

در اکثر برنامه های سازمانی، مثل برنامه های مدیریت آرشیو اسناد، همواره این نیاز جزو خواسته های کاربران بوده که بتوانند به صورت مستقیم و از طریق تنها یک کلیک، تصویر مورد نظر را اسکن کرده و به صورت خودکار در برنامه وارد کنند؛ یعنی بدون اینکه نیاز باشد با استفاده از یک برنامه دیگر ابتدا تصویر را اسکن کرده و سپس در فرم وب، فایل اسکن شده را Browse کنند. این نیاز اساساً به معنی دسترسی به سخت افزار کاربر از طریق مرورگر می باشد که به دلایل متعددی امکان پذیر نیست! مشکلات امنیتی ایجاد شده در این راه بسیار جدی است. اما خوشبختانه راه هایی برای رسیدن به این هدف وجود دارند:

1- **ActiveX**: که به صورت native فقط در IE پشتیبانی می شود (در نسخه های جدیدتر IE نیاز به [بروز رسانی پلاگین ActiveX controls](#) می باشد) و برای استفاده از آن در مرورگرهای Firefox و Chrome هم باید از پلاگین های جانبی روی هر مرورگر استفاده کرد که مثلاً برای اجرای بر روی Firefox، باید افزونه [IE Tab](#) را نصب کرد که تنها کارش این است که سایت را از طریق موتور IE در پنجره ی فایرفاکس اجرا کند! که البته این مورد مثل این می ماند که سایت در IE باز شده باشد که ممکن است زیاد خوشایند نباشد؛ در غیر این صورت باید پروژه را از اول بر مبنای اجرای بر روی IE طراحی و پیاده سازی کرد. در ضمن از مشکلات اجرای پلاگین نوشته شده توسط برنامه نویسی در IE و نسخه های مختلف آن هم چشم پوشی می کنیم. یکی از مزیت های این روش نسبت به بقیه این هست که دانلود و نصب پلاگین مورد نیاز به صورت خودکار و توسط مرورگر انجام می شود.

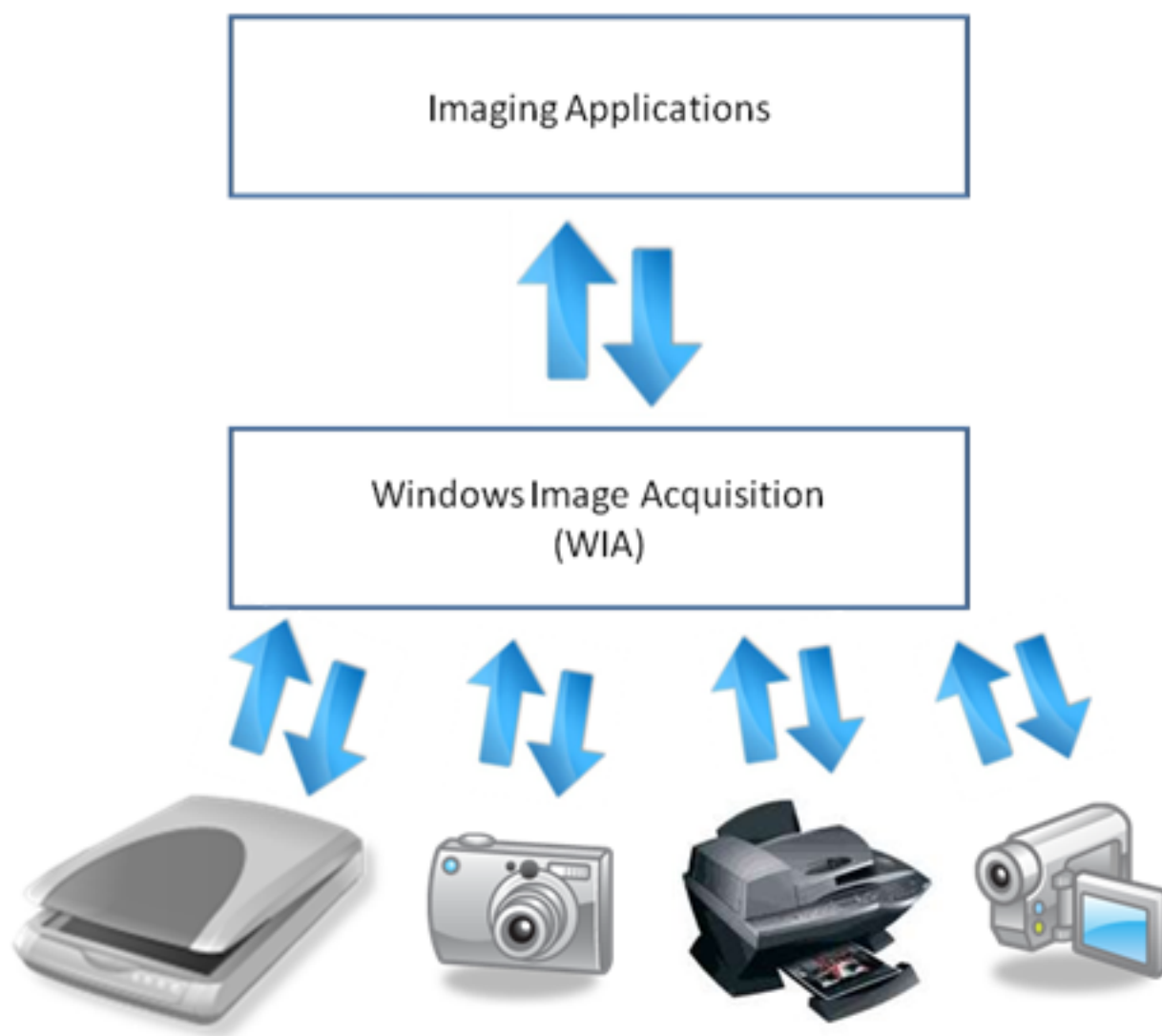
2- **استفاده از یک کامپوننت جانبی**: مثل کامپوننت های [LEADTOOLS](#) که ابزارهای متنوع و SDK های بسیار قدرتمندی را برای اینکار و کارهای دیگر، مانند کار با اسکن تصاویر مغزی، پردازش تصویر، بارکد خوان ها، مدیریت اسناد و ... فراهم کرده است. قیمت این ابزار بسیار زیاد است و در برخی موارد امکانات فراهم آورده بسیار بیشتر است از خواسته ی ما. این ابزار، هم در HTML & Javascript و هم در DotNet قابل استفاده است و مستندات نسبتاً خوبی هم در این زمینه ارائه کرده است. همچنین کامپوننت [Dynamsoft](#) که باز هم غیر رایگان هست و قیمت بالایی نیز دارد.

اگر روال کار کامپوننت های بالا را مورد بررسی قرار دهید (از طریق اجرای Demo ها، [اینجا](#) و [اینجا](#)) متوجه خواهید شد که هر دو نیاز به نصب یک سرویس یا App سمت کلاینت جهت اجرای دستورات خود دارند. پس می شود اینطور نتیجه گرفت که انجام اینکار بدون اینکه چیزی سمت کاربر نصب شود، ممکن نیست و در هر دو، لینک نصب فایل exe سرویس برای دانلود قرار داده شده است. بر این اساس به راه حل سومی خواهیم رسید که خودمان این سرویس را جهت تعامل با اسکنر سمت کاربر طراحی و پیاده سازی نماییم.

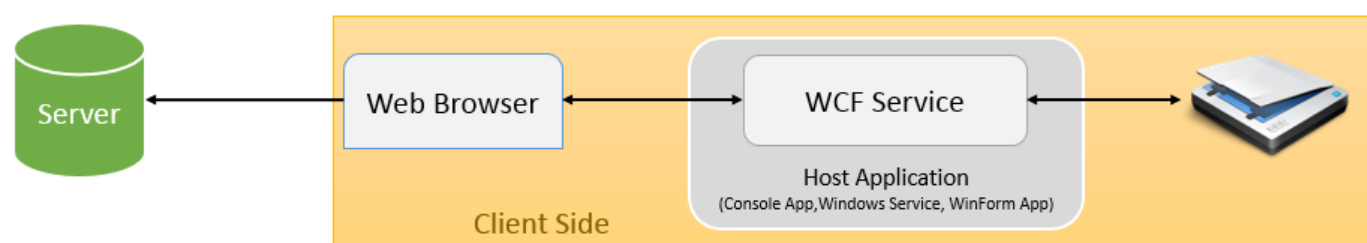
اما چطور ممکن است که با اجرای یک فایل exe سمت کاربر (با این فرض که کاربر در یک دامین مطمئن قرار دارد و می شود درخواست نصب سرویس را نمود) این امکان را برای کاربر فراهم نمود که با یک کلیک در مرورگر، اسکنر به صورت خودکار اسکن را آغاز کرده و سپس تصویر حاصل را به یکی از کنترلرهای ما در سمت سرور ارسال نماید؟

برای اینکار ما با دو صورت مساله مواجه هستیم؛ اول اینکه چطور تصویر را سمت کاربر اسکن کنیم و دوم اینکه چطور این تصویر را به سرور ارسال نماییم!

برای مساله ی اول از کتابخانه [\(Windows Image Acquisition \(WIA](#) استفاده خواهیم نمود که این کتابخانه به ما این امکان را میدهد تا با سخت افزارهایی که از [TWAIN](#) پشتیبانی می کنند، بتوانیم ارتباط برقرار نماییم.



برای مسالهای دوم هم نیاز به پیاده سازی یک WCF Service و اجرای آن (هاست کردن) در سمت کلاینت داریم. در واقع با صدا زدن متدهای این سرویس، از کتابخانه‌ی بالا استفاده کرده و اسکن را انجام می‌دهیم.



ادامه دارد...

در قسمت قبل « [کار با اسکنر در برنامه‌های تحت وب \(قسمت اول\)](#) » دیدی از کاری که قرار است انجام دهیم، رسیدیم. حالا سراغ یک پروژه‌ی عملی و پیاده‌سازی مطالب مطرح شده می‌رویم.

ابتدا پروژه‌ی WCF را شروع می‌کنیم. ویژوال استودیو را باز کرده و از قسمت WCF > Visual C# > New Project یک پروژه‌ی WCF Service Application جدید را مثلاً با نام "WcfServiceScanner" ایجاد نمایید. پس از ایجاد، دو فایل IService1.cs و ScannerService موجود را به IScannerService تغییر نام دهید. سپس ابتدا محتویات کلاس اینترفیس IScannerService را به صورت زیر تعریف نمایید :

```
[ServiceContract]
public interface IScannerService
{
    [OperationContract]
    [WebInvoke(Method = "GET",
        BodyStyle = WebMessageBodyStyle.Wrapped,
        RequestFormat = WebMessageFormat.Json,
        ResponseFormat = WebMessageFormat.Json,
        UriTemplate = "GetScan")]
    string GetScan();
}
```

در اینجا ما فقط اعلان متدهای مورد نیاز خود را ایجاد کرده‌ایم. علت استفاده از Attribute ایی با نام [WebInvoke](#) ، مشخص نمودن نوع خروجی به صورت Json است و همچنین عنوان آدرس مناسبی برای صدا زدن متد. پس از آن کلاس ScannerService را مطابق کدهای زیر تغییر دهید:

```
public class ScannerService : IScannerService
{
    public string GetScan()
    {
        // TODO Add code here
    }
}
```

تا اینجا فقط یک WCF Service معمولی ساخته‌ایم. در ادامه به سراغ کلاس WIA برای ارتباط با اسکنر می‌رویم. بر روی پروژه‌ی خود راست کلیک کرده و Add Reference را انتخاب نموده و سپس در قسمت COM، گزینه‌ی Microsoft Windows Image Acquisition Library v2.0 را به پروژه‌ی خود اضافه نمایید. با اضافه شدن این ارجاع به پروژه، دسترسی به فضای نام WIA برای ما امکان پذیر می‌شود که ارجاعی از آن را در کلاس ScannerService قرار می‌دهیم.

```
using WIA;
```

اکنون متد GetScan را مطابق زیر اصلاح می‌نماییم:

```
public string GetScan()
{
    var imgResult = String.Empty;
    var dialog = new CommonDialogClass();
    try
    {
        // نمایش فرم پیشفرض اسکنر
        var image = dialog.ShowAcquireImage(WiaDeviceType.ScannerDeviceType);

        // ذخیره تصویر در یک فایل موقت
        var filename = Path.GetTempFileName();
    }
}
```

```
image.SaveFile(filename);
var img = Image.FromFile(filename);

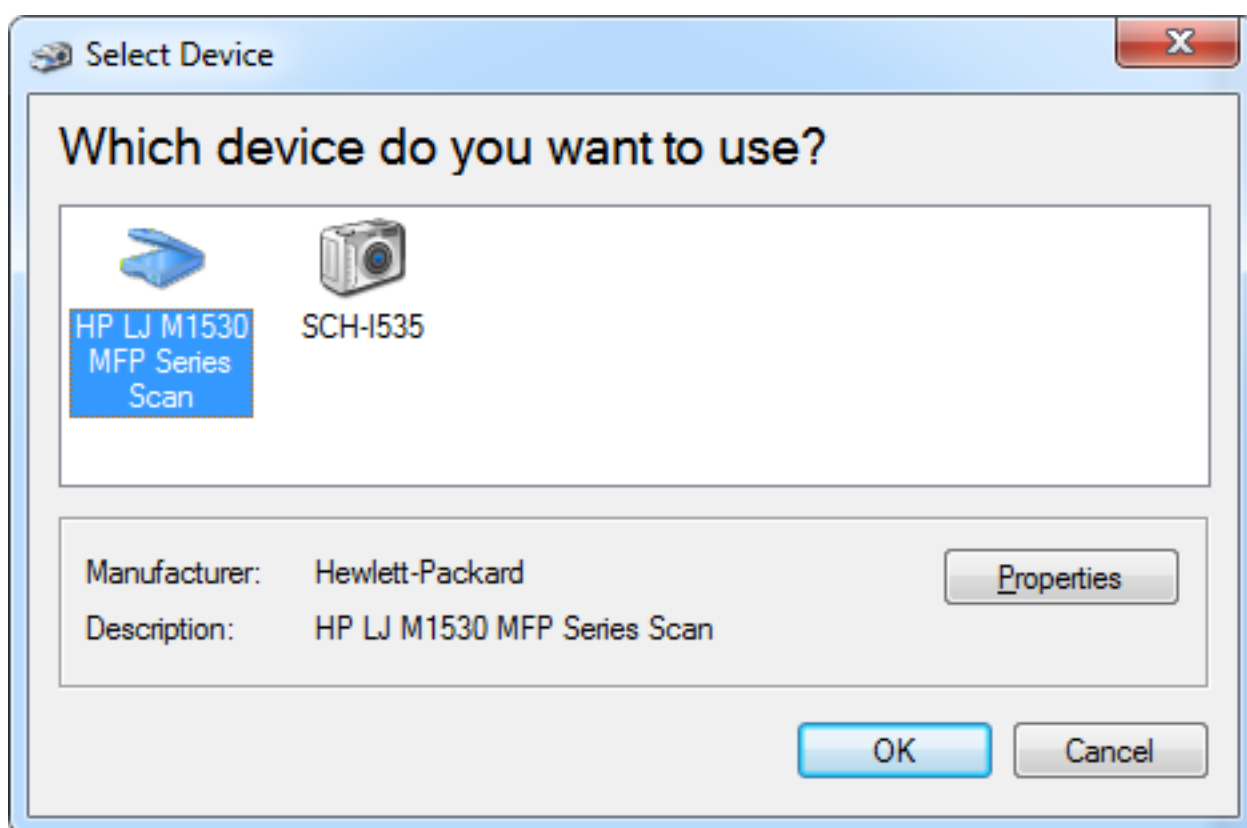
// img تبدیل تصویر به Base64 جهت ارسال سمت کاربر و نمایش در تگ
imgResult = ImageHelper.ImageToBase64(img, ImageFormat.Jpeg);
}
catch
{
    // از آنجایی که امکان نمایش خطا وجود ندارد در صورت بروز خطا رشته خالی
    // بازگردانده می شود که به معنای نبود تصویر می باشد
}

return imgResult;
}
```

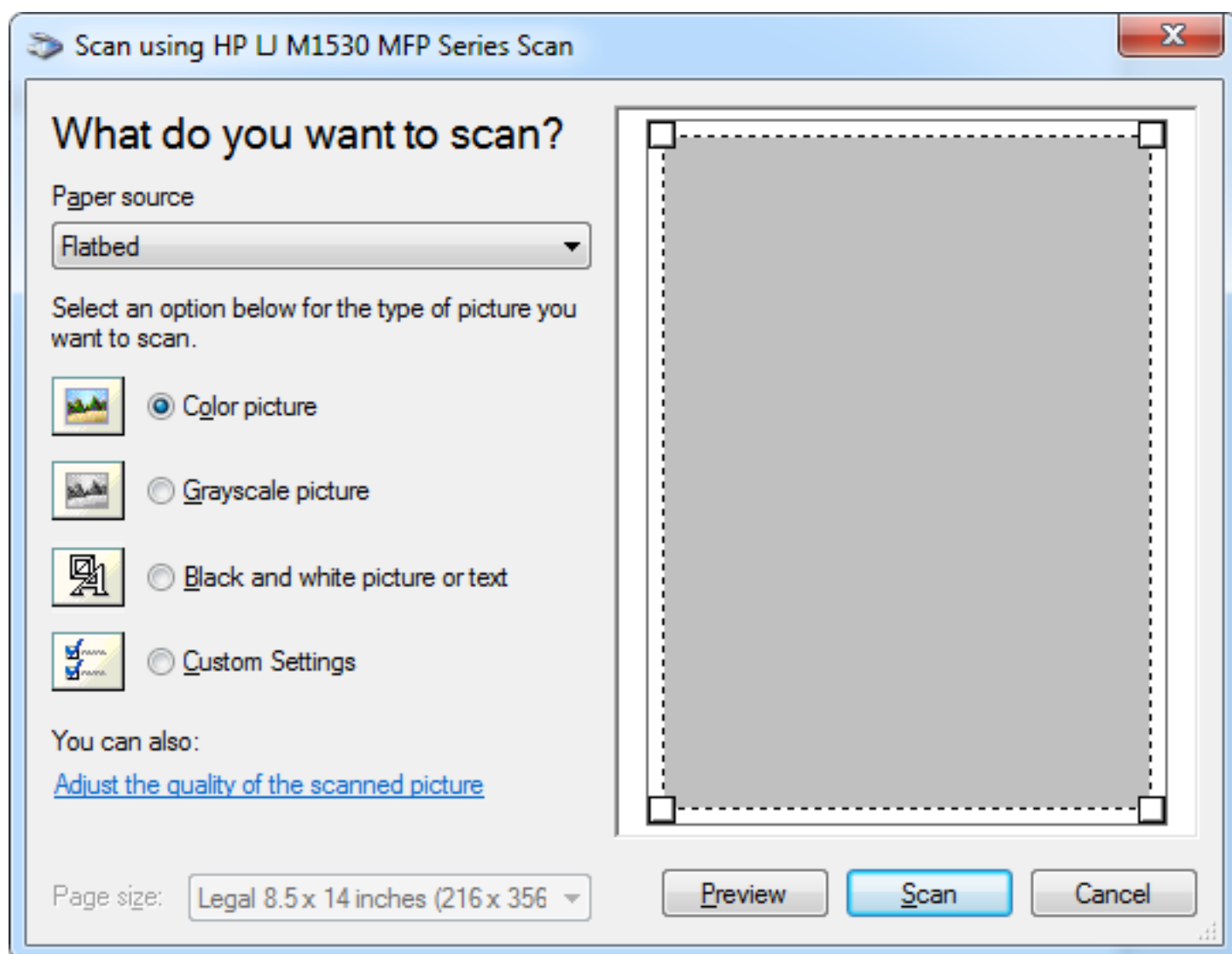
دقت داشته باشید که کدها را در زمان توسعه بین Try..Catch قرار ندهید چون ممکن است در این زمان به خطاهایی برخورد کنید که نیاز باشد در مرورگر آنها را دیده و رفع خطا نمایید.

کلاس اصلی در اینجا جهت نمایش فرم کار با اسکنر می باشد و متدهای مختلفی را جهت ارتباط با اسکنر در اختیار ما قرار می دهد که بسته به نیاز خود می توانید از آنها استفاده کنید. برای نمونه در مثال ما نیز متد اصلی که مورد استفاده قرار گرفته، [ShowAcquireImage](#) می باشد که این متد، فرم پیش فرض دریافت اسکنر را به کاربر نمایش می دهد و کاربر از طریق آن می تواند قبل از شروع اسکن، یکسری تنظیمات را انجام دهد.

این متد ابتدا به صورت خودکار فرم تعیین دستگاه اسکنر ورودی را نمایش داده :



و سپس فرم پیش فرض اسکنرهای TWAIN را جهت تعیین تنظیمات اسکن نمایش می دهد که این امکان نیز در این فرم فراهم است تا دستگاه های Feeder یا Flated انتخاب گردند.



خروجی این متد همان عکس اسکن شده است که از نوع [WIA.ImageFile](#) می باشد و ما پس از دریافتش، ابتدا آن را در یک فایل موقت ذخیره نموده و سپس با استفاده از یک متد کمکی آن را به فرمت Base64 برای درخواست کننده اسکن ارسال می نماییم.

کدهای کلاس کمکی ImageHelper:

```
public static string ImageToBase64(Image image, System.Drawing.Imaging.ImageFormat format)
{
    if (image != null)
    {
        using (MemoryStream ms = new MemoryStream())
        {
            // Convert Image to byte[]
            image.Save(ms, format);
            byte[] imageBytes = ms.ToArray();

            // Convert byte[] to Base64 String
            string base64String = Convert.ToBase64String(imageBytes);
            return base64String;
        }
    }
    return String.Empty;
}
```

توجه داشته باشید که خروجی این متد قرار است توسط callback یک متد جاوا اسکریپتی مورد استفاده قرار گرفته و احیاناً عکس مورد نظر در صفحه نمایش داده شود. پس بهتر است که از قالب تصویر به شکل Base64 استفاده گردد. ضمن اینکه پلاگین های JQuery مرتبط با ویرایش تصویر هم از این قالب پشتیبانی می کنند. ([اینجا](#))

این مثال به ساده ترین شکل نوشته شد. کلاس دیگری هم در [اینجا](#) وجود دارد و در صورتیکه از اسکندر نوع Feeder استفاده می کنید، می توانید از کدهای آن استفاده کنید.

کار ما تا اینجا در پروژه ی WCF Service تقریباً تمام است. اگر پروژه را یکبار Build نمایید برای اولین بار احتمالاً پیغام خطاهای زیر ظاهر خواهند شد:

Error List				
4 Errors 0 Warnings 0 Messages				
Description	File	Line	Column	Project
1 Interop type 'WIA.CommonDialogClass' cannot be embedded. Use the applicable interface instead.	ScannerService.svc.cs	23	13	WcfServiceScanner
2 The type 'WIA.CommonDialogClass' has no constructors defined	ScannerService.svc.cs	23	40	WcfServiceScanner
3 Interop type 'WIA.CommonDialogClass' cannot be embedded. Use the applicable interface instead.	ScannerService.svc.cs	23	44	WcfServiceScanner
4 'WIA.CommonDialogClass' does not contain a definition for 'ShowAcquireImage' and no extension method 'ShowAcquireImage' accepting a first argument of type 'WIA.CommonDialogClass' could be found (are you missing a using directive or an assembly reference?)	ScannerService.svc.cs	27	36	WcfServiceScanner

جهت رفع این خطا، در قسمت Reference های پروژه خود، WIA را انتخاب نموده و از Properties های آن خصوصیت Embed Interop Types را به False تغییر دهید؛ مشکل حل می شود.

به سراغ پروژه ی ویندوز فرم جهت هاست کردن این WCF سرویس می رویم. می توانید این سرویس را بر روی یک [Console App](#) یا [Windows Service](#) هم هاست کنید که در اینجا برای سادگی مثال، از WinForm استفاده می کنیم.

یک پروژه ی WinForm جدید را ایجاد کنید و سپس از قسمت Add Reference > Solution به مسیر پروژه ی قبلی رفته و dll های آن را به پروژه خود اضافه نمایید.

Form1.cs را باز کرده و ابتدا دو متغیر زیر را در آن به صورت عمومی تعریف نمایید:

```
private readonly Uri _baseAddress = new Uri("http://localhost:6019");
private ServiceHost _host;
```

برای استفاده از کلاس [ServiceHost](#) لازم است تا ارجاعی به فضای نام System.ServiceModel داده شود. متغیر _baseAddress نگه دارنده ی آدرس ثابت سرویس اسکندر در سمت کلاینت می باشد و به این ترتیب ما دقیقاً می دانیم باید سرویس را با کدام آدرس در کدهای جاوا اسکریپتی خود فراخوانی نماییم.

حال در رویداد Form_Load برنامه، کدهای زیر را جهت هاست کردن سرویس اضافه می نماییم:

```
private void Form1_Load(object sender, EventArgs e)
{
    _host = new ServiceHost(typeof(WcfServiceScanner.ScannerService), _baseAddress);
    _host.Open();
}

private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    _host.Close();
}
```

همین چند خط برای هاست کردن سرویس روی آدرس localhost و پورت 8010 کامپیوتر کلاینت کافی است. اما یکسری تنظیمات مربوط به خود سرویس هم وجود دارد که باید در زمان پیاده سازی سرویس، در خود پروژه ی سرویس، ایجاد می گردید. اما از آنجا که ما قرار است سرویس را در یک پروژه ی دیگر هاست کنیم، بنابراین این تنظیمات را باید در همین پروژه ی WinForm قرار دهیم.

فایل App.Config پروژه ی WinForm را باز کرده و کدهای آنرا مطابق زیر تغییر دهید:

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>

  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="BehaviourMetadata">
          <serviceMetadata httpGetEnabled="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <service name="WcfServiceScanner.ScannerService"
        behaviorConfiguration="BehaviourMetadata">
        <endpoint address=""
          binding="basicHttpBinding"
          contract="WcfServiceScanner.IScannerService" />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

اکنون پروژه‌ی هاست آماده اجرا می‌باشد. اگر آنرا اجرا کنید و در مرورگر خود آدرس ذکر شده را وارد کنید، صفحه‌ی زیر را مشاهده خواهید کرد که به معنی صحت اجرای سرویس اسکندر می‌باشد.

ScannerService Service

You have created a service.

To test this service, you will need to create a client and use it to call the service. You can do this using the svcutil.exe tool from the command line with the following syntax:

```
svcutil.exe http://localhost:6019/?wsdl
```

You can also access the service description as a single file:

```
http://localhost:6019/?singleWsdl
```

This will generate a configuration file and a code file that contains the client class. Add the two files to your client application and use the generated client class to call the Service. For example:

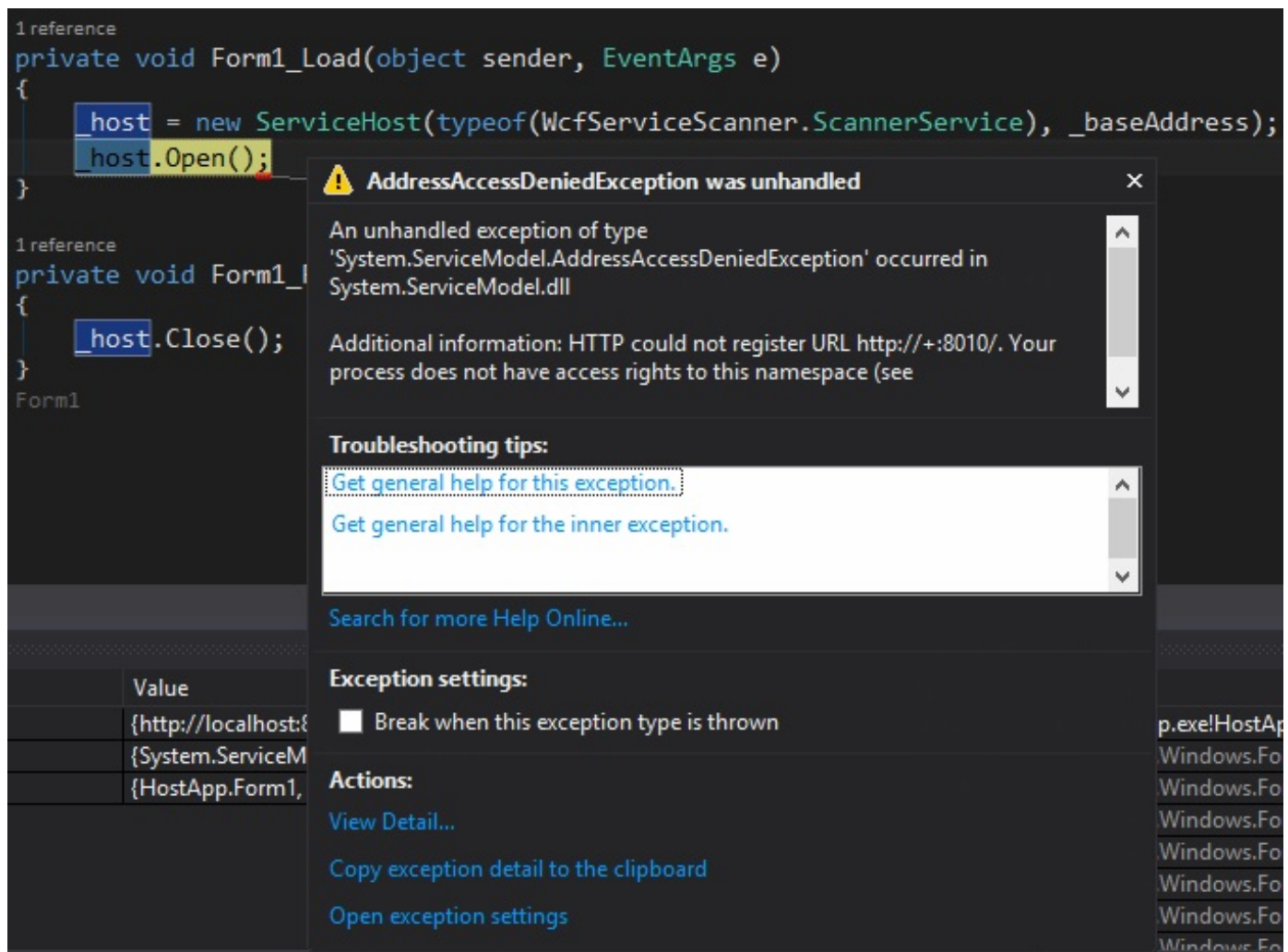
C#

```
class Test
{
    static void Main()
    {
        ScannerServiceClient client = new ScannerServiceClient();

        // Use the 'client' variable to call operations on the service.

        // Always close the client.
        client.Close();
    }
}
```

اگر موفق به اجرا نشدید و احیاناً با خطای زیر مواجه شدید، اطمینان حاصل کنید که ویژوال استودیو Run as Administrator باشد. مشکل حل خواهد شد.

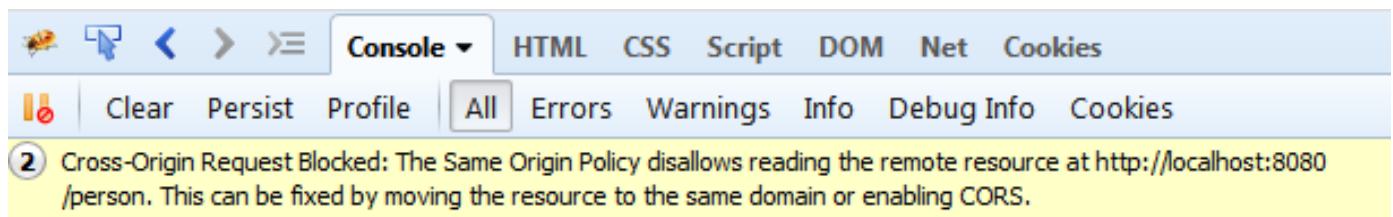


به سراغ پروژه‌ی بعدی، یعنی وب سایت خود می‌رویم. یک پروژه‌ی MVC جدید ایجاد نمایید و در View مورد نظر خود، کدهای زیر را جهت صدا زدن متد `GetScan` اضافه می‌کنیم.

(از آنجا که کدها به صورت جاوا اسکریپت می‌باشد، پس مهم نیست که حتما پروژه MVC باشد؛ یک صفحه‌ی HTML ساده هم کافی است).

```
<a href="#" id="get-scan">Get Scan</a>
<img src="" id="img-scanned" />
<script>
    $("#get-scan").click(function () {
        var url = 'http://localhost:6019/';
        $.get(url, function (data) {
            $("#img-scanned").attr("src", "data:image/Jpeg;base64, " + data.GetScanResult);
        });
    });
</script>
```

دقت کنید در هنگام دریافت اطلاعات از سرویس، نتیجه به شکل `GetScanResult` خواهد بود. الان اگر پروژه را اجرا نمایید و روی لینک کلیک کنید، اسکندر شروع به دریافت اسکن خواهد کرد اما نتیجه‌ای بازگشت داده نخواهد شد و علت هم مشکل امنیتی [CORS](#) می‌باشد که به دلیل دریافت اطلاعات از یک دامین دیگر رخ می‌دهد و اگر با Firebug درخواست را بررسی کنید متوجه خطا به شکل زیر خواهید شد.



راه حل های زیادی برای این مشکل ارائه شده است، و متأسفانه بسیاری از آنها در شرایط پروژهای ما جوابگو نمی باشد (به دلیل هاست روی یک پروژه ویندوزی). تنها راه حل مطمئن (تست شده) استفاده از یک کلاس سفارشی در پروژه WCF Service می باشد که مثال آن در [اینجا](#) آورده شده است. برای رفع مشکل به پروژه WcfServiceScanner بازگشته و کلاس جدیدی را به نام CORSSupport ایجاد کرده و کدهای زیر را به آن اضافه کنید:

```
public class CORSSupport : IDispatchMessageInspector
{
    Dictionary<string, string> requiredHeaders;
    public CORSSupport(Dictionary<string, string> headers)
    {
        requiredHeaders = headers ?? new Dictionary<string, string>();
    }

    public object AfterReceiveRequest(ref System.ServiceModel.Channels.Message request,
    System.ServiceModel.IClientChannel channel, System.ServiceModel.InstanceContext instanceContext)
    {
        var httpRequest = request.Properties["httpRequest"] as HttpRequestMessageProperty;
        if (httpRequest.Method.ToLower() == "options")
            instanceContext.Abort();
        return httpRequest;
    }

    public void BeforeSendReply(ref System.ServiceModel.Channels.Message reply, object
    correlationState)
    {
        var httpResponse = reply.Properties["httpResponse"] as HttpResponseMessageProperty;
        var httpRequest = correlationState as HttpRequestMessageProperty;

        foreach (var item in requiredHeaders)
        {
            httpResponse.Headers.Add(item.Key, item.Value);
        }
        var origin = httpRequest.Headers["origin"];
        if (origin != null)
            httpResponse.Headers.Add("Access-Control-Allow-Origin", origin);

        var method = httpRequest.Method;
        if (method.ToLower() == "options")
            httpResponse.StatusCode = System.Net.HttpStatusCode.NoContent;
    }
}

// Simply apply this attribute to a DataService-derived class to get
// CORS support in that service
[AttributeUsage(AttributeTargets.Class)]
public class CORSSupportBehaviorAttribute : Attribute, IServiceBehavior
{
    #region IServiceBehavior Members

    void IServiceBehavior.AddBindingParameters(ServiceDescription serviceDescription,
    ServiceHostBase serviceHostBase, System.Collections.ObjectModel.Collection<ServiceEndpoint> endpoints,
    BindingParameterCollection bindingParameters)
    {
    }

    void IServiceBehavior.ApplyDispatchBehavior(ServiceDescription serviceDescription,
    ServiceHostBase serviceHostBase)
    {
        var requiredHeaders = new Dictionary<string, string>();

        //Chrome doesn't accept wildcards when authorization flag is true
    }
}
```

```
//requiredHeaders.Add("Access-Control-Allow-Origin", "*");
requiredHeaders.Add("Access-Control-Request-Method", "POST,GET,PUT,DELETE,OPTIONS");
requiredHeaders.Add("Access-Control-Allow-Headers", "Accept, Origin, Authorization, X-
Requested-With,Content-Type");
requiredHeaders.Add("Access-Control-Allow-Credentials", "true");
foreach (ChannelDispatcher cd in serviceHostBase.ChannelDispatchers)
{
    foreach (EndpointDispatcher ed in cd.Endpoints)
    {
        ed.DispatchRuntime.MessageInspectors.Add(new CORSSupport(requiredHeaders));
    }
}

void IServiceBehavior.Validate(ServiceDescription serviceDescription, ServiceHostBase
serviceHostBase)
{
}

#endregion
}
```

فضاهای نام لازم برای این کلاس به شرح زیر می باشد:

```
using System.ServiceModel;
using System.ServiceModel.Channels;
using System.ServiceModel.Description;
using System.ServiceModel.Dispatcher;
```

کلاس ScannerService را باز کرده و آنرا به ویژگی

```
[CORSSupportBehavior]
public class ScannerService : IScannerService
{
```

مزین نمایید.

کار تمام است، یکبار دیگر ابتدا پروژه ی WcfServiceScanner و سپس پروژه هاست را Build کرده و برنامه ی هاست را اجرا کنید. اکنون مشاهده می کنید که با زدن دکمه ی اسکن، اسکندر فرم تنظیمات اسکن را نمایش می دهد که پس از زدن دکمه ی Scan، پروسه آغاز شده و پس از اتمام، تصویر اسکن شده در صفحه ی وب سایت نمایش داده می شود.

نظرات خوانندگان

نویسنده: امیر بختیاری
تاریخ: ۱۳۹۳/۱۱/۲۱ ۱:۴۸

با سلام و تشکر خدمت شما دوست گرامی
من تمام مراحل را به دقت ایجاد کردم ولی بازم همون خطایی که خودتون هم عنوان کردید را می‌ده

Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at http://localhost:13748/. This can be fixed by moving the resource to the same domain or enabling CORS.

فایل پروژه را به طور کامل می‌ذارم ببینید علت چیه [WcfServiceScanner.zip](#)

با تشکر مجدد

نویسنده: شروین ایرانی
تاریخ: ۱۳۹۳/۱۱/۲۱ ۹:۵

با سلام؛ آیا امکانش هست که سورس پروژه‌ی مثال رو هم قرار بدید؟ تشکر

نویسنده: امیر بختیاری
تاریخ: ۱۳۹۳/۱۱/۲۱ ۱۰:۳۱

سورس پروژه را از نظر قبلی می‌تونید دریافت کنید