

Witam. Dzisiejsza lekcja jest jedna z ostatnich przed rozpoczęciem tak długo wyczekiwanej przez wszystkich zabawy z pakietem DirectX. Dzisiaj powiemy sobie o bardzo ważnej sprawie jaka jest COM (Component Object Model). Dowiemy się co to jest COM, dlaczego i jak tego używać.

Może niektórzy nie wiedza, ale cały pakiet DirectX został oparty jak na owe czasy swojego powstawania nowej technologii MS jaka był COM. Czym jest tak naprawdę ten enigmatycznie brzmiący skrót ? Czy znajomość tego przyda nam się w przyszłości ? Przekonajmy się na własnej skórze.

Ogólnie

COM to niezależny od platformy, rozprowadzany przez Microsoft zorientowany obiektowo system do tworzenia binarnych komponentów (fajnie brzmi :), które mogą współpracować z określonymi elementami naszych aplikacji - tak przynajmniej twierdzi oficjalnie MS. Dla nas jego przenośność czy fakt, że stanowi podstawę dla takich technologii jak OLE czy ActiveX nie będzie miał wielkiego znaczenia więc skupimy się raczej na praktycznym jego wykorzystaniu w programowaniu aplikacji korzystających z naszego upragnionego pakietu dla zagorzałych giercowników :). Obiekty COM'a można oczywiście pisać w różnych językach programowania, ale zastosowanie języków zorientowanych obiektowo, takich jak C++ znacznie ułatwia ich zaimplementowanie. COM pierwotnie został zaprojektowany do używania go przez użytkowników C++ i VB ale jak to w Windows bywa, wszystko się zmienia... Wymaganiem technologii COM jest to aby możliwe było w danym języku można było tworzyć struktury wskaźników i jawnie bądź nie wywoływać funkcje poprzez wskaźniki.

Obiekty, Interfejsy, Metody

W naszych przykładach będziemy mieli do czynienia z obiektami. Czym one są w technologii COM ? Otóż, każdy obiekt (w sensie programu) zawiera w sobie pewne zbiory danych oraz funkcji, które działają na tych danych. Obiekty COM'a charakteryzują się tym, że dostęp do danych obiektu realizowany jest wyłącznie poprzez jeden lub kilka zbiorów funkcji powiązanych z obiektem. Te zbiory funkcji są nazywane Interfejsami a poszczególne funkcje z danego zbioru nazywane są metodami. Wymaganiem COM'a jest również to, że dostęp do funkcji z danego interfejsu jest możliwy tylko poprzez wskaźnik do danego interfejsu. Brzmi trochę strasznie, ale na pewno wyjaśni się to przy bliższym omówieniu. Poza wyznaczeniem samego standardu dla obiektów już utworzonych (binarnych) technologia COM daje nam do ręki pewne podstawowe interfejsy, które dziedziczy każdy nowo tworzony obiekt COM-a oraz dostarcza kilka funkcji API, które pomagają obsługiwać te obiekty

Interfejs

Interfejs jak powiedzieliśmy wcześniej jest zbiorem pewnych funkcji (metod), które należą do obiektu i dają możliwość operowania na danych należących do obiektu. Nie wszystkie metody danego obiektu należą do jednego interfejsu. Ponieważ obiekt może mieć kilka interfejsów są one pogrupowane w pewne grupy funkcjonalne, ale to już zależy oczywiście od programisty piszącego taki obiekt, może sobie swoje metody pogrupować np. według nazw i poumieszczać je w odpowiednich interfejsach, ale dla własnej i innych wygody najlepiej jest to właśnie robić ze względu na to co robią a nie na długość ich nazw :)). Jest to trochę inne podejście do tego znanego np. z C++, gdzie pod pojęciem interfejsu rozumiemy wszystkie funkcje udostępniane przez daną klasę. Można jeszcze dodać, że oczywiście nie wszystkie funkcje danego obiektu muszą być umieszczone w którymś z interfejsów, mogą one pozostać niewidoczne dla kogoś kto będzie wykorzystywał taki obiekt, ale jak to zrobimy to już nasza prywatna spawa. Pragnę zaznaczyć, że raczej nie będziemy się zajmowali tworzeniem obiektów COM a tylko nauczymy się z nich korzystać, ale jeśli ktoś jest chętny to... wie co należy zrobić :)). Jeśli ktoś będzie używał naszego obiektu (jak my obiektów DirectX-a) to dostępny będzie tylko jeden interfejs danej chwili, z którego będzie mógł wykorzystywać wszystkie funkcje. Ktoś może zapytać: No dobrze a co jeśli obiekt ma więcej interfejsów ? Jest to pytanie jak najbardziej na miejscu i już za moment się wyjaśni jak korzystać z tego dobrodziejstwa. Jest kilka warunków pod którymi można tworzyć takie interfejsy ale my nie będziemy się tym zajmować, bo nie to jest celem niniejszych lekcji. To temat na niejedną książkę, ale bardzo dobrze technologia COM-a jest opisana np. w MSDN-ie więc jak kogoś to fascynuje to radzę tam zaglądnąć.

IUnknown

Obiekty COM można oczywiście dziedziczyć. Jednak dziedziczenie w tej technologii nie oznacza dziedziczenia kodu. Jeśli chcemy podziedziczyć sobie po jakimś obiekcie COM to dostajemy do ręki binarna jego wersję z udostępnionymi interfejsami. Ponieważ interfejs nie jest powiązany z implementacją (kodem) więc nie dziedziczymy kodu tylko interfejsy. W obiektach COM nie ma dziedziczenia selektywnego. Jeśli jeden interfejs dziedziczy z innego to dziedziczy wszystkie jego metody łącznie z ich definicjami. Wszystkie predefiniowane interfejsy łącznie z tymi, które my zdefiniujemy dziedziczą z bardzo ważnego interfejsu zwanego IUnknown (nazwy interfejsów w porządku napisanym obiekcie powinny zaczynać się od dużej litery I). Interfejs też zawiera trzy metody wirtualne:

```
QueryInterface()  
AddRef()  
Release()
```

Każdy obiekt COM musi implementować ten interfejs ponieważ dzięki niemu a zwłaszcza jednej z jego funkcji, a

mianowicie **QueryInterface()** będziemy mogli się poruszać i uzyskiwać wskaźniki do pozostałych interfejsów naszego obiektu ! Natomiast funkcje **AddRef()** i **Release()** będą służyć do sterowania czasem życia naszego obiektu, co jest niemiernie ważną sprawą. Przy budowie obiektów istnieje kilka obostrzeń co do tych funkcji, ale po wszelkie szczegóły jak zwykle odsyłam do MSDN-a. My natomiast zajmiemy się teraz...

Wykorzystanie

Tytuł jakże brzydko brzmiący, ale jakże ważny, dla nas użytkowników obiektów COM. Teraz dowiemy się jak wykorzystać interfejs **IUnknown** i jego metodę **QueryInterface()** aby uzyskać dostęp do innych interfejsów naszego obiektu. Kiedy mamy już wskaźnik do określonego interfejsu obiektu to możemy bez problemów dostać się do innych jego metod, interfejsów czy nawet danych. Podstawowy problem to ten jak dostać się do jakiegokolwiek interfejsu :)). Wiecie ? Jeśli nie to nie będę was trzymał dłużej w niepewności i zdradzam już te tajemnice. Można to zrobić na ... 4 sposoby ! My będziemy używać jednego a po resztę wiecie gdzie się zgłosić. My po prostu wywołamy funkcję, która da nam do ręki wskaźnik do określonego interfejsu. Będzie to funkcja dostarczana przez pakiet DirectX. Mając wskaźnik do jednego z interfejsów możemy bez problemu odwołać się do innego. Jak ? Każdy interfejs w obiekcie COM dziedziczy z **IUnknown** a skoro tak to na pewno posiada taką metodę jak **QueryInterface()** prawda ? (i my wiemy dlaczego :). Metoda ta jak napisałem powyżej służy do uzyskiwania wskaźników do wszystkich interfejsów jakie posiada dany obiekt. Działa ona w bardzo prosty sposób. My wywołując metodę **QueryInterface()** kažemy jej po prostu zapytać obiekt o to czy posiada taki czy inny interfejs. Jeśli obiekt posiada szukany przez nas interfejs to funkcja musi zwrócić nam do niego wskaźnik. I tu zaczyna się cała zabawa :)). Jeśli obiekt posiada wiele interfejsów to możemy sobie po nich skakać do woli i wywoływać wszystkie funkcje jakie udostępnia obiekt COM. Czyż nie wspaniale ? Oczywiście jeśli chcemy aby funkcja działała poprawnie (czyli zwracała nam szukany wskaźnik) to musimy jako jeden z jej parametrów podać nazwę interfejsu o który pytamy. Nazwy interfejsów jak już nadmieniałem zaczynają się na literkę 'I', co najlepiej widać po nazwie **IUnknown** a jak będziemy być może mieli okazję się przekonać w DirectX jest tak samo. Pełna lista parametrów metody (bo już nie powinniśmy dla elegancji mówić o funkcjach a raczej o metodach) **QueryInterface()** wygląda następująco:

```
HRESULT QueryInterface( REFID iid, void** ppvObject )
```

REFID to .. w zasadzie trudno powiedzieć co to jest, bo na szybko nie doszukałem się co to za typ. Ale w każdym razie dokumentacja twierdzi że to identyfikator interfejsu, do którego pragniemy uzyskać wskaźnik. Tutaj właśnie podajemy nazwę szukanego przez nas interfejsu. Drugi parametr to jak już pewnie większość się domyśliła miejsce na wskaźnik. Jeśli interfejs o podanej przez nas nazwie istnieje w obiekcie to ta zmienna będzie zawierać wskaźnik do niego. Jakiego ma być typu ? To zależy, musimy oczywiście wiedzieć (z dokumentacji) do jakiego typu interfejsu pragniemy się odwołać. W lekcjach opisujących programowanie DirectX wyjaśni się to już na przykładach.

Życie

Bardzo ważną sprawą przy obiektach COM jest czas ich istnienia w programie. Każdy obiekt ma pewien wewnętrzny licznik, który jest inkrementowany przy wywołaniu funkcji, która zwraca wskaźnik do któregoś z jego interfejsów (np. **QueryInterface()**). Robi to wspomniana gdzieś powyżej funkcja **AddRef()**, jedna z metod interfejsu **IUnknown**. Dlaczego powinniśmy zwracać na to uwagę ? Otóż, jak się łatwo domyśleć trzecia z funkcji tegoż interfejsu - **Release()** powoduje dekrementację tego licznika. I tu bardzo ważna rzecz. Jeśli licznik ten osiągnie wartość 0 to w tym momencie obiekt jest niszczone (usuwany z pamięci). Wiec jeśli najdzie nas kiedyś ochota w programie wywołać jawnie metodę **Release()** (co będzie bardzo możliwe) to musimy uważać, żeby przypadkiem nie wyzerować sobie licznika obiektu, z którym aktualnie pracujemy lub który będzie nam potrzebny. Jeśli wywołamy za dużo razy te funkcje to możemy sobie wyobrazić co będzie się działo, zwłaszcza że Windows szczególnie lubi takie błędy, ale ambitni i ciekawi mogą oczywiście spróbować - nic tak nie uczy jak doświadczenie :). Należy więc tej funkcji używać bardzo rozsądnie i nie zaciemniać sobie niepotrzebnie kodu, bo są to błędy szczególnie trudne do znalezienia.

No i to chyba na razie tyle. Mam nadzieję, że teraz już COM nie będzie wzbudzał w większości z was przerażenia, a jak się okaże później w DirectX jest to bardzo pożyteczne i ułatwia po prostu życie. Być może dużo roboty, nazwy i zmienne kosmiczne, ale jak widać da się zrozumieć. No i przede wszystkim stanowi wstęp do zrozumienia OLE (może komuś się przyda) oraz ActiveX. A może komuś wpadnie do głowy napisać coś o tworzeniu takich obiektów ? Chętnie zobaczę jakieś tutoriale na ten temat, bo jest iście fascynujący ;).