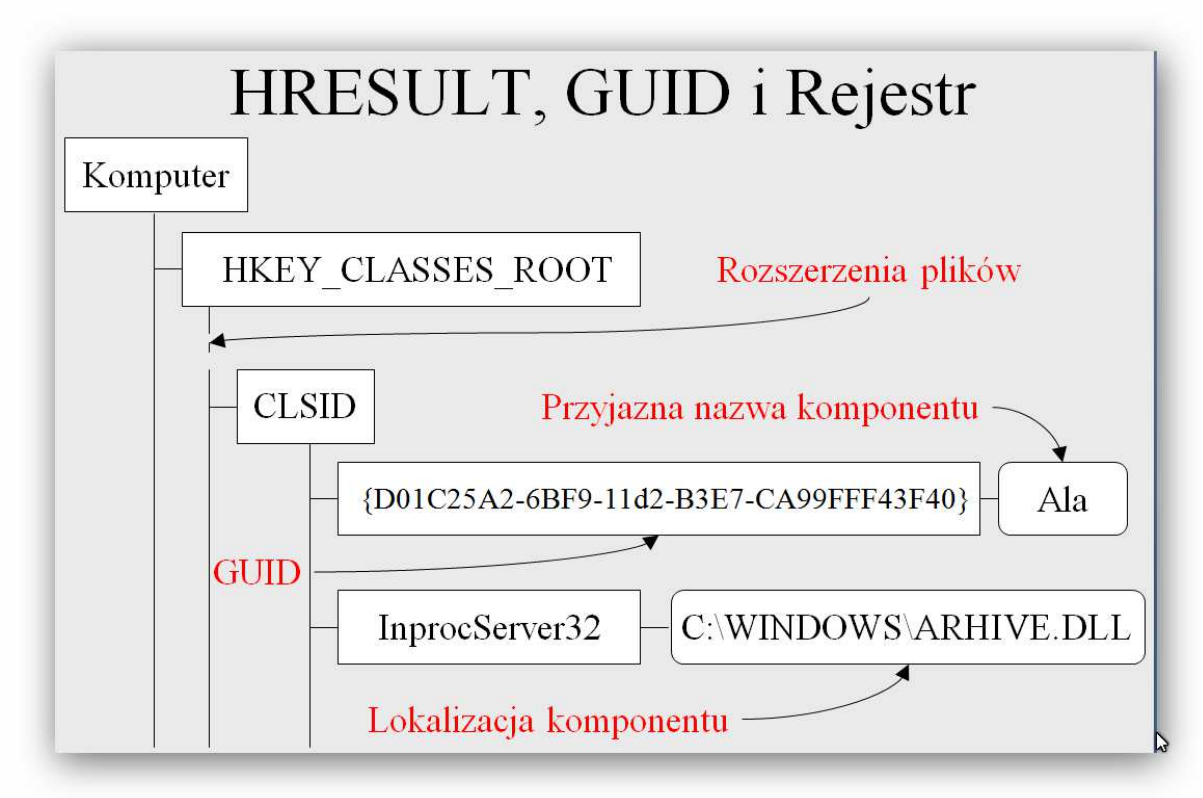
Tworzenie komponentów ActiveX

Kontrolki można tworzyć w różnych językach i technologiach np. MFC, ATL, C++, C#, Borland Delphi czy Visual Basic. Efektem końcowym jest kontrolka niezależna od języka programowania typu OLE ostatecznie zaklasyfikowanymi do technologii COM dziś rozszerzona do COM+ i DCOM (rozproszona wywoływana na różnych komputerach) którą można używać w dowolnym programie współpracującym z kontrolkami tego typu.

Rejestrowanie kontrolek ActiveX
Każda kontrolka ma przypisany unikalny identyfikator (**CLSID**) class ID.
Jest to liczba 128 bitowa zapisana heksalnie np.
{03437DCB-1719-4119-B253-02B4939A2CF4}

Zarejestrowane w systemie kontrolki ActiveX są wyszczególnione w gałęzi
HKEY_LOCAL_MACHINE\SOFTWARE\CLASSES\CLSID



Do zarządzania rejestrowaniem kontrolek w systemie wykorzystujemy odpowiednie narzędzia systemu Windows aby zarejestrować serwer kontrolki activex.

HRESULT, GUID i Rejestr

- STDAPI DllRegisterServer()
- STDAPI DllUnregisterServer()
- RegSvr32.exe

Jeśli kontrolka nie posiada programu instalacyjnego to należy ją zarejestrować programem **regsvr32.exe**,

np. **regsvr32.exe MyCtrl.ocx**.

Rejestracja kontrolki polega na zapisaniu w rejestrze Windows większości informacji dotyczących obiektu (np. lokalizacja kontrolki na dysku). Informacje te są wykorzystywane za każdym razem, kiedy aplikacja używa obiektu ActiveX.

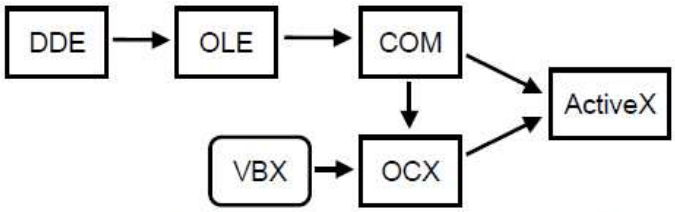
Program **regsvr32.exe** ponadto wywołuje funkcję **DLLSelfRegister**.

Parametry programu **regsvr32.exe**:

```
Regsvr32 [/u] [/s] [/n] [/i[:cmdline]] dllname
 /s - Silent; display no message boxes
 /u - Unregister server
 /i - Call DllInstall passing it an optional [cmdline];
      when used with /u calls dll uninstall
 /n - do not call DllRegisterServer; this option must
      be used with /i
```

3

Technologia ActiveX



OLE (Object Linking and Embedding) – osadzanie i łączenie obiektów. Technologia pozwalająca na umieszczanie jednego obiektu wewnątrz drugiego i prezentowania go w takiej formie, jak czyni to aplikacja macierzysta.

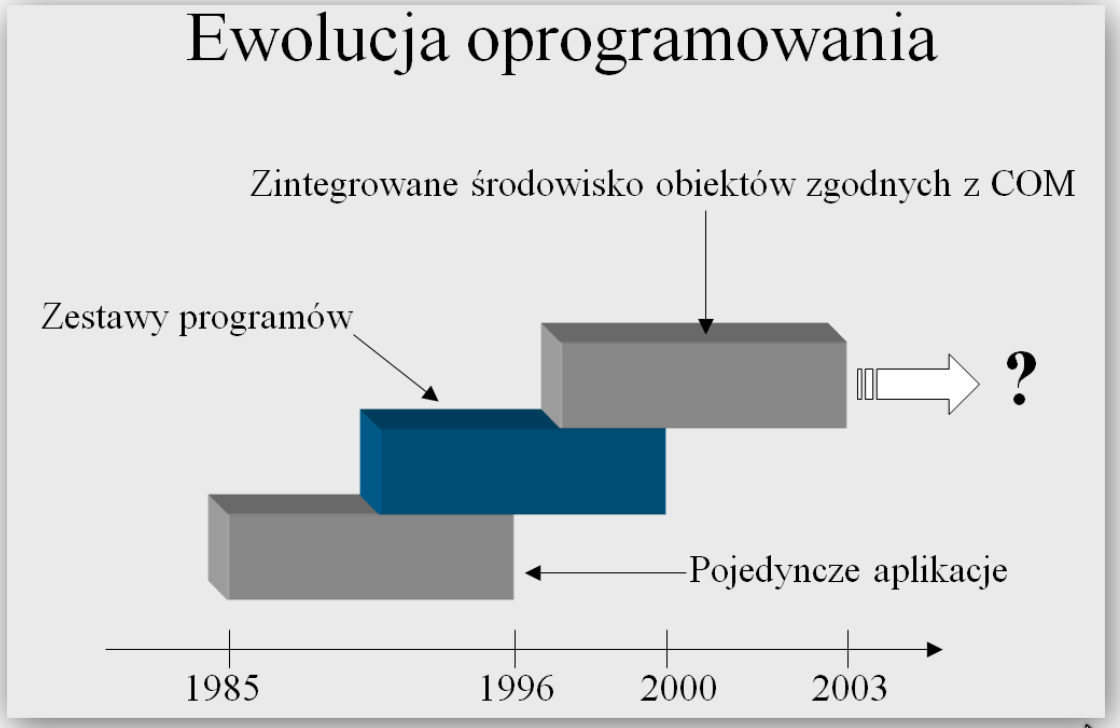
COM (Component Object Model) – model programowania zorientowanego obiektowo, udostępniającego programiście technologię OLE.

VBX (Visual Basic Controls) – kontrolki wizualne utworzone za pomocą Visual Basic (16-bitowe).

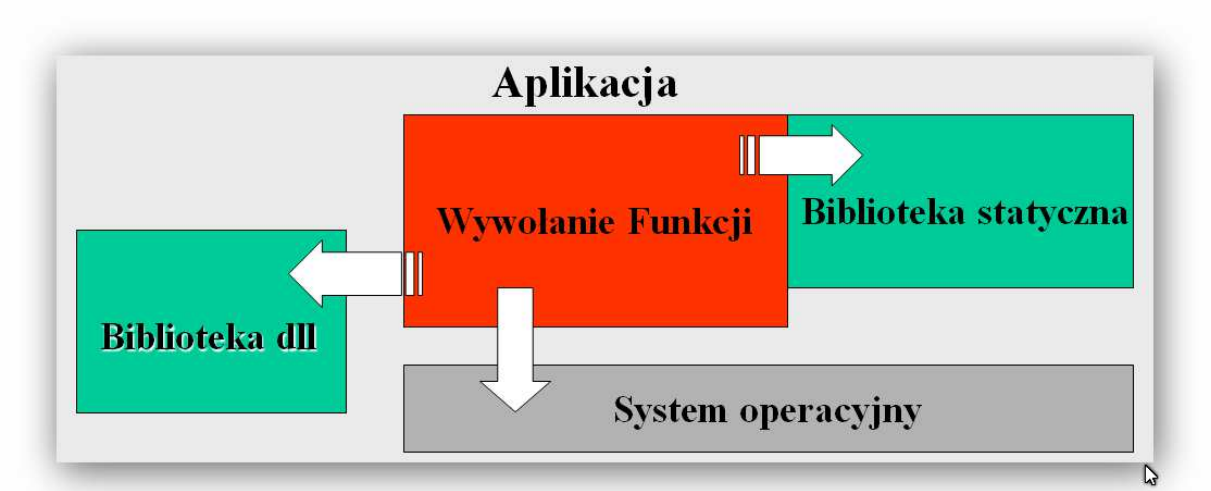
OCX (OLE Control Extension) – kontrolki wizualne przeniesione na platformę 32-bitową, wykorzystujące technologię OLE.

ActiveX – promowana przez Microsoft technologia pozwalająca na stosowanie obiektów OLE wewnątrz innych dokumentów w sposób niezależny od systemu operacyjnego.

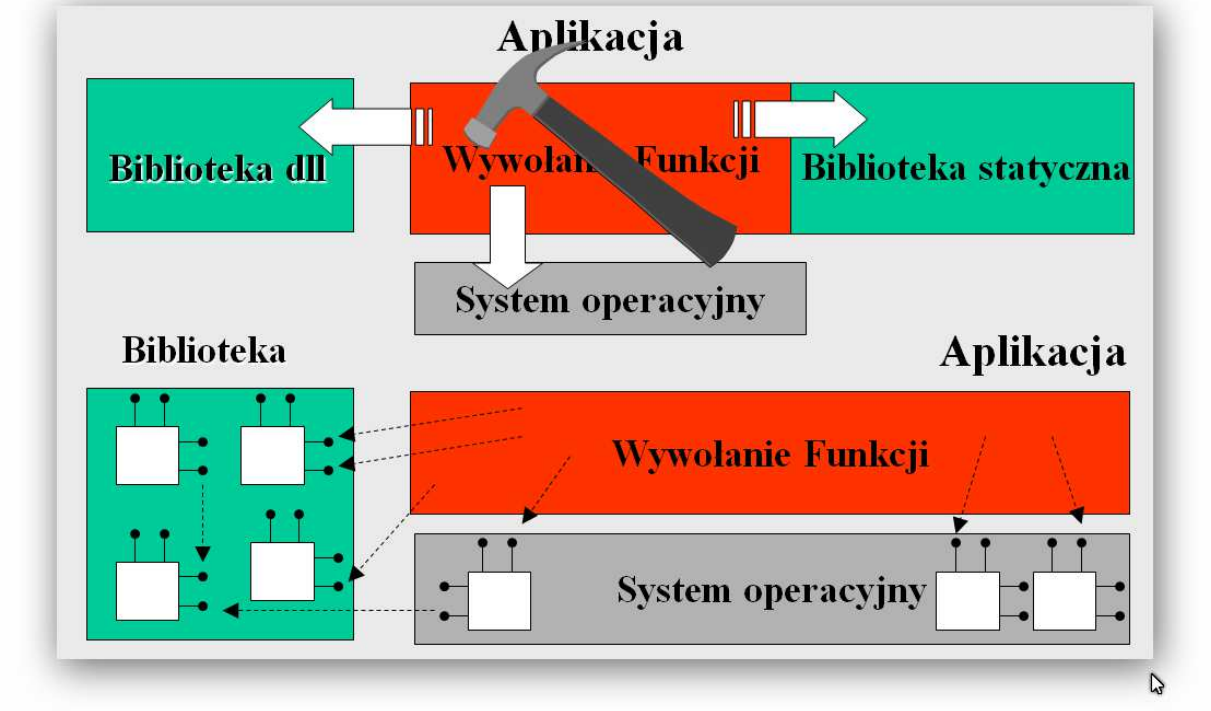
Ewolucja oprogramowania



Klasyczne programowanie DLL bez technologii COM(ActiveX)



- duży rozmiar kodu wynikowego
- każda aplikacja korzystająca z biblioteki posiada ją w sobie
- duże kłopoty przy zmianie kodu biblioteki

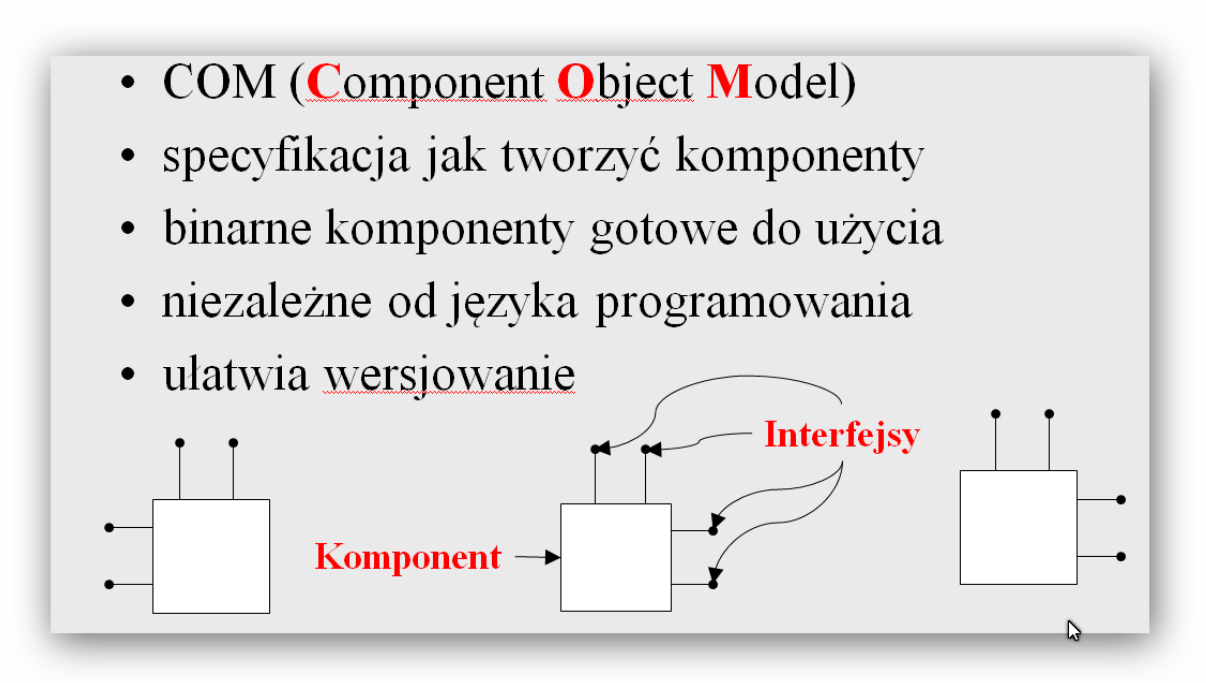


W nowym podejściu COM mamy podziała komponentu na różne części, które fizycznie mogą być trzech miejscach. W aplikacji, w wspólnej dla wszystkich bibliotece i w systemie operacyjnym.

Interfejs – zbiór logicznie powiązanych właściwości, metod i zdarzeń obejmujących określoną funkcjonalność (wygląd obiektu, współdziałanie z otaczającą aplikacją, wywoływanie zdarzeń).

Technologia COM określa, w jaki sposób aplikacje i pojedyncze komponenty mogą współpracować poprzez użycie interfejsów.

Technologia ActiveX definiuje z kolei płaszczyznę zbudowaną na szczycie modelu COM, określając jakiego typu interfejsy powinny wspierać różne obiekty, a także jak różne obiekty powinny ze sobą współpracować niezależnie od rodzaju systemu operacyjnego.

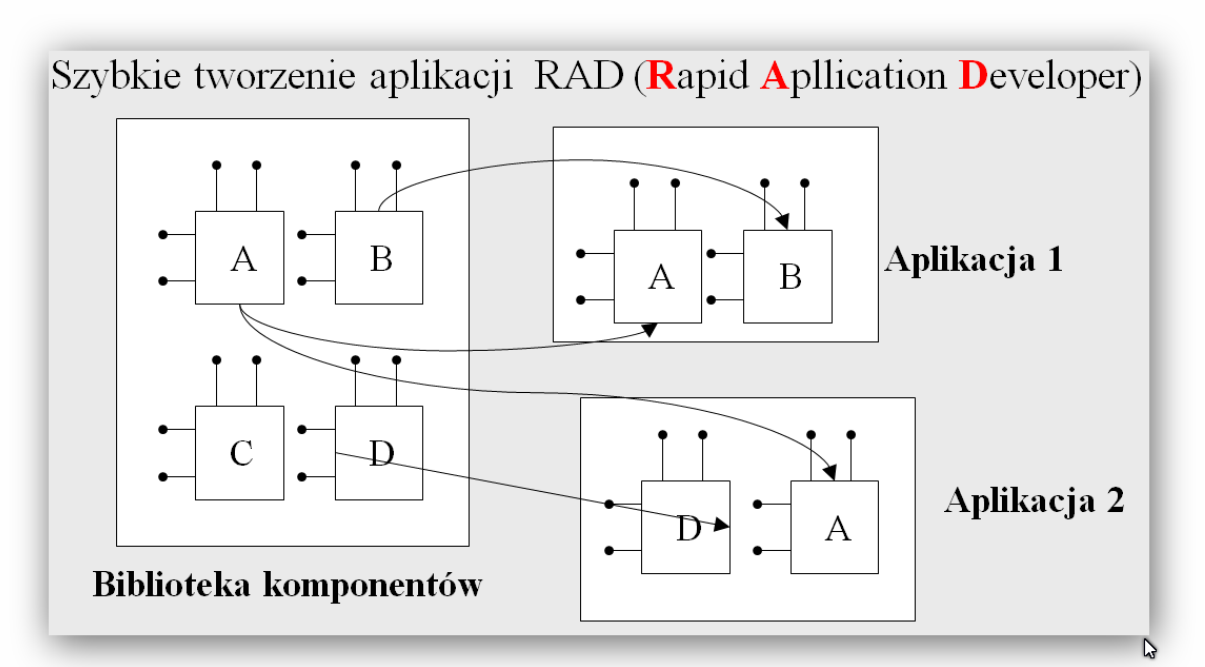


COM (Component Object Model) jest opracowaną przez Microsoft technologią umożliwiającą efektywną komunikację między aplikacjami

Podczas programowanie często używa się zamiast słowa komponent **coclass** (skrót od component object class) określający że dana klasa jest komponentem a nie zwykła klasą.

Component Object Model definiuje:

binarny standard wywoływania funkcji między komponentami, struktury interfejsów udostępnianych przez poszczególne obiekty, mechanizmy jednoznacznej identyfikacji komponentów i ich interfejsów.

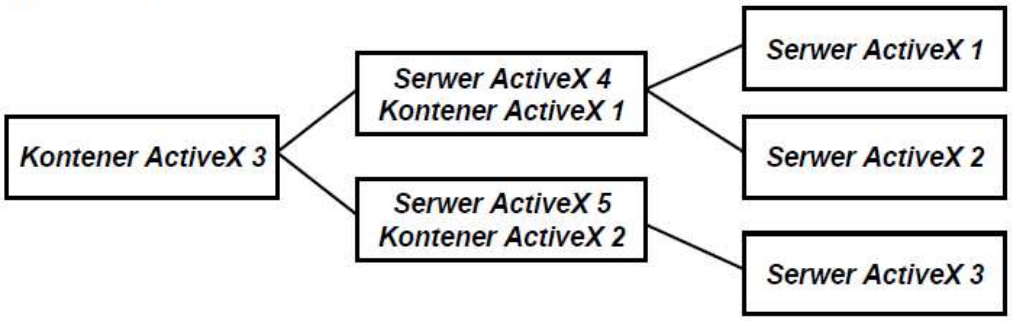


Jedną z kluczowych technologii występujących w obiektach ActiveX jest ***automatyzacja***.

Automatyzacja pozwala aplikacji, będącej **serwerem ActiveX** (ang. ActiveX server), na osadzanie się wewnątrz innej aplikacji stanowiącej **kontener ActiveX** (ang. ActiveX container), aktywowanie się i kontrolowanie należącej do niej części interfejsu użytkownika lub dokumentu. Kiedy użytkownik wykona niezbędne zmiany i przejdzie do innej części programu nie kontrolowanej przez aplikację osadzoną, ta samoczynnie kończy swoją pracę.

Przykłady:
- Edycja równania MS Equation lub arkusza MS Excel w dokumencie MS Word.
- Przeglądanie plików pdf w przeglądarce internetowej.
- Wykorzystanie komponentów graficznych CVI w aplikacjach Visual C++

Każdy obiekt ActiveX, który może mieć osadzone w sobie inne obiekty ActiveX, jest kontenerem ActiveX. Zatem jedna aplikacja może być równocześnie kontenerem oraz serwerem ActiveX.



Kontrolka ActiveX jest szczególnym przykładem serwera ActiveX, który nie jest aplikacją i nie jest w stanie funkcjonować samodzielnie. Może być jednak osadzony w innej aplikacji, czyniąc ją automatycznie kontenerem ActiveX.

Trzeba pamiętać o tym że osadzając kontrolkę czy kreując ją ma ona pewną stałą grupę właściwości, które muszą być określone w każdej kontrolce. Zapewniają wyminę podstawowych informacji o wyglądzie kontrolki. I dopiero do tych standardowych programista może dodawać swoje.

Każdy obiekt ActiveX posiada **właściwości**, które określają jego wygląd oraz sposób interakcji z otoczeniem (kontenerem).

Podstawowe grupy właściwości:

Ambient – zawierają informacje dotyczące parametrów kontenera związanych z jego wyglądem. Są one udostępniane kontrolce ActiveX w celu umożliwienia dopasowania jej wyglądu do wyglądu kontenera. Kontrolka nie może zmieniać wartości tych właściwości. Przykładowe właściwości należące do tej grupy to ***BackColor***, ***ForeColor***, ***Font***, ***ScaleUnits*** (nazwa jednostki odległości używanej przez kontener).

Control – właściwości ustawiane przez kontrolkę podczas jej osadzania w kontenerze. Zawierają podstawowe parametry związane z typem pracy i wyglądem, np. ***Enabled***, ***Hwnd*** (uchwyt do okna kontrolki), ***Caption***, ***Appearnace*** (FALSE dla kontrolki 2-D, TRUE dla kontrolki 3-D).

Kontrolka utworzona w oparciu o bibliotekę MFC zawiera 9 podstawowych właściwości typu Control. Są to tzw. ***Stock Properties***. Dodając funkcjonalność kontrolki ActiveX programista określa natomiast nowe właściwości, czyli tzw. ***Custom Properties***.

Extended - dodatkowe właściwości ustawiane przez kontener np. ***Visible***, ***Parent***.

Programowanie interfejsów

COM – Interfejsy- Interfejs jest zestawem prototypów funkcji składowych komponentu COM (metod).

Nazwy interfejsów przyjęło się poprzedzać przedrostkiem I, np.: ILookup

Interfejs można zdefiniować na bazie innego interfejsu, czyli zastosować dziedziczenie (ale wyłącznie jednobazowe)

Interfejs nie posiada własnej implementacji.

Każdy komponent może implementować wiele interfejsów -oferować wiele zestawów usług.

Klienty (aplikacje lub inne komponenty) odwołują się do interfejsów za pośrednictwem wskaźników.

Każdy interfejs posiada własny, unikalny identyfikator

- Globally Unique Identifier (GUID).

Obiekty COM udostępniają swoje funkcje obiektom zewnętrznym za pośrednictwem interfejsów

Interfejs **IUnknown** jest wymagany w każdym obiekcie ActiveX i służy do ustalenia jakie inne interfejsy są przez obiekt wspierane.

Interfejs **IUnknown** udostępnia trzy podstawowe metody:

QueryInterface – zwrócenie tablicy wskaźników do wspieranych interfejsów.
AddRef, Release – Zwiększenie/zmniejszenie licznika odwołań do interfejsu.
Jeżeli licznik odwołań zmniejszy się do zera interfejs jest usuwany z pamięci.
Metody stosowane w celach diagnostycznych.

Ponadto obiekty AvtiveX powinny posiadać zdolność do samorejestrowania się.

Budowane przez programistę interfejsy są podstawie określonych ściśle w technologii OLE interfejsów dziedziczą z nich główne cechy i są różne dla Kontenera i Serwera ActiveX. Są zdefiniowane dla czterech obszarów funkcjonalności.

Rodzaje interfejsów:

Kontener ActiveX

- IOleInPlaceFrame
- IOleInPlaceUIWindow
- IOleInPlaceSite
- IOleClientSaite
- IAdviseSink
- IOleControlSite
- **IDispatch (properties)**
- IPropertyNotifySink
- **IDispatch (events)**

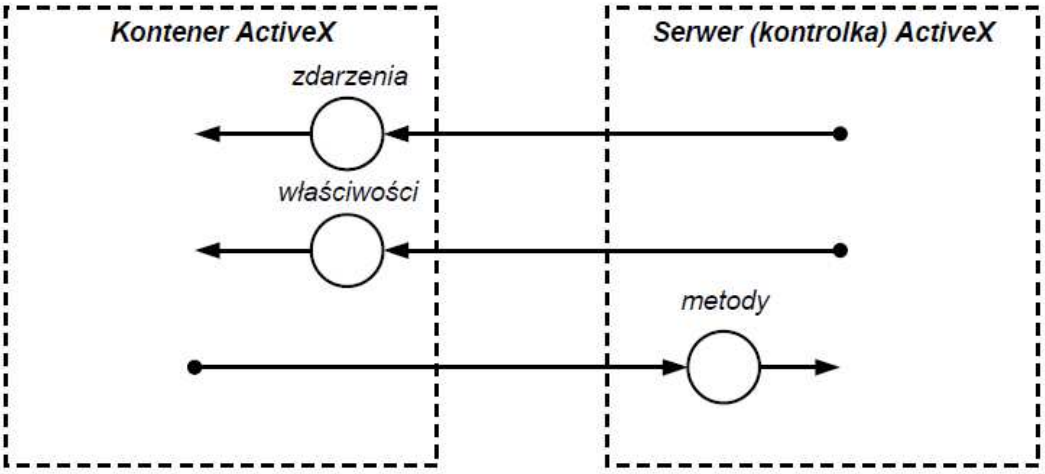
Cztery obszary funkcjonalności:

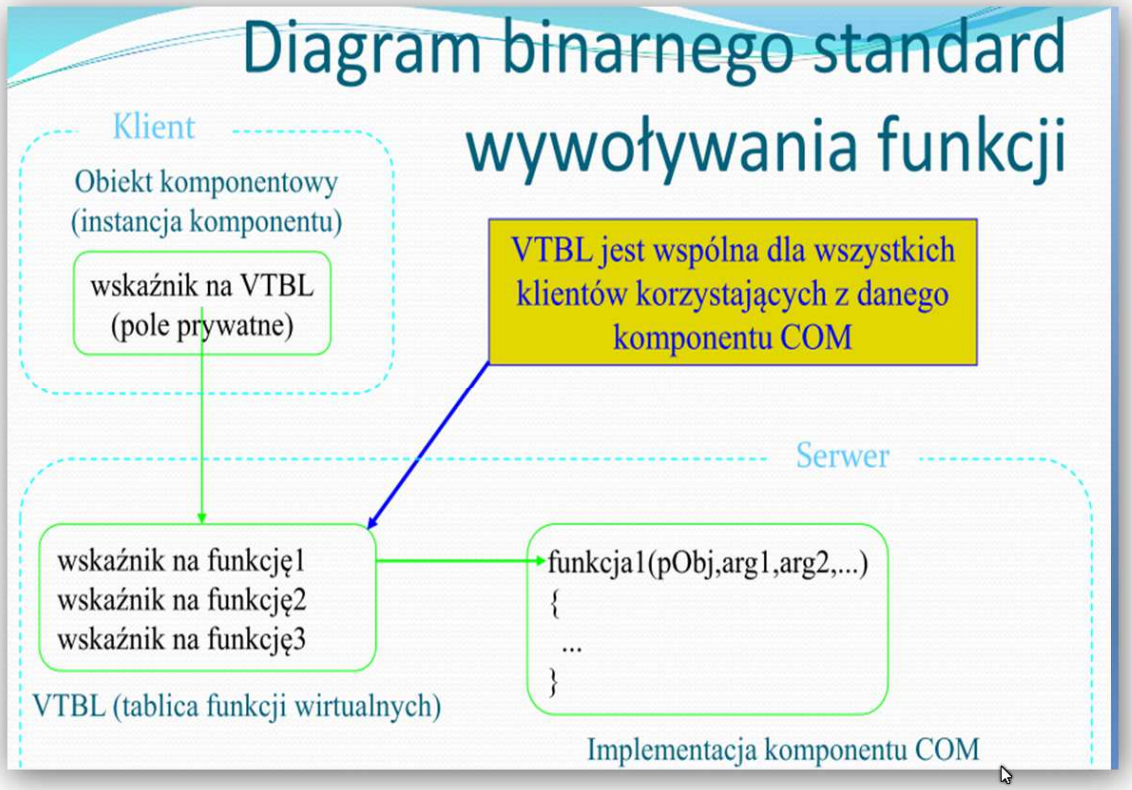
- GUI
- Metody
- Zdarzenia
- Właściwości

Serwer (kontrolka) ActiveX

- IOleInPlaceActiveObject
- IOleInPlaceObject
- IOleObject
- IRunnableObject
- IDataObject
- IViewObject2
- IOleCache2
- IPersistStorage
- IPersistStreamInit
- ISpecifyPropertyPages
- IConnectionPointContainer
- IConnectionPoint
- IProvideClassInfo2
- **IDispatch (methods)**
- IOleControl

Wzajemne oddziaływanie między serwerem a kontenerem ActiveX odbywa się głównie przez trzy interfejsy **IDispatch**. Jeden z tych interfejsów należy do kontrolki, pozostałe do kontenera.





Podstawowym elementem umożliwiającym automatyzację jest interfejs **IDispatch** (*dispinterface*). Został utworzony, aby umożliwić nie obiektowym językom programowania (np. Visual Basic) odwoływanie się do obiektów COM.

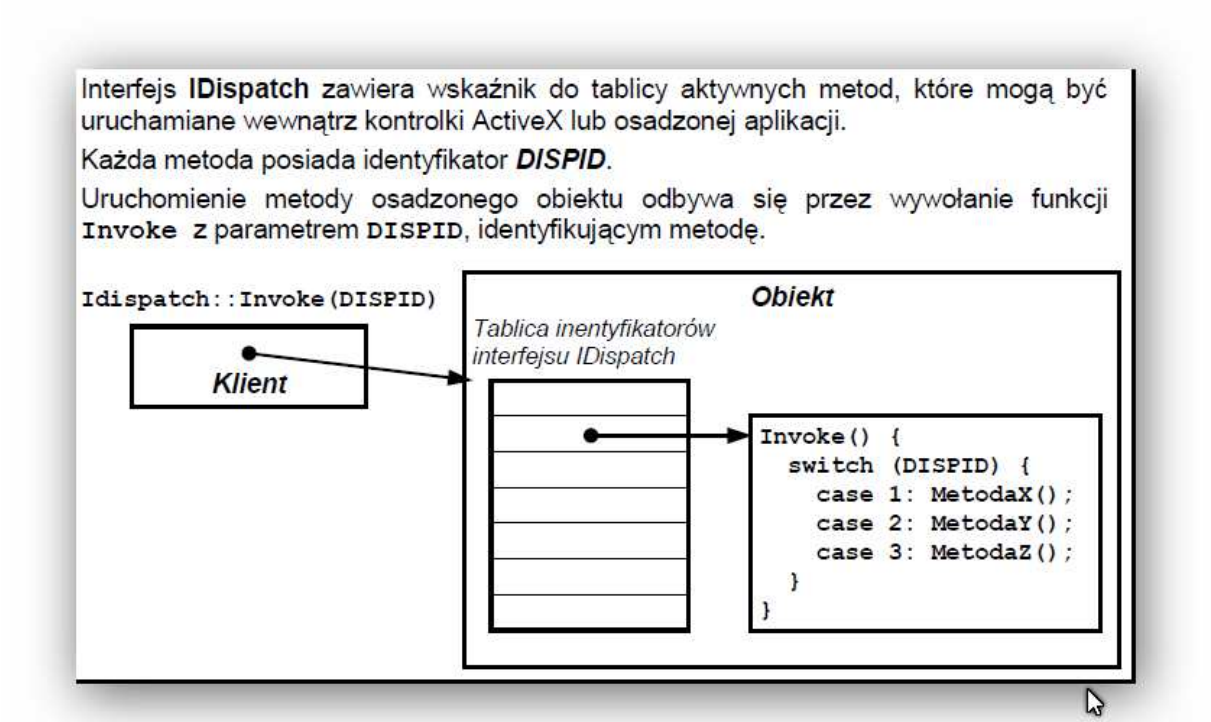
Interfejs **IDispatch** udostępnia cztery podstawowe metody:

1. GetTypeInfoCount – służy do sprawdzania, ile typów zostało zadeklarowanych w bibliotece typów danego obiektu COM;

2. GetTypeInfo – zwraca informacje, z których możemy dowiedzieć się, ile metod oferuje konkretny obiekt, jakie są ich nazwy i parametry ich wywołań; czasami zawiera opisy metod;

3. GetIDsOfNames – zwraca uchwyt (wskaźnik) do metody, której nazwa została przekazana jako parametr wywołania funkcji;

4. Invoke – powoduje wykonanie metody, odczyt lub zmianę wartości właściwości o identyfikatorze pobranym przez funkcję **GetIDsOfNames**.



Funkcja **Invoke** operuje na wskaźnikach i odpowiednimi flagami wywołując funkcje pozwala na komunikację dwustronną.

DISPATCH_METHOD – wywołanie metody
DISPATCH_PROPERTYGET – pobranie wartości właściwości
DISPATCH_PROPERTYPUT – ustawienie wartości właściwości
DISPATCH_PROPERTYPUTREF - ustawienie wartości właściwości poprzez zmienną referencyjną

```
HRESULT Invoke(
        DISPID dispIdMember, REFIID  riid, LCID  lcid,
        WORD  wFlags, DISPPARAMS FAR*  pDispParams,
        VARIANT FAR*  pVarResult, EXCEPINFO FAR*  pExcepInfo,
        unsigned int FAR*  puArgErr);
```

Parametry funkcji `Invoke`:

`DISPID dispIdMember` - identyfikator (metody, właściwości)

`REFIID riid - IID_NULL`

`LCID lcid -` identyfikator narodowościowy, wykorzystywany przez funkcję **GetIDsOfNames**

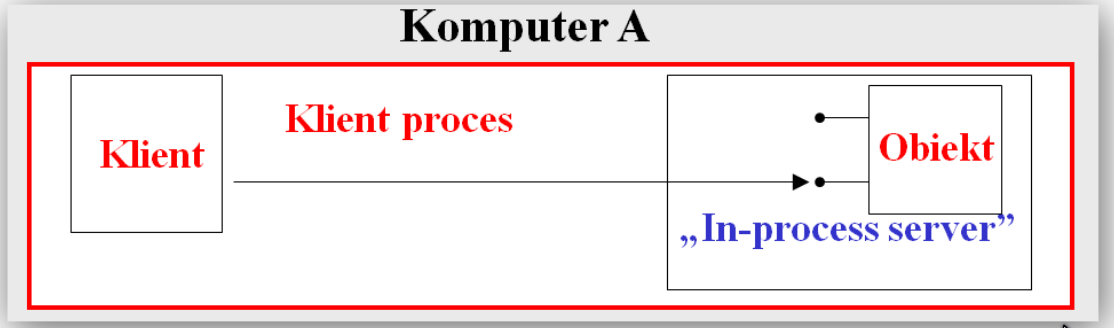
`WORD wFlags - przyczyna wywołania funkcji:`

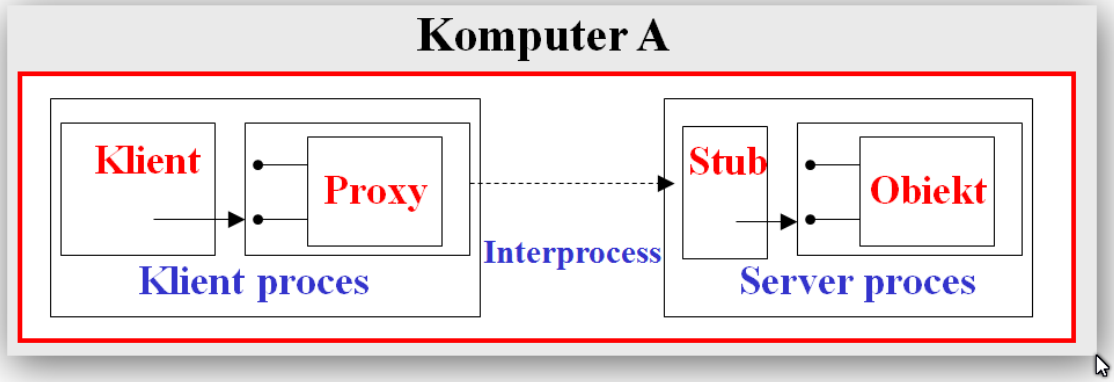
- `DISPATCH_METHOD` – wywołanie metody
- `DISPATCH_PROPERTYGET` – pobranie wartości właściwości
- `DISPATCH_PROPERTYPUT` – ustawienie wartości właściwości
- `DISPATCH_PROPERTYPUTREF` – ustawienie wartości właściwości poprzez zmienną referencyjną

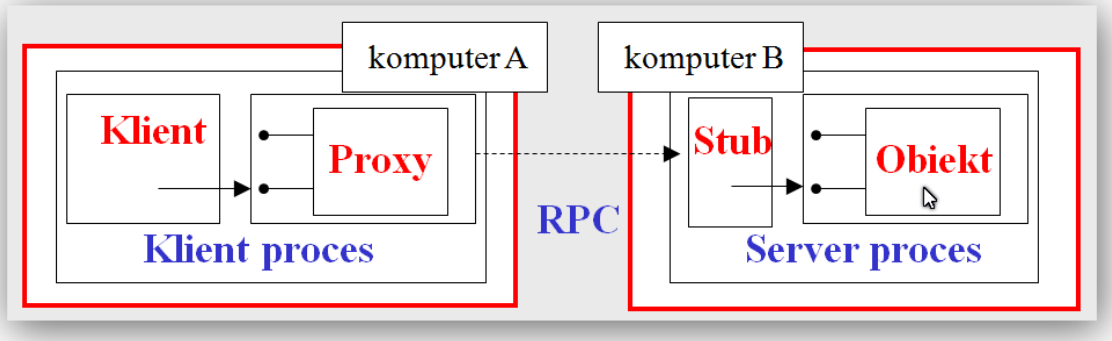
Przykładowe wartość zwracana przez funkcję `Invoke`:

Return value	Meaning
S_OK	OK
DISP_E_BADPARAMCOUNT	Niewłaściwa liczba parametrów
DISP_E_BADVARTYPE	Niewłaściwy typ parametru
DISP_E_EXCEPTION	Wystąpił wyjątek
DISP_E_MEMBERNOTFOUND	Właściwość nie istnieje, lub jest tylko do odczytu
DISP_E_NONAMEDARGS	Implementacja interfejsu **IDispatch** nie zezwala na stosowanie nazw parametrów
DISP_E_PARAMNOTFOUND	Nie znaleziono parametru

Trzy możliwości wywołania kontrolki w zależności od miejsca jej położenia.
W tym samym procesie, w innym procesie na tej samej maszynie, na różnych komputerach DCOM.





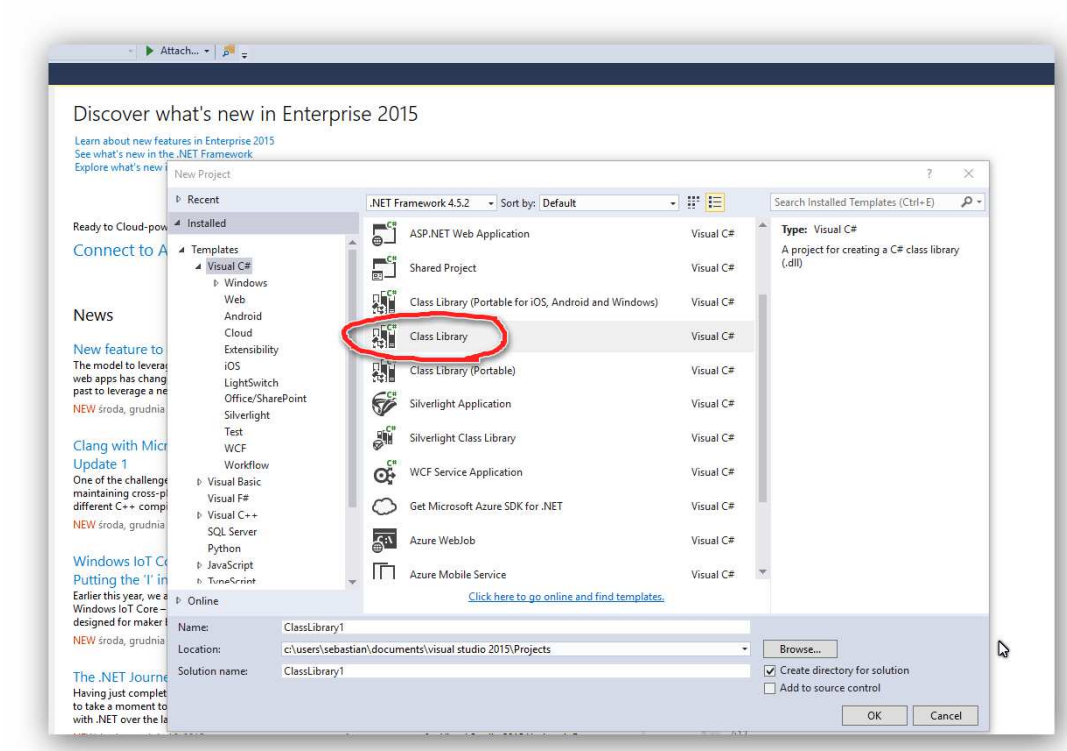


Przykład w C# z MSDN

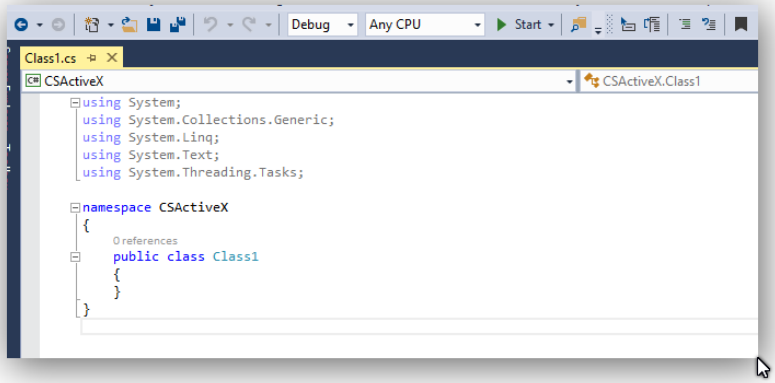
https://code.msdn.microsoft.com/csactivex-b86194f8

W przypadku tworzenia obiektów COM ze względów na ich rejestracje w systemie najlepiej uruchomić visual studio w trybie administratora.

Wybieramy projekt typu Class Library w C#

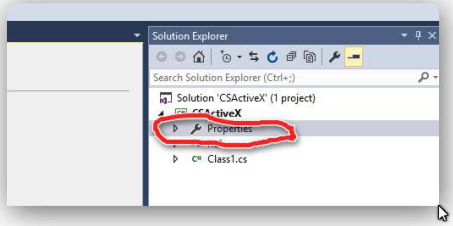


Zmieniamy nazwę projektu np. na CSActiveX.

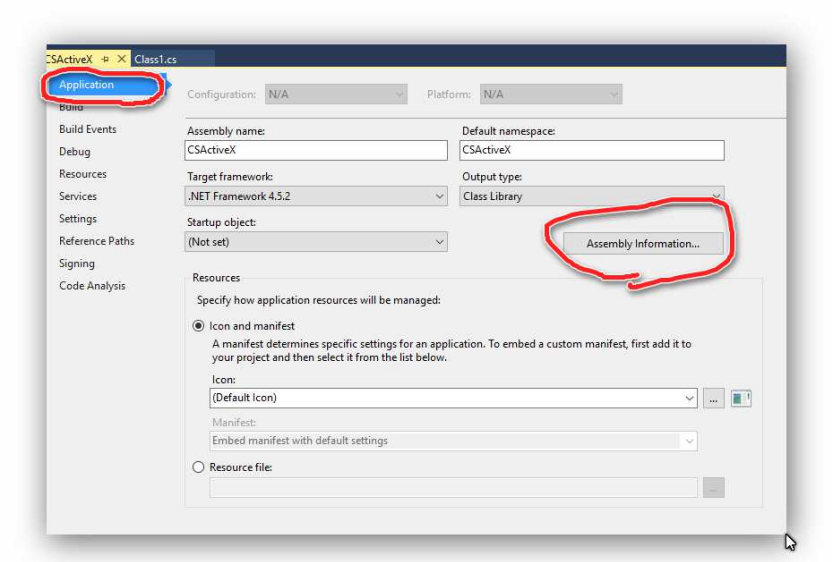


Należy teraz przełączyć na tryb wizualny obiektu COM ustawiając **.NET assembly COM-visible**.

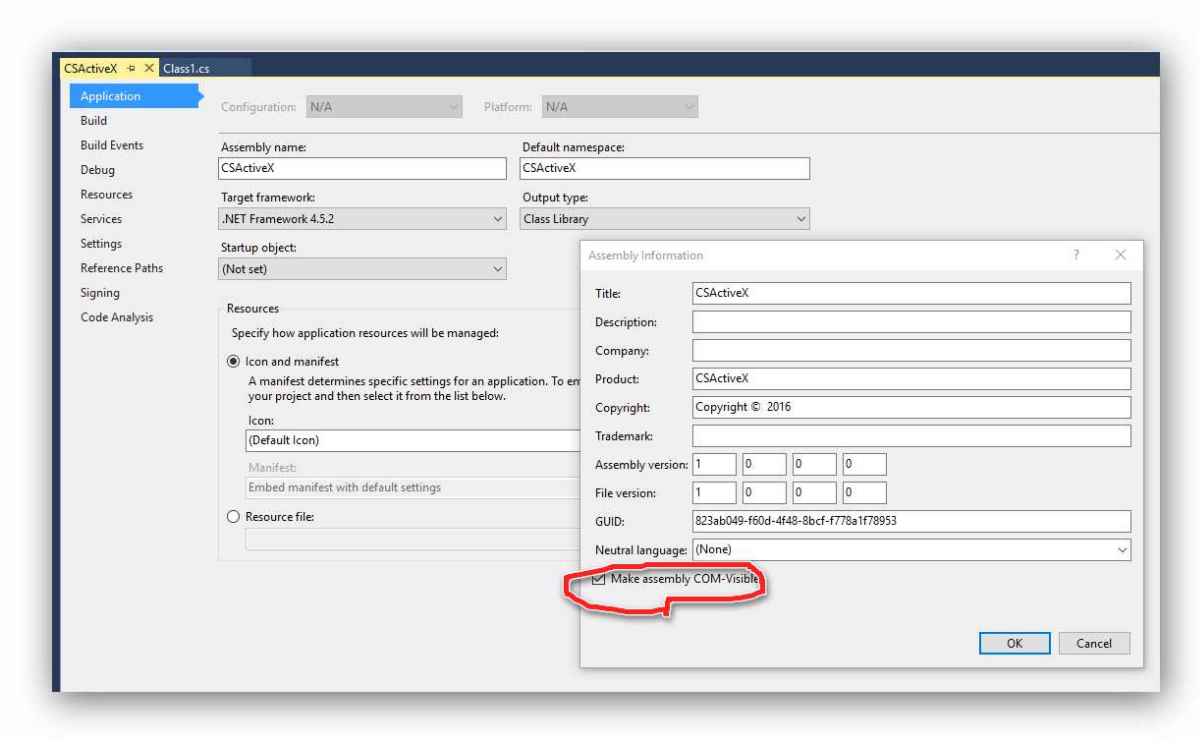
Dostęp do własności naszego projektu mamy w opcji Project -> CSAciveX Property lub z boku w Solution Explorer



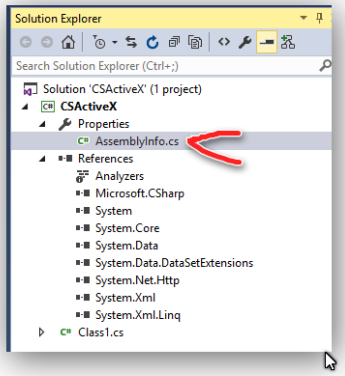
W opcjach tych mamy pewne parametry co nam ułatwią zarządzanie kontrolką.



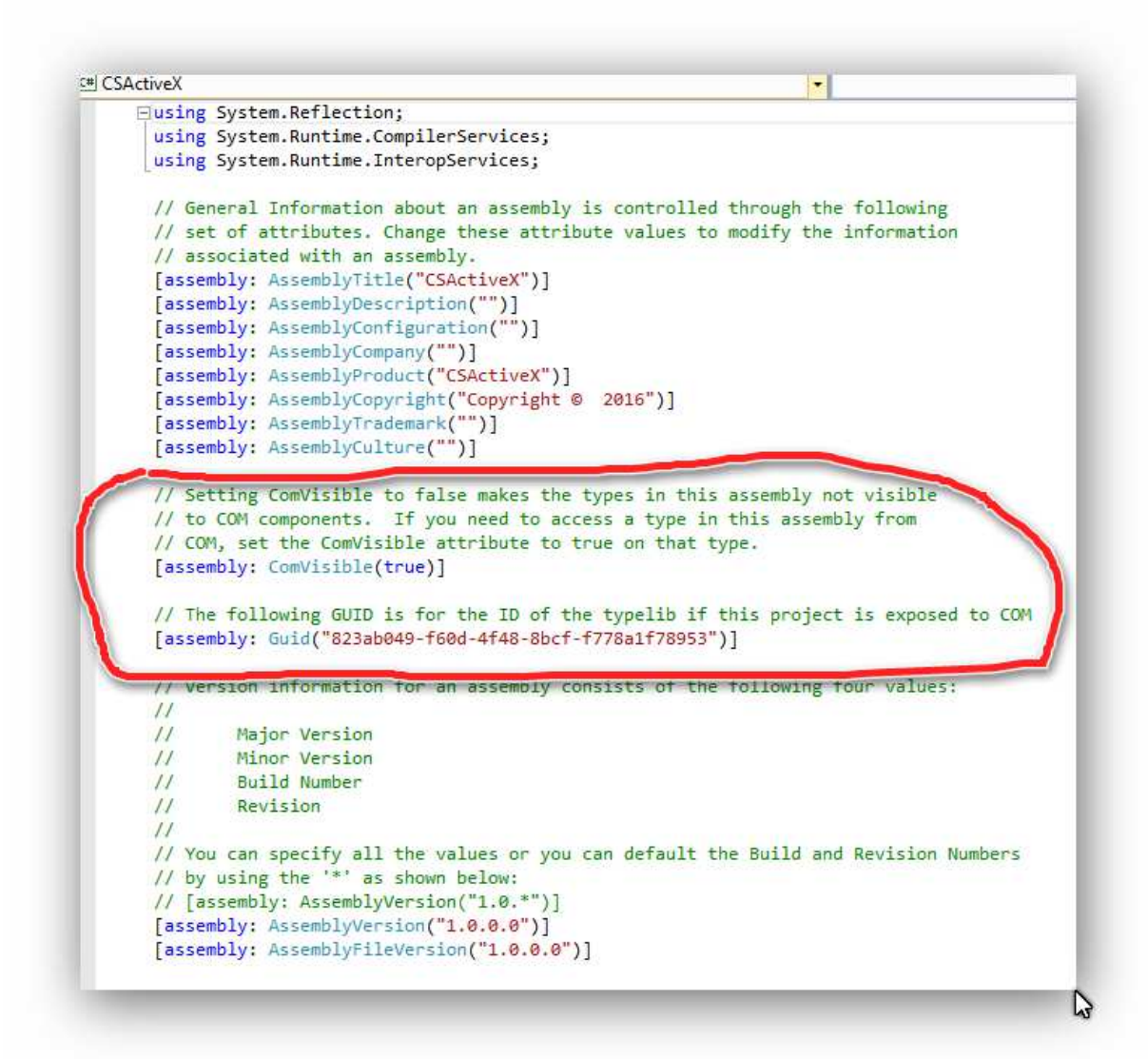
Klikamy na przycisk **Assembly Information** i wybieramy **Make Assembly COM-Visible**



Po włączeniu tej opcji w pliku konfiguracji assemblyInfo.cs



Pojawią się wpisy dotyczące przełączenia w tryb VisibleCOM



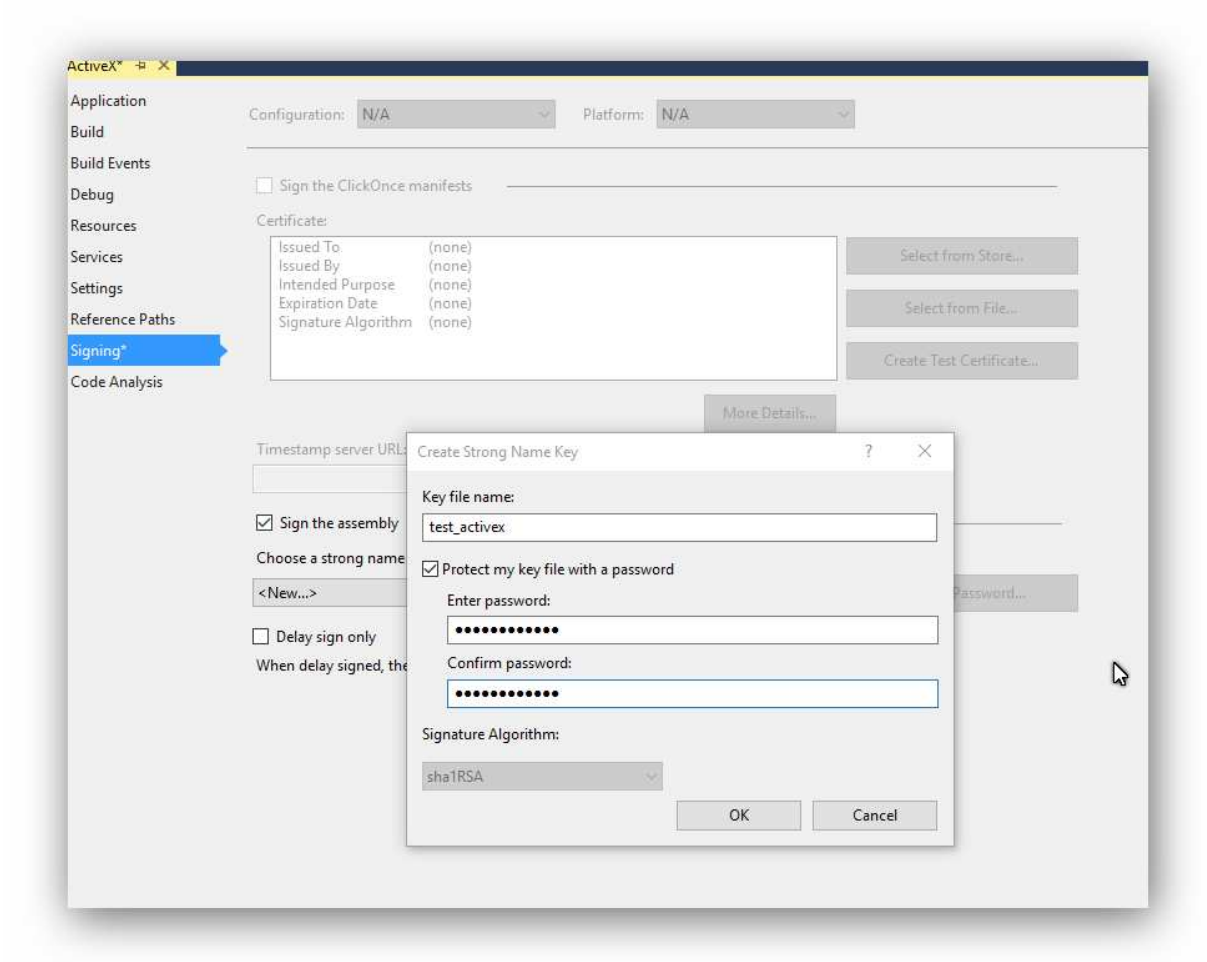
2. Dodanie referencji od obłsugi COM/ACTIVEX **System.Runtime.InteropServices Namespace**

- W pliku projektu AssemblyInfo.cs

using System.Runtime.InteropServices;

Komponenty ActiveX powinny być podpisane

Sign your assembly with a strong name / podpisanie obiektu com
Własności projektu -> Signing
Ustawimy -> Sign the assembly
Ustawiamy -> New (key file name)



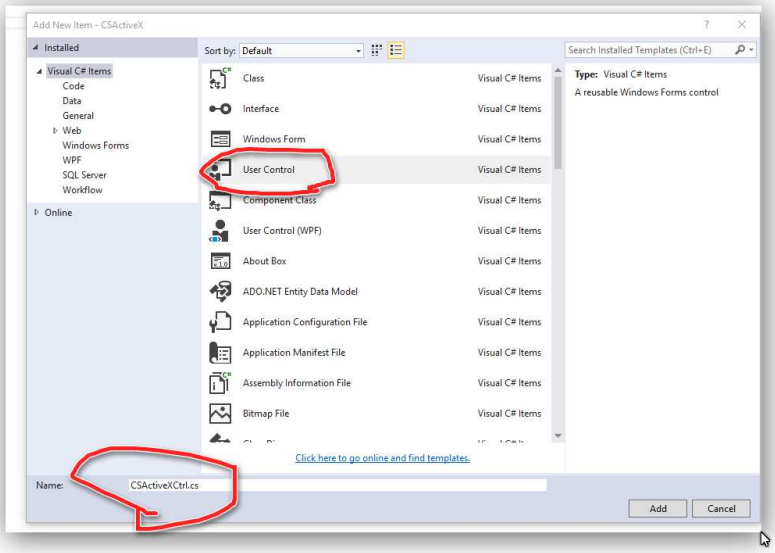
Można skorzystać z MSDN pobrać gotową klasę pomostową ActiveXCtrlHelper.cs co pomaga rejestrować i odrejestrować komponent oraz dostarcza pomost miedzy kontenerem w typach OLE a

obiektami .NET. Jest w nim zbiór gotowych funkcji do połączenia zdarzeń miedzy różnymi platformami OLE COM i .NET.

Aby ją dodać dodajemy klasę do projektu Project -> Add Class

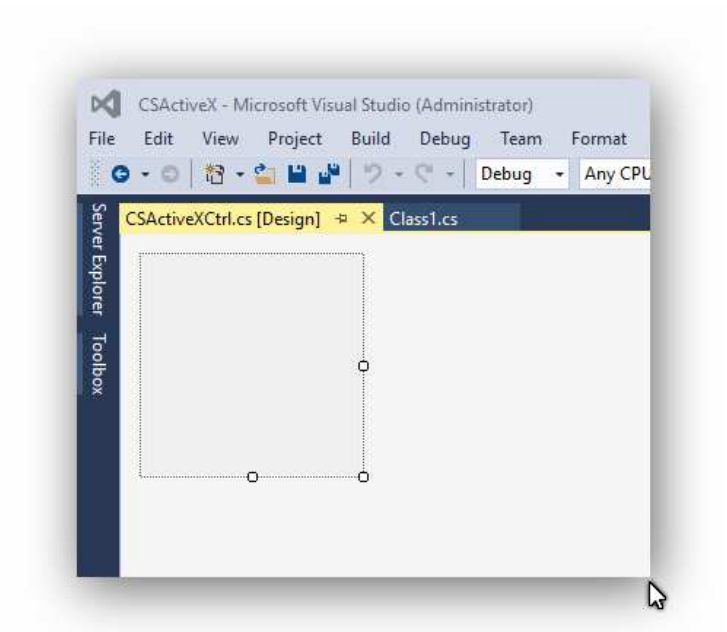
i dodajemy **ActiveXCtrlHelper** class (gotowy kod na dysku) i dodajemy do projektu

Dodajemy do projektu klasę kontrolną do budowy kontrolek wizualnych UI framwork w Project -> User control na niej oprzemy zbudowanie kontrolki ActiveX



Nazwiemy ją CSAciveXCtrl.cs

Pojawi nam się płótno do kreowania wizualnej części komponentu [Design]



W naszym przypadku zbudujemy komponent na tej klasie i stanie się ona naszą coclassą czyli (componet object class). Będziemy ją parametryzować aby zamienić ją ze zwykłej klasy Framework na klasę COM (ActiveX).

Przed dodaniem kolejnych własności typowych dla ActiveX sprawdzamy czy mamy podstawowy zestaw asemblacji aby mieć możliwość przypisywania parametrów ActiveX do klas. I dla pliku klasy związanego z UI u nas CSActiveXCtrl.cs dodajemy odpowiednie **using**

```
#region Using directives
    using System;
    using System.Collections.Generic;
    using System.ComponentModel;
    using System.Data;
    using System.Drawing;
    using System.Text;
    using System.Windows.Forms;
    using System.Runtime.InteropServices;
    using Microsoft.Win32;
    using System.Reflection;
    using System.Security.Permissions;
    #endregion
```

Oczywiście ta klasa pozwoli nam dodawać już kontrolki wizualne framework do projektu na jej płótnie ale nie jest na razie powiązana z ActiveX interfejsami wykonamy to w późniejszym etapie.

Budowa interfejsu dla ActiveX czyli zbioru metod i właściwości jakie chcemy aby dana kontrolka oferowała w kontenerze dla innego programu, które z niej ma skorzystać zaczynamy od zdefiniowania wewnątrz klasy CSAciveXCtrl powiązanego z nią interfejsu. W ramach kontrolki activeX może być wiele interfejsów i każdy z nich ma swój niepowtarzalny identyfikator GUID, który po rejestracji systemie Windows będzie po tym numerze rozpoznawany. System Windows porusza się po numerze GUID a nie nazwie. Nazwa jest tylko do przejścia dla programisty z nazwy na GUID ale wszystkie odwołania są w rzeczywistości do GUID to tak jakby w kodzie wpisać zamiast nazwy interfejsu jego GUID było by nam się ciężko po tych cyfrach się poruszać stąd kompilator automatycznie zmianie nazwy programisty na odpowiednie GUID w wywołaniach kofunkcji. Jest analogia to IP i DNS dokładnie tak samo w rzeczywistości poruszmy się po IP a jak nie mamy IP pytamy po nazwie DNS jakie jest jego IP.

Teraz w CSAciveXCtrl.cs dodajemy publiczny interfejs Project-> Add Class -> Interface

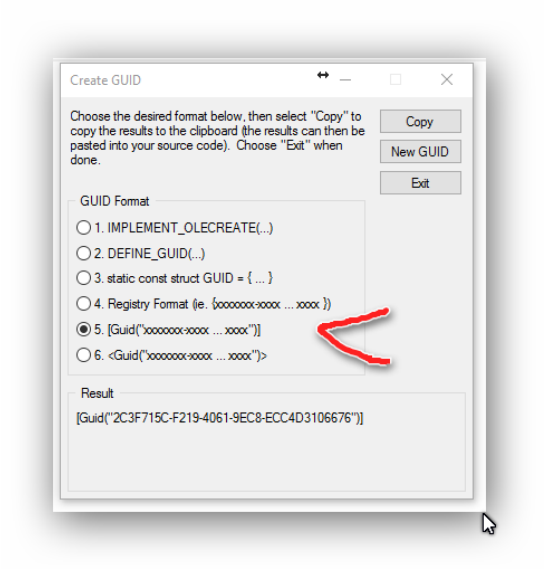
W naszym projekcie nadajmy mu nazwę `AxCSActiveXCtrl`

będzie ona odpowiedzialna za właściwości jakie chcemy aby otrzymał programista przy wkładaniu kontrolki do swojego projektu. Wygodnie jest nie tworzyć nowej pozycji w projekcie przez Project -> Add tylko w klasie CSActiveXCtrl.cs ręcznie dodawać kolejne interfejsy ponieważ są one automatycznie z tą klasą związane a dodanie przez Project -> Add spowoduje dodanie nie związanego interface z tą klasą tylko osobnego w osobnym pliku cs.

```
public interface axCSActiveXCtrl
  {
  }
```

Dodajemy również do interfejsu niepowtarzalny identyfikator globalny GIUD czyli tak naprawdę niepowtarzalnej nazwy interfejsu. Niepowtarzalnej dla wszystkich zarejestrowanych activex w systemie w naszym i innych komputerach.

Teraz do interfejsu musimy dodać identyfikator niepowtarzalny GUID. Tools -> Create GUID



Dodajemy pierwszy interface odpowiedzialny za metody i properties w naszym projekcie

```
[Guid("4C21D85F-F3EB-41D1-AE4E-5D8AA34BBC4A")]
 public interface axCSActiveXCtrl
  {
     // properties standardowe przewidzane przez COM dla komponentów
         ....
    // properties własne programisty
         ...
  }
```

Dodajemy jeszcze kolejny interfejs odpowiedzialny za same zdarzenia w komponencie.

```
public interface AxCSActiveXCtrlEvents
{
}
```

W coclassie musimy jeszcze dodać który z naszych interfejsów jest odpowiedzialny za komunikacje zdarzeń

pomost w ActiveX, który interfejs realizuje obsługę zdarzeń dla komponentu ActiveX

```
[ComSourceInterfaces(typeof(AxCSActiveXCtrlEvents))]
```

Jest coklasą realizująca w naszym programie obsługą zdarzeń Event

Np. AxCSActiveXCtrlEvents

Czyli coclassa przyjmuje następujące konfiguracje

Musimy jeszcze określić z jakiego typu bazowego OLE nasza klasa korzysta. W ActiveX technologia COM pozwala z określonych ściśle typów budować interfejsy i nadawać im funkcjonalność

Dla **interfejsów** w C# musimy podać jakiego typu jest ten interfejs

```
    [InterfaceType(ComInterfaceType.InterfaceIsIDispatch)]
```

Dla **coclass** w C# powiązanych z technologią COM (ActiveX) należy podać jakiego jest typy

```
    [ClassInterface(ClassInterfaceType.None)]
```

Dla coclassy kompletna parametryzacja dla COM

```
    [ProgId("DemoCSharpActiveX.HelloWorld")]
    [ClassInterface(ClassInterfaceType.None)]
    // przypisanie który interfejs jest od zdarzeń
    [ComSourceInterfaces(typeof(AxCSActiveXCtrlEvents))]
    [Guid("80B59B58-98EA-303C-BE83-D26E5D8D6794")]
    // dodanie co coclassy naszych interfejsów po przecinku bez interfejsu od Events
    public partial class CSActiveXCtrl : UserControl, AxCSActiveXCtrl
    {

        ……..

    }
```

Dodawanie properties do komponentu

W activeX nie ma od strony kontenera (programu, w którym wykorzystano kontrolkę) nie ma dostępu do zmiennych / properties programu bezpośrednio. Dostęp realizowany jest wyłącznie przez metody/funkcje interfejsu. Przy czym w interfejsie musimy zapewnić pewne standardowe podstawowe własności obowiązkowe, do których ma od razu dostęp program, który korzysta z kontrolki ActiveX i pierwszy te parametry ustawia. Dotyczą one podstawowych własności komponentu wizulanych czyli Visible, Enable, kolor tła BackColor i kolor pierwszo planowy ForeColor.

```
public interface axCSActiveXCtrl

  {

      #region Properties
```

```
        // standardowe własności w kontrolce COM
                bool Visible { get; set; }
                bool Enabled { get; set; }
                int ForeColor { get; set; }
                int BackColor { get; set; }


        //Nasza  nowa  własna properties np.

               float FloatProperty { get; set; }
        #endregion

}
```

musimy zdefiniować to w następujący sposób.

Czyli aby dodać do projektu properties musimy dodać w dwóch miejscach.

1. Najpierw prototyp metody w interfejsie u nas AxCSActiveXCtrl

```
float FloatProperty { get; set; }   // Custom property
```

2. I drugą częścią jest dodanie implementacji tej properties już nie w interfejsie tylko w coclassie czyli CSActiveXCtrl w naszym projekcie

```
private float fField = 0;
    public float FloatProperty
    {
        get { return this.fField; }
        set { this.fField = value; }
    }
```

Dla properties standardowych w komponencie COM dostęp do niech jest realizowany poprzez technologie OLE/COM co pozwoli nam zapewnić zgodność typów wywołania metod.

Implementacja w coclass dostępu do properties standardowych

```
        public new int ForeColor
        {
            get { return ActiveXCtrlHelper.GetOleColorFromColor(base.ForeColor);
}
            set { base.ForeColor =
ActiveXCtrlHelper.GetColorFromOleColor(value); }
        }

       public new int BackColor
        {
            get { return ActiveXCtrlHelper.GetOleColorFromColor(base.BackColor);
}
            set { base.BackColor =
ActiveXCtrlHelper.GetColorFromOleColor(value); }
        }
```

Dodawanie metod do komponentu

Czyli aby dodać do projektu metodę musimy dodać w dwóch miejscach.

1. Najpierw prototyp metody w **interfejsie** u nas AxCSActiveXCtrl

```
string HelloWorld();
```

2. I drugą częścią jest dodanie **implementacji** tej metody już nie w interfejsie tylko w coclassie czyli CSActiveXCtrl w naszym projekcie

```
public string HelloWorld()
{
        return "HelloWorld";
}
```

Nasz w projekcie wygląda następująco

```
[Guid("D4B8539E-3839-3913-8B1A-C551A9930864")]
    public interface AxCSActiveXCtrl
    {
        #region Properties


        bool Visible { get; set; }          // Typical control property
        bool Enabled { get; set; }          // Typical control property
        int ForeColor { get; set; }         // Typical control property
        int BackColor { get; set; }         // Typical control property
        float FloatProperty { get; set; }   // Custom property / nasza własna properties


        #endregion


        #region Methods


        void Refresh();                     // standardowa metoda COM
        string HelloWorld();                // nasza własna

        #endregion
    }
```

Dodawanie Event / Zdarzeń do komponentu

Do obsługi zdarzeń trzeba zbudować pomost pomiędzy dwoma platformami obsługi zdarzeń OLE/COM i .NET tu z pomocą przychodzi klasa wcześniej zaimplementowana ActiveXCtrlHelper.cs

Etap dodawania zdarzeń też można podzielić na kilka etapów. Zdarzenia w komponencie COM również dzielimy na standardowe oraz własne.

Mamy już interfejs **AxCSActiveXCtrlEvents** odpowiedzialny za same zdarzenia w komponencie.

Do głównej coclassy projektu należy dodać, który z interfejsów odpowiada za obsługę zdarzeń.
[ComSourceInterfaces(typeof(AxCSActiveXCtrlEvents))]

Proces dodawania zdarzenia.
Dla przykładu dodane zostanie zdarzenie reagujące na zmianę wartości w polu properties FloatProperty

1. Dodanie do interfejsu od zdarzeń w projekcie **AxCSActiveXCtrlEvents** W przypadku interface opertego na bazowym interfejsie IDispatch (events) konieczne jest nadanie identyfikatorów dla każdego zdarzenia

```
DispId(2)]
  void FloatPropertyChanging(float NewValue, ref bool Cancel);
```

dla klas standardowych np. Click() również musimy nadać DispId

```
[DispId(1)]
void Click();
```

2. Dodanie do coclassy zmienną typu delegacja do przechowywania nowego typu zdarzeń obsługi zdarzenia event handler w platformie .NET. Delegacja musi być publiczna i w trybie ComVisible(false)

```
[ComVisible(false)]
public delegate void FloatPropertyChangingEventHandler(float NewValue, ref bool Cancel);
```

3. Dodanie do coclassy zdarzenie Event opartego na nowym typie zdarzenia z punku 2
   `public event FloatPropertyChangingEventHandler FloatPropertyChanging;`
4. Dodanie do obsługi SET w properties dla przykładowego pola FloatProperty wywołania obsługi zdarzenia

```
if (null != FloatPropertyChanging)
    FloatPropertyChanging(value, ref cancel);
```

Czyli kod kompletny do zmiany get,set properties z obsługą nowego zdarzenia

```
public float FloatProperty
 {
     get { return this.fField; }
     set
     {
         bool cancel = false;
         if (null != FloatPropertyChanging)
             FloatPropertyChanging(value, ref cancel);
         if (!cancel)
         {
             this.fField = value;
             this.lbFloatProperty.Text = value.ToString();
         }
     }
 }
```

Dla standardowego zdarzenia Click procedura jest podobna

```
[ComVisible(false)]
public delegate void ClickEventHandler();
public new event ClickEventHandler Click = null;

void CSActiveXCtrl_Click(object sender, EventArgs e)
{
        if (null != Click) Click(); wywołanie Click event.
}
```

Dodanie obsługi metody jest w konstruktorze coclassy

```
base.Click += new EventHandler(CSActiveXCtrl_Click);
```

Nasz interfejs od zdarzeń w projekcie przyjmuje postać.

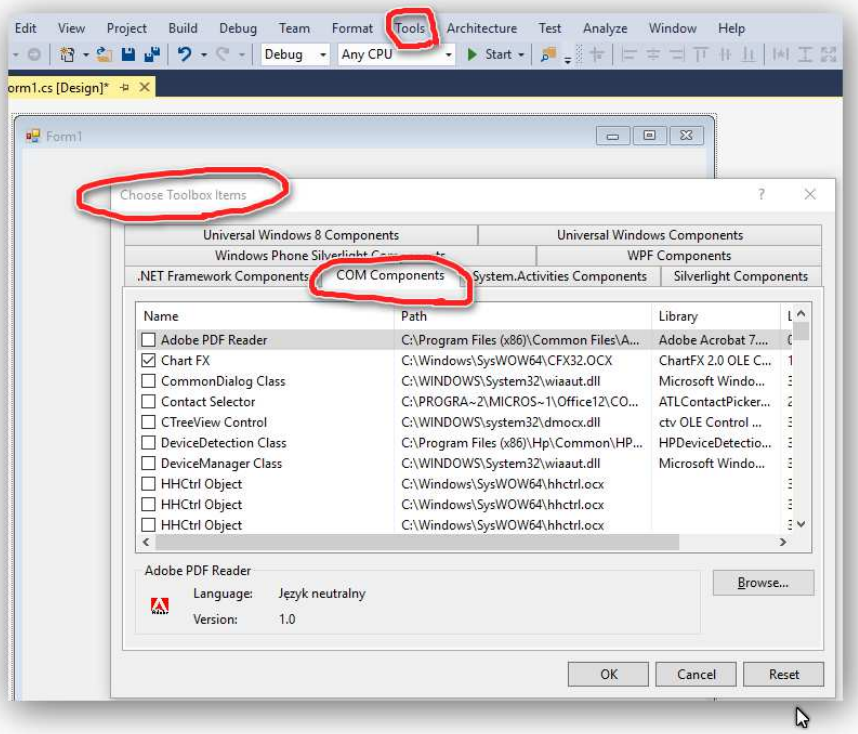
```
    // dodajemy pomost w ActiveX, który interfejs realizuje obsługę zdarzeń dla komponetu
ActiveX
    // Nie powtrzalny identyfikator interfeksu GUID
    [Guid("901EE2A0-C47C-43ec-B433-985C020051D5")]
    [InterfaceType(ComInterfaceType.InterfaceIsIDispatch)]
    // punbliczny interface do zdarzeń
    public interface AxCSActiveXCtrlEvents
    {
        #region Events

        // w przypadku inerface IDispatch konieczne jest nadanie identyfikatorów dla każdego
zdarzenia
        // identyfikatory dla metody
        [DispId(1)]
        void Click();

        [DispId(2)]
        void FloatPropertyChanging(float NewValue, ref bool Cancel);

        #endregion
    }
```

Dodawanie zarejestrowanych komponentów do innych projektów

Tools -> Choose toolbox item …

Wybieramy COM components

Zobaczymy zarejestrowane w systemie serwery ActiveX



Program zaimportuje przez automatyzacje i utworzy TLB struktury opisu obiektu. Po wyborze na formie zaznaczamy obszar gdzie wstawić część wizualną obiektu.