc391841601)

[2.1.4. FoundationAttribute 6](#_Toc391841602)

[2.1.5. ImlSemanticAttibute 6](#_Toc391841603)

[2.1.6. IsAbstract 6](#_Toc391841604)

[2.1.7. OutcomeAttribute 6](#_Toc391841605)

[2.1.8. PropertyAttribute 7](#_Toc391841606)

[2.2. Interfejsy 7](#_Toc391841607)

# Wprowadzenie

Biblioteka IML.dll zawiera bazowy model obiektowy języka IML (ang. Implementable Modeling Language) [1]. Krótki opis języka IML przedstawiono w dokumencie [Język IML – krótka charakterystyka](IMLShortDesc.docx). Elementy języka są w bibliotece IML.dll zdefiniowane jako klasy i interfejsy obiektowe. W modelu bazowym zdefiniowano podstawowe elementy, które mogą służyć jako klasy bazowe dla innych modułów języka.

## Poziomy „meta” modelu

W specyfikacji semantyki pierwszej wersji języka UML [2] określono cztery poziomy modelowania:

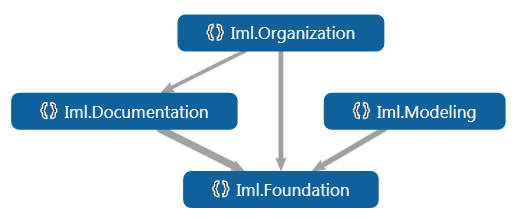
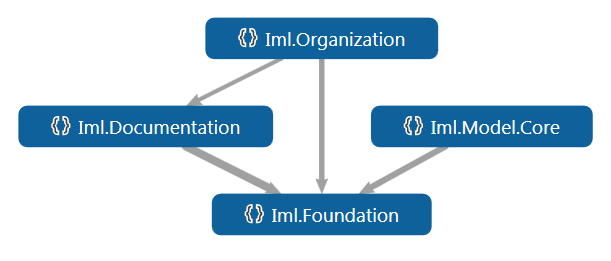
* poziom 0 – rzeczywisty, modelowany świat,
* poziom 1 – model świata zapisany w języku UML,
* poziom 2 („metamodel”) – model języka UML opisujący sposób tworzenia modelu,
* poziom 3 („meta-metamodel”) – model języka modelowania opisujący sposób definiowania języka UML.

W kolejnych wersjach ten czterowarstwowy model języka przestał być wspominany, ale w implementacji IML nie tylko opracowano poziom 2 powyższego modelu (model języka), ale też elementy poziomu 3. Na poziomie 2 wykorzystano klasy i interfejsy obiektowe języka C#, a na poziomie 3 – definiowane przez programistę atrybuty tych klas (ang. custom attributes).

## Przestrzenie nazw

W bibliotece IML.dll zdefiniowano kilka przestrzeni nazw, które ułatwiają organizację elementów (rys. 1):

* Iml.Foundation – zawierającą podstawowe elementy infrastruktury IML,
* Iml.Documentation – zawierającą definicje dokumentu i jego elementów,
* Iml.Modeling – zawierającą bazowe definicje dla definiowania konkretnych modeli języka IML,
* Iml.Organization – zawierającą elementy organizacyjne, które definiują organizację projektu.



Rys. 1. Zależności wewnętrzne między przestrzeniami nazw IML

## Klasy a interfejsy

Elementy infrastruktury języka IML są definiowane jako klasy i implementują podstawową funkcjonalność (jak np. sprawdzenie unikatowości nazw). W pewnych przypadkach jednak model obiektowy języka wymagałby podania dwóch klas bazowych dla definicji elementu. Ponieważ w C# nie ma dziedziczenia wielokrotnego, to niektóre elementy muszą być definiowane jako interfejsy.

Przykład interfejsu stanowi element nazwany (INamedElement), który odróżnia nazwę wewnętrzną (Name) od nazwy reprezentacyjnej (DisplayName) elementu. Umożliwia ponadto sprawdzenie, w jakim języku jest wyrażona nazwa reprezentacyjna.

public interface INamedElement

{

Object Instance { get; }

string Name { get; }

string DisplayName { get; }

string Language { get; }

}

## Pliki partial i shared

Projekt IML został zaimplementowany tak, aby jego elementy mogły być wykorzystywane we frameworku RIA (ang. Rich Internet Applications) dla Silverlight [3]. W tym frameworku na podstawie kodu źródłowego pisanego przez programistę dla serwera WCF jest generowany kod dla klienta Silverlight. Generator analizuje kod zapisany po stronie serwera w plikach z rozszerzeniem .cs i generuje odpowiadające im pliki po stronie serwera. Generowane pliki implementują tylko funkcjonalność związaną z przekazywaniem danych (i ich aktualizacją) między serwerem a klientem. Pliki z rozszerzeniem .partial.cs pozostają nieanalizowane po stronie serwera. Pliki z rozszerzeniem .shared.cs są kopiowane bez zmian na stronę klienta (rys. 2).

.cs

.shared.cs

.partial.cs

.shared.cs

.cs

generator

WCF RIA

strona klienta

strona serwera

Rys. 2. Przenoszenie kodu ze strony serwera na stronę klienta w WCF RIA Framework

Dlatego część logiki, która powinna być realizowana tylko po stronie serwera jest zapisana w plikach .partial.cs. Zwłaszcza to dotyczy właściwości publicznych, które nie podlegają przenoszeniu na stronę klienta. Część wspólna logiki jest zapisana w plikach .shared.cs.

# Bazowe elementy infrastruktury języka IML

Przestrzeń nazw Iml.Foundation definiuje szereg klas abstrakcyjnych (i kilka interfejsów), które stanowią później podstawę do definiowania konkretnych elementów. W tej samej przestrzeni nazw zdefiniowano też atrybuty, które mogą być używane do specyfikacji szczególnych cech elementów.

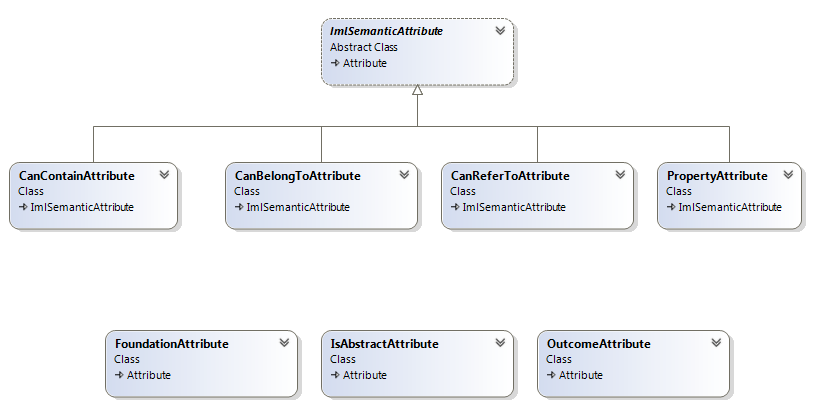
## Atrybuty

Listę zdefiniowanych atrybutów przedstawia tab. 1.

Tab. 1. Atrybuty zdefiniowane w pakiecie Iml.Foundation

|  |  |
| --- | --- |
| **Atrybut** | **Znaczenie** |
| CanBelongToAttribute | Jakie inne elementy semantyczne mogą zawierać dany element semantyczny |
| CanContainAttribute | Jakie inne elementy semantyczne może zawierać dany element semantyczny. |
| CanReferToAttribute | Do jakich innych elementów semantycznych może odwoływać się dany element semantyczny przez referencję |
| FoundationAttribute | Jakiej klasy elementy mogą być elementami bazowymi dla danej klasy elementów |
| ImlSemanticAttribute | Abstrakcyjna klasa atrybutów dla elementów IML opisująca ich cechy semantyczne. |
| IsAbstractAttribute | Klasa atrybutu umożliwiająca oznaczenie klasy elementu jako klasy abstrakcyjnej |
| OutcomeAttribute | Jakiej klasy elementy mogą być rozwiązaniem dla danej klasy elementów bazowych |
| PropertyAttribute | Jakie właściwości może posiadać dany element semantyczny |

Wśród atrybutów można wyróżnić grupę atrybutów definiujących semantykę elementów języka. Tworzą one strukturę dziedziczenia z klasą ImlSemanticAttribute jako klasą nadrzędną (rys. 3).



Rys. 3. Struktura dziedziczenia atrybutów semantycznych

Uwaga

Atrybuty: Foundation, Outcome i Property są przeznaczone dla konkretnych modeli IML i nie są jeszcze wykorzystywane. Ich użyteczność nie została potwierdzona.

### CanBelongToAttribute

Klasa atrybutu umożliwiająca określenie, jakie inne elementy semantyczne mogą zawierać dany element semantyczny. Określenie to obejmuje:

* klasę zawieranego elementu,
* stereotyp zawieranego elementu,
* semantykę zawartości (rolę),
* grupę elementów (przedział tabelki definicji)
* atrybuty wymagalności (dziedziczone z klasy ImlSemanticAttribute).

Relacja semantyczna określana przez tę klasę atrybutu jest komplementarna do relacji określanej przez CanContainAttribute.

Przykład

W poniższym przykładzie wykorzystano atrybut CanBelongTo do zaznaczenia, że element klasy Comment (komentarz) może należeć do elementu modelu (klasy ModelElement). Element skomentowany pełni w tej relacji rolę "annotatedElement", a komentarz – "ownedComment". Komentarz należy do grupy "ownedElement".

Listing 1

[CanBelongTo(typeof(ModelElement), Semantics = "annotatedElement", BackwardSemantics = "ownedComment", Subsets = "ownedElement")]

public class Comment : ModelElement

{

...

}

### CanContainAttribute

Klasa atrybutu umożliwiająca określenie, jakie inne elementy semantyczne może zawierać dany element semantyczny. Określenie to obejmuje:

* klasę zawieranego elementu,
* stereotyp zawieranego elementu,
* semantykę zawartości (rolę),
* grupę elementów (przedział tabelki definicji)
* atrybuty wymagalności (dziedziczone z klasy ImlSemanticAttribute).

Pozostałe atrybuty są przeniesione z definicji UML.

Relacja semantyczna określana przez tę klasę atrybutu jest komplementarna do relacji określanej przez CanBelongToAttribute.

Przykład

Poniżej zadeklarowano, że element klasy Table może zawierać elementy klasy TableRow. Dalej podany atrybut ContentProperty jest informacją dla serializatora XAML, aby nie generował elementu XML oznaczającego właściwość Rows reprezentującą zbiór wierszy. Elementy XML TableRow są bezpośrednio zawarte w elemencie Table.

[CanContain(typeof(TableRow),Ordered=true,GroupName="Rows")]

[ContentProperty("Rows")]

public class Table: Block

{

...

}

### CanReferToAttribute

Klasa atrybutu umożliwiająca określenie, do jakich innych elementów semantycznych może odwoływać się dany element semantyczny przez referencję. Określenie to obejmuje:

* klasę wskazywanego elementu,
* stereotyp wskazywanego elementu,
* semantykę referencji (rolę wskazywanego elementu),
* grupę referencji (przedział tabelki definicji)
* atrybuty wymagalności.

Pozostałe atrybuty są przeniesione z definicji UML.

Przykład

W poniższym przykładzie zadeklarowano, że element Classifier może odwoływać się do innego klasyfikatora przez relację o semantyce "general”. Może również odwoływać się do elementu klasy NamedElement przez relację "inheritedMember".

[CanReferTo(typeof(Classifier), Semantics="general")]

[CanReferTo(typeof(NamedElement), Semantics="inheritedMember", Readonly = true, Subsets = "member")]

public abstract partial class Classifier

{

...

}

### FoundationAttribute

Klasa atrybutu umożliwiająca oznaczenie jakie elementy mogą stanowić podstawę do tworzenia elementów danej klasy. Określa się:

* klasę elementu podstawy
* poziom uszczegółowienia, od którego element wejściowy jest wymagany

Atrybut ten może być stosowany przy generowaniu pewnych elementów na podstawie innych. Komplementarny do atrybutu klasy OutcomeAttribute.

### ImlSemanticAttibute

Abstrakcyjna klasa atrybutów dla elementów IML opisująca ich cechy semantyczne. Definiuje pola określające kiedy dana cecha elementu jest dozwolona, a kiedy jest wymagana. Definiuje też meta-atrybuty przeniesione z definicji UML. Wszystkie atrybuty odnoszące się do semantyki dziedziczą z tej klasy.

### IsAbstract

Klasa atrybutu umożliwiająca oznaczenie klasy elementu jako klasy abstrakcyjnej. Potrzebna, gdy klasa abstrakcyjna nie jest przenoszona na stronę klienta przez RIA. Wówczas trzeba ją oznaczyć jako zwykłą klasę i oznaczyć tym atrybutem.

Przykład

Taką klasą abstrakcyjną z punktu widzenia semantyki IML, ale konkretną ze względu na RIA, jest klasa NamedElement.

[DataContract]

[IsAbstract]

public partial class NamedElement: Element, INamedElement

{

...

}

### OutcomeAttribute

Klasa atrybutu umożliwiająca oznaczenie jakie elementy mogą stanowić wynik przy tworzeniu innych elementów na podstawie elementów danej klasy. Określa się:

* klasę elementu wynikowego
* poziom uszczegółowienia, od którego element wejściowy jest wymagany

Atrybut ten może być stosowany przy generowaniu pewnych elementów na podstawie innych. Komplementarny do atrybutu klasy FoundationAttribute.

### PropertyAttribute

Klasa atrybutu umożliwiająca określenie, jakie właściwości może posiadać dany element semantyczny. Określenie to obejmuje: nazwę właściwości oraz atrybuty wymagalności.

## Interfejsy

Listę zdefiniowanych interfejsów przedstawia tab. 2.

Tab. 2. Atrybuty zdefiniowane w pakiecie Iml.Foundation

|  |  |
| --- | --- |
| **Atrybut** | **Znaczenie** |
| INamedElement | Musi być implementowany przez element, który ma nazwę |
| INotifyObjectChanged | Służy do powiadamiania innych elementów o zmianie obiektu i jego właściwości |
| IReferencedElement | Musi być implementowany przez element akceptujący odwołania z innych elementów |
| IReferencingElement | Musi być implementowany przez element tworzący odwołania z innych elementów |

### INamedElement

Ten interfejs musi być implementowany przez element, który ma nazwę. Definiuje:

* nazwę własną
* nazwę reprezentacyjną
* język, w którym jest wyrażona nazwa reprezentacyjna

### INotifyObjectChanged

W .NET Framework jest zdefiniowany interfejs INotifyPropertyChanged, który musi implementować obiekt, który informuje inne obiekty o zmianie jakiejś swojej właściwości. Zdarzenie PropertyChanged przekazuje jednak tylko nazwę zmienionej właściwości. W IML potrzebne jest zdarzenie, które przekazuje nie tylko nazwę właściwości, ale też referencję do obiektu, w którym właściwość została zmieniona. Tę rolę pełni interfejs INotifyObjectChanged, który może być wykorzystywany zwłaszcza przez kolekcje. Kolekcja klasy ObservableCollection wprawdzie implementuje interfejs INotifyPropertyChanged, ale informuje o zmianie własnych właściwości, a nie właściwości swoich elementów.

### IReferencedElement

Obiekt, do którego odwołują się inne elementy, musi implementować ten interfejs, aby rejestrować przychodzące do niego odwołania. Przy zmianie swojej właściwości powinien poinformować o tym inne elementy.

### IReferencingElement

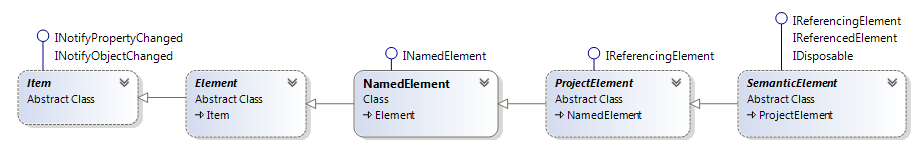
Obiekt, który tworzy referencje do innych elementów musi implementować ten interfejs, aby rejestrować wychodzące z niego odwołania.

## Klasy

### Podstawowe klasy elementów projektowych

W module IML zdefiniowano cały szereg klas abstrakcyjnych reprezentujące elementy projektu (rys. 4):

* Klasa Item – abstrakcyjna klasa bazowa dla wszystkich elementów modelu danych IML. Implementuje interfejsy INotifyPropertyChanged oraz INotifyObjectChanged.
* Klasa Element – podstawowa klasa elementu IML. Deklaruje unikatowy identyfikator. Umożliwia nawiązanie relacji zawierania między elementem tej klasy, a innym elementem nadrzędnym (właścicielskim).
* Klasa NamedElement – pierwsza nieabstrakcyjna klasa reprezentująca dowolny element, który ma nazwę. Implementuje ona interfejs INamedElement.
* Klasa ProjectElement – abstrakcyjna klasa łącząca elementy semantyczne i elementy prezentacyjne. Wspólną cechą są stereotyp.
* Klasa SemanticElement – abstrakcyjna klasa reprezentująca element semantyczny. Może zawierać inne elementy semantyczne. Może nawiązywać relacje z innymi elementami semantycznymi. Element ma identyfikator referencyjny, który jest czytelny dla człowieka, ale ma pewną formalną składnię.

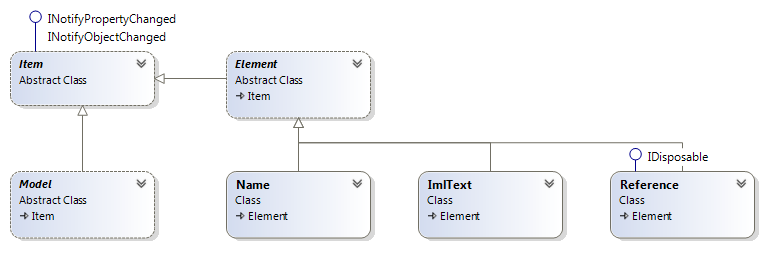


Rys. 4. Podstawowe klasy elementów projektowych

### Inne klasy podstawowe

Inne klasy podstawowe (rys. 5) to:

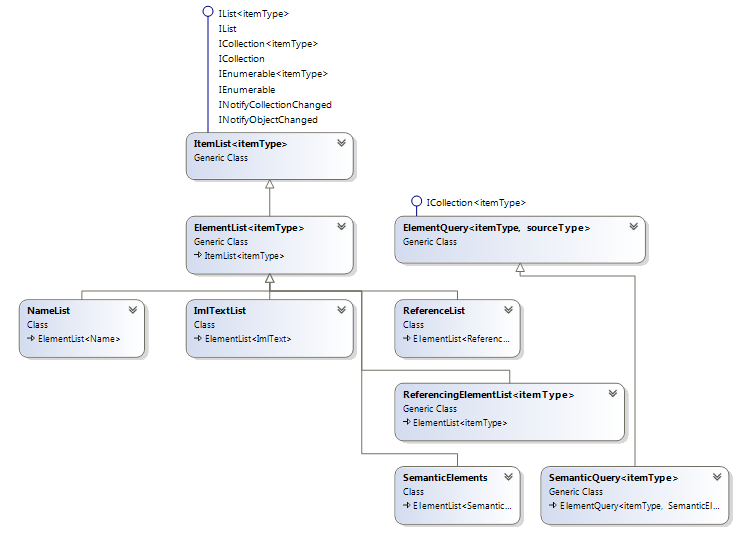
* klasa Model – abstrakcyjna klasa modelu. Pobiera klasy potomne względem SemanticElement i udostępnia taką listę klas.
* klasa Name – klasa reprezentująca nazwę elementu. Element może mieć wiele nazw w różnych językach. W jednym języku może mieć główną nazwę oraz wiele aliasów.
* klasa ImlText – klasa reprezentująca tekst, który może być prosty lub formatowany. Klasa ImlText ma atrybut Language do określenia języka, w którym podany był dany tekst.
* klasa Reference – klasa reprezentująca referencję między elementami. Rozwiązanie referencji polega na znalezieniu elementu docelowego.



Rys. 5. Inne klasy podstawowe

### Klasy kolekcji

Klasy kolekcji (rys. 6) organizują listy obiektów. W większości wywodzą się z klasy ItemList, ale niektóre z klasy ElementQuery.



Rys. 6. Klasy kolekcji

W przestrzeni nazw Iml.Foundation zdefiniowano następujące klasy kolekcji:

* klasa ItemList – uogólniona lista elementów. Sama należy do jakiegoś elementu właścicielskiego i przypisuje do niego własne elementy składowe.
* klasa ElementList – uogólniona lista elementów. Umożliwia operacje na identyfikatorach unikatowych,
* klasa NameList – lista nazw. Pilnuje, aby nie było nazw pustych oraz aby w danym języku była tylko jedna nazwa główna.
* klasa ImlTextList – lista elementów typu ImlText. Umożliwia wybór wersji językowej tekstu.
* klasa ReferenceList – lista referencji. Redefiniuje operacje na identyfikatorach.
* klasa ReferencingElementList – lista elementów z referencjami wychodzącymi. Definiuje operacje rozwiązywania referencji.
* klasa SemanticElements – lista elementów semantycznych. Potrafi rozwiązać referencje swoich elementów.
* klasa ElementQuery – kwerenda dająca dostęp do innej, źródłowej listy elementów i wybierająca z niej pewne elementy.
* klasa SemanticQuery – kwerenda elementów semantycznych.

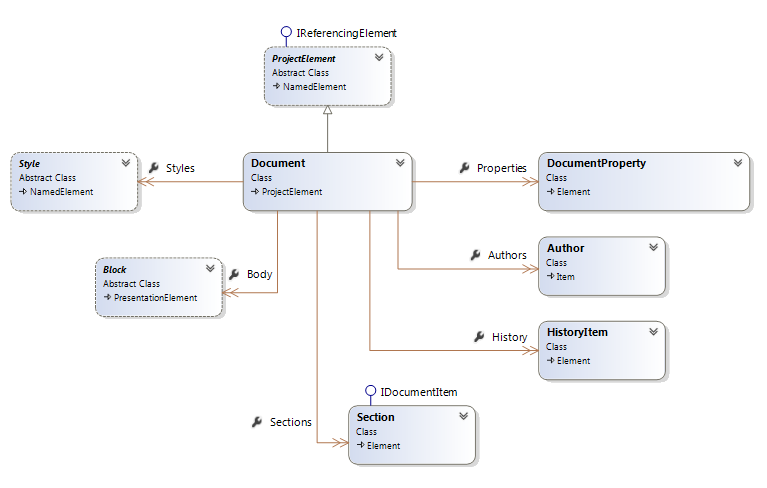
### Klasy pomocnicze

Klasy pomocnicze zdefiniowane w przestrzeni nazw Iml.Foundation to:

* klasa GuidGenerator – klasa pomocnicza tworząca identyfikator globalnie unikatowy (GUID) z dowolnego unikatowego łańcucha.
* klasa RefIDGenerator – klasa pomocnicza tworząca unikatowy identyfikator referencyjny dla podanego prefiksu.
* klasa ReferenceTypeConverter – konwerter referencji do/z postaci tekstowej. Podaje TargetID referencji.

## Model obiektowy dokumentu

Model obiektowy dokumentu jest zdefiniowany w przestrzeni nazw Iml.Documentation. Sam dokument jest zdefiniowany jako klasa pochodna od ProjectElement (rys. 7). Dokument może mieć swoje właściwości, implementuje je kolekcja Properties. Jeden dokument może mieć wielu autorów, informacje o nich udostępnia kolekcja Authors. Dokument może też mieć swoją historię (kolekcja History). Zawartość dokumentu jest reprezentowana przez ciało główne (Body), które jest kolekcją bloków oraz listę sekcji (kolekcja Sections). Sposób prezentacji treści jest określany przez kolekcję styli (Styles).



Rys. 7. Otoczenie obiektowe dokumentu

|  |  |  |  |
| --- | --- | --- | --- |
| **Właściwość** | **Typ** | **Znaczenie** | **R/W** |
| AutomaticallyUpdate | bool | Czy styl jest automatycznie aktualizowany | r/w |
| Borders | Borders | Kolekcja obramowań | r/w |
| BuiltIn | bool | Czy styl jest wbudowany | r/o |
| Description | string | Opis stylu (tworzony z jego właściwości | r/o |
| Font | Font | Formatowanie znaków | r/w |
| Frame | Frame | Formatowanie ramki | r/o |
| Hidden | bool | Czy styl jest ukryty (zarezerwowane) | r/w |
| InUse | bool | Czy styl wbudowany został użyty w dokumencie | r/o |
| LanguageID | WdLanguageID | Identyfikator języka | r/w |
| LanguageIDFarEast | WdLanguageID | Identyfikator języka dalekowschodniego | r/w |
| Linked | bool | Czy styl jest stylem powiązanym, który może być użyty zarówno przy formatowaniu znaków, jak i akapitów | r/s |
| ListLevelNumber | int | Poziom listy | r/s |
| ListTemplate | ListTemplate | Formatowanie listy | r/s |
| Locked | bool | Czy styl jest zablokowany | r/w |
| NameLocal | string | Nazwa stylu w języku użytkownika | r/w |
| NoProofing | int | Czy sprawdzanie składni i gramatyki jest ignorowane | r/w |
| NoSpaceBetween‍­‍ParagraphsOfSameStyle | bool | Czy odstępy między akapitami w tym samym stylu są usuwane | r/w |
| ParagraphFormat | ParagraphFormat | Formatowanie akapitu | r/w |
| Priority | int | Priorytet sortowania na liście stylów | r/w |
| QuickStyle | bool | Czy styl należy do listy szybkich stylów | r/w |
| Shading | Shading | Cieniowanie | r/o |
| Table | TableStyle | Właściwości tablicy | r/o |
| Type | WdStyleType | Typ stylu | r/o |
| UnhideWhenUsed | bool | Czy styl ukryty ma zostać odkryty jeśli jest użyty | r/w |
| Visibility | bool | Zarezerwowane | r/w |
| BaseStyle | dynamic | Styl bazowy | r/s |
| LinkStyle | dynamic | Styl powiązany | r/s |
| NextParagraphStyle | dynamic | Styl następnego akapitu | r/s |