Obiekty w plikach wykonywalnych, *marshaling*

Komponent w pliku exe

Odczyt `IClassFactory` komponencie umieszczonym w pliku dll ładowanym w przestrzeń adresową klienta następuje poprzez wywołanie eksportowanej funkcji `DllGetClassObject()`. Podobnie rejestracja i wyrejestrowanie komponentu odbywa się przez eksportowane funkcje `DllRegisterServer()` i `DllUnregisterServer()`. Ze względu na fakt, że pliki uruchamialne nie eksportują funkcji i nie mogą być uruchomione w przestrzeni adresowej innej aplikacji, dostęp do obiektów odbywa się w inny sposób.

Komponent w pliku exe

Komponent aktywowany jest w pliku wykonywalnym na żądanie przez system COM poprzez uruchomienie aplikacji komponentu z parametrem *-Embedding*. Możliwe są również parametry *-RegServer* i *-UnregServer* rejestrujący i odrejestrowywujący komponent (odpowiednik funkcji `DllRegisterServer` i `DllUnregisterServer`). Komponent rejestrowany jako aplikacja powinien zostać zdefiniowany w kluczu `LocalServer32` (`InprocServer32` dla serwerów w plikach dll)

Aplikacja może również działać tradycyjnie, gdy nie podano żadnego parametru.

Komponent w pliku exe

Ponieważ aplikacje nie mają możliwości bezpośredniego przekazania referencji do `IClassFactory`, proces ten odbywa się za pośrednictwem systemu COM. W trybie *Embedding* komponent powinien uruchomić system COM, utworzyć *class factory* i zarejestrować go za pomocą funkcji `CoRegisterClassObject()`. Jeśli aplikacja obsługuje wiele komponentów, rejestracja musi być przeprowadzona dla każdego z nich.

```
CoInitializeEx(NULL,COINIT_MULTITHREADED);
IClassFactory *pClassFactory = new CFactory();

DWORD dwRegister;
CoRegisterClassObject(CLSID_Stopwatch, pClassFactory,
  CLSCTX_LOCAL_SERVER, REGCLS_MULTIPLEUSE, &dwRegister);
```

Komponent w pliku exe

```
void CoRegisterClassObject(
  REFCLSID rclsid,     //CLSID to be registered
  IUnknown * pUnk,     //Pointer to the class object
  DWORD dwClsContext,  //Context for running executable code
  DWORD flags,         //How to connect to the class object
  LPDWORD lpdwRegister //Pointer to the value returned
);
```

dwClsContext definiuje zakres osiągalności obiektu:

- **CLSCTX_INPROC_SERVER - w przestrzeni adresowej aplikacji komponentu**
- **CLSCTX_LOCAL_SERVER - w obrębie lokalnego komputera**
- **CLSCTX_REMOTE_SERVER - zdalnie**

Komponent w pliku exe

`flags` określa sposób połączenia do komponentu:

`REGCLS_SINGLEUSE` - po połączeniu klienta z obiektem, zarejestrowany obiekt *class factory* przestaje być widoczny dla kolejnych klientów. Kolejna próba utworzenia obiektu powoduje uruchomienie osobnej aplikacji komponentu.

`REGCLS_MULTIPLEUSE` - po połączeniu klienta z obiektem, zarejestrowany obiekt *class factory* wciąż jest widoczny dla kolejnych klientów. Ta sama aplikacja komponentu może więc obsługiwać wielu klientów.

Komponent w pliku exe

Po rejestracji aplikacja powinna przejść w stan oczekiwania. Aplikacja powinna wyjść ze stanu oczekiwania gdy:

- **nie istnieją już żadne instancje obiektów**
- ***class factory* ma licznik zablokowań równy zero**
- **aplikacja nie jest pod kontrolą użytkownika (dla aplikacji wizualnych)**

Przed zakończeniem działania dla każdego zarejestrowanego obiektu należy wywołać `CoRevokeClassObject()` i usunąć obiekt `IClassFactory`:

```
CoRevokeClassObject(dwRegister);
pClassFactory->Release();
CoUninitialize();
```

Komponent w pliku exe

```
HANDLE hEvent;
int main(int argc, char** argv)
{
  if (argc >1)
  {
    char* szToken = strtok(argv[1], "-/";
    if (stricmp(szToken, "Embedding") == 0)
    {
      CoInitializeEx(NULL,COINIT_MULTITHREADED);
      IClassFactory *pClassFactory = new CFactory();

      DWORD dwRegister;
      CoRegisterClassObject(CLSID_Stopwatch, pClassFactory,
        CLSCTX_LOCAL_SERVER, REGCLS_MULTIPLEUSE, &dwRegister);

      hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);
      WaitForSingleObject(hEvent, INFINITE);
      CoRevokeClassObject(dwRegister);
      pClassFactory->Release();
      CoUninitialize();
    }
  }
  return 0;
}
```

Komponent w pliku exe

```
unsigned long CStopwatch::Release()
{
  if (InterlockedDecrement( &m_nReferenceCount ) == 0)
  {
    delete this;
    if (InterlockedDecrement( &g_nServerLockCount ) == 1)
      SetEvent(hEvent);
    return 0;
  }
  return m_nReferenceCount;
}
```

Komponent jako serwis

Komponent, który ma być uruchamiany jako serwis, rejestruje w systemie komponenty podobnie jak aplikacja. Powinna ona nastąpić gdy aplikacja uruchomiona jest bez parametrów. Dopuszczalne są parametry *-RegServer* i *-UnregServer*.

Należy pamiętać, że serwis ma specyficzną procedurę uruchamiania. Powinien on rejestrować obiekty jako `REGCLS_MULTIPLEUSE` i nie powinien kończyć działania gdy wszystkie obiekty komponentów zostały zniszczone lecz na sygnał systemu zatrzymującego serwis.

Rejestracja komponentu dll, exe i serwisu

Komponent w pliku wykonywalnym i jako serwis rejestrowany jest w:
`HKEY_CLASSES_ROOT\CLSID\{`*`CLSID`*`}`
{*`CLSID`*} jest identyfikatorem klasy.
Klucz `LocalServer32` wskazuje na lokalizację pliku wykonywalnego zawierającego komponent. Wartość typu *string* `AppID` nim zawarty wskazuje na identyfikator {*`AppID`*} umieszczony w:
`HKEY_CLASSES_ROOT\AppID\{`*`AppID`*`}`
Wartości zawarte w tym kluczu określają sposób uruchomienia i prawa dostępu.

Rejestracja komponentu dll, exe i serwisu

Możliwe są następujące wartości:

- `AccessPermission` - zawiera prawa dostępu
- `ActivateAtStorage` - aktywuje komponent na serwerze, na którym znajduje się zapisany jego stan
- `AuteticationLevel` - ustala sposób autentyzacji
- `LaunchPermisions` - zawiera prawa uruchamiania
- `LocalService` - aktywacja komponentu następuje jako serwis Windows NT
- `ServiceParameters` - parametry przekazane serwisowi podczas uruchamiania
- `RemoteServerName` - określa nazwę komputera, na którym aktywowany będzie serwer
- `RunAs` - określa nazwę użytkownika użytego do aktywacji
- `DllSurogate` - określa aplikację użytą do załadowania komponentu w postaci dll (gdy pusty, użyty jest plik dllhost.exe). Komponent taki musi być zarejestrowany w `HKEY_CLASSES_ROOT\CLSID\{`*`CLSID`*`}\LocalServer32`

Marshaling

Mashaling jest to sposób przekazywania wywołań metod wraz z parametrami pomiędzy klientem i serwerem COM. Każde wywołanie kodowane jest jako zestaw bajtów przesyłanych do serwera poprzez kanał transmisyjny. Odpowiedź serwera jest również kodowana i odsyłana do klienta. Proces kodowania i dekodowania odbywa się w module *proxy* i *stub* odpowiednio klienta i serwera. Zadaniem *proxy* jest emulacja wirtualnej tablicy funkcji interfejsu komponentu, przy czym funkcje, na które wskazują elementy *vtable* zajmują się kodowaniem funkcji i parametrów, wysyłaniem do kanału transmisyjnego, oczekiwaniem na odpowiedź, a gdy taka nadejdzie dekodują ją i zwracają rezultat procesowi wywołującemu. Kod *stub* nasłuchuje na kanale transmisyjnym, odbiera nadchodzące dane, identyfikuje funkcję i parametry, wywołuje metodę komponentu, koduje odpowiedź i odsyła ją kanałem do *stub*.

Marshaling

Wyróżnia się trzy typy mashalingu:
- **standardowy**
- **poprzez informację w *Type Library***
- **niestandardowy**

Standardowy marshaling jest to sposób transmisji danych domyślnie generowany przez kompilator MIDL. Obejmuje on wszystkie podstawowe typy danych i samodefiniujące się struktury danych. Kompilator MIDL generuje dla pliku Test.idl pliki:
- **`Test.h` z deklaracją klas abstrakcyjnych interfejsów**
- **`Test_i.c` zawierający definicje stałych CLSID**
- **`Test_p.c` zawierający kod *proxy/stub***
- **`dlldata.c` z kodem dla pliku dll zawierającego *proxy/stub***
- **`Testps.def` z definicją eksportowanych funkcji**

Marshaling

Kompilacja tych plików do pliku dll (preprocesor musi mieć zdefiniowane `REGISTER_PROXY_DLL` i `_WIN32_DCOM`) powoduje wygenerowanie biblioteki-komponentu *proxy/stub* dla zdefiniowanych interfejsów.

Eksportowane funkcje biblioteki *proxy/stub* `DllRegisterServer` i `DllUnregisterServer` rejestrują komponent *proxy/stub* i interfejsy obsługiwane przez tą bibliotekę (`HKEY_CLASES_ROOT\Interface\{`*`IID`*`}`, gdzie `{`*`IID`*`}` identyfikuje interfejs, a wartość klucza `ProxyStubClsid32` wskazuje na zarejestrowany komponent *proxy/stub*).

W przypadku gdy komponent znajduje się w pliku dll możliwe jest zawarcie w nim również komponentu *proxy/stub*.

Dla standardowego marshalingu, aby komponent mógł uruchamiany na zdalnej maszynie, komponent proxy/stub musi być tam również zarejestrowany.

Marshaling

Dla określonej, dość ograniczonej grupy typów danych opracowano tzw. *type library marshaler*:

- unsigned char
- BYTE
- VARIANT_BOOL
- double
- float
- int
- long
- short
- CY
- CURRENCY
- BSTR
- DATE
- SCODE
- enum type
- VARIANT
- IDispatch*
- IUnknown*
- SAFEARRAY(type)
- type*

Jest on oparty na typach ograniczonych przez atrybut `oleautomation` interfejsu. Ogranicza to znacznie zbiór dostępnych typów, jednak wraz z interfejsem `IDispatch` gwarantuje to kompatybilność komponentu z językami skryptowymi (VBScript, JScript).

Marshaling

Niestandardowy marshaler służy do przesyłania danych o nietypowej strukturze, najczęściej definiowanej poza plikiem IDL. Wymaga to realizacji interfejsu IMarshal wraz z interfejsem komponentu i zrealizowanie operacji na poziomie kanału RPC.