# Part 4 – Deploying Webresources using spkl and debugging inside the browser

This is part of the course 'Scott's guide to building Power Apps JavaScript Web Resources using TypeScript'.

Now that you have created and tested your webresource locally, you are ready to deploy it to your Dataverse environment and hook it up to the Model Driven App.

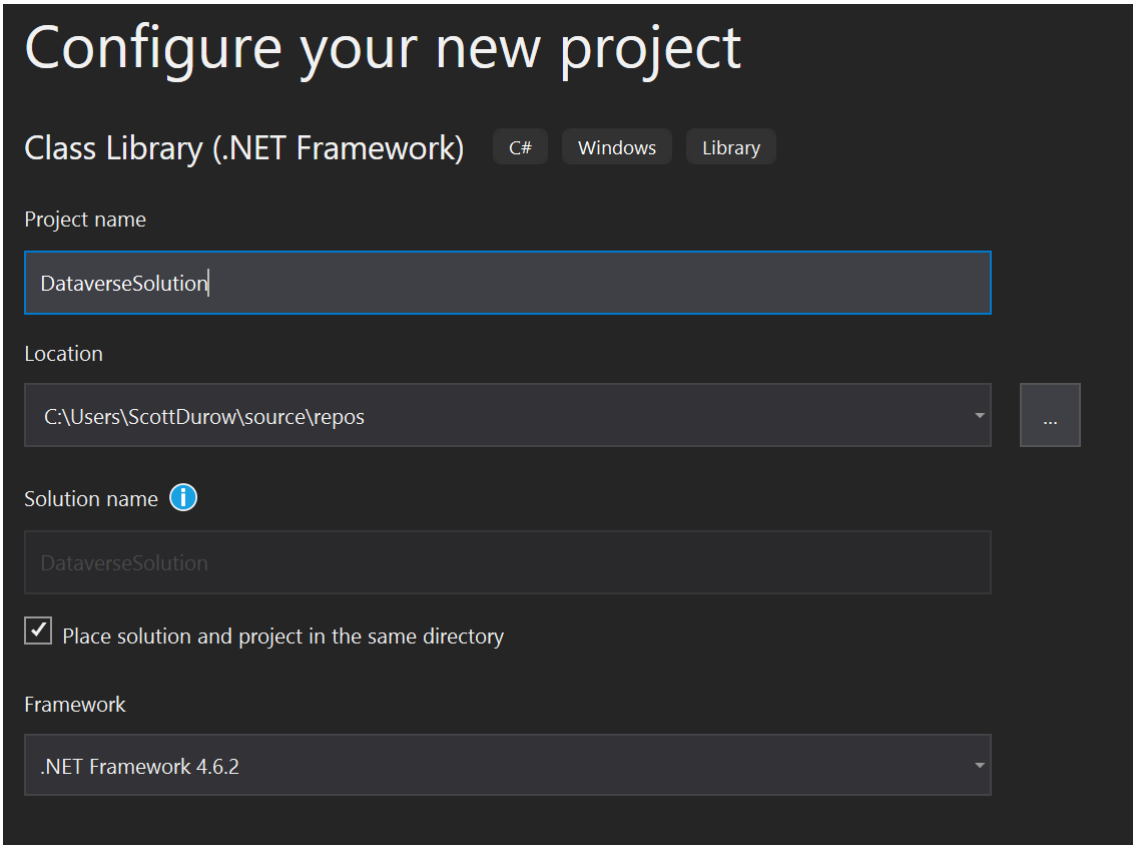
## Deploying using spkl

`spkl` is a command line utility (https://github.com/scottdurow/SparkleXrm/wiki/spkl) that is distributed via NuGet. It allows you to perform common Dataverse tasks such as deploying Plugins and Webresources via a set of simple command files.

### Create Visual Studio Project

The `spkl` NuGet package can easily be downloaded and run using Visual Studio's NuGet integration. Start by using Visual Studio to create a new C# Class Library Project (it does not really matter what type as long as it targets the .Net Framework and not .Net Core).

> **Note**: You can use Visual Studio Community Edition (https://visualstudio.microsoft.com/downloads/ ) if you don't have Visual Studio already installed.

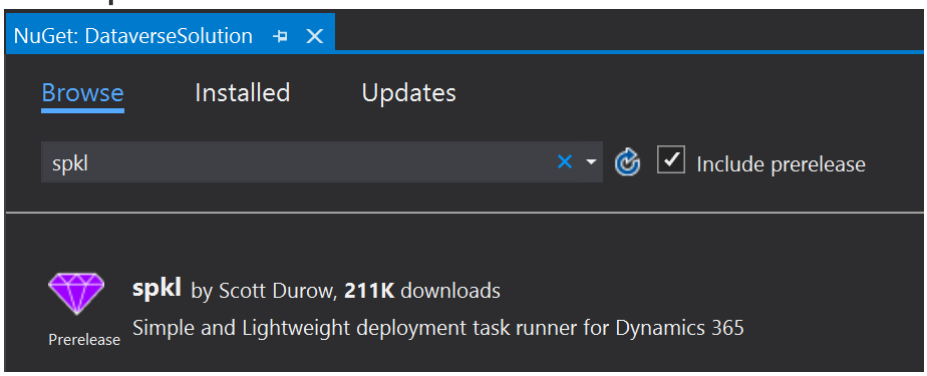



Select the location of your project to be in the same folder as the VSCode project that you have created for your TypeScript.

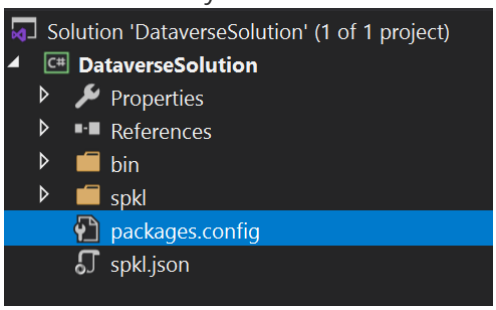
In the newly created project:

1. Right-Click on the **Project** -> **Manage NuGet Packages**

2. Select the **Browse** tab and type **spkl**. If you want to use the more recent beta features, select **Include prerelease**:




3. Select **spkl** and then **Install**
4. You will now see a **spkl** folder and `spkl.json` in your project. You can delete the other two c# files since they are not needed:




## Edit the `spkl.json`

You next need to tell `spkl` where your built JavaScript files are to deploy.

Open the `spkl.json` and replace it with the following:

```
{
    "webresources": [
        {

            "profile": "default,debug",
            "solution": "SOLUTION_UNIQUE_NAME",
            "files": [
                {
                    "uniquename": "dev1_/js/clienthooks.js",
                    "file": "../clientjs/dist/clienthooks.js",
                    "description": "JavaScript for Forms and Commandbar Actions"
                }
            ]
        }
    ]
}
```

The webresources section tells `spkl` where the files are you want to deploy.

Replace `SOLUTION_UNIQUE_NAME` with your specific solution name. The deployed webresource will be added to this solution.

> **Note**: Later in this series we will add another section under solutions to control how the solution will be unpacked and the JavaScript picked up from the build output.

## Deploying the webresources

You can now easily deploy your webresource using the batch file found under `spkl`.

If you are working in VSCode, you don't need to have Visual Studio open, you can simply use the terminal to change the directory using the relative path to your spkl folder:

```
cd ..\DataverseSolution\spkl
deploy-webresources.bat
```

> **Note:** It is important that you run the batch files when in the folder that contains the `spkl.json` or a child directory of that location. `spkl` will search for the nearest `spkl.json` to use.

```
C:\repos\jswebresources\DataverseSolution\spkl>deploy-webresources.bat
Using 'C:\repos\jswebresources\DataverseSolution\packages\spkl.1.0.633-beta\tools\spkl.exe'

                __   .__
  ____________ |  | _|  |
 /  ___/\____ \|  |/ /  |
 \___ \ |  |_> >    <|  |__
/____  >|   __/|__|_ \____/
     \/ |__|        \/
spkl Task Runner v1.0.633.1     Tasks v1.0.633.1
Office365 Login
Note:  Use /l switch for on-premises or legacy login
(0) Add New Server Configuration
(1) dev1demos14.crm3.dynamics.com,  demos14,  scott.durow@dev1demos14.onmicrosoft.com

Specify the saved server configuration number (0-1) [1] : 1

Deploying WebResources
Using Config 'C:\repos\jswebresources\DataverseSolution'
Updating Webresource '../clientjs/dist/clienthooks.js' -> 'dev1_/js/clienthooks.js'
Adding to solution 'supersolution'
Deployed 1 webresource(s)
Publishing 1 webresources...
Publish complete.
Processed 1 section(s)
Processed 1 config(s)
Press any key to continue . . .
```

The webresource is deployed or an existing one updated. It is added to the solution if not already there and then the changes are published.

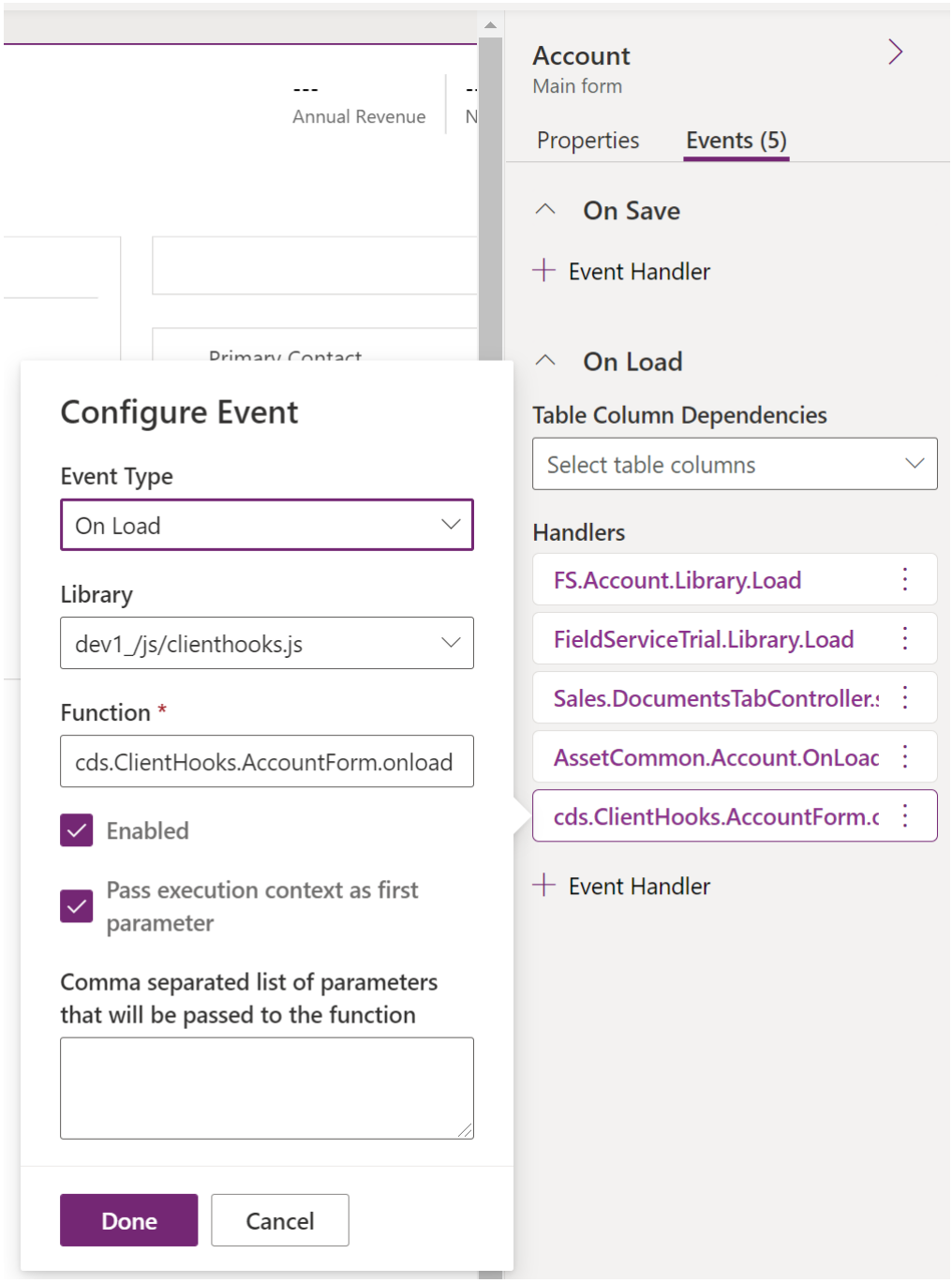
You can then hook up your webresource to the necessary form events. In this example we register it on the Account form for the Onload event:

**Event Type**: On Load

**Library**: `dev1_/js/clientooks.js`

**Function**: `cds.ClientHooks.AccountForm.onload`

**Pass execution context:** Checked




# Debugging in the browser

When you have webpack running in watch mode (`npm start`) the output JavaScript will update each time you make a change. Instead of constantly deploying your webresource with each change you can easily use a tool called Fiddler to redirect requests to your webresource on the server to the local file that is being updated by webpack. See https://docs.microsoft.com/en-us/dynamics365/customerengagement/on-premises/developer/streamline-javascript-development-fiddler-autoresponder

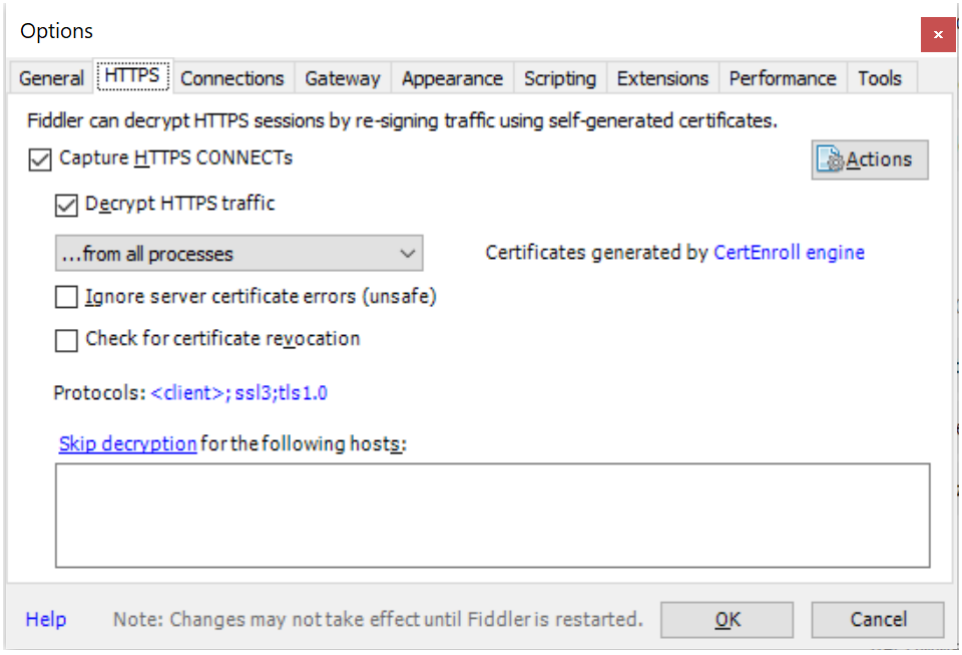
## Download and install Fiddler Classic

There is a new version of Fiddler called 'Fiddler Everywhere' – but instead you should download 'Fiddler Classic'

Download and install from: https://www.telerik.com/fiddler/fiddler-classic

After installation, you must enable HTTPS decryption so that we can intercept the traffic between the browser and the Dataverse environment.

1. Open **Tools** -> **Options** -> **HTTPS** Tab
2. Check the **Decrypt HTTPS traffic** and accept the prompts to install the Fiddler Certificate.




3. You can now select the **AutoResponder** tab, and select **Add Rule**
4. Set the Match string to be the webresource name and the Path to the full path of your webpack generated output:

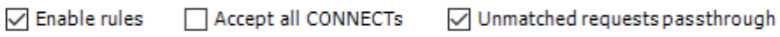
- `dev1_/js/clienthooks.js` - Enter the unique name of your webresource.
- Enter the full path of your `dist/ClientHooks.js`

```
<img src="media/Part 4 - Deploying and browser
debugging/ab3a009fb2a013645d6c913ac8b2ca96-1621041308204-1621041350541.png"
alt="ab3a009fb2a013645d6c913ac8b2ca96-1621041308204-1621041350541"
style="zoom:50%;" />
```

**Note:** This can also be a regex if you had multiple webresources at the same location:

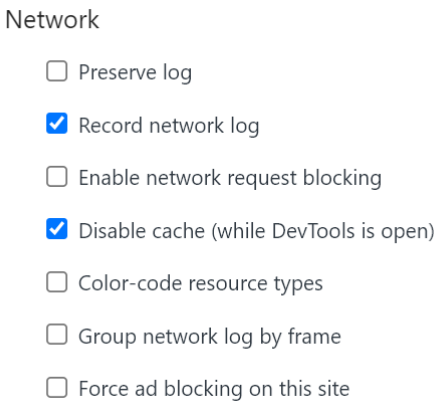
```
StringToMatch: REGEX:(?insx).+\/dev1_\/js\/(?'fname'[^?]*.js)`
File: `C:\MyProject\dist\${fname}
```

Be sure to select **Enable rules** and **Unmatched requests passthrough**.

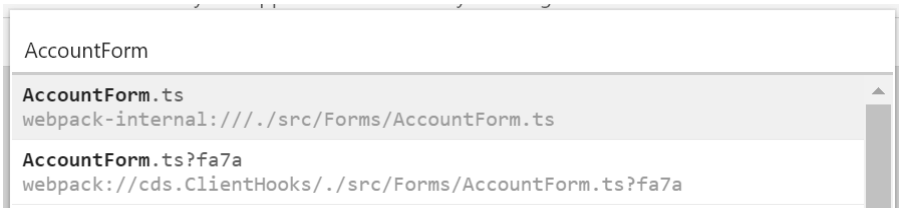



Now when you refresh your browser, the webpack generated file will be loaded into the browser.

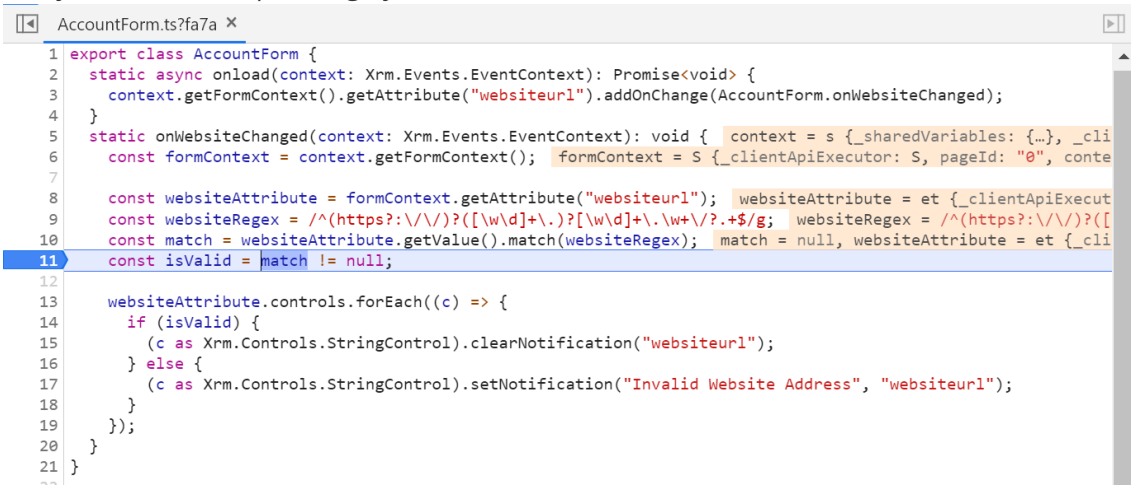
5. Navigate your Dataverse App in the browser and press F12 (or `Ctrl + Shift + I`) to open the Developer Tools.
6. In The Developer Tools open the settings and check **Disable cache (while DevTools is open)**. This will ensure that the latest version is always requested when you refresh a page.




7. Ensure you have webpack running in watch mode (`npm start`)
8. Navigate to an Account Record in your Model Driven App to load the Webresource.
9. In the Developer Tools use Open File (`Ctrl + P`) and type the name of a TypeScript file, you should see two versions of it – one prefixed with `webpack-internal://` and the other just with `webpack://`




10. Be sure to select the file with the `webpack:` prefix. This is the original TypeScript source that is output under the `sourceMappingURL` because your `webpack.dev.js` has the setting `devtool: "eval-source-map"`.
    The `webpack-internal` source is simply the JavaScript that was created by the TypeScript compiled will look somewhat different to your original TypeScript depending on which TypeScript language features you are using and the target you are using. Using ES5 is the most common if you want to IE11 (still used by the App for Outlook when running inside some versions of Office at the time of writing) [https://docs.microsoft.com/en-us/office/dev/add-ins/concepts/browsers-used-by-office-web-add-ins](https://docs.microsoft.com/en-us/office/dev/add-ins/concepts/browsers-used-by-office-web-add-ins)
11. You can now set a break point on the `AccountForm.ts`, refresh the Model Driven record and you can then step through your code.




12. If you make a change to your TypeScript, webpack will pick up the modification and regenerate the JavaScript. Refreshing the Model Driven App page will then pick up the latest version (provided Fiddler is running and you have the Developer Tools open).

13. When you are stepping through code, you may find that it will step into the underlying webpack-internal code. This is because the TypeScript transpiled files are output using the `eval-source-map` plugin and labelled with `sourceURL=webpack-internal`.

Once you are happy with the JavaScript webresource, you can then build for distribution and deploy the final version. Later in this series we will look at how to build and deploy your JavaScript webresource as part of a build/release pipeline.

```
npm run dist
```

Change directory to the spkl folder at the command prompt:

```
deploy-webresource.bat
```

This will deploy your built web resource without the source-maps and with the code minified for production deployment.

# Troubleshooting

- **TypeScript file not found** - If you don't see your `AccountForm.ts` inside the browser, you may need to perform a Hard Refresh `Ctrl-Shift-R`
- Breakpoint not hit - You may be looking at a stale version of your `AcountForm.ts` - try searching again and ensure that it is prefixed with `webpack` - and not `wepack-internal`.

# Up Next...

Now we are ready to look at some more advanced topics such as early bound types and calling the `WebApi`.