WCF configuration scheme

O *schema* de configuração do *Windows Communication Foundation (WCF)* inclui as seguintes partes principais:

```
<configuration>
     <system.serviceModel>

        <bindings>

        </bindings>

        <services>

        </services>

        <behaviors>

        </behaviors>

    </system.serviceModel>
</configuration>
```

http://msdn.microsoft.com/en-us/library/ms733099.aspx

WCF configuration scheme

http://msdn.microsoft.com/en-us/library/ee358768.aspx

"Configuring Windows Communication Foundation (WCF) services can be a complex task. There are many different options and it is not always easy to determine what settings are required. While configuration files increase the flexibility of WCF services, they also are the source for many hard to find problems. .NET Framework version 4 addresses these problems and provides users a way to reduce the size and complexity of service configuration. "

Simplified Configuration

In WCF configuration files, the <system.serviceModel> section contains a <service> element for each service hosted. The <service> element contains a collection of <endpoint> elements that specify the endpoints exposed for each service and optionally a set of service behaviors. The <endpoint> elements specify the address, binding, and contract exposed by the endpoint, and optionally binding configuration and endpoint behaviors. The <system.serviceModel> section also contains a <behaviors> element that allows you to specify service or endpoint behaviors.

The following example shows the <system.serviceModel> section of a configuration file.

Simplified Configuration

```
<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
       <behavior name="MyServiceBehavior">
          <serviceMetadata httpGetEnabled="true"/>
          <serviceDebug includeExceptionDetailInFaults="false" />
       </behavior>
    </serviceBehaviors>
  </behaviors>
  <bindings>
    <basicHttpBinding>
      <binding name="MyBindingConfig" maxBufferSize="100"
                                      maxReceiveBufferSize="100" />
    </basicHttpBinding>
  </bindings>
  <services>
    <service behaviorConfiguration="MyServiceBehavior" name="MyService">
      <endpoint address="" binding="basicHttpBinding“ contract="ICalculator“
                bindingConfiguration="MyBindingConfig" />
      <endpoint address="mex" binding="mexHttpBinding"
                contract="IMetadataExchange"/>
    </service>
  </services>
  </system.serviceModel>
```

Endpoint de exportação da *metadata* do serviço

Simplified Configuration – Default Endpoint/Binding

- ✓ Cada vez que pretendemos criar um serviço WCF, precisamos de definir o *endpoint* e o *binding*. O WCF 4.0 introduziu o conceito de *Default Endpoint* que evita estar repetidamente a definir os *settings* comuns do *endpoint* / *binding*.
- ✓ O *default endpoint* escolhe o *binding* apropriado baseado no formato do URL do endereço base, de acordo com uma tabela predefinida no ficheiro de configuração do .NET 4 da máquina (*Machine.config.comments*).

```
<protocolMapping>
  <add scheme="http" binding="basicHttpBinding" bindingConfiguration="" />
  <add scheme="net.tcp" binding="netTcpBinding" bindingConfiguration="" />
  <add scheme="net.pipe" binding="netNamedPipeBinding" bindingConfiguration="" />
  <add scheme="net.msmq" binding="netMsmqBinding" bindingConfiguration="" />
</protocolMapping>
```

- ✓ Por exemplo se definirmos

```
<service name="DefaultEndPoint.HelloService“
         behaviorConfiguration="DefaultEndPoint.HelloServiceBehavior" >
  <host>
    <baseAddresses>
      <add baseAddress="http://localhost:8732/HelloService/" />
    </baseAddresses>
  </host>
</service>
```

Exemplo: *DefaultEndPoint.zip*

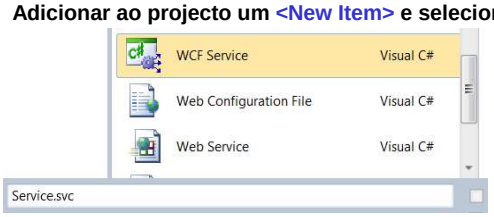
- ✓ O binding será `basicHttpBinding`

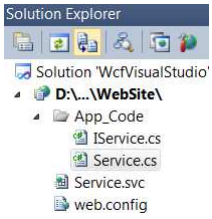
Criação de um serviço no Visual Studio 2010

Exemplo: *WcfVisualStudio.zip*

1. **Criar uma solução vazia no Visual Studio;**
2. **Adicionar um <New Web Site> e selecionar <ASP.NET Empty Web Site>, não esquecendo de definir a localização do site no *File System***

3. **Adicionar ao projecto um <New Item> e selecionar <WCF Service>**





4. **Será gerado um Serviço com:**
   1. **o contrato *Iservice.cs***
   2. **a classe para implementar o serviço *Service.cs***

Criação de um serviço no Visual Studio 2010

```
[ServiceContract]
public interface IService
{
  [OperationContract]
  void DoWork();
}
```

Contrato

```
public class Service : IService
{
  public void DoWork()
  {
  }
}
```

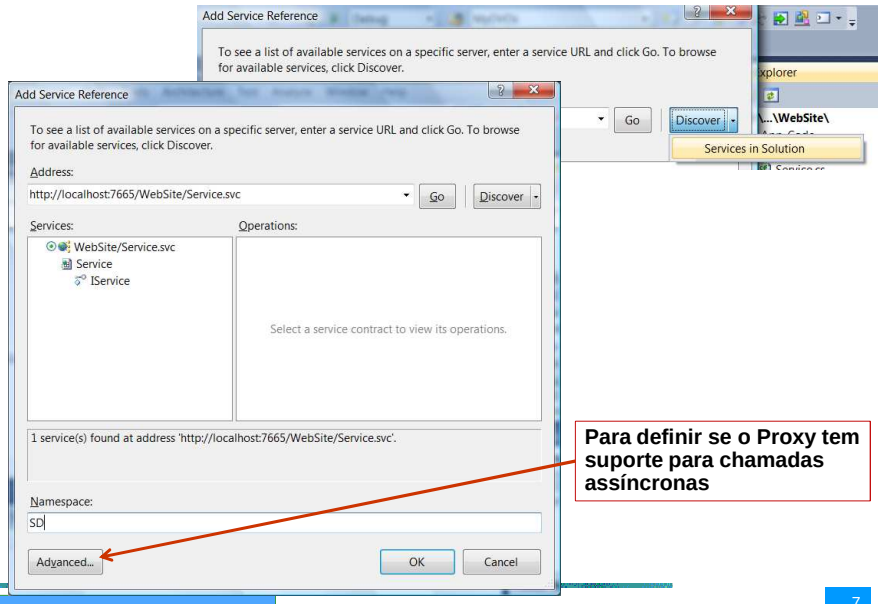
Serviço

web.config

```
<configuration>
    <system.web>
        <compilation debug="false" targetFramework="4.0" />
    </system.web>
    <system.serviceModel>
        <behaviors>
            <serviceBehaviors>
                <behavior name="">
                    <serviceMetadata httpGetEnabled="true" />
                    <serviceDebug includeExceptionDetailInFaults="false" />
                </behavior>
            </serviceBehaviors>
        </behaviors>
        <serviceHostingEnvironment multipleSiteBindingsEnabled="true" />
    </system.serviceModel>
</configuration>
```

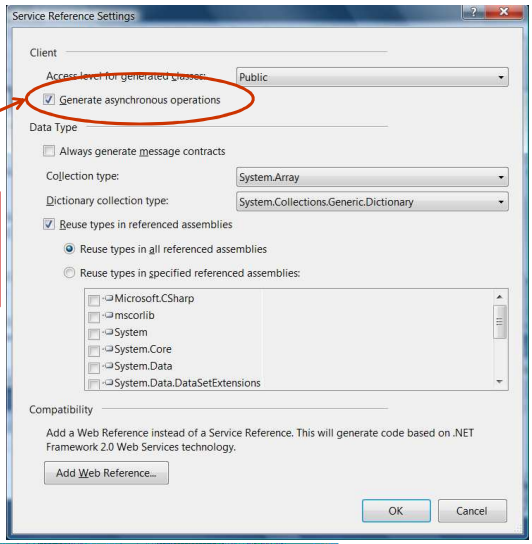
Note que, como o serviço fica alojado no servidor http do Visual Studio, então o endereço base define o *binding* por omissão `basicHttpBinding`

Adicionar uma referência para um serviço no Visual Studio 2010



Para definir se o Proxy tem suporte para chamadas assíncronas

Gerar proxy com suporte para chamadas assíncronas



Para que o *proxy* gerado em <Add Service Reference> tenha suporte para chamadas assíncronas

Configuração do cliente (gerado automaticamente)

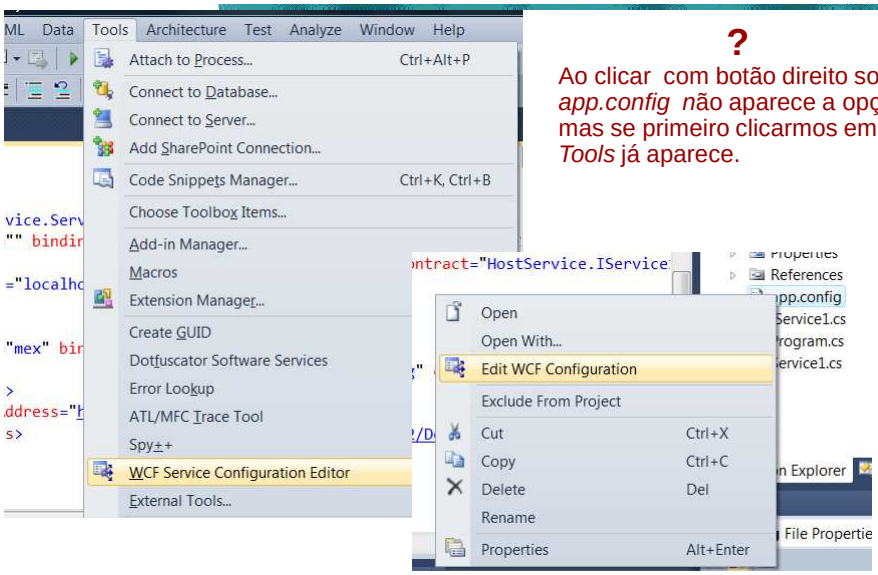
```
<configuration>
   <system.serviceModel>
     <bindings>
         <basicHttpBinding>
            <binding name="BasicHttpBinding_IService" closeTimeout="00:01:00"
               openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00"
               allowCookies="false" bypassProxyOnLocal="false"
               hostNameComparisonMode="StrongWildcard" maxBufferSize="65536"
               maxBufferPoolSize="524288" maxReceivedMessageSize="65536"
                messageEncoding="Text" textEncoding="utf-8" transferMode="Buffered"
                useDefaultWebProxy="true">
                <readerQuotas maxDepth="32" maxStringContentLength="8192"
                                 maxArrayLength="16384" maxBytesPerRead="4096"
                                                  maxNameTableCharCount="16384" />
                <security mode="None">
                    <transport clientCredentialType="None" proxyCredentialType="None"
                            realm="" />
                    <message clientCredentialType="UserName" algorithmSuite="Default" />
                </security>
            </binding>
         </basicHttpBinding>
     </bindings>
     <client>
       <endpoint address="http://localhost:7665/WebSite/Service.svc"
            binding="basicHttpBinding" bindingConfiguration="BasicHttpBinding_IService"
            contract="SD.IService" name="BasicHttpBinding_IService" />
     </client>
   </system.serviceModel>
</configuration>
```


Adicionar um *WCF Service* a qualquer aplicação

Se adicionarmos um WCF Service numa aplicação qualquer (neste caso aplicação consola *HostService*) o Visual Studio adiciona as seguintes configurações:

```
<configuration>
  <system.serviceModel>
    <behaviors>
            <serviceBehaviors>
                <behavior name="">
                    <serviceMetadata httpGetEnabled="true" />
                    <serviceDebug includeExceptionDetailInFaults="false" />
                </behavior>
            </serviceBehaviors>
    </behaviors>
    <services>
      <service name="HostService.Service1">
        <endpoint address="" binding="wsHttpBinding" contract="HostService.IService1">
                    <identity>
                        <dns value="localhost" />
                    </identity>
        </endpoint>
        <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
          <host>
            <baseAddresses>
              <add
       baseAddress="http://localhost:8732/Design_Time_Addresses/HostService/Service1/" />
            </baseAddresses>
          </host>
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

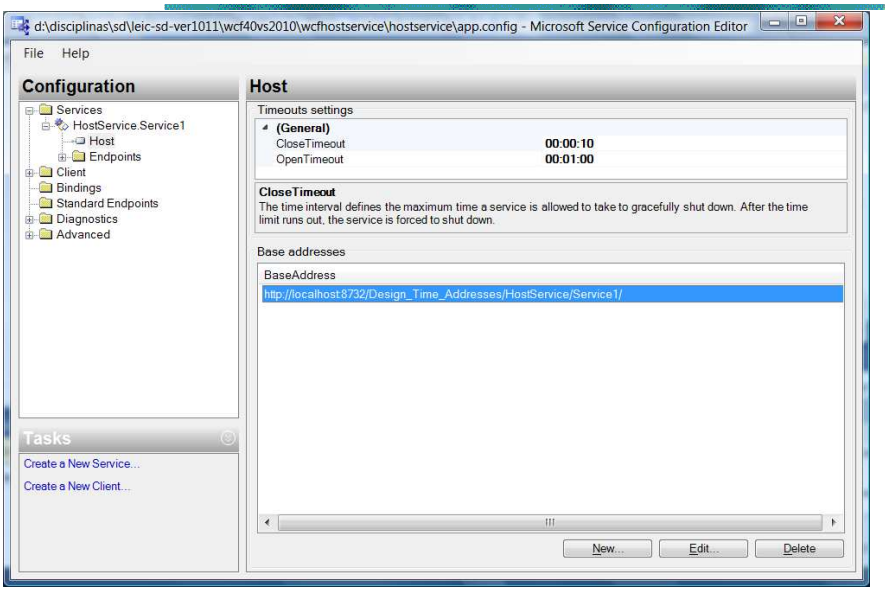
Alterar configurações com a *tool WCF Service Configuration Editor*



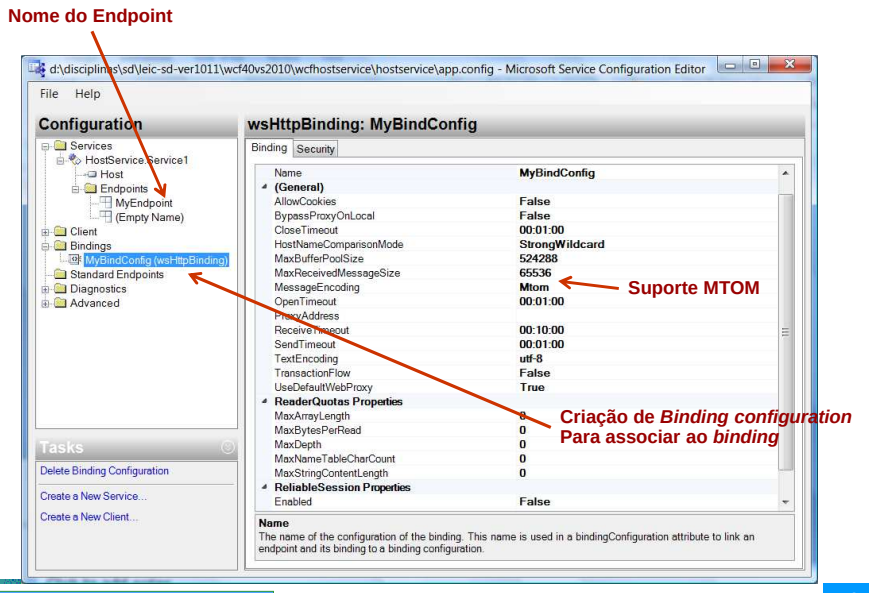
?

Ao clicar com botão direito sobre *app.config* não aparece a opção, mas se primeiro clicarmos em *Tools* já aparece.

Alterar configurações com a *tool WCF Service Configuration Editor*



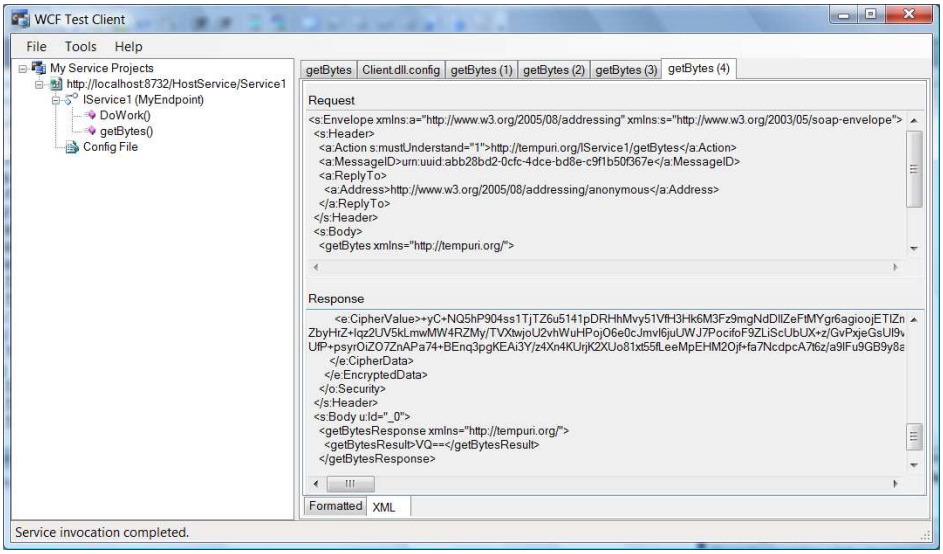
Alterar configurações com a *tool WCF Service Configuration Editor*



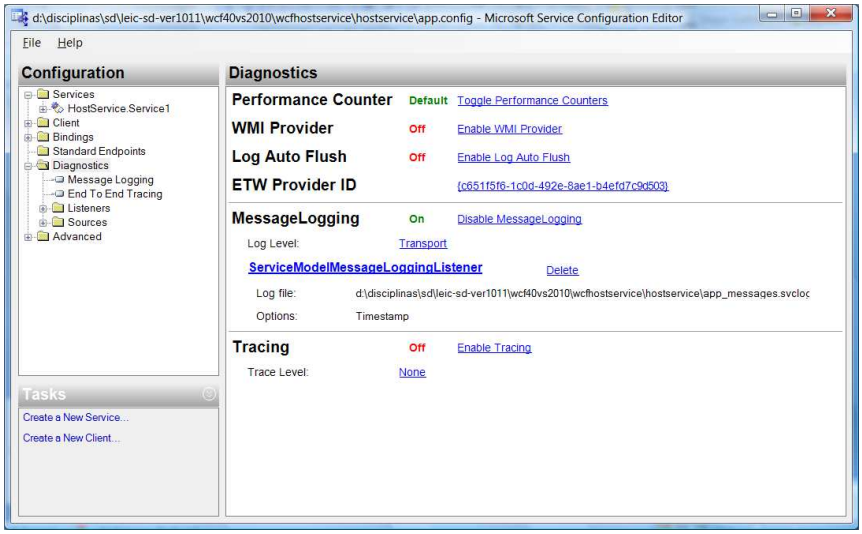
Configurações após alterações

```
<system.serviceModel>
    <bindings>
            <wsHttpBinding>
                <binding name="MyBindConfig" messageEncoding="Mtom" />
            </wsHttpBinding>
    </bindings>
    <behaviors>
       <serviceBehaviors>
                <behavior name="">
                    <serviceMetadata httpGetEnabled="true" />
                    <serviceDebug includeExceptionDetailInFaults="false" />
                </behavior>
            </serviceBehaviors>
    </behaviors>
    <services>
       <service name="HostService.Service1">
         <endpoint address="" binding="wsHttpBinding" bindingConfiguration="MyBindConfig"
                name="MyEndpoint" contract="HostService.IService1">
                <identity>     <dns value="localhost" />   </identity>
         </endpoint>
         <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
          <host>
            <baseAddresses>
                  <add baseAddress="http://localhost:8732/HostService/Service1/" />
            </baseAddresses>
          </host>
       </service>
    </services>
 </system.serviceModel>
```

Tool WCF Test Client – wcfTestClient.exe



Configuração para *Message Logging* num ficheiro



Duplex Binding

Exemplo: *DuplexBinding.zip*

```
namespace HostDuplexService
{
    [ServiceContract(SessionMode=SessionMode.Required,
                     CallbackContract=typeof(IClientReceiver))]
    public interface IDuplexService
    {
        [OperationContract(IsInitiating=true)]
        string init(); // inicia session no servidor
        [OperationContract(IsTerminating=true)]
        void exit(); // termina session no servidor
        [OperationContract]
        void SendMsg(string msg);
    }

    // CallBack interface in client
    public interface IClientReceiver
    {
        [OperationContract(IsOneWay = true)]
        void newAnnounce(string msg);
    }
}
```


Duplex Binding Service

```
namespace HostDuplexService
{
  [ServiceBehavior(InstanceContextMode=InstanceContextMode.Single)]
  public class DuplexService : IDuplexService
  {
     public string init()
     {
           return "Duplex Service is alive";
     }

    public void SendMsg(string msg)
    {
       string sessionID = OperationContext.Current.SessionId;
       IClientReceiver cli =
             OperationContext.Current.GetCallbackChannel<IClientReceiver>();
       cli.newAnnounce("New announcement from:" + sessionID + ":" + msg);
    }

    public void exit()
    {

    }
  }
}
```

A partir de **OperationContext.Current** podemos obter informação sobre o contexto de chamada do serviço.

Duplex Binding Service - Config

```
<system.serviceModel>
        <behaviors>
            <serviceBehaviors>
                <behavior name="HostDuplexService.DuplexBehavior">
                    <serviceMetadata httpGetEnabled="true" />
                    <serviceDebug includeExceptionDetailInFaults="false" />
                </behavior>
            </serviceBehaviors>
        </behaviors>
        <services>
            <service behaviorConfiguration="HostDuplexService.DuplexBehavior"
                name="HostDuplexService.DuplexService">
                <endpoint address="" binding="wsDualHttpBinding" bindingConfiguration="DuplexBinding"
                    contract="HostDuplexService.IDuplexService">
                    <identity>
                        <dns value="localhost" />
                    </identity>
                </endpoint>
                <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
                <host>
                    <baseAddresses>
                        <add baseAddress="http://localhost:8732/HostDuplexService/DuplexService/" />
                    </baseAddresses>
                </host>
            </service>
        </services>
      <bindings>
        <wsDualHttpBinding>
           <binding name="DuplexBinding">
              <security mode="None" />
           </binding>
        </wsDualHttpBinding>
      </bindings>
    </system.serviceModel>
```


Cliente do Duplex Binding Service

```
namespace ConsClient
{
    [CallbackBehavior(ConcurrencyMode = ConcurrencyMode.Multiple)]
    public class Receiver : IDuplexServiceCallback
    {
        public void newAnnounce(string msg)
        {
            Console.WriteLine(msg);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Receiver rec = new Receiver(); // cria um objecto para receber callbacks
            DuplexServiceClient svc = new DuplexServiceClient(new InstanceContext(rec));
            Console.WriteLine(svc.init());
            for (; ; )
            {
                Console.Write("msg to send?");
                string msg = Console.ReadLine();
                if (string.Compare(msg, "exit") == 0) break;
                svc.SendMsg(msg);
            }
            svc.exit();
        }
    }
}
```

Por omissão, o nome da interface de *Callback* é gerada no *proxy* com o nome da interface do serviço mais o sufixo "*Callback*"

Ao instanciar o proxy para o serviço cria um contexto com o objecto que recebe os callbacks

Exemplo: *Binding Context*

- O WCF tem 3 *bindings* que possibilitam a existência de contextos adicionais (*custom contexts*): BasicHttpContextBinding, NetTcpContextBinding e WSHttpContextBinding que derivam dos respectivos *binding* base.
- O contexto é um dicionário de strings com pares *<Key, Value>* passados implicitamente como *header* das mensagens SOAP;
- O contexto só pode ser afectado pelo cliente antes da usar o *proxy* para o serviço a primeira vez. Depois disso o contexto é *cached* e qualquer tentativa para o modificar resulta numa excepção.

Exemplo: *BindingContexts.zip*

Cliente que define um contexto

```
static void Main(string[] args)
{
  ServiceCtxClient prx = new ServiceCtxClient();
  IContextManager ctxManager = prx.InnerChannel.GetProperty<IContextManager>();
  IDictionary<string, string> context = ctxManager.GetContext();

  context["clientName"] = "luis";
  context["AccessID"] = "QWERTY";
  ctxManager.SetContext(context);

  Console.WriteLine( prx.DoWork() ); // invoca o serviço

       // Não é possível mudar o contexto após o channel ter sido aberto
       //context["clientName"] = "xx";
       //context["AccessID"] = "XXXXX";
       //ctxManager.SetContext(context);

        Console.ReadLine();
}
```

```
[ServiceContract]
public interface IServiceCtx
{
    [OperationContract]
    string DoWork();
}
```

Mensagem SOAP com passagem do contexto

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
            xmlns:a="http://www.w3.org/2005/08/addressing">
  <s:Header>
     <a:Action s:mustUnderstand="1">
           http://tempuri.org/IServiceCtx/DoWork
     </a:Action>
     <a:MessageID>urn:uuid:2985c05b-b03b-45dc-8f58-f82c27e7cb84</a:MessageID>
     <a:ReplyTo>
        <a:Address>http://www.w3.org/2005/08/addressing/anonymous</a:Address>
     </a:ReplyTo>
     <Context xmlns="http://schemas.microsoft.com/ws/2006/05/context">
          <Property name="clientName">luis</Property>
          <Property name="AccessID">QWERTY</Property>
     </Context>
     <a:To s:mustUnderstand="1">
           http://localhost:8732/HostService/ServiceCtx/
     </a:To>
  </s:Header>
  <s:Body>
      <DoWork xmlns="http://tempuri.org/">
      </DoWork>
  </s:Body>
</s:Envelope>
```


Serviço acede ao contexto

```
public class ServiceCtx : IServiceCtx
{
  public string DoWork() // Service Operation
  {
     string clientName=null; string accessID=null;

      MessageProperties msgProp=
                          OperationContext.Current.IncomingMessageProperties;
      ContextMessageProperty ctxProperty=
              msgProp[ContextMessageProperty.Name] as ContextMessageProperty;

      if (ctxProperty.Context.ContainsKey("clientName"))
      {
          clientName=ctxProperty.Context["clientName"];
          Console.WriteLine("client name: " + clientName);
      }
      if (ctxProperty.Context.ContainsKey("AccessID"))
      {
           accessID = ctxProperty.Context["AccessID"];
           Console.WriteLine("  Access ID:" + accessID);
      }
      return clientName + ":" + accessID;
  }
}
```

```
[ServiceContract]
public interface IServiceCtx
{
    [OperationContract]
    string DoWork();
}
```