

# Projektowanie aplikacji ASP.NET

## Wykład 11/15

### ClickOnce

Wiktor Zychła 2018/2019

---

#### Spis treści

2	ClickOnce .....	2
3	WebServices .....	5
3.1	Usługa .....	5
3.2	Proxy.....	7
3.3	Adres zwrotny dla usługi .....	11

## 2 ClickOnce

Technologia [ClickOnce](#) pozwala na uruchamianie aplikacji desktopowych bezpośrednio z serwera aplikacji.

Motywacja: aplikacja w przeglądarce ma dostęp wyłącznie do API ustandaryzowanych w ramach HTML5. W szczególności aplikacja przeglądarkowa nie ma swobodnego dostępu do systemu plików oraz do zasobnika certyfikatów. To uniemożliwia wykonanie w technologii przeglądarkowej takiej aplikacji, która wykona **podpisanie cyfrowe** dokumentu elektronicznego certyfikatem użytkownika – klucz prywatny certyfikatu jest dostępny wyłącznie na maszynie użytkownika i nie ma technicznie żadnego sposobu żeby dostać się do tego klucza prywatnego z poziomu przeglądarki, ani też nie ma sposobu żeby ten klucz wysłać na serwer (to drugie jest akurat oczekiwane, nie powinno być takiej możliwości).

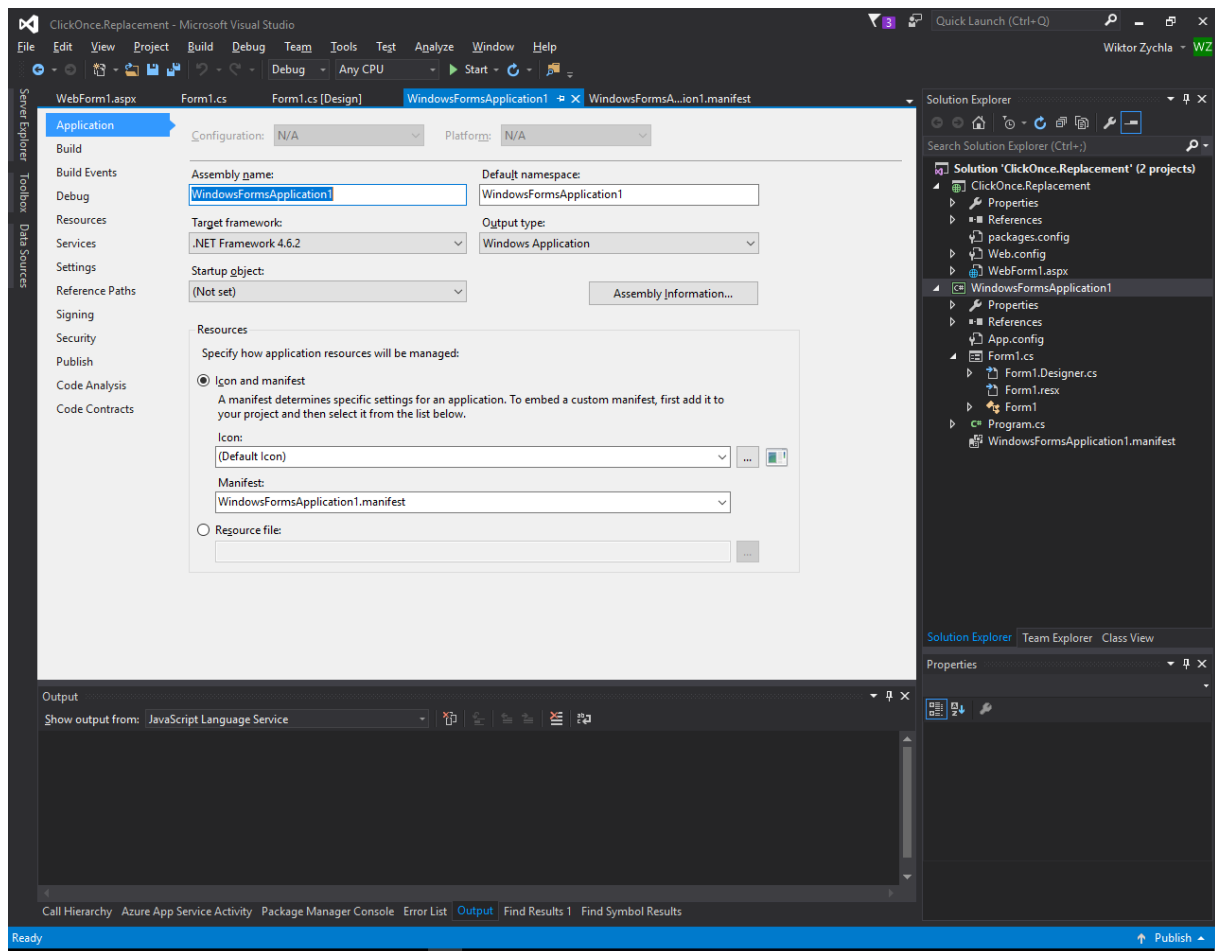
To oznacza, że w celu implementacji jednej z ważnych funkcjonalności oprogramowania przemysłowego – czyli podpisywania plików certyfikatami – należy użytkownikowi dostarczyć klasyczną aplikację desktopową.

ClickOnce daje użyteczną alternatywę, z uwagi na dobrą charakterystykę:

	Klasyczna aplikacja desktopowa	Aplikacja ClickOnce
Wytwarzanie	.NET/C#	.NET/C#
Wymagania	Wymaga .NET Framework na maszynie użytkownika	Wymaga .NET Framework na maszynie użytkownika
Dostęp do API	Dostęp do całego API .NET	Dostęp do całego API .NET w tym API niedostępne dla aplikacji przeglądarkowych, np. <ul style="list-style-type: none"><li>• dostęp do systemu plików</li><li>• dostęp do zasobnika certyfikatów</li></ul>
Instalacja	Pakiet instalacyjny, np. *.msi	Plik *.application i pliki *.deploy umieszczone na serwerze aplikacyjnym, dostępne z przeglądarki wspierającej ClickOnce (natywnie - Microsoft Edge, Microsoft Internet Explorer, pozostałe przeglądarki – pluginy)
Aktualizacja	Dodatkowy pakiet instalacyjny z aktualizacją	Automatyczna aktualizacja po umieszczeniu nowej wersji aplikacji na serwerze

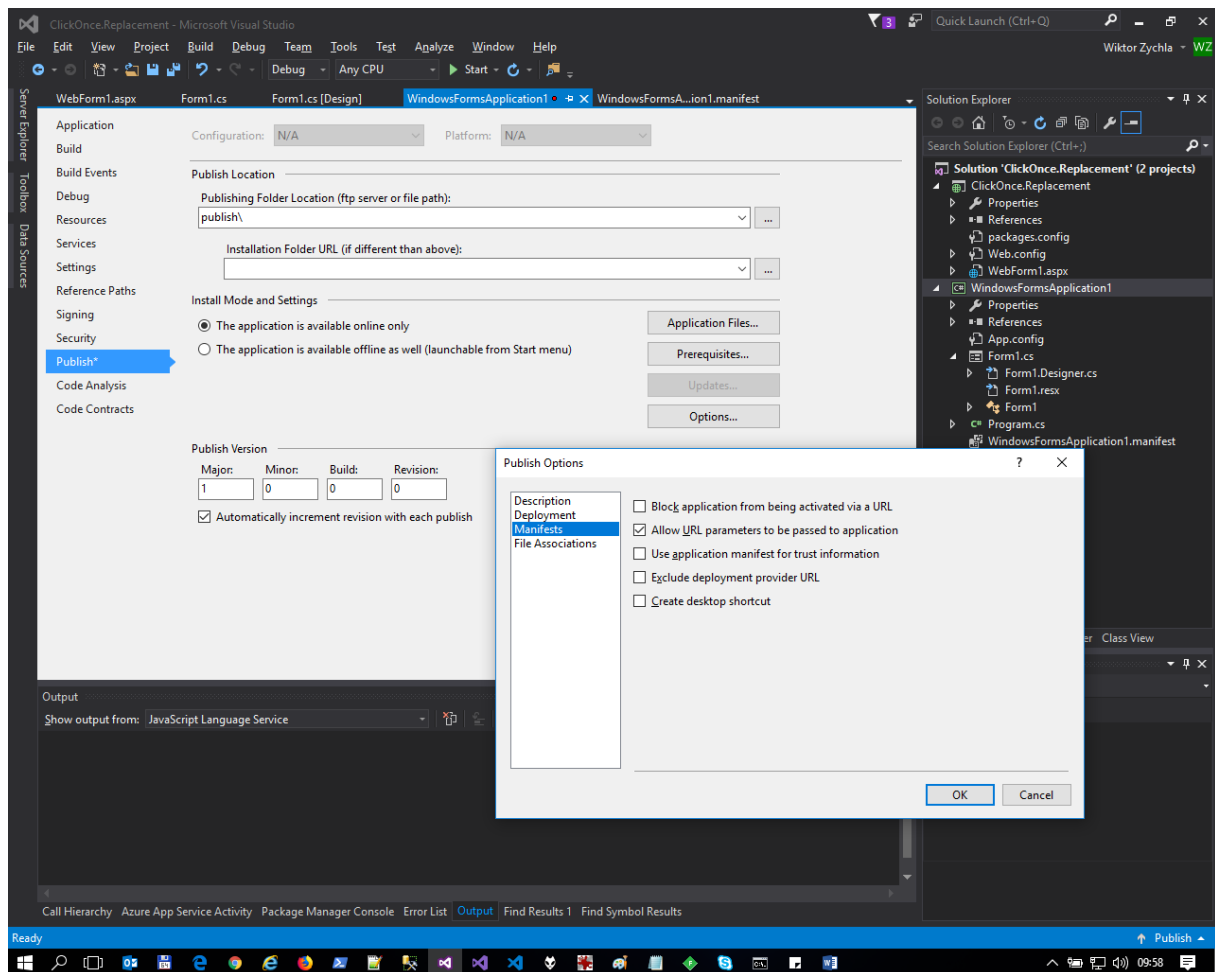
Wytworzenie aplikacji ClickOnce polega na przygotowaniu zwykłej aplikacji desktopowej i opublikowaniu jej na serwerze aplikacyjnym.

Krok 1. Na panelu właściwości projektu aplikacji desktop należy wyszukać zakładkę Publish

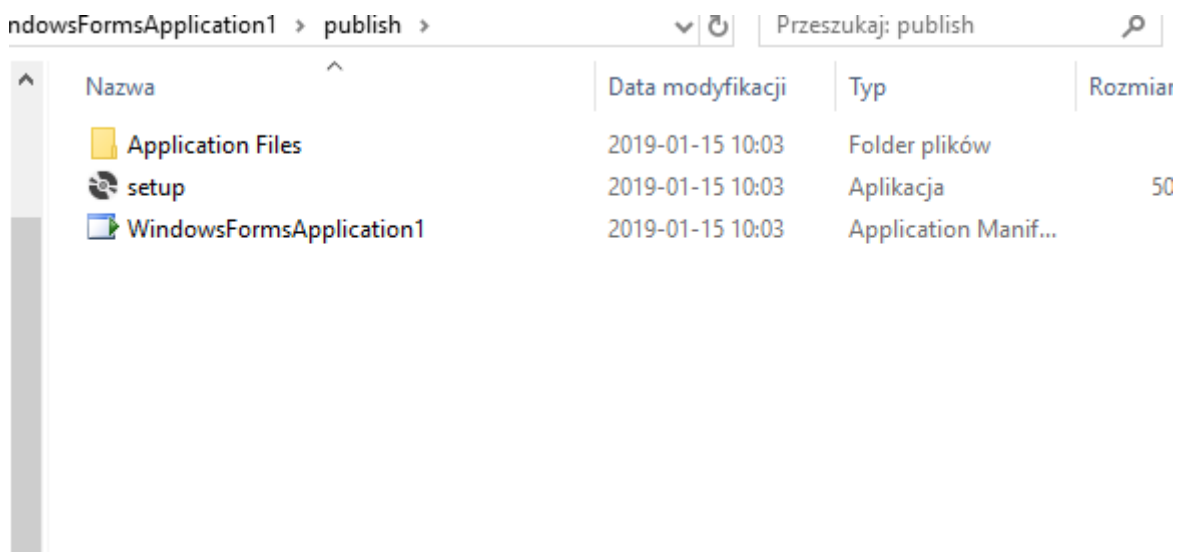


## Krok 2. Na zakładce Publish należy

- zaznaczyć opcję Application is available online only
- ustawić wersję publikowanej aplikacji (uwaga, niezależna od wersjonowania samego pliku \*.exe)
- w Options/Deployment upewnić się że wybrano „use \*.deploy file extension”
- w Options/Manifests zaznaczyć „Allow URL parameters to be passed to application”



Krok 3. Ustawić folder docelowy i opublikować aplikację do wybranego foldera aplikacji web (Publish Now). Folderem docelowym może być podfolder foldera bieżącej aplikacji lub bezpośrednio – folder aplikacji web. W tym pierwszym przypadku zawartość podfoldera należy ręcznie przekopiować do foldera aplikacji web.



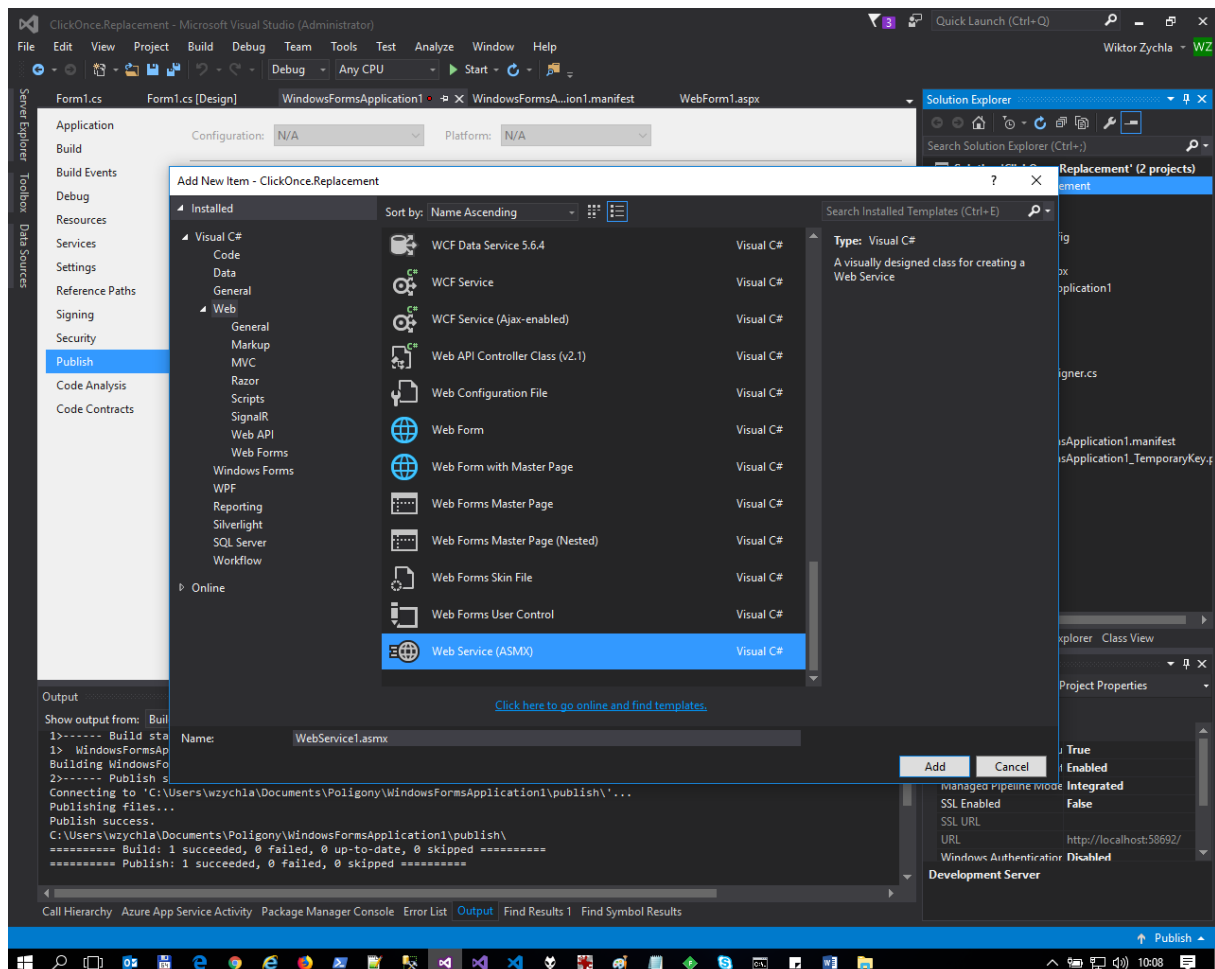
Krok 4. Aby uruchomić aplikację w przeglądarce, użytkownik aplikacji nawiguje do pliku \*.application, w powyższym przykładzie byłoby to WindowsFormsApplication1.application. Odnośnik do pliku \*.application może być umieszczony bezpośrednio na stronie web.

## 3 WebServices

### 3.1 Usługa

Aplikacja ClickOnce uruchamia się w środowisku użytkownika i ma nieograniczony dostęp do zasobów lokalnych ale w przeciwieństwie do aplikacji web – brakuje jej możliwości komunikacji z serwerem. Do tego celu należy dla aplikacji przygotować usługę aplikacyjną, najszybciej – usługę typu WebService.

Krok 1. W projekcie aplikacji Web – dodaj nowy element, w gałęzi Web, komponent ASMX WebService



Krok 2. Uzupełnienie logiki aplikacji, na przykład

```
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[System.ComponentModel.ToolboxItem(false)]
public class WebService1 : System.Web.Services.WebService
{
    [WebMethod]
    public string HelloWorld( string data )
    {
        return "Hello World " + data;
    }
}
```

Ciało klasy dziedziczącej z **WebService** reprezentuje usługę, z którą będzie się komunikować aplikacja. Ciało metody **HelloWorld** to jedna z metod udostępnianych przez usługę.

Klient usługi komunikuje się z nią przez http/https, za pomocą dialektu SOAP, który w praktyce sprowadza się do żądania typu POST którego argumentem jest odpowiednio spreparowany XML.

Usługa zawołana w trybie GET pokazuje informację o tym jak ją wołać:

The screenshot shows a web browser window displaying the WSDL documentation for a web service named 'WebService1'. The browser's address bar shows the URL 'http://localhost:58692/WebService1.asmx'. The page content includes:

- A link to 'HelloWorld'.
- A warning: 'Ta usługa sieci Web używa obszaru http://tempuri.org/ jako domyślnego obszaru nazw. Zalecenie: przed opublikowaniem tej usługi XML sieci Web zmień domyślny obszar nazw.'
- Explanatory text: 'Każda usługa XML sieci Web musi mieć unikatowy obszar nazw, aby aplikacje klienckie odróżniały ją od innych usług w sieci Web. Dla usług XML sieci Web, które są opracowywane, jest dostępny obszar http://tempuri.org/, ale opublikowane usługi XML sieci Web powinny korzystać z bardziej trwałego obszaru nazw.'
- Text: 'Usługa XML sieci Web powinna być identyfikowana przez obszar nazw kontrolowany przez użytkownika. Na przykład jako część obszaru nazw może służyć nazwa domeny internetowej firmy. Mimo że wiele obszarów nazw usług XML sieci Web przypomina adresy URL, nie muszą one wskazywać rzeczywistych zasobów w sieci Web. (Obszary nazw usług XML sieci Web są identyfikatorami URI).'
- Text: 'Domyślny obszar nazw dla usług XML sieci Web tworzonych przy użyciu architektury ASP.NET można zmienić za pomocą właściwości Namespace atrybutu WebService. WebService jest atrybutem stosowanym do klasy zawierającej metody usług XML sieci Web. Poniżej podano przykładowy kod ustawiający obszar nazw jako „http://microsoft.com/webservices/”:'
- Code snippets for C#, Visual Basic, and C++ showing how to set the namespace for a web service class.
- Links for further information: 'Namespaces in XML', 'WSDL Specification', and 'RFC 2396'.

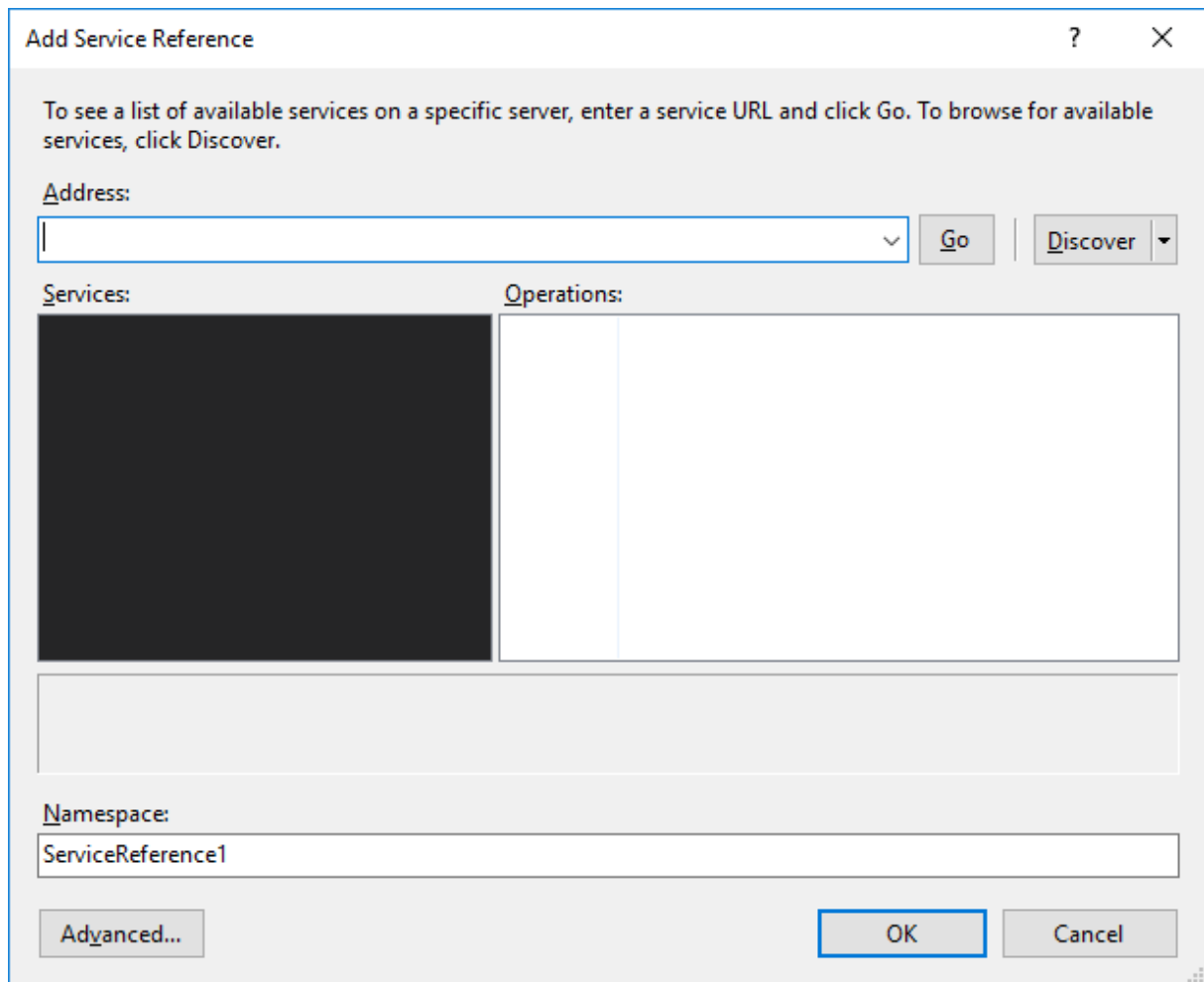
W szczególności, po wybraniu metody, można zobaczyć dokumentację standardu wywołania w dialektach SOAP 1.1, SOAP 1.2 i http POST. To oznacza, że usługa jest całkowicie interoperacyjna – klienta usługi można napisać w [dowolnej technologii wspierającej gniazda TCP](#).

### 3.2 Proxy

Jeśli klientem usługi WebService jest aplikacja .NET (a tak jest w przypadku ClickOnce!), to można zautomatyzować przygotowanie kodu usługi klienckiej typu proxy. Jest to możliwe dzięki standardowi WSDL (o tym więcej za tydzień), który dokumentuje kontrakt usługi w sposób formalny.

W praktyce należy

Krok 1. W aplikacji desktop dodać referencję do usługi WebService – prawy przycisk na „references” i „Add Service Reference”. Opcjonalnie można użyć narzędzia **wSDL.exe** lub **svcutil.exe** z linii poleceń.



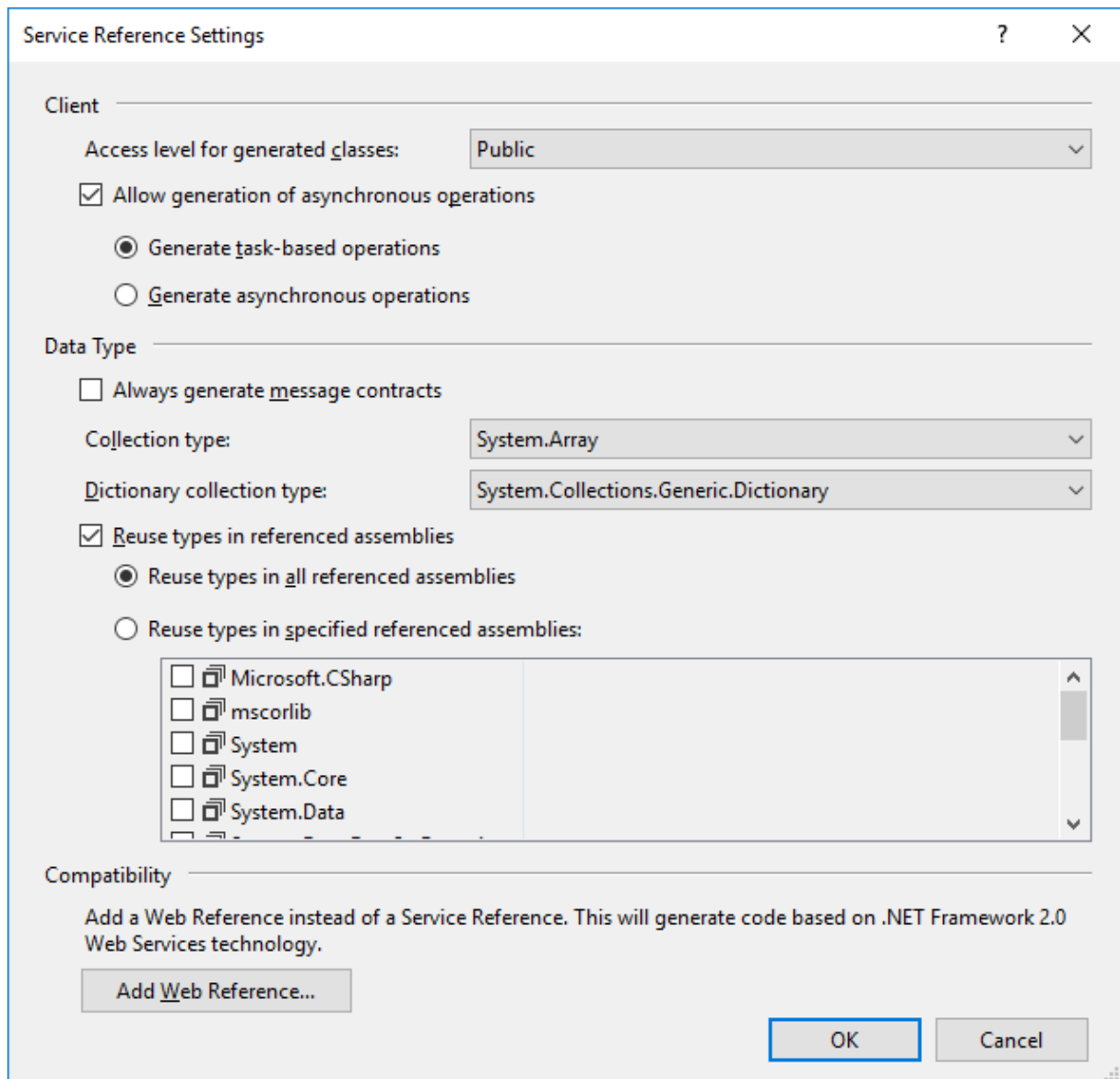
Z uwagi na wsparcie dla WCF, aktualne wersje Visual Studio pozwalają generować klasy proxy na dwa sposoby:

- Klasy proxy „starego” typu, przeznaczone pierwotnie do komunikacji z usługami WebServices – te klasy proxy dziedziczą z systemowej klasy **SoapHttpClientProtocol**
- Klasy proxy „nowego” typu przeznaczone do komunikacji z usługami WCF – te dziedziczą z **ClientBase**

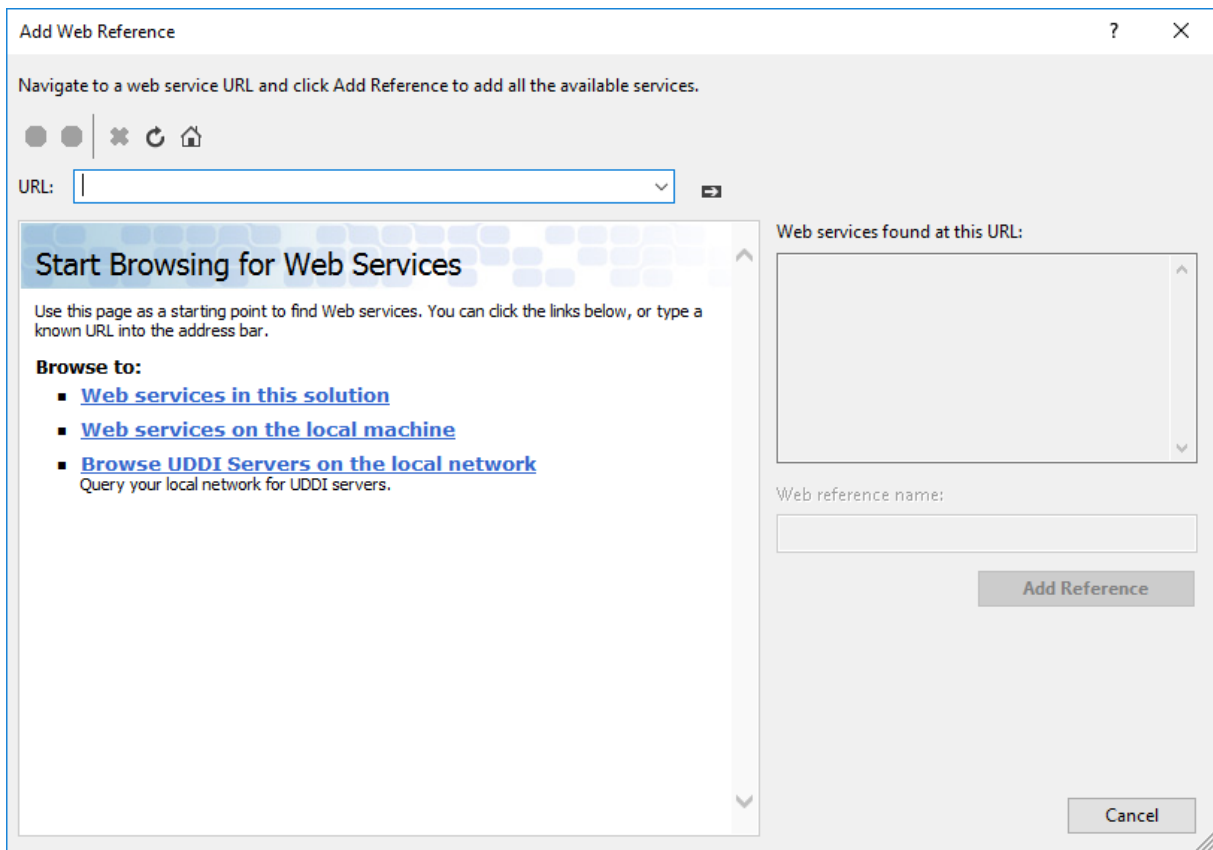
W praktyce z uwagi na ten sam kontrakt (SOAP) klas proxy można używać zamiennie – klasa proxy do usługi Webservice zadziała z usługą WCF, podobnie jak klasa proxy WCF.

Ten kreator który widać na powyższym obrazie służy do generowania proxy „nowego” typu. Aby dostać się do generatora proxy starego typu, należy kliknąć „Advanced”

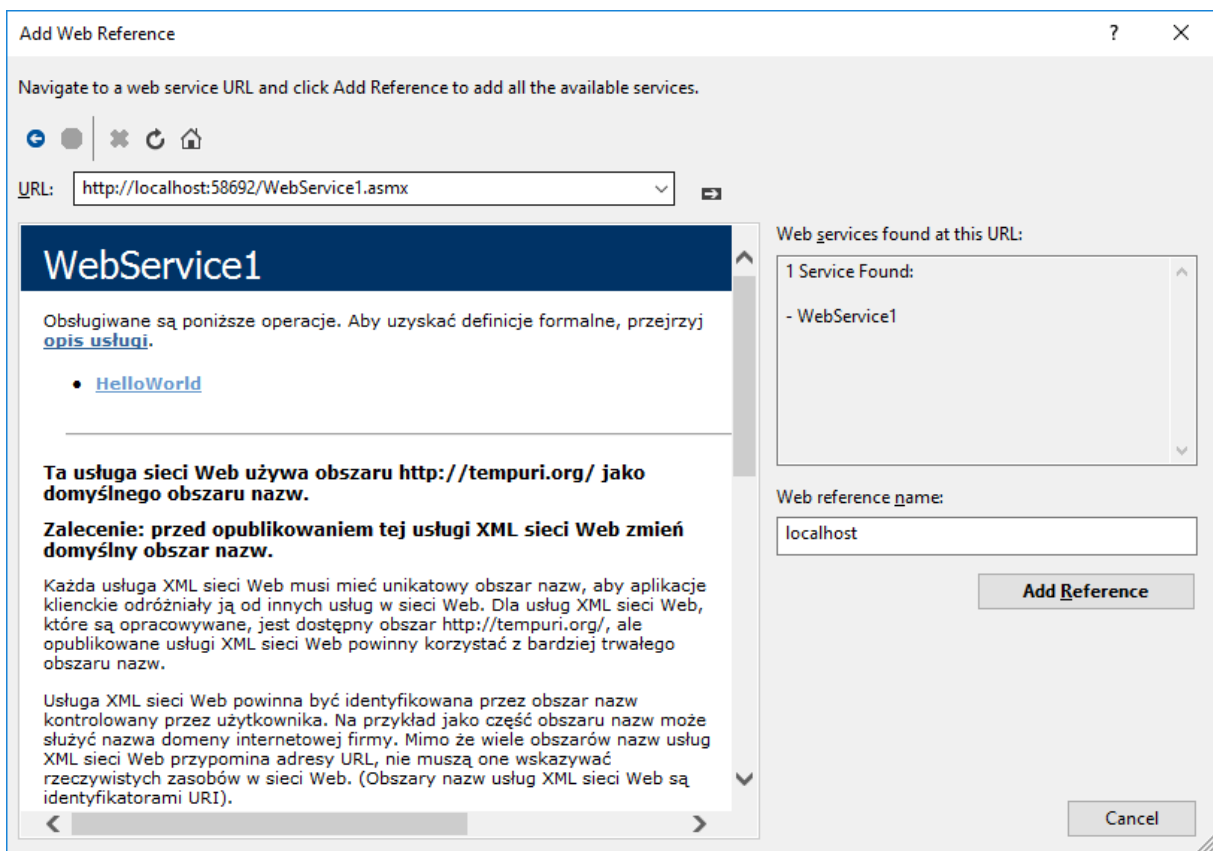




a tu „Add Web Reference”



Tu wystarczy wybrać „Web services in this solution”, wybrać usługę, wskazać dla niej alias dla proxy:



Wygenerowany kod proxy dla klienta można zobaczyć w pliku Reference.cs:

```

4 #pragma warning disable 1591
5
6 namespace WindowsFormsApplication1.localhost {
7     using System;
8     using System.Web.Services;
9     using System.Diagnostics;
10    using System.Web.Services.Protocols;
11    using System.Xml.Serialization;
12    using System.ComponentModel;
13
14    /// <remarks/>
15    [System.CodeDom.Compiler.GeneratedCodeAttribute("System.Web.Services", "4.7.3062.0")]
16    [System.Diagnostics.DebuggerStepThroughAttribute()]
17    [System.ComponentModel.DesignerCategoryAttribute("code")]
18    [System.Web.Services.WebServiceBindingAttribute(Name="WebService1Soap", Namespace="http://tempu
19    1 reference
20    public partial class WebService1 : System.Web.Services.Protocols.SoapHttpClientProtocol {
21
22        private System.Threading.SendOrPostCallback HelloWorldOperationCompleted;
23
24        private bool useDefaultCredentialsSetExplicitly;
25
26        /// <remarks/>
27        0 references
28        public WebService1() {
29            this.Url = global::WindowsFormsApplication1.Properties.Settings.Default.WindowsFormsApp
30            if ((this.IsLocalFileSystemWebService(this.Url) == true)) {
31                this.UseDefaultCredentials = true;
32                this.useDefaultCredentialsSetExplicitly = false;
33            }
34            else {
35                this.useDefaultCredentialsSetExplicitly = true;
36            }
37        }
38    }
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Wygenerowane proxy pozwala na wywołanie kodu usługi Webservice z poziomu aplikacji desktop. Należy zwrócić uwagę na to że proxy rzeczywiście odwzorowuje sygnaturę metody mimo tego, że jej wywołanie oznacza w rzeczywistości:

- Utworzenie żądania SOAP
- Wysłanie żądania w trybie POST
- Odebranie odpowiedzi SOAP
- Sparsowanie odpowiedzi

```

localhost.WebService1 proxy = new localhost.WebService1();
proxy.HelloWorld()
}
string localhost.WebService1.HelloWorld(string data)
HttpListener listener:

```

Oba sposoby generowania, stary i nowy, są dostępne z linii poleceń:

- Stary – **wsdl.exe**
- Nowy – **svcutil.exe**

### 3.3 Adres zwrotny dla usługi

Wygenerowane proxy ma jedną niefortunność – adres usługi w klasie proxy jest zapisany jako adres miejsca w którym leżała usługa z której generowano proxy. Jest to zapewne localhost na jakimś porcie deweloperskim.

Po przeniesieniu aplikacji ClickOnce na serwer, usługa proxy dalej odwołuje się do localhost i to jest błąd – użytkownik na localhost **nie ma** serwera aplikacyjnego.

Powstaje więc pytanie – skąd aplikacja ClickOnce miałaby wiedzieć z jakiego adresu jest wywoływana, tak żeby **zmienić** adres docelowy dla proxy na taki, pod jakim rzeczywiście dostępna jest usługa?

Odpowiedź na to pytanie brzmi następująco: aplikacja ClickOnce może się dowiedzieć skąd została pobrana, przez **System.Deployment.Application.ApplicationDeployment.CurrentDeployment**. Typową praktyką jest odczytanie adresu pobrania aplikacji ClickOnce a następnie zbudowanie adresu usługi **relatywnie** w stosunku do adresu pobrania:

```
// aplikacja jest pobrana stąd
var deploymentUri = System.Deployment.Application.ApplicationDeployment.CurrentDeployment.ActivationUri;

// WebService jest relatywnie w stosunku do adresu pobrania
var serviceUri = new Uri(deploymentUri, "./WebService1.asmx");

localhost.WebService1 proxy = new localhost.WebService1();
proxy.Url = serviceUri.ToString();

var result = proxy.HelloWorld("foo");
```