

Ćwiczenie 4.2

WCF – kontrakty danych, operacje jednokierunkowe i zwrotne (callbacks).

Autor: Mariusz Fraś

1 Cele ćwiczenia

Celem ćwiczenia jest:

1. Zapoznanie się z kontraktami danych - specyfikacja przekazywania w komunikatach złożonych zestawów danych.
 2. Poznanie kolejnych opcji konfigurowania serwisu i typu operacji usług, w tym operacji jednokierunkowych.
 3. Poznanie sposobu definiowania i obsługi usług typu callback (lub inaczej duplex) – usługi ze zwrotnym wywołaniem procedury obsługi odpowiedzi na żądanie.
- **Pierwsza część zadania** jest do wykonania według podanej instrukcji i ewentualnych poleceń prowadzącego zajęcia laboratoryjne.
 - **Druga część zadania** jest do przygotowania i oddania lub do wg wykonania wg. poleceń prowadzącego na kolejnych zajęciach.

2 Zadanie – część I

Zadanie polega na realizowaniu usługi wykorzystującej kontrakt danych (**DataContract**) oraz usługi realizującej operacje synchronicznie i asynchronicznie wykonywane w serwisie wielowątkowo. Ponadto zadanie polega na zrealizowaniu usługi asynchronicznie zwracającej wynik wykonania operacji, ze zwrotnym wywołaniem – tzw. kontrakt typu **Callback**.

Zadanie składa się z kilku następujących etapów:

1. Utworzenie usługi zawierającej kontrakt danych i klienta korzystającego z niej.
2. Definicja serwisu z operacją typu żądanie-odpowiedź i operacji jednokierunkowej (**OneWay** - żądanie bez odpowiedzi), oraz deklaracja serwisu jako wielowątkowego.
3. Następnie rozbudowa klienta dla wywołania dodatkowych operacji.
4. Zdefiniowanie usługi z kontraktem typu **Callback**, z usługami jednokierunkowymi.
5. Rozbudowa klienta o uchwyt (ang. *handler*) z operacjami, które będą wywoływane przez serwer po zakończeniu usługi, w celu zwrócenia rezultatu działania.

Uwaga: aplikacje należy uruchamiać z poziomu konsoli – to pozwoli łatwiej przyswoić materiał (zwłaszcza kwestie konfigurowania serwisów).

1. Wstępne przygotowanie solucji (projektów)	<ul style="list-style-type: none"> • Utwórz nową solucję, a w niej kolejno 3 projekty: <ul style="list-style-type: none"> - kontrakt usługi – projekt WCF Library, - host usługi – projekt aplikacji konsolowej, - klient usługi – projekt aplikacji konsolowej. • Skonfiguruj wstępnie kolejno 3 projekty (podobnie jak w poprzednim ćwiczeniu): <ul style="list-style-type: none"> - referencję w hoście do System.ServiceModel i do kontraktu, - referencję w kliencie do System.ServiceModel. <p>Uwaga: nie konfiguruj Connected Services References dopóki kontrakt i host nie będą gotowe.</p>
2. Definiowanie kontraktu usługi i kontraktu danych	<p>Zdefiniuj w projekcie kontraktu – w pliku opisującym interfejs usługi IService1.cs – kontrakt usługi mającej operację (metodę) przyjmującą parametry typu złożonego (złożony zestaw danych) i zwracającą parametr typu złożonego – typ opisany kontraktem danych (DataContract). W prezentowanym przykładzie są to dane opisujące liczbę zespoloną (mającą część rzeczywistą i urojoną).</p> <ul style="list-style-type: none"> • Wpisz opis interfejsu (kontraktu usługi) IComplexCalc zawierający metodę addCNum: <pre>[ServiceContract] public interface ICalculator { [OperationContract] ComplexNum addCNum(ComplexNum n1, ComplexNum n2); }</pre>

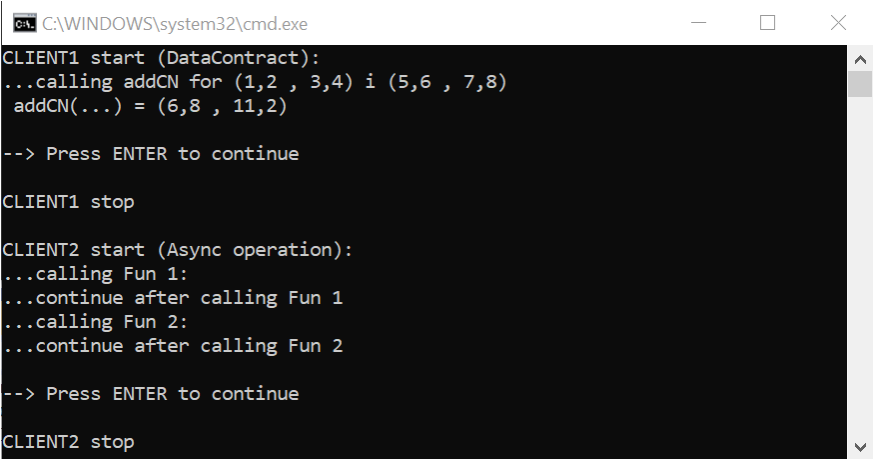
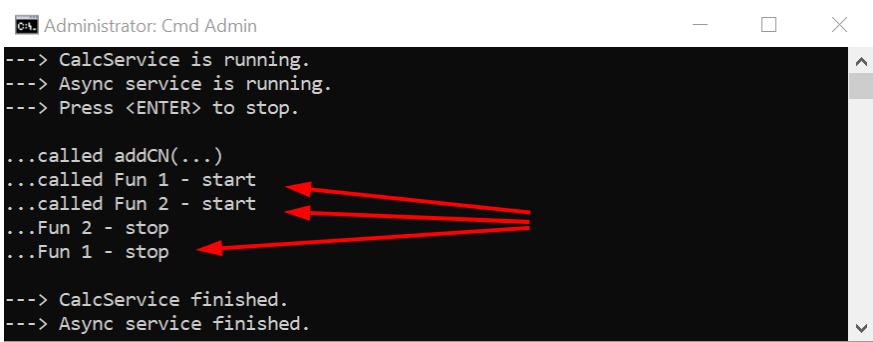
	<ul style="list-style-type: none"> Poniżej zdefiniuj kontrakt danych ([DataContract]) odpowiadający liczbie zespolonej ComplexNum (czyli złożony zestaw danych) przekazywanej i zwracanej przez metodę addCNum usługi, t.j: <ul style="list-style-type: none"> klasę ComplexNum – ze specyfikatorem [DataContract], elementy [DataMember] – są przekazywane w wiadomościach, tu także wykorzystana możliwa opcja - konstruktor klasy ComplexNum. <pre> [DataContract] public class ComplexNum { string description = "Complex number"; [DataMember] public double real; [DataMember] public double imag; [DataMember] public string Desc { get { return description; } set { description = value; } } public ComplexNum(double r, double i) { this.real = r; this.imag = i; } } </pre>
3. Implementacja kontraktu usługi	<p>Zaimplementuj kontrakt – klasę MyComplexCalc (z wymaganymi metodami).</p> <ul style="list-style-type: none"> W pliku Service1.cs wpisz kod klasy MyComplexCalc implementującej interfejs IComplexCalc. <pre> public class MyComplexCalc : IComplexCalc { public ComplexNum addCNum(ComplexNum n1, ComplexNum n2) { Console.WriteLine("...called addCNum(...)"); return new ComplexNum(n1.real + n2.real, n1.imag + n2.imag); } } </pre> <ul style="list-style-type: none"> Dodaj odpowiednie deklaracje using ... stosownie do potrzeb (np. z menu kontekstowego prawego klawisza myszki).
4. Implementacja i konfiguracja hosta usługi.	<p>Utwórz aplikację konsolową hostującą usługę WCF.</p> <ul style="list-style-type: none"> W projekcie hosta w pliku konfiguracyjnym zdefiniuj podstawowe elementy serwisu – punkt końcowy, adres bazowy, binding, itp. jednym ze sposobów poznanych w poprzednim ćwiczeniu. Np.:

	<ul style="list-style-type: none"> o <u>Programując w kodzie C# hosta.</u> Otwórz plik Program.cs i wpisz kod realizujący następujące funkcje: <ul style="list-style-type: none"> ▪ Utworzenie URI z bazowym adresem serwisu. ▪ Utworzenie obiektu ServiceHost. ▪ Utworzenie (dodanie do hosta) punktu końcowego (endpoint). ▪ Skonfigurowanie hosta serwisu aby udostępniał metadane. ▪ Uruchomienie serwisu. <p>(w miejsce kropek ... dopisz odpowiedni kod)</p> <pre>Uri baseAddress1 = new Uri(...); ServiceHost myHost1 = new ServiceHost(...); ServiceEndpoint endpoint1 = myHost1.AddServiceEndpoint(...); // Metadata: ServiceMetadataBehavior smb = new ServiceMetadataBehavior(); mb.HttpGetEnabled = true; myHost1.Description.Behaviors.Add(smb); try { myHost1.Open(); Console.WriteLine("--> ComplexCalculator is running."); Console.WriteLine("--> Press <ENTER> to stop.\n"); Console.ReadLine(); //wait for stop myHost1.Close(); Console.WriteLine("--> ComplexCalculator finished"); } catch (CommunicationException ce) { Console.WriteLine("Exception occurred: {0}", ce.Message); myHost1.Abort(); } }</pre> <p><u>albo częściowo skonfiguruj serwis w pliku App.config:</u></p> <ul style="list-style-type: none"> o Otwórz plik App.config i wpisz w węźle <configuration> (za węzłem <startup>) kod definiujący niezbędne elementy: <ul style="list-style-type: none"> • Uruchom z konsoli serwis (host) i sprawdź działanie.
5. Tworzenie klienta proxy i implementacja klienta	<p><u>Utworzenie kodu klienta proxy (proxy client) z użyciem opcji Add Service Reference.</u></p> <ul style="list-style-type: none"> • W projekcie klienta, dodaj referencję serwisową do zdefiniowanej usługi. Uwaga: pamiętaj o uruchomieniu najpierw serwisu! <p><u>Implementacja klienta (aplikacji):</u></p> <ul style="list-style-type: none"> • W pliku Program.cs wpisz kod klienta realizujący następujące działania: <ul style="list-style-type: none"> o Utworzenie instancji klienta proxy. o Przygotowanie parametrów do wywołania operacji serwisu. o Wywołanie operacji usługi (z użyciem klienta proxy). o Zamknięcia klienta proxy.

	<pre> class Program { static void Main(string[] args) { ComplexCalcClient client1 = new ComplexCalcClient("WSHttpBinding_IComplexCalc"); ComplexNum cnum1 = new ComplexNum(); cnum1.real= 1.2; cnum1.imag= 3.4; ComplexNum cnum2 = new ComplexNum(); cnum2.real = 5.6; cnum2.imag = 7.8; Console.WriteLine("\nCLIENT1 - START"); Console.WriteLine("...calling addCNum(...)"); ComplexNum result1 = client1.addCNum(cnum1, cnum2); Console.WriteLine(" addCNum(...) = ({0},{1})", result1.real, result1.imag); Console.WriteLine("--> Press ENTER to continue"); Console.ReadLine(); client1.Close(); Console.WriteLine("CLIENT1 - STOP"); } } </pre> <ul style="list-style-type: none"> • Utwórz odpowiednie referencje i zlikwiduj sygnalizowane błędy.
6. Testowanie działania aplikacji	<ul style="list-style-type: none"> • Uruchom serwis (aplikację hostującą usługę WCF) i klienta w oddzielnych oknach konsoli i skontroluj działanie aplikacji. • Zakończ działanie wszystkich aplikacji.
7. Serwis wielowątkowy i operacje asynchroniczne	<p>Zdefiniuj w dotychczasowym projekcie usługi, drugą usługę z operacją asynchroniczną.</p> <p>Note: <i>Presented here asynchronous requests pattern is one of several available.</i></p> <ul style="list-style-type: none"> • W pliku interfejsu, dopisz interfejs drugiej usługi zawierającej deklaracje dwóch metod: <ul style="list-style-type: none"> - operację Fun1 – operację typu asynchronicznego bez odpowiedzi (operację jednokierunkową) – typu OneWay - operację Fun2 standardową operację typu żądanie-odpowieź. Obie funkcje nic nie zwracają. <pre> [ServiceContract] public interface IAsyncService { [OperationContract(IsOneWay = true)] void Fun1(String s1); [OperationContract] void Fun2(String s2); } </pre>

	<ul style="list-style-type: none"> • Zaimplementuj w kontrakcie nową usługę i obie operacje. W pliku Service1.cs dopisz kod klasy AsyncService: implementującej interfejs IAsyncService (są to nazwy własne). <ul style="list-style-type: none"> ○ Zadeklaruj zachowanie serwisu jako wielowątkowe. ○ Dopisz kod operacji Fun1 i Fun2 ○ Umieść w nich uśpienie procesu na kilka sekund w celu symulacji długiego czasu przetwarzania. <pre>[ServiceBehavior(ConcurrencyMode=ConcurrencyMode.Multiple)] public class AsyncService : IAsyncService { public void Fun1(String s1) { Console.WriteLine("...called Fun 1 - start"); Thread.Sleep(4*1000); // sleep for 4 sec. (4000 ms) Console.WriteLine("...Fun 1 - stop"); return; } public void Function2(String s2) { Console.WriteLine("...called Fun 2 - start "); Thread.Sleep(2*1000); // sleep for 2 sec. (2000 ms) Console.WriteLine("...Fun 2 - stop"); return; } }</pre> <p>Uwaga: zachowanie wielowątkowe jest niezbędne do realizacji żądań równoległe (natychmiastowe obsługiwane kolejnych żądań). W szczególnych przypadkach w zależności od wiązania i serwisu, klient mimo to może być blokowany do czasu ukończenia poprzedniego żądania</p>
8. Rozbudowa hosta dla drugiej usługi	<p>Dopisz w kodzie aplikacji hostującej uruchomienie drugiego serwisu.</p> <ul style="list-style-type: none"> • W pliku Program.cs dopisz w odpowiednich miejscach kod realizujący następujące funkcje: <ul style="list-style-type: none"> ○ Utworzenie URI z bazowym adresem drugiego serwisu. Podaj inny niż wcześniej adres (port) dla usługi. ○ Utworzenie obiektu hosta drugiego serwisu. ○ Dodanie punktu końcowego z wiązaniem BasicHttpBinding. ○ Skonfigurowanie hosta serwisu aby udostępniał metadane. ○ Uruchomienie drugiego serwisu. <p>Uwaga: tu kropki [...] wskazują już istniejące fragmenty kodu.</p> <pre>static void Main(string[] args) { [...] Uri baseAddress2 = new Uri(...); ServiceHost myHost2 = new ServiceHost(...); ServiceEndpoint endpoint2 = myHost2.AddServiceEndpoint(...); myHost2.Description.Behaviors.Add(smb); }</pre>

	<pre> try { [...] myHost2.Open(); Console.WriteLine("--> Async service is running"); [...] myHost2.Close(); Console.WriteLine("--> Async service finished"); [...] } catch (CommunicationException ce) { [...] myHost2.Abort(); } } </pre> <p>Uważaj aby NIE zatrzymać serwisu przed odesłaniem wyników !</p> <ul style="list-style-type: none"> • Skompiluj i uruchom z konsoli serwis (host) i sprawdź jego działanie.
9. Rozbudowa klienta dla drugiej usługi.	<p><u>Utworzenie kodu proxy (z użyciem opcji Add Service Reference).</u></p> <ul style="list-style-type: none"> • W projekcie aplikacji konsolowej klienta, dodaj referencję serwisową do drugiej usługi. Pamiętaj o uruchomieniu najpierw serwisu ! <p><u>Implementacja/rozbudowa klienta (aplikacji):</u></p> <ul style="list-style-type: none"> • W pliku Program.cs dopisz kod realizujący następujące działania: <ul style="list-style-type: none"> ○ Utworzenie instancji klienta proxy do drugiej usługi. ○ Wywołanie operacji Fun1 i Fun2 (10 ms delays are added to properly display the order of service activities) ○ Zamknięcia klienta proxy. <pre> static void Main(string[] args) { [HERE THE PREVIOUS CODE] Console.WriteLine("CLIENT2 - START (Async service)"); AsyncServiceClient client2 = new AsyncServiceClient("BasicHttpBinding_IAsyncService"); Console.WriteLine("...calling Fun 1"); client2.Fun1("Client2"); Thread.Sleep(10); Console.WriteLine("...continue after Fun 1 call"); Console.WriteLine("...calling Fun 2"); client2.Fun2("Client2"); Thread.Sleep(10); Console.WriteLine("...continue after Fun 2 call"); Console.WriteLine("--> Press ENTER to continue"); Console.ReadLine(); client2.Close(); Console.WriteLine("CLIENT2 - STOP"); } </pre>

<p>10. Testowanie aplikacji</p>	<ul style="list-style-type: none"> • Uruchom serwis (aplikację hostującą usługę WCF) w jednym oknie konsoli. • Uruchom klienta w drugim oknie konsoli. • Skontroluj wyniki działania. <ul style="list-style-type: none"> ◦ Zwróć uwagę na momenty czasowe działania klienta i serwisu przy wywoływaniu metod Funkcja1 i Funkcja2 oraz kolejność komunikatów w serwisie. ◦ Zwróć uwagę, że po wywołaniu Funkcja1 natychmiast realizowano kolejne działania (wywołanie funkcji 2). Po wywołaniu funkcji 2 trzeba czekać na jej zakończenie. • Wynik działania powinien być podobny do poniższego: <p>Okno klienta:</p>  <p>Okno hosta:</p>  <ul style="list-style-type: none"> • Zakończ działanie wszystkich aplikacji.
<p>11. Wstępne przygotowanie projektu trzeciej usługi</p>	<p>Trzecia usługa będzie zwracać wyniki działań poprzez wywołania zwrotne do klienta (ang. <i>callbacks</i>), czyli asynchronicznie. To pozwala klientowi od razu realizować działania bez blokowania się w oczekiwaniu na wyniki wywołania operacji. Wyniki zostaną zwrócone przez serwis po wykonaniu operacji.</p> <p>W aplikacji ten sam host będzie uruchamiał dwie usługi – poprzednią i tą trzecią. Trzecia usługa będzie realizowana w oddzielnym projekcie.</p>

	<ul style="list-style-type: none"> • Dodaj do solucji projekt WCF Library trzeciej usługi (np. o nazwie CallbackService). • Dodaj w hoście referencję do tego drugiego pliku kontraktu.
12. Definiowanie kontraktu usługi z operacjami typu callback	<p>Zdefiniuj w projekcie kontraktu – w pliku opisującym interfejs usługi IService1.cs – kontrakt usługi typu Callback mającej dwie operacje (metody). W tym celu definiuje się:</p> <ul style="list-style-type: none"> - operacje typu OneWay, - zachowanie usługi typu CallbackContract specyfikujące typ interfejsu zwrotnego (tu: specyfikujemy go jako ISuperCalcCallback) – interfejs dla obsługi wywołań zwrotnych - ten interfejs ICallbackHandler musi być implementowany w kliencie, - zachowanie można zdefiniować jako atrybut kontraktu serwisu (w [ServiceContract]), - dodatkowo określimy wymaganie działania instancji serwisu w ramach sesji. <ul style="list-style-type: none"> • Zdefiniuj kod interfejsu kontraktu usługi ISuperCalc zawierający metody/operacje asynchroniczne (z callbackiem) Factorial (obliczanie silni) i DoSomething: (symulacja obliczania czegoś), definiując dodatkowo atrybut serwisu CallbackContract i wymaganie trybu działania w sesji: <pre> [ServiceContract (SessionMode = SessionMode.Required, CallbackContract=typeof(ISuperCalcCallback))] public interface ISuperCalc { [OperationContract(IsOneWay = true)] void Factorial(double n); [OperationContract(IsOneWay = true)] void DoSomething(int sec); } </pre> • Zdefiniuj tutaj również interfejs ISuperCalcCallback zawierający opis metod wywoływanych u klienta w celu przekazania rezultatów wykonania operacji Factorial i DoSomething – tj. zawierający metodę FactorialResult dla silni i metodę DoSomethingResult dla obliczeń czegoś. Dopisz w tym samym pliku drugi interfejs: <pre> public interface ISuperCalcCallback { [OperationContract(IsOneWay = true)] void FactorialResult(double result); [OperationContract(IsOneWay = true)] void DoSomethingResult(string result); } </pre>

13.Implementacja kontraktu usługi	<p>Zaimplementuj kontrakt – klasę implementującą każdą z wymaganych metod interfejsu ISuperCalc.</p> <ul style="list-style-type: none"> W pliku Service1.cs. wpisz kod klasy MySuperCalc implementującej interfejs ISuperCalc. Dla usługi definiuje się także zachowanie InstanceContextMode=PerSession oznaczające tworzenie instancji obiektu (instancji usługi) dla każdej sesji. W konstruktorze pobierany jest uchwytu (handler) dla callback'u. <pre>[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession, ConcurrencyMode = ConcurrencyMode.Multiple)] public class MySuperCalc : ISuperCalc { double result; ISuperCalcCallback callback = null; public MySuperCalc() { callback = OperationContext.Current.GetCallbackChannel <ISuperCalcCallback>(); } public void DoSomething(int sec) { Console.WriteLine("...called DoSomething({0})", sec); if (sec > 2 & sec < 10) Thread.Sleep(sec * 1000); else Thread.Sleep(3000); callback.DoSomethingResult("Calculatons lasted " + sec + " second(s)"); } public void Factorial(double n) { Console.WriteLine("...called Factorial({0})", n); Thread.Sleep(1000); result = 1; for (int i = 1; i <= n; i++) result *= i; callback.FactorialResult(result); } }</pre> <ul style="list-style-type: none"> W obu metodach na końcu wywołujemy metody callback'owe w kliencie.
14.Rozbudowa hosta dla trzeciej usługi	<p>Dopisz w kodzie aplikacji hostującej uruchomienie trzeciego serwisu.</p> <ul style="list-style-type: none"> W pliku Program.cs dopisz w odpowiednich miejscach kod realizujący następujące funkcje: <ul style="list-style-type: none"> Utworzenie URI z bazowym adresem trzeciego serwisu. Utworzenie obiektu hosta trzeciego serwisu. Dodanie punktu końcowego z wiązaniem WSDualHttpBinding.

	<ul style="list-style-type: none"> o Zdefiniowanie metadanych serwisu. o Uruchomienie trzeciego serwisu. <p>Uwaga: tu kropki [...] oddzielają już istniejące fragmenty kodu.</p> <pre>static void Main(string[] args) { [...] Uri baseAddress3 = new Uri(...); ServiceHost myHost3 = new ServiceHost(typeof(MySuperCalc), baseAddress3); WSDualHttpBinding myBinding3 = new WSDualHttpBinding(); ServiceEndpoint endpoint3 = myHost3.AddServiceEndpoint(typeof(ISuperCalc), myBinding3, "ThirdService"); myHost3.Description.Behaviors.Add(smb); try { [...] myHost3.Open(); Console.WriteLine("--> Callback SuperCalc is running."); [...] myHost3.Close(); Console.WriteLine("--> Callback SuperCalc finished"); } catch (CommunicationException ce) { [...] myHost3.Abort(); } }</pre> <ul style="list-style-type: none"> • Uruchom z konsoli serwis (host) i sprawdź działanie.
15.Rozbudowa klienta dla wykorzystania trzeciej usługi	<ul style="list-style-type: none"> • Dodaj w kliencie referencję serwisową do trzeciej usługi: <p>Uwaga: pamiętaj o uruchomieniu najpierw aplikacji hostującej usługę!</p> <ul style="list-style-type: none"> • Dodaj do projektu klienta nową klasę (np. o nazwie SuperCalcCallback), w której zdefiniowane będą operacje wywoływane zwrotnie przez serwis, aby odesłać wyniki działania jego operacji usług. <pre>class SuperCalcCallback : ISuperCalcCallback { public void FactorialResult(double result) { //here the result is consumed Console.WriteLine(" Factorial = {0}", result); } public void DoSomethingResult(string info) { //here the result is consumed Console.WriteLine(" Calculations: {0}", info); } }</pre>

	<ul style="list-style-type: none"> Otwórz plik Program.cs i dopisz kod w funkcji main realizujący następujące działania: <ul style="list-style-type: none"> Utworzenie obiektu uchwytu (handlera) z operacjami odbioru wyników od serwisu. Utworzenie instancji klienta proxy (<i>proxy client</i>). Wywołanie operacji usługi z klienta (proxy). Zamknięcia klienta <p>Odbiór i wypisywanie wyników będzie asynchroniczny – inicjowany przez serwis.</p> <pre>static void Main(string[] args) { [HERE THE PREVIOUS CODE] Console.WriteLine("\nCLIENT3 - START (Callbacks):"); SuperCalcCallback myCbBHandler = new SuperCalcCallback (); InstanceContext instanceContext = new InstanceContext(myCbHandler); SuperCalcClient client3 = new SuperCalcClient(instanceContext); double value1 = 10; Console.WriteLine("...call of Factorial({0})...", value1); client3.Factorial(value1); int value2 = 5; Console.WriteLine("...call of DoSomething..."); client3.DoSomething(value2); value1 = 20; Console.WriteLine("...call of Factorial({0})...", value1); client3.Factorial(value1); Console.WriteLine("--> Client must wait for the results"); Console.WriteLine("--> Press ENTER after receiving ALL results"); Console.ReadLine(); client3.Close(); Console.WriteLine("CLIENT3 - STOP"); }</pre>
16. Testowanie działania aplikacji	<ul style="list-style-type: none"> Uruchom serwis (aplikację hostującą usługi) w jednym oknie konsoli. Uruchom klienta w drugim oknie konsoli. Skontroluj wyniki działania. Zwróć uwagę na momenty czasowe działania serwisu i klienta. Ostateczny wynik działania powinien być podobny do poniższego:

Okno klienta:

```
C:\WINDOWS\system32\cmd.exe

CLIENT1 start (DataContract):
...calling addCN for (1,2 , 3,4) i (5,6 , 7,8)
  addCN(...) = (6,8 , 11,2)

--> Press ENTER to continue

CLIENT1 stop

CLIENT2 start (Async operation):
...calling Fun 1:
...continue after calling Fun 1
...calling Fun 2:
...continue after calling Fun 2

--> Press ENTER to continue

CLIENT2 stop

CLIENT3 start:
...calling Factorial(10)...
...calling DoSomething...
...calling Factorial(20)...
--> Client must wait to NOT finish before receiving results (callback)
--> Press ENTER after (!) you receive ALL results
  Factorial = 3628800
  Factorial = 2,43290200817664E+18
  SomethingDone: Calculations took 5 seconds

CLIENT3 - stop
```

Okno hosta:

```
Administrator: Cmd Admin

---> CalcService is running.
---> Async service is running.
---> Callback service SuperCalc is running
---> Press <ENTER> to stop.

...called addCN(...)
...called Fun 1 - start
...called Fun 2 - start
...Fun 2 - stop
...Fun 1 - stop
...called Factorial(10)
...called DoSomething(5)
...called Factorial(20)
...Factorial(10) - finished
...Factorial(20) - finished
...DoSomething(5) - finished

---> CalcService finished.
---> Async service finished.
---> SuperCalc service finished.
```

- Zakończ działanie wszystkich aplikacji..

3 Zadanie – część II

Szczegółowe wymagania i wybór zadania (A, B lub inne) określa prowadzący grupy.

A. Przećwiczyć prezentowane w ćwiczeniu techniki.

1. Definiowanie, implementacja i wykorzystywanie kontraktów danych.
2. Operacje jednokierunkowe - typu OneWay.
3. Tworzenie serwisu i klienta z użyciem kontraktu serwisu typu Callback..

B. Przykładowe zadanie do wykonania

Napisać aplikację WCF - serwis i klienta spełniające następujące wymagania:

1. Klient jest aplikacją konsolową lub graficzną oraz:

- a. Wywoływanie operacji następuje po wybraniu akcji – np. po naciśnięciu odpowiedniego przycisku lub wybraniu opcji menu.
- b. Przesyłane dane są podawane przez użytkownika (są pobierane z kontrolek w aplikacji (dla aplikacji graficznej) lub wpisywane w konsoli przez użytkownika).
- c. Wyniki działania wywoływanych operacji są wypisywane na ekranie (w odpowiednim polu okna aplikacji (ewentualnie w kilku oddzielnych polach) lub w konsoli).

Uwaga: aplikacja graficzna może być troszkę wyżej punktowana!

2. Serwis (usługa) jest hostowany w aplikacji konsolowej oraz:

- a. Usługa utrzymuje (przechowuje) jakiś magazyn danych (zbiór/lista rekordów danych itp.), które można modyfikować (dodawać, usuwać,...) za pomocą operacji.
- b. Rekord danych powinien zawierać **dane różnego typu**.
- c. Usługa ma operację, która dodaje rekord danych. Ta operacja przyjmuje parametr określony za pomocą **kontraktu danych** (typ określający cały rekord).
- d. Usługa ma operację zwracającą wartości zestawu danych, oraz operacje zwracające wybrane pojedyncze dane.
- e. Usługa ma jakąś **operację typu callback (duplex)**
– np. operację która coś wyszukuje lub sprawdza (np. czy w zestawie występuje jakaś podana wartość albo cecha albo podzbiór rekordów lub ile jest rekordów jakiegoś rodzaju itp.). Czyli:
 - Operacja po wywołaniu natychmiast zwraca sterowanie. Klient może w tym czasie wykonywać inne działania.
 - Po wykonaniu operacji informuje klienta o wyniku działania operacji (np. czy dane występują, wyświetla znalezione dane, lub podaje jakieś inne informacje, itp. – zależnie od tego jaka to operacja). Te działanie jest realizowane poprzez wywołanie zwrotne (wywołanie *callback* w kliencie).
 - Dodać kilkusekundową. zwłokę w operacji dla zauważenia efektu działania, aby można było pokazać, jej asynchroniczność.

3. Zestaw danych (znaczenie, typy) należy sobie wybrać indywidualnie. Rekord zestawu danych powinien się składać z **danych różnego typu!**

Uwaga: klient nie może przechowywać listy danych głównie u siebie i jedynie kopiować ją na serwer. To serwer przechowuje dane i w razie potrzeby udostępnia je klientowi.

Uwaga: Aby utrzymywać trwale dane zdefiniować zachowanie usługi jako Single, lub zdefiniować zmienną dla przechowywania danych jako statyczną.