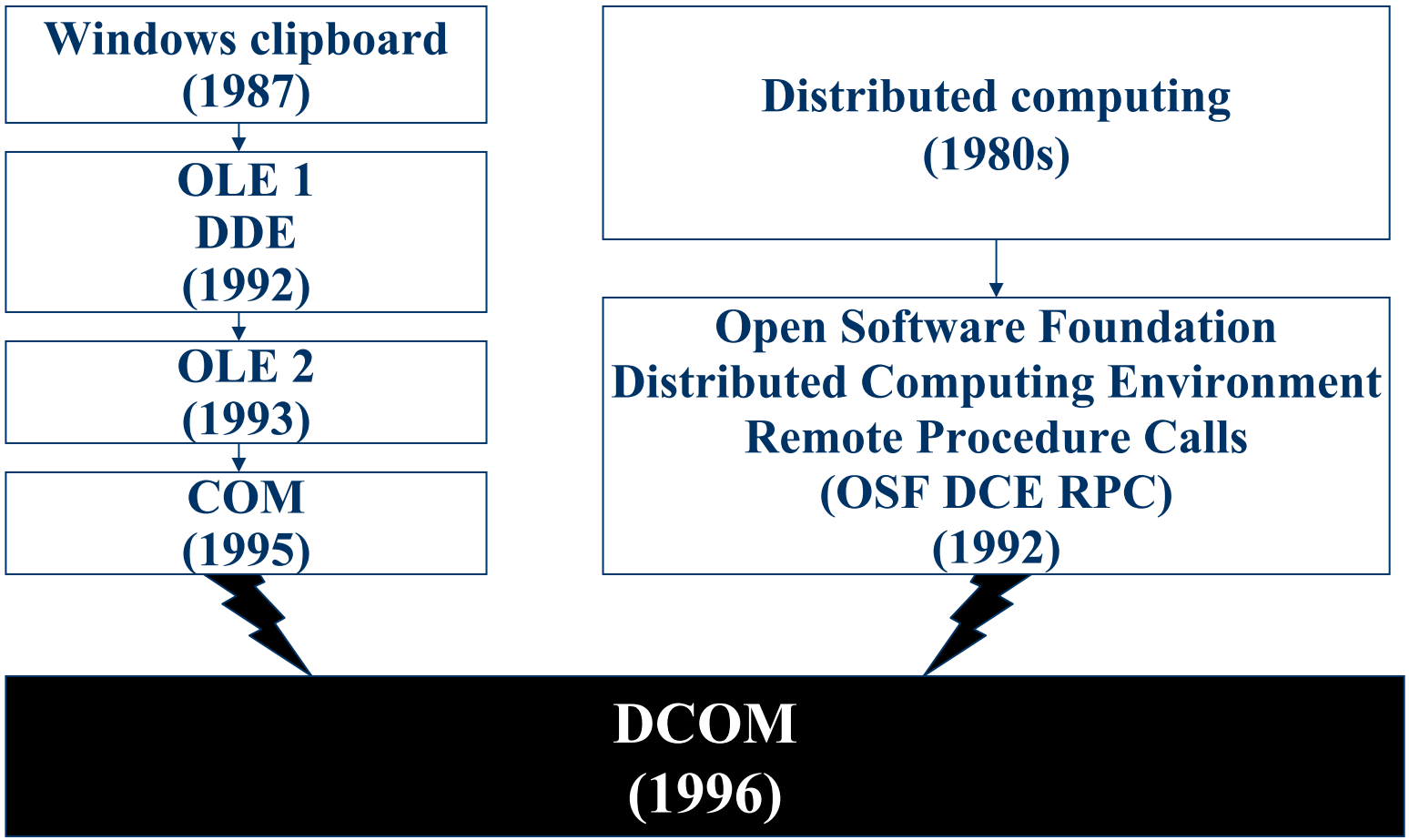
# COM jako standard

# Droga do DCOM



# Co to jest COM?

COM – Component Object Model:

- Definiuje standardowy sposób tworzenia komponentów
- Definiuje binarny standard komponentu

# Cechy COM

- Dedykowany do programowania zorientowanego obiektowo:
  - Enkapsulacja danych i kodu
  - Polimorfizm
  - Dziedziczenie
- Luźne powiązanie z aplikacją
- Bezpieczna zmiana wersji komponentu
- Przeźroczystość lokalizacji

# DLL a COM

- DLL jest również binarnym standardem
- DLL również w ograniczony sposób przeprowadza enkapsulację
- DLL nie definiuje takich zagadnień jak:
  - Alokacja pamięci
  - Czas życia obiektów
  - Standardowy dostęp do obiektów

DLL jest wystarczającym standardem aby mógł zawierać w sobie komponenty COM.

# Interfejs COM

Klient i serwer komunikują się ze sobą za pomocą interfejsów. Komponent poprzez jeden lub większą liczbę interfejsów wyraża swoją funkcjonalność i jedynie poprzez nie może być udostępniany.

Interfejs jest realizowany w postaci wirtualnych tablic funkcji (*vtable*).

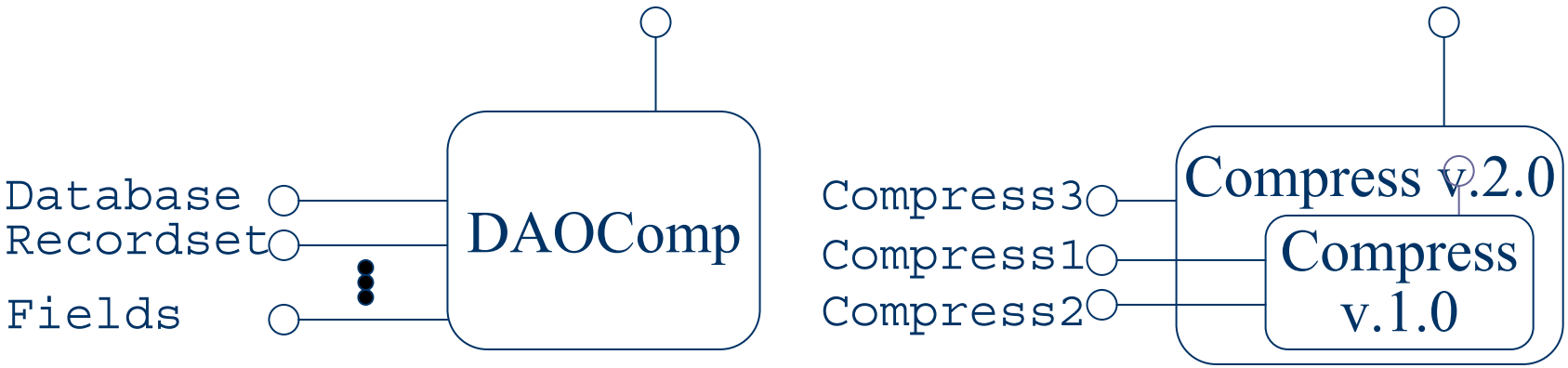
# Interfejs COM

Interfejs jest zestawem definicji funkcji, tzn. ich nazw, zwracanych wartości i parametrów.

Odpowiednikiem interfejsu w C++ są abstrakcyjne klasy. Interfejs może zawierać jedynie metody. Dostęp do danych może odbywać się jedynie poprzez funkcje zdefiniowane w interfejsie. Dane dostępne poprzez specjalnie zdefiniowane metody nazywa się właściwościami (*properties*).

# Interfejs COM

Komponent może zawierać więcej niż jeden interfejs. Wielokrotne interfejsy mogą służyć rozszerzaniu funkcjonalności lub stanowić kolejne wersje komponentu.



# Interfejs COM

Gdy komponent został już udostępniony do ogólnego użytku, interfejs nie powinien być modyfikowany zarówno pod względem struktury (vtable), jak również funkcjonalności. Konieczność jego zmiany wymaga stworzenia nowego interfejsu.

# Interfejs COM

Interfejs nie jest unikalny w systemie. Kilka komponentów może zawierać ten sam interfejs (polimorfizm).

Klient spodziewa się, że interfejs udostępniany przez komponent jest w pełni zrealizowany i może korzystać ze wszystkich metod. Jeżeli komponent udostępnia dany interfejs, musi on być w pełni zaimplementowany.

# Interfejs COM

COM definiuje dużą liczbę standardowych interfejsów. Wszystkie ich nazwy zaczynają się literą I:

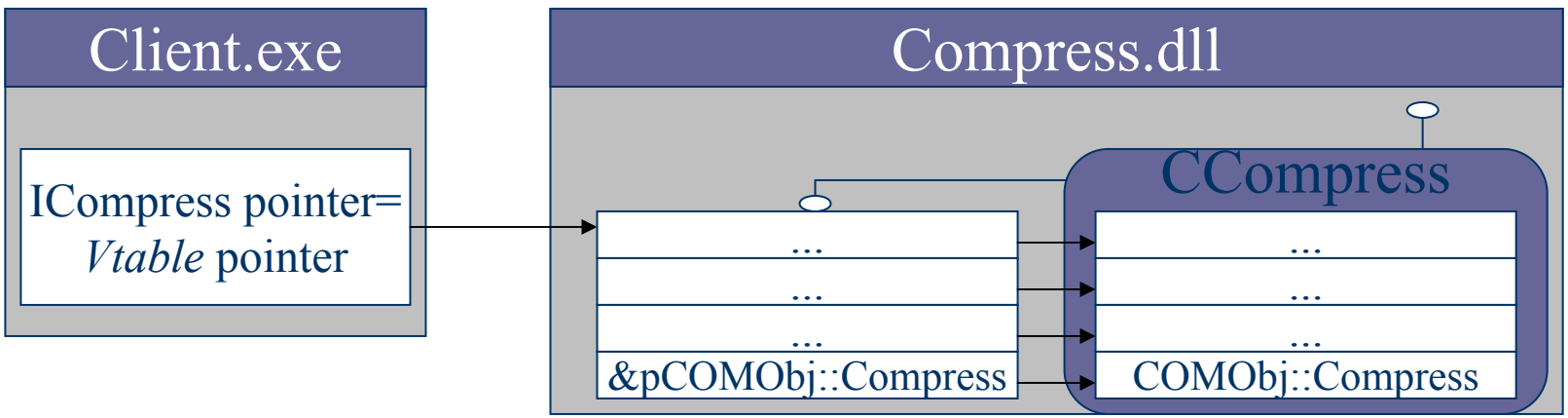
IClassFactory, IDispatch, IUnknown

Taka forma nazewnictwa nie jest wymagana dla własnych interfejsów, jednak podobnie jak dla nazewnictwa klas świadczy o dobrym stylu programisty. Implementacja interfejsu najczęściej jest tą samą nazwą zaczynającą się od C:

CCompress (implementacja interfejsu ICompress)

# Interfejs COM

W standardzie COM interfejs jest skonstruowany jako *vtable*. Ta tablica wskaźników do funkcji ma ściśle określony układ (standard binarny). *Vtable* ma identyczną strukturę jak dla klas abstrakcyjnych w C++. Dostęp do metod serwera COM następuje poprzez wskaźnik do *vtable* komponentu, a nie poprzez bezpośrednie wywoływanie funkcji.



# Komponent

Każdy komponent spełniający wymagania standardu COM musi:

- wywodzić się z bazowego interfejsu **`IUnknown`**, który realizuje podstawową funkcjonalność każdego interfejsu,
- posiadać odpowiadający mu obiekt *Class Factory* odpowiedzialny tworzenie instancji komponentu. Ma on cechy typowego komponentu (wywodzi się z **`IUnknown`**), nie posiada jednak swojego obiektu *Class Factory*

# IUnknown

IUnknown jest standardowym interfejsem COM. Interfejs ten **musi** być zaimplementowany przez wszystkie obiekty COM.

IUnknown definiuje 3 metody:

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

# IUnknown::QueryInterface()

- Sprawdza czy żądany przez klienta interfejs jest implementowany przez komponent
- Jeśli żądany interfejs istnieje, zwraca do niego wskaźnik. W przeciwnym przypadku zwracany jest kod błędu E_NOINTERFACE.

# Klasa/interfejs `IUnknown` Cechy `QueryInterface`

Podstawowe reguły `QueryInterface`

- Zestaw interfejsów komponentu nie może się zmienić w czasie w trakcie jego działania.
- Za pomocą każdego interfejsu w komponencie można zażądać dowolnego innego interfejsu w tym komponencie.
- Kolejne żądania tego samego interfejsu z jednego wątku klienta w ramach tej samej instancji komponentu powinny zwrócić ten sam wskaźnik.

Z powyższych wynikają następujące cechy `QueryInterface`:

- Symetria (*symmetry*): dla istniejącego interfejsu IA operacja zażądania interfejsu IA powinna się powieść
- Zwrotność (*reflexivity*): jeśli dla istniejącego interfejsu IA żądanie interfejsu IB przebiegło poprawnie, to z interfejsu IB operacja zażądania interfejsu IA powinna się powieść
- Przechodniość (*transitivity*): jeśli dla istniejącego interfejsu IA żądanie interfejsu IB przebiegło poprawnie, a następnie dla interfejsu IB żądanie interfejsu IC przebiegło poprawnie, to z interfejsu IC operacja zażądania interfejsu IA powinna się powieść

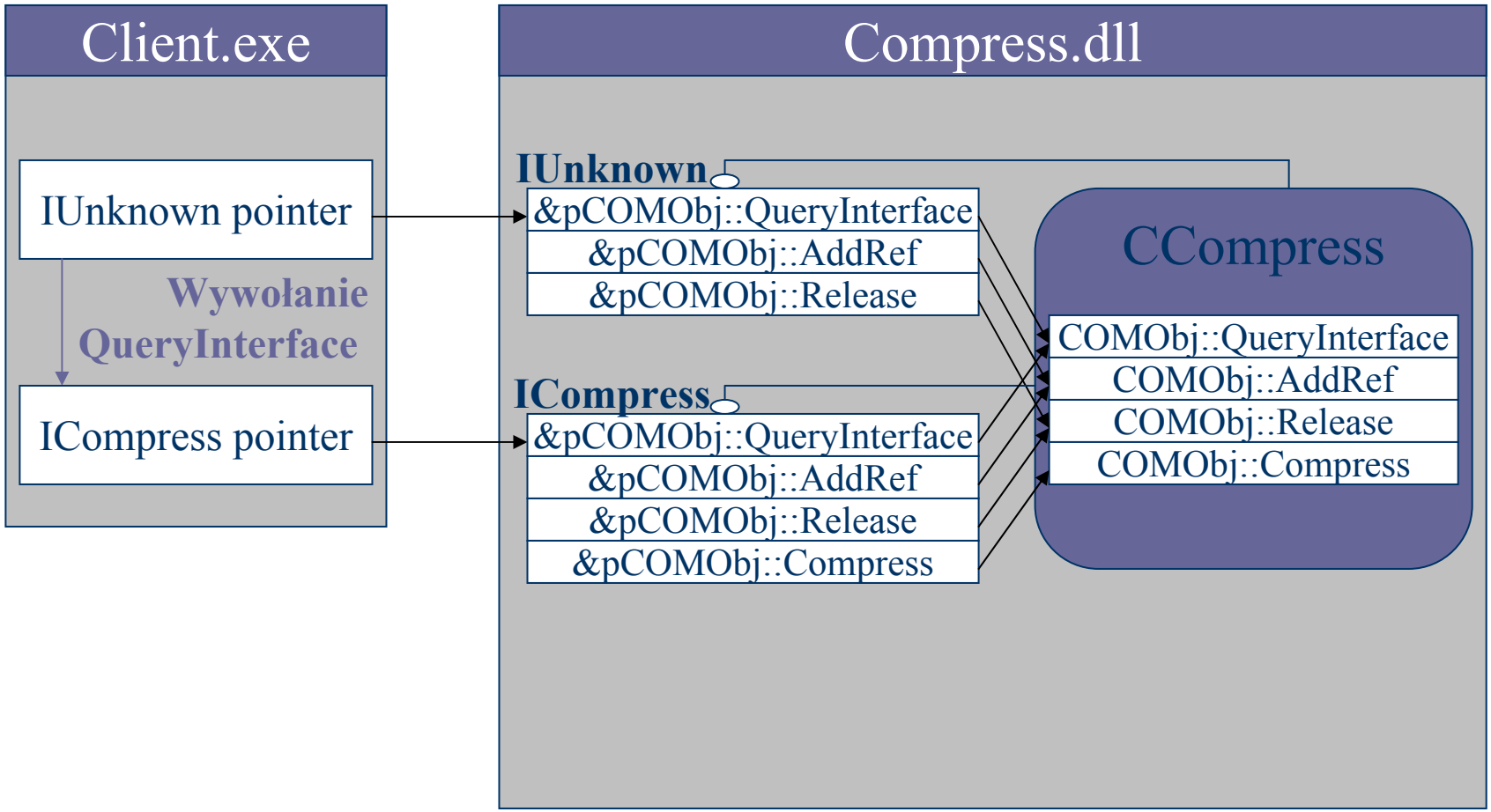
# IUnknown::AddRef(), IUnknown::Release()

Metody `AddRef()` i `Release()` odpowiedzialne są za zarządzanie licznikiem odwołań (*reference counter*).

Każdy obiekt, który łączy się z komponentem lub tworzy kopię wskaźnika do interfejsu powinien wywołać `AddRef()`, gdy natomiast nie zamierza dalej z niego korzystać – `Release()`. Gdy licznik odwołań osiąga wartość 0, komponent może wywołać swój destruktor.

**`QueryInterface()` zwracając wskaźnik do interfejsu zwiększa licznik odwołań (wywołuje wewnętrznie `AddRef()`).**

# IUnknown


Client.exe
IUnknown pointer
Wywołanie
QueryInterface
ICompress pointer
Compress.dll
IUnknown
&pCOMObj::QueryInterface
&pCOMObj::AddRef
&pCOMObj::Release
ICompress
&pCOMObj::QueryInterface
&pCOMObj::AddRef
&pCOMObj::Release
&pCOMObj::Compress
CCompress
COMObj::QueryInterface
COMObj::AddRef
COMObj::Release
COMObj::Compress

# Interfejs IUnknown

```
class IUnknown
{
public:
  virtual HRESULT __stdcall QueryInterface(
        REFIID riid,
        void **ppvObject) = 0;
  virtual unsigned long __stdcall AddRef() = 0;
  virtual unsigned long __stdcall Release() = 0;
};
```

# IClassFactory

Każdy komponent COM posiada skojarzoną *Class Factory*. Jest on odpowiedzialny za konstrukcję komponentu i czas życia modułu.

```
class IClassFactory : public IUnknown
{
public:
    virtual HRESULT __stdcall CreateInstance(
         IUnknown *pUnkOuter,
         REFIID riid,
         void **ppvObject) = 0;
    virtual HRESULT __stdcall LockServer(
         BOOL fLock) = 0;

);
```

# Przykład – Stopwatch serwer
## Klasa wirtualna `IStopwatch`

Na przykładzie realizacji klasy CStopwatch przedstawione zostaną podstawowe elementy komponentu COM.

Rozważmy następującą klasę abstrakcyjną:

```
class CStopwatch
{
public:
  virtual HRESULT __stdcall Start() = 0;
  virtual HRESULT __stdcall ElapsedTime(float *Time) = 0;
}
```

# Przykład – Stopwatch serwer
## Implementacja (klasa `CStopwatch`)

$$ElapsedTime = \frac{StopCounter - StartCounter}{Frequency} - Overhead$$

```
class CStopwatchHFT : public CStopwatch
{
public:
   CStopwatchHFT();
   virtual ~CStopwatchHFT();
private:
  // The frequency of the counter
  // returned by QueryPerformanceCounter()
  LARGE_INTEGER m_nFrequency;

  // The counter value when the start method was last called.
  LARGE_INTEGER m_nStartTime;
public:
  HRESULT __stdcall Start();
  HRESULT __stdcall ElapsedTime(float *Time);
};
```

# Przykład – Stopwatch serwer
## Implementacja (klasa `CStopwatch`)

```
CStopwatchHFT::CStopwatchHFT()
{
  // Initialize the member variables
  m_nStartTime.QuadPart = 0;

  // Save the frequency of the performance counters
  QueryPerformanceFrequency ((LARGE_INTEGER*) &m_nFrequency);
}

HRESULT __stdcall CStopwatchHFT::Start()
{
  if (QueryPerformanceCounter((LARGE_INTEGER*) &m_nStartTime))
    return S_OK;
  else
    return E_FAIL;
}
```

# Przykład – Stopwatch serwer
## Implementacja (klasa `CStopwatch`)

```
HRESULT __stdcall CStopwatchHF::ElapsedTime(float *Time)
{
  HRESULT hr;
  LARGE_INTEGER nStopTime;

  // Set the stop time immediately so that a minimum amount of
  // timer code is executed between the start and stop times
  if ( !QueryPerformanceCounter((LARGE_INTEGER*) &nStopTime)
       || (m_nStartTime.QuadPart == 0)) {
    // Either QPC failed or
    // start was not called before stop
    hr = E_FAIL;
  } else {
    *Time = (float) (nStopTime.QuadPart - m_nStartTime.QuadPart);
    *Time = (*Time / m_nFrequency.QuadPart);
    hr = S_OK;
  }
  return hr;
}
```

# Przykład – Stopwatch serwer
## Zarządzanie obiektem `CStopwatch`

Klient nie zarządza bezpośrednio konstrukcją i zwalnianiem obiektu serwera.

```
extern "C" HRESULT __stdcall DllGetClassObject(
        LPVOID* ppv)
{
  *ppv = static_cast<Cstopwatch*>(new CStopwatchHFT);
  return S_OK;
}
```

# Przykład – Stopwatch serwer
## Zarządzanie obiektem `CStopwatch`

```
class CStopwatch
{
public:
  unsigned long __stdcall Release() =0;
  virtual HRESULT __stdcall Start() =0;
  virtual HRESULT __stdcall ElapsedTime(float *Time) =0;
}

unsigned long __stdcall CStopwatchHFT::Release()
{
  delete this;
  return 0;
}
```

# Przykład – Stopwatch serwer
## Moduł DLL zawierający `CStopwatch`

```
extern "C"
BOOL __stdcall DllMain(HINSTANCE hInstance, DWORD dwReason,
  LPVOID /*lpReserved*/)
{
  return TRUE;
}



LIBRARY      "Timers.DLL"
EXPORTS
    DllGetClassObject   @1 PRIVATE
```

# Przykład – Stopwatch klient

```
// StopwatchClient.cpp :
//   Defines the entry point for the console application.
//

// std::out etc
#include <iostream>
#include "..\Timers\Stopwatch.h"

#define TIMERSDLL "..\\Timers\\Timers.dll"

typedef HRESULT (__stdcall *DLLGETCLASSOBJECT)(LPVOID* ppv);
```

# Przykład – Stopwatch klient

```
// Instantiates a stopwatch object and returns it by reference
HRESULT CreateInstance(void** ppv)
{
  HRESULT hr = E_FAIL;
  HINSTANCE hinstDll;
  DLLGETCLASSOBJECT DllGetClassObject;

  hinstDll = LoadLibrary(TIMERSDLL);
  if (hinstDll == NULL){
    std::cout << "Unable to load \"" TIMERSDLL "\"" << std::endl;
  } else {
    DllGetClassObject = (DLLGETCLASSOBJECT) GetProcAddress(
        hinstDll, "DllGetClassObject");
    if (DllGetClassObject != NULL)
      hr = DllGetClassObject(ppv);
  }
  return hr;
}
```

# Przykład – Stopwatch klient

```
int main(int argc, char* argv[])
{
  float nElapsedTime;
  CStopwatch* pStopwatch  = NULL;
  hr = CreateInstance((void**) &pStopwatch);

  if (!SUCCEEDED(hr)){
    std::cout << "ERROR: Unable to create Stopwatch!!";
  } else {
    pStopwatch->Start();
    pStopwatch->ElapsedTime(&nElapsedTime);
    std::cout << "The overhead time is " << nElapsedTime << std::endl;
    pStopwatch->Release();
    pStopwatch = NULL;
  }
  return 0;
}
```

# Przykład – Stopwatch serwer
## Klasa/interfejs `IUnknown`

```
class IStopwatch : public IUnknown
{
public:
  // unsigned long __stdcall Release() =0;
  virtual HRESULT __stdcall Start() =0;
  virtual HRESULT __stdcall ElapsedTime(float *Time) =0;
}

class CStopwatchHFT : public IStopwatch
{
  ...
public:
  // IUnknown methods
  HRESULT __stdcall QueryInterface(
    REFIID riid,
    void **ppvObject);
  unsigned long __stdcall AddRef();
  unsigned long __stdcall Release();
  ...
};
```

# Przykład – Stopwatch serwer
## Metoda `QueryInterface()`

Komponent i każdy interfejs posiadają unikalny identyfikator (UUID):

```
// {EEBF6D1E-8EF1-4acf-9E5F-4D95E01D698A}
const IID IID_IStopwatch =
  { 0xeebf6d1e, 0x8ef1, 0x4acf,
    { 0x9e, 0x5f, 0x4d, 0x95, 0xe0, 0x1d, 0x69, 0x8a } };

// {83DC3C46-1259-4f95-A2D1-CD11A8819E2E}
const CLSID CLSID_Stopwatch =
  { 0x83dc3c46, 0x1259, 0x4f95,
    { 0xa2, 0xd1, 0xcd, 0x11, 0xa8, 0x81, 0x9e, 0x2e } };
```

Standardowe interfejsy posiadają predefiniowane UUID, np.:

```
//IUnknown {00000000-0000-0000-C000-000000000046}
```

# Przykład – Stopwatch serwer
## Metoda `QueryInterface()`

```
HRESULT __stdcall CStopwatchHFT::QueryInterface(
  REFIID riid,
  void **ppvObject)
{
  HRESULT hr = S_OK;

  if (riid == IID_IUnknown) {
    *ppvObject = static_cast<IUnknown*>(static_cast<CStopwatch*>(this));
  } else if (riid == IID_IStopwatch) {
    *ppvObject = static_cast<IStopwatch*>(this);
  } else {
    ppvObject = NULL;
    hr = E_NOINTERFACE;
  }

  if (SUCCEEDED(hr))
    (static_cast<IUnknown*>(*ppvObject))->AddRef();

  return hr;
}
```

# Przykład – Stopwatch serwer
## Zarządzanie obiektem `CStopwatch`

```
extern "C" HRESULT __stdcall DllGetClassObject(
  REFCLSID rclsid,
  REFIID riid,
  LPVOID* ppv)
{
 HRESULT hr;

 if (rclsid == CLSID_Stopwatch) {
    CStopwatchHFT* stopwatch = new CStopwatch;
    hr = stopwatch->QueryInterface(riid, ppv);
  } else {
    hr = CLASS_E_CLASSNOTAVAILABLE;
  }
  return hr;
}
```

# Przykład – Stopwatch serwer
## Metody `AddRef()`, `Release()`

```
class CStopwatchHFT : public IStopwatch
{ ...
private:
 // Reference counting
 long m_nReferenceCount;
  ...
};

CStopwatchHFT::CStopwatchHFT ()
{

  // Initialize the member variables
  m_nStartTime.QuadPart = 0;
  m_nReferenceCount = 0;

  // Save the frequency of the performance counters
  QueryPerformanceFrequency ((LARGE_INTEGER*) &m_nFrequency);

}
```

# Przykład – Stopwatch serwer
## Metody `AddRef()`, `Release()`

```
unsigned long __stdcall CStopwatchHFT::AddRef()
{
  return InterlockedIncrement( &m_nReferenceCount );
}

unsigned long __stdcall CStopwatchHFT::Release()
{
  if (InterlockedDecrement( &m_nReferenceCount ) == 0) {
    delete this;
    return 0;
  }
  return m_nReferenceCount;
}
```

# Przykład – Stopwatch serwer
## Metody `AddRef()`, `Release()`

Zasady, których należy przestrzegać w związku z licznikiem odwołań do komponentu:

1. Gdy do zmiennej przypisujemy wskaźnik do interfejsu, `AddRef()` powinien zostać wywołany na rzecz tego interfejsu
2. `Release()` powinien zostać wywołany, gdy niezerowy wskaźnik do interfejsu wychodzi poza zasięg zmiennej, lub gdy zostaje przypisany nowy wskaźnik
3. Pominięcie niektórych wywołań `AddRef()` i `Release()` jest możliwe, gdy mamy dodatkowe informacje o czasie życia komponentu

# Przykład – Stopwatch klient

```
// Instantiates a stopwatch object and returns it by reference
HRESULT CreateInstance(REFCLSID rclsid, REFIID riid, void** ppv)
{
  HRESULT hr = E_FAIL;
  HINSTANCE hinstDll;
  DLLGETCLASSOBJECT DllGetClassObject;

  hinstDll = LoadLibrary(TIMERSDLL);
  if (hinstDll == NULL){
    std::cout << "Unable to load \"" TIMERSDLL "\"" << std::endl;
  } else {
    DllGetClassObject = (DLLGETCLASSOBJECT) GetProcAddress(
        hinstDll, "DllGetClassObject");
    if (DllGetClassObject != NULL)
      hr = DllGetClassObject(rclsid, riid, ppv);
  }
  return hr;
}
```

# Przykład – Stopwatch klient

```
int main(int argc, char* argv[])
{
  float nElapsedTime;
  IUnknown* pUnknown  = NULL;
  CStopwatch* pStopwatch  = NULL;

  hr=CreateInstance(CLSID_Stopwatch, IID_IUnknown, (void**) &pUnknown);

  if (!SUCCEEDED(hr)){
    std::cout << "ERROR: Unable to create Stopwatch!!";
  } else {
    hr=pUnknown->QueryInterface(IID_IStopwatch, (void**) &pStopwatch);
    if (!SUCCEEDED(hr)){
       std::cout << "ERROR: Unable to retrive IStopwatch!!";
    } else {
       pStopwatch->Start();
       pStopwatch->ElapsedTime(&nElapsedTime);
       std::cout << "The overhead time is " << nElapsedTime << std::endl;
       pStopwatch->Release();
       pStopwatch = NULL;
    }
    pUnknown->Release();
  }
  return 0;
}
```

# Przykład – Stopwatch serwer
## Komponent `IClassFactory`

```
class CStopwatchClassFactory : public IClassFactory
{
public:
  CStopwatchClassFactory();
  virtual ~CStopwatchClassFactory();

private:
  // Reference counting
  long m_nCFReferenceCount;

public:
  // IUnknown methods
  HRESULT __stdcall QueryInterface(REFIID riid, void **ppvObject);
  unsigned long __stdcall AddRef();
  unsigned long __stdcall Release();

  // IClassFactory
  HRESULT __stdcall CreateInstance(
    IUnknown *pUnkOuter,
    REFIID riid,
    void **ppvObject);
  HRESULT __stdcall LockServer(BOOL fLock);
};
```

# Przykład – Stopwatch serwer
## Komponent `IClassFactory`

```
extern long g_nServerLockCount;

CStopwatchClassFactory::CStopwatchClassFactory()
{
 m_nCFReferenceCount = 0;
}

CStopwatchClassFactory::~CStopwatchClassFactory()
{
}

HRESULT __stdcall CStopwatchClassFactory::QueryInterface(
  REFIID riid, void **ppvObject)
{
  HRESULT hr = S_OK;

  if (riid == IID_IUnknown)
    *ppvObject=static_cast<IUnknown*>(static_cast<IClassFactory*>(this));
  else if (riid == IID_IClassFactory)
    *ppvObject = static_cast<IClassFactory*>(this);
  else {
    ppvObject = NULL;
    hr = E_NOINTERFACE;
  }
  if (SUCCEEDED(hr))
    (static_cast<IUnknown*>(*ppvObject))->AddRef();
  return hr;
}
```

# Przykład – Stopwatch serwer
## Komponent `IClassFactory`

```
HRESULT __stdcall CStopwatchClassFactory::CreateInstance(
  IUnknown *pUnkOuter, REFIID riid, void **ppvObject )
{
  HRESULT hr;
  CStopwatchHFT* pStopwatch = new CStopwatchHFT;

  if (pUnkOuter != NULL)
    return CLASS_E_NOAGGREGATION;

  hr = pStopwatch->QueryInterface(riid, ppvObject);
  if (FAILED(hr))
    delete pStopwatch;

  return hr;
}

HRESULT __stdcall CStopwatchClassFactory::LockServer( BOOL fLock )
{
  if (fLock)
    InterlockedIncrement( &g_nServerLockCount );
  else
    InterlockedDecrement( &g_nServerLockCount );
  return S_OK;
}
```

# Przykład – Stopwatch serwer
## Komponent `IClassFactory`

```
unsigned long __stdcall CStopwatchClassFactory::AddRef()
{
  if(InterlockedIncrement( &m_nCFReferenceCount ) == 1)
    InterlockedIncrement( &g_nServerLockCount );
  return m_nReferenceCount;
}

unsigned long __stdcall CStopwatchClassFactory::Release()
{
  if (InterlockedDecrement( &m_nCFReferenceCount ) == 0)
  {
    delete this;
    InterlockedDecrement( &g_nServerLockCount );
    return 0;
  }
  return m_nReferenceCount;
}
```

# Przykład – Stopwatch serwer
## Zarządzanie licznikiem odwołań modułu

```
unsigned long __stdcall CStopwatchHFT::AddRef()
{
  if(InterlockedIncrement( &m_nReferenceCount ) == 1)
    InterlockedIncrement( &g_nServerLockCount );
  return m_nReferenceCount;
}

unsigned long __stdcall CStopwatchHFT::Release()
{
  if (InterlockedDecrement( &m_nReferenceCount ) == 0)
  {
    delete this;
    InterlockedDecrement( &g_nServerLockCount );
    return 0;
  }
  return m_nReferenceCount;
}
```

# Przykład – Stopwatch serwer
## Czas życia modułu

Każdy moduł zawierający komponenty COM ma zaimplementowaną funkcję `DllCanUnloadNow()`. Informuje ona system czy dany moduł nie posiada już żadnych zaalokowanych komponentów i czy może być on usunięty.

```
HRESULT __stdcall DllCanUnloadNow(void)
{
    return (g_nServerLockCount == 0 ) ? S_OK : S_FALSE;
}
```

# Przykład – Stopwatch serwer
## Zarządzanie komponentem

```
extern "C" HRESULT __stdcall DllGetClassObject(
  REFCLSID rclsid,
  REFIID riid,
  LPVOID* ppv)
{
 HRESULT hr;

 if (rclsid == CLSID_Stopwatch) {
   CStopwatchClassFactory* pStopwatchClassFactory =
      new CStopwatchClassFactory;
   hr = pStopwatchClassFactory->QueryInterface(riid, ppv);
   if(FAILED(hr))
      delete pStopwatchClassFactory;
  } else {
    hr = CLASS_E_CLASSNOTAVAILABLE;
  }
  return hr;
}
```

# Przykład – Stopwatch serwer

```
; Timers.def : Declares the module parameters.

LIBRARY      "Timers.DLL"

EXPORTS
    DllGetClassObject   @1 PRIVATE
    DllCanUnloadNow     @2 PRIVATE
```

# Przykład – Stopwatch klient

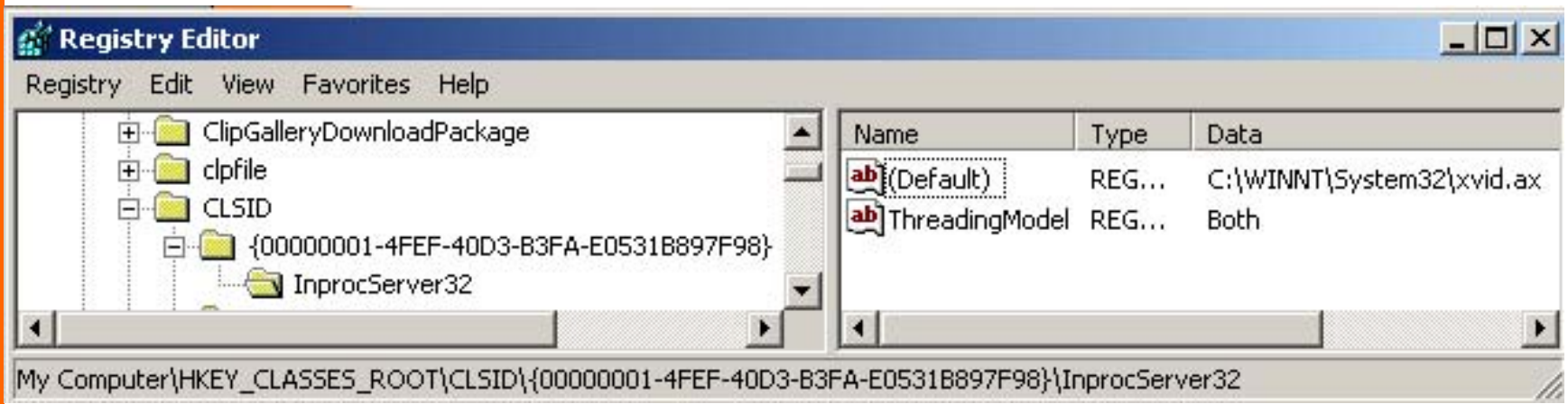
```
HRESULT CreateInstance(REFCLSID rclsid, REFIID riid, void** ppv)
{
  HRESULT hr = E_FAIL;
  HINSTANCE hinstDll;
  DLLGETCLASSOBJECT DllGetClassObject;
  IClassFactory* pClassFactory;

  hinstDll = LoadLibrary(TIMERSDLL);
  if (hinstDll == NULL){
    std::cout << "Unable to load \"" TIMERSDLL "\"" << std::endl;
  } else {
    DllGetClassObject = (DLLGETCLASSOBJECT) GetProcAddress(
        hinstDll, "DllGetClassObject");
    if (DllGetClassObject != NULL){
      hr=DllGetClassObject(rclsid, IID_IClassFactory, (void**)&pClassFactory);
      if (SUCCEEDED(hr)){
        hr=pClassFactory->CreateInstance(NULL, riid, ppv);
        pClassFactory->Release();
      }
      pClassFactory=NULL;
    }
  }
  return hr;
}
```

# Funkcje systemowe zarządzające obiektami COM

Do zarządzania obiektami COM istnieją specjalne funkcje systemowe odpowiedzialne za ładowanie bibliotek, inicjalizację i deinicjalizację obiektów.

Korzystają one z informacji zawartych w rejestrze systemowym, który wiąże identyfikator obiektu z właściwym plikiem i sposobem jego uruchomienia.



# CoInitialize(), CoInitializeEx(), CoUninitialize()

CoInitialize(), CoInitializeEx():

- Inicjalizuje bibliotekę COM w bieżącym wątku.
- Specyfikuje model współpracy dla biblioteki (wielowątkowy, jednowątkowy,...)

CoUninitialize():

- Deaktyewuje bibliotekę COM w bieżącym wątku.

# CoCreateInstance()

Na podstawie ustawień w rejestrze ładuje moduł COM, inicjalizuje wskazany obiekt i zwraca jego wskaźnik.

```
HRESULT CoCreateInstance(
 REFCLSID rclsid, //Class identifier (CLSID) of the object
 LPUNKNOWN pUnkOuter, //Pointer to controlling IUnknown
 DWORD dwClsContext, //Context for running executable code
 REFIID riid, //Reference to the identifier of the interface
 LPVOID * ppv //Address of output variable that receives
              // the interface pointer requested in riid
);

rclsid          - np. CLSID_Stopwatch
riid            - najczęściej IDD_IUnknown
dwClsContext    - sposób uruchomienia serwera
```

# Przykład – Stopwatch klient
## Funkcje systemowe COM

```
int main(int argc, char* argv[])
{
  HRESULT hr;
  IStopwatch* pStopwatch = NULL;

  CoInitialize(NULL);

  hr = CoCreateInstance(
    CLSID_Stopwatch,
    NULL,
    CLSCTX_ALL,
    IID_IStopwatch,
    (void**) &pStopwatch);
```

```
  if (!SUCCEEDED(hr)){
     std::cout << "ERROR: Unable to
              create Stopwatch!!\n";
  } else {
   pStopwatch->Start();
     ...
   pStopwatch->ElapsedTime(&nElapsedTime);
   std::cout << "The overhead time is "
             << nElapsedTime << std::endl;
   pStopwatch->Release();
   pStopwatch = NULL;
  }
   CoUninitialize();
   return 0;
}
```