Microsoft Interface Definition Language

IDL

IDL (Interface Definition Language) kompilowany jest przez MIDL.exe:

- Tworzy pliki nagłówkowe klas abstrakcyjnych dla zdefiniowanych w pliku *.idl interfejsów
- Tworzy kod *proxy* i *stub* dla serwera i klienta
- Tworzy pliki definicji UUID (*universally unique identifier*) – unikalnych identyfikatorów dla interfejsów, komponentów i *type library*
- Tworzy pliki ułatwiające rejestrację komponentu
- Tworzy skompilowaną wersję IDL – *type library*

Type Library (TLB)

Type library zawiera wszystkie informacje potrzebne do ekstrakcji struktury interfejsu. Biblioteka ta może stanowić osobny plik *.tlb, najczęściej jednak jest on przechowywany w zasobach pliku *.dll lub *.exe, w którym znajduje się odpowiadający mu komponent.

UUID

System Windows do identyfikacji komponentów stosuje UUID – 128 bitowy klucz. Liczba unikalnych wartości takiego klucza i sposób jego generacji powoduje, że wygenerowany losowo klucz jest z ogromnym prawdopodobieństwem unikalny.

- 2^{128} – umożliwia utworzenie 2 396 199 449 901 404 018 kombinacji na sekundę przez najbliższe 4.5 miliarda lat

- Generacja UUID oparta jest na:
  - 48 bitowym numerze karty sieciowej
  - Aktualnym czasie systemowym wyrażonym w 100ns odstępach od 15 października 1582

Przykładowy plik IDL

```
// Timers.idl : IDL source for Timers.dll
//

// This file will be processed by the MIDL tool to
// produce the type library (Timers.tlb) and marshalling code.

import "oaidl.idl";
import "ocidl.idl";
[
  object,
  uuid(EEBF6D1E-8EF1-4acf-9E5F-4D95E01D698A),
  helpstring("IStopwatch Interface"),
  pointer_default(unique)
]
interface IStopwatch : IUnknown
{
  [helpstring("Starts the timer")]
    HRESULT Start();

  [helpstring("Returns the number of seconds that have \
               passed since Start was called")]
    HRESULT ElapsedTime([out, retval] float* Time);
};
```

Przykładowy plik IDL

```
[
  uuid(D7EBA784-2BF1-11D3-A48B-0000861C844E),
  version(1.0),
  helpstring("Timers 1.0 Type Library")
]
library TIMERSLib
{
  importlib("stdole32.tlb");
  importlib("stdole2.tlb");

  [
    uuid(83DC3C46-1259-4f95-A2D1-CD11A8819E2E),
    helpstring("Stopwatch Class")
  ]
  coclass Stopwatch
  {
    [default] interface IStopwatch;
  };
};
```

Interface Definition Language
`import, #include`

```
import "filename";
#include "filename"
```

Dyrektywa `import` i `#include` włącza do pliku IDL zawartość innego pliku. Jej działanie jest analogiczne do dyrektywy `#include "filename"` języka C++.

Interface Definition Language
`interface`

Interfejs definiowany jest poprzez słowo kluczowe `interface`. Ogólna forma definicji ma następującą postać:

```
[ header ]
interface name : inherited_interface
{
     interface definition
};
```

Interface Definition Language
`interface: header`

W części *`header`* definiowane są atrybuty interfejsu. Dla interfejsu COM zdefiniowane są m.in. następujące atrybuty:

- **object**
- **local**
- **uuid()**
- **hidden**
- **helpstring()**
- **pointer_default()**
- **version**

```
[
  object,
  uuid(EEBF6D1E-8EF1-4acf-9E5F-4D95E01D698A),
  helpstring("IStopwatch Interface"),
  pointer_default(unique),
]
```

Interface Definition Language
`interface: header: object, local`

Atrybut `object` oznacza, że definiowany będzie interfejs COM. Dla tego typu interfejsu:
- **komponent będzie mógł być wywoływany zdalnie***
- **każda metoda musi zwracać HRESULT**

***o ile nie zdefiniowano atrybutu `local`, przy którym kompilator MIDL nie generuje kodu proxy i stub.**

Gdy nie został zdefiniowany atrybut `object` przyjmowany jest interfejs typu DCE RPC.

Interface Definition Language
`interface: header: uuid`

Atrybut `uuid()` przypisuje definiowanemu interfejsowi określony numer GUID. Ten numer jest unikalnym numerem identyfikującym interfejs. Atrybut ten jest wymagany dla interfejsu COM.

Interface Definition Language
`interface: header: helpstring, hidden`

Atrybut `hidden` powoduje, że dany interfejs nie będzie eksponowany przez narzędzia programistyczne. Będzie on natomiast poprawnie implementowany i kompilowany.
Zdefiniowany opis w atrybucie `helpstring()` używany jest przez narzędzia programistyczne do podania bardziej szczegółowych informacji o interfejsie niż sama jego nazwa. Maksymalna długość opisu wynosi 255 znaków.

Interface Definition Language
`interface: header: pointer_default`

Atrybut `pointer_default()` definiuje domyślny sposób translacji wskaźników dla argumentów metod. Sposób ten może być też indywidualnie definiowany dla każdego parametru.

`ref`:

- **Wskaźniki muszą zawsze wskazywać na prawidłowo zaalokowany obszar pamięci (nie dopuszcza się wskaźnika NULL)**
- **Wskaźnik nie zmienia się w trakcie wywołania (przed i po wskazuje na ten sam obszar pamięci)**
- **Serwer nie alokuje nowej pamięci. Zwracane wartości zostają zapisane w obszarze zaalokowanym przed wywołaniem**
- **Nie dopuszcza się aby różne zmienne wskaźnikowe wskazywały na ten sam obszar pamięci**
- **Zmienna z tym atrybutem nie może być używana do parametrów typu `out`**

Interface Definition Language
`interface: header: pointer_default`

`ptr`:

- **Dopuszczalne są wskaźniki NULL.**
- **Wskaźnik może ulec zmianie w trakcie wywołania z NULL na non-NULL (może zostać zaalokowany obszar pamięci)**
- **Dopuszcza się aby różne zmienne wskaźnikowe wskazywały na ten sam obszar pamięci**

Używanie tego typu wskaźników wiąże się z większym nakładem czasowym podczas wywołań, aby zidentyfikować dane wskazywane przez wskaźnik, czy nie jest on NULL i czy inna zmienna nie wskazuje na ten sam obszar.

Interface Definition Language
`interface: header: pointer_default`

`unique`:

- **Dopuszczalne są wskaźniki NULL**
- **Wskaźnik może ulec zmianie w trakcie wywołania (pamięć może zostać zaalokowana lub zwolniona)**
- **Gdy wskaźnik zmienia się z non-NULL na NULL pamięć nie jest zwalniana automatycznie (adres pamięci musi być zapamiętany przed wywołaniem funkcji)**
- **Nie dopuszcza się aby różne zmienne wskaźnikowe wskazywały na ten sam obszar pamięci**
- **Zmienna z tym atrybutem nie może być stosowana do określania rozmiarów struktur, np. tablic.**

Interface Definition Language
interface: methods

```
[ attributes ] HRESULT name( param-list );
```

```
interface IStopwatch : IUnknown
{
  [helpstring("Starts the timer")]
    HRESULT Start();

  [helpstring("Returns the number of seconds that have \
               passed since Start was called")]
    HRESULT ElapsedTime([out, retval] float* Time);
};
```

Interface Definition Language
`interface: methods: attributes`

Pole atrybutów definiuje cechy specyficzne dla danej metody. Najczęściej używanymi atrybutami dla interfejsów COM są:

- **hidden**
- **local, call_as()**
- **helpstring()**
- **propget**
- **propput**
- **propputref**
- **id()**

Interface Definition Language

`methods: attributes: hidden, helpstring, local, call_as()`

Atrybuty `hidden` i `helpstring` mają analogiczne znaczenie jak dla interfejsu.

Atrybuty `local` i `call_as()` pozwalają na ograniczenia dostępności określonej metody tylko gdy komponent pracuje w przestrzeni adresowej klienta (`local`) oraz pozwalają na wskazanie, że inna metoda ma być używana w przypadku wywołań zdalnych (`call_as()`).

Interface Definition Language

`methods: attributes: propget, propput, propputref`

Atrybuty `propget`, `propput` i `propputref` wskazują, że metoda służy do dostępu do właściwości. W takim przypadku nazwa metody określa nazwę właściwości. Dwie oddzielne metody o tej samej nazwie muszą zostać zdefiniowane jeżeli właściwość jest zarówno do odczytu (`propget`) jak i do zapisu (`propput`, `propputref`).

Atrybut `propputref` używany jest w przypadku, gdy argument przekazywany jest przez referencję a nie przez wartość (np. referencja do obiektu).

Interface Definition Language
`methods: attributes: id`

Atrybut `id()` definiuje identyfikator dla funkcji dla interfejsu dualnego (`IDispatch`). Argumentem atrybutu jest 32-bitowa wartość bez znaku.

Interface Definition Language
`methods: parameters`

Każdy parametr ma oprócz typu danej i nazwy zdefiniowane atrybuty. Mają one wyższy priorytet od atrybutów definiowanych dla metody. Definiują one sposób przekazywania danych do i z serwera. Podstawowymi atrybutami są:

- ref
- ptr
- unique
- in
- out
- retval
- iid_is()

Interface Definition Language

`parameters: attributes: ref, ptr, unique`

Atrybuty `ref, ptr, unique` mają identyczne znaczenie jak dla atrybutu `pointer_default` interfejsu. Zmieniają one domyślne ustawienie indywidualnie dla określonego parametru.

Interface Definition Language
`parameters: attributes: in, out`

Atrybuty `in`, `out` określają kierunek przepływu danych dla argumentu (do, z serwera lub w obie strony).
Jeżeli atrybut jest typu `out` lub `in, out` to argument musi być przekazywany przez referencję. Przekazywanie argumentów prostych z atrybutem `in` może odbywać się poprzez wartość.

Interface Definition Language
`parameters: attributes: retval`

Jeden z parametrów typu `out` może posiadać dodatkowy atrybut typu `retval`. W języku C++ w przypadku użycia tzw. *smart pointers* dla interfejsu atrybut ten powoduje, że wskazany argument będzie użyty jako wartość zwracana przez funkcję.
Gdy używany jest tzw. *raw interface*, parametr ten nie ma wpływu na postać funkcji w C++.

Interface Definition Language
`parameters: attributes: iid_is`

Atrybut `iid_is` używany jest dla argumentów przekazujących wskaźnik do różnych interfejsów za pomocą wspólnego bazowego interfejsu.
Atrybut ten wskazuje na argument, który określa GUID aktualnie żądanego interfejsu aby właściwie przeprowadzić jego *marshalling*.

Interface Definition Language
`library`

Biblioteka typów (*Type Library*) definiowana jest poprzez słowo kluczowe `library`. Ogólna forma definicji ma następującą postać:

```
[ header ]
library name
{
        library definition
};
```

Interface Definition Language
`library: header`

W części *`header`* definiowane są atrybuty biblioteki. Zdefiniowane są następujące atrybuty:

- **control**
- **uuid()**
- **hidden**
- **helpstring()**
- **version**

```
[
  uuid(D7EBA784-2BF1-11D3-A48B-0000861C844E),
  version(1.0),
  helpstring("Timers 1.0 Type Library")
]
```

Interface Definition Language

`library: header: uuid, hidden, helpstring, version, control`

Atrybuty `uuid, hidden, helpstring, version` mają analogiczne znaczenie jak dla atrybutów interfejsu.

Atrybut `control` wskazuje, że biblioteka dotyczy obiektów takich jak ActiveX COM (*controls*). Jej zawartość będzie prezentowana jedynie podczas przeglądania komponentów dla projektów GUI.

Interface Definition Language
`library: library definition`

W bloku definicji biblioteki typów znajduje się lista interfejsów i CoClass, które zostaną w niej zawarte (nazwy interfejsów pojawiających się wewnątrz bloku `coclass` nie muszą być specyfikowane ponownie w bloku `library`). Możliwe jest włączenie innej biblioteki poprzez polecenie `importlib`.

```
library TIMERSLib
{
        importlib("stdole32.tlb");
        ...
        interface IStopwatch;
        ...
        [ header ]
        coclass name
        {
                coclass definition
        }
};
```

Interface Definition Language
coclass

CoClass (*Component Class*) definiuje grupę interfejsów związanych z określonym komponentem.

```
library TIMERSLib
{
      importlib("stdole32.tlb");
      ...
      interface IStopwatch;
      ...
      [ header ]
      coclass name
      {
            coclass definition
      }
};
```

Interface Definition Language
`coclass: header`

Blok *`header`* może zawierać następujące atrybuty:

- **licensed**
- **control**
- **uuid()**
- **hidden**
- **helpstring()**
- **version**

CoClass z atrybutem `licensed` może być jedynie używana przez klientów posiadających odpowiedni klucz (licencję). Serwer musi posiadać interfejs `IClassFactory2`.

Znaczenie pozostałych atrybutów jest analogiczne jak dla nagłówka biblioteki typów.

Interface Definition Language
`coclass: coclass definition`

Blok definicji `coclass` zawiera listę interfejsów wraz z atrybutami:

```
[ attributes ] dispinterface name;
[ attributes ] interface name;
```

Zdefiniowane są następujące atrybuty:
- **source**
- **default**
- **restricted**

Interfejs określony słowem `dispinterface` musi posiadać zdefiniowany tzw. dualny interfejs (posiadać zaimplementowany interfejs `IDispatch`).

Interface Definition Language

`coclass: definition: source, default, restricted`

Atrybut `source` definiuje interfejs, który musi zostać zaimplementowany po stronie klienta dla wywołań zwrotnych (*call-back*). Serwer powinien jednocześnie zawierać interfejs `IConnectionPointContainer`.

Atrybut `default` określa domyślny interfejs dla makrojęzyków (np. Visual Basic). Atrybut ten może występować wyłącznie raz dla interfejsów bez atrybutu `source` i raz dla interfejsów typu `source`.

Atrybut `restricted` wyklucza interfejs z zestawu dostępnych dla makrojęzyków. Nie może on jednocześnie występować z atrybutem `source`.