IDL EKSPRESOWE WPROWADZENIE

Jarosław Francik

PRZYKŁAD DEFINICJI INTERFEJSU:

```
import "unknwn.idl";
[    object,
     uuid(B5DB3493-9925-4633-B7DB-30DA5E75D347),
     helpstring("przykładowy interfejs IA"),
     pointer_default(unique)
]
interface IA : IUnknown
{    HRESULT fun([in] int x, [out] int &r);
};
```

OBJAŚNIENIA:

Powyższa definicja odpowiada następującej klasie C++:

```
interface IA : public IUnknown
{    virtual void __stdcall fa() = 0;
};
```

Specyfikacja klasy jest poprzedzona fragmentem znajdującym się w nawiasach kwadratowych. Są to rozszerzenia IDL w stosunku do C++:

`import`	wstawia plik w IDL (w tym przypadku: definicję klasy IUnknown)
`object`	wyróżnik obiektu COM (IDL ma zastosowania szersze niż tylko COM)
`uuid`	identyfikator GUID
`helpstring`	zwięzły opis w języku naturalnym, udostępniany na poziomie biblioteki typów
`pointer_default`	domyślny tryb marshalingu wskaźników; dostepne wartości to:
	`ref` wskaźniki są traktowane jak referencje: nie mogą zawierać NULL, nie mogą być zmieniane, w obrębie funkcji nie mogą być tworzone aliasy
	`unique` wskaźniki mogą zawierać NULL, mogą być zmieniane, ale w obrębie funkcji nie mogą być tworzone aliasy
	`ptr` wszystko powyższe jest dozwolone

Specyfikacja funkcji składowych. Wszystkie funkcji w IDL muszą zwracać wartość HRESULT. Poszczególne parametry mogą zawierać dodatkowe określniki, stanowiące rozszerzenia IDL w stosunku do C++. Są one niezbędne dla zapewnienia prawidłowego marshalingu parametrów.

Do zdefiniowania parametrów marshalingu są potrzebne:
- adres przesyłanej zmiennej,
- kierunek przesyłu argumentu,
- rozmiar przesyłanego bloku.

Pierwszy z powyższych parametrów jest znany podczas wykonania programu.

Kierunek przesyłu argumentu określa się za pomocą następujących określników:

`in`	wyróżnik argumentu wejściowego funkcji
`out`	wyróżnik argumentu wyjściowego (wyniku) funkcji (na ogół wskaźnika lub referencji)
`in, out`	wyróżnik argumentu będącego jednocześnie wejściowym i wyjściowym

Rozmiar przesyłanego bloku jest najczęściej jednoznacznie określany na podstawie typu przesyłanych danych. W kolejnych podpunktach zostaną przedstawione elementy języka IDL pozwalające na określenie rozmiaru przesyłanych danych w przypadkach, gdy nie może on być ustalony tylko na podstawie typu danych.

Przesyłanie łańcuchów wymaga użycia określnika *string*. Należy pamiętać, że COM obsługuje wyłącznie łańcuchy *Unicode* (*wchar_t* zamiast *char*):

```
HRESULT strfun([in, string] wchar_t *pIn, [out, string] wchar_t **pOut);
```

Przesyłanie tablic wymaga określenia rozmiaru tablicy za pomocą określnika *size_is*. Pozwala on na podanie rozmiaru posługując się innym parametrem, który powinien zawierać rozmiar tablicy:

```
HRESULT arrfunIn([in] long nSize, [in, size_is(nSize)] long array[]);
HRESULT arrfunOut([out, in] long *nSize, [out, size_is(nSize)] long array[]);
```

należy zauważyć, że parametr użyty w *size_in* może być tylko wejściowy *[in]* lub wejściowo-wyjściowy *[in, out]*. Wynika z tego, że COM będzie zawsze wiedzieć, jak dużą tablicę udostępnił klient. Serwer może zapełnić tylko część tej tablicy, stosownie ustawiając parametr *nSize*. Z tego względu warto rozważyć następującą modyfikację funkcji *arrfunOut*, w której rozdzielono parametr wejściowy i wyjściowy:

```
HRESULT arrfunOut2([in] nSizeIn, [out, size_is(nSizeIn)] long array[],
                                                    [out] long *nSizeOut);
```

Przekazywanie struktur jest łatwe, gdyż IDL pozwala na bezpośrednie definiowanie struktur:

```
typedef struct Date
{     int D; int M; int Y;
} ;
// ....
      HRESULT structfun([in] Date birthdate);
```

Sprawa się komplikuje, gdy elementami struktury są wskaźniki na inne struktury, które również powinny być odwzorowane w ramach marshalingu. Kompilator IDL musi dokładnie wiedzieć, na co wskazuje wskaźnik – nie należy więc raczej stosować typu *void**. Jeśli przekazywany jest wskaźnik na interfejs COM, dobrym pomysłem jest przekazanie – w osobnym parametrze – jego IID. Służy temu specjalny określnik *iid_is*:

```
HRESULT intfun([in] const IID &iid, [out, iid_is(iid)] IUnknown **ppv);
```

Oczywiście mniej więcej to samo można uzyskać przekazując typ interfejsu wprost, np.:

```
HRESULT intfun_naive([out] IMyInterface **ppv);
```

rozwiązanie takie zemści się jednak, gdy zamiast wskaźnika na *IMyInterface* zechcemy przesłać coś, co wskazuje na jakąś klasę wywiedzoną z *IMyInterface*.

PRZYKŁAD DEFINICJI BIBLIOTEKI TYPÓW:

```
[     uuid(3221CCFF-68B2-442b-AFDC-FA9C767F1FC4),
      version(1.0),
      helpstring("Przykładowa biblioteka typów")
]
library MyTypeLib
{     importlib("stdole32.tlb") ;
      [
            uuid(E843265D-B374-4d56-A980-0EC5E9188B47),
            helpstring("Klasa komponentu")
      ]
      coclass Component
      {
            [default] interface IA;
            interface IB;
      };
} ;
```

OBJAŚNIENIA:

`library` definiuje bibliotekę typów o podanym GUID (LIBID). Biblioteka może zawierać jeden lub więcej komponentów – klas składników

`coclass` definiuje klasę składników o podanym GUID (CLSID). Klasa może udostępniać jeden lub więcej interfejsów. Jeden z nich może być domyślny, (istotne dla języków makr i *VB*).

Opis powyższy ma charakter wstępny; w szczególności pominięto zagadnienia związane z definiowaniem dispinterfejsów (*dipinterface, propget, propput, dual, oleautomation*). Wyczerpujący opis języka IDL można znaleźć w MSDN; dobrym sposobem nauki IDL jest też analizowanie różnych dostępnych definicji klas.