**München - Office JS - 12.10.2020**

**Einleitung**

**Danke**

* Danke Euch
* Danke René.
* Was macht René sonst noch, wenn er nicht Excelt? Lustige Bildchen in FB posten

**Über mich**

* MVP, Reconnect
* Sites, Translator, Ticker, Soccer.
* Translator, weltweit einzigartig
* 420 K Downloads vom Soccer, damals 2010 bei der WM

**Über mich - Tools**

* Überwiegender Teil meines Tools beinhaltet VBA.
* Meine Tools haben alle eine eigene UI, erstellt in VBA.
* Multilanguage in allen Tools.
* Datenbanken in den meisten Tools.
* KMUs und große Unternehmen.

**Überblick**

**Office-Erweiterungen**

* Es gibt 3 Sorten von Add-Ins
* VBA Add-Ins, die dann auf VBA basieren sowie nur für Desktop-Rechner verfügbar sind, wobei Windows die bevorzugte Plattform ist. Mac ist da eher eingeschränkt.
* Sogenannte COM Add-Ins, die mit C++, .Net und VSTO entwicklelt werden können. Auch hier nur Desktop und Windows als Plattform.
* Und schließlich nun Office JS Add-Ins, die in JavaScript erstellt werden. Diese Add-Ins laufen dann auf quasi allen Plattformen, also z.B. Desktop & Online.

**Office-Add-Ins**

* Unter Office-Add-Ins ist eine Plattform zu verstehen, jedenfalls meint das MS so. Und wenn bei MS von Office-Add-In gesprochen wird, sind in der Regel nun JavaScript-Add-Ins gemeint.
* Office-Add-Ins sind Webanwendungen, funktionieren also wie eine Website, die in die Anwendung als Add-In integriert wird.
* Das simpelste Add-In lässt anhand als statische Webseite realisieren, also wo beispielswiese nur einen Text ausgegeben wird. Das ist aber nicht besonders sinnvoll, denn Webanwendungen verwenden in der Regel einen Server.
* Von daher wird dann im Add-In meist ein Server angebunden, der z.B. Daten liefert. Womit die Daten vom Server ausgeliefert werden, ist dem Add-In erstmal egal. Der Server kann zum Beispiel PHP verwenden, um Daten abzurufen.
* Mit Office-JS ist nun die JavaScript-Schnittstelle zu den einzelnen Anwendungen gemeint. Das heißt, über Office JS lassen sich dann Features in der Anwendung ansprechen, z.B. in Excel ein Diagramm erstellen oder aktualisieren. Diese Schnittstelle kann dann als API gesehen werden.

**Office-Add-Ins-Typen**

* Wie sehen jetzt Office Add-Ins aus?
* Office-Add-Ins können Objekte im Dokument oder Aufgabenbereiche sein. Die Bedienung erfolgt wie auf einer Webseite.
* Links z.B. das Microsoft Visio Data Visualizer Add-In, was Visio-Diagramme in ein Excel-Dokument einbettet bzw. anzeigt.
* Rechts das Feature Ideen aus Excel, was eigentlich ein Add-In ist. Auch Script Lab, ist ein Office-Add-In mit Aufgabenbereichen.
* Das Menüband lässt sich auch mit Office-Add-Ins ergänzen.

**Office JS API**

* Die Office JS API ist in mehrere Teile aufgeteilt.
* Die Common API enhält allgemeine und anwendungsüber­greifende Objekte und Methoden, wie z.B. zur Authentifizierung, Dialoge, Event-Handler, Dokumente usw.
* Und dann gibt es pro Anwendung eine spezifische API, die dann die Features der Anwendung anspricht.
* Die am aktivsten entwickelte API ist die für Excel, die schon bei Version 1.12 ist.
* Wir hätten noch APIs für PowerPoin,t, Word, Outlook, Visio, OneNote und Project.
* Interessanterweise heißt die API zu Outlook Mailbox.
* Bei Excel wäre noch anzumerken, dass es noch eine speziell Excel Online API gibt, die bei Version 1.1. steht.
* Zu jeder Versionnummer bestehen Anforderungen an die Versionsnummer und den Build der Anwendung. Zum Beispiel setzt die Version 1.12 für Excel voraus, dass mindestens Excel 365 in der Version von September 2020 installiert ist.
* Add-Ins für Excel 2016 (Kaufversion) laufen nur mir API-Version 1.1 und für Excel 2019 Kaufversion nur bis Version 1.8 der API.

**Office JS & VBA**

* Wenn wir mal VBA mit Office JS vergleichen, sind wesentliche Punkte folgendes.
* Die Lernkurve in Office JS ist recht hoch, denn neben JavaScript als solches, müssen auch Technologien zur Entwicklung von Webanwendungen erlernt werden.
* Klarer Vorteil für Office Add-Ins. Läuft dann auch im Web.
* VBA hat eine veraltete IDE, für JavaScript usw. gubt eine ganze Reihe an modernen Tools, wie z.B. Visual Studio oder Visual Studio Code.
* VBA erfährt nur Updates, wenn z.B. Excel um neue Features ergänzt wird. Was "Neues" wie z.B. eine moderne IDE ist nicht geplant. VBA ist übrigens nicht tot, wie manche meinen. Sagt selbst Microsoft.
* Das Objektmodell für Office JS entspricht noch nicht dem von VBA, es fehlt einiges.
* Nebenbei, Office JS ist meines Wissens langsamer als VBA, was übrigens ja technologisch bedingt auch nicht gerade das schnellste Pferd ist.

**Office JS Entwicklertools**

* Rein theoretisch braucht man für die Entwicklung eines Office Add-Ins, z.B. für Excel, nur einen Texteditor, wie z.B. Notepad++.
* Macht allerdings nicht besonders viel Freude. Denn Syntaxhighlighting, IntelliSense usw. sind dann doch etwas bequemer.
* Es gibt eine ganz Reihe an Tools, je nachdem ob man Einsteiger oder Fortgeschrittener ist, oder wieviel Geld man ausgeben will, kommen einige Tools in Betracht.
* Wenn man Anfänger ist, empfehle ich definitiv Script Lab. Script Lab stelle ich später noch detaillierter vor.
* Möchte man mehr in die Richtung der Erstellung professioneller Add-Ins gehen, sind Visual Studio Code oder Visual Studio eine gute Wahl.
* Visual Studio ist kostenlos.
* Visual Studio kostet schon was, bietet dann aber z.B. spezielle Vorlagen, um Office Add-Ins zu entwickeln.

**Konzepte**

**Aufbau**

* Ein JavaScript Office-Add-In folgt einem generellen Aufbau und besteht aus mindestens zwei Komponenten
* Ein sogenanntes Manifest, was eine XML-Datei ist, definiert grundlegende Eigenschaften zum Add-In. Dazu gleich mehr. Das Manifest muss übrigens immer im Root-Verzeichnis liegen,
* Die Website, also das entsprechende Projekt. Dort ist man dann mit der z.B. Ordnerstruktur frei.
* Im Bild mal eine solche Ordnerstruktur, wie diese von einem Generator-Tool in Visual Studio generiert wurde.
* Wie zu sehen, beinhaltet das Webprojekt einige HTML, CSS und TypeScript-Dateien sowie noch andere relevante Dateien für das Projekt, z.B. Node.JS Module.

**Manifest**

* Das Manifest ist eine XML-Datei, wo Eigenschaften des Add-In festgelegt werden. Zum Beispiel eine ID in Form einer GUID, ein Name für das Add-In, eine Beschreibung, eine Version, Icons, die unterstützten Hosts, Einträge im Menüband, wenn notwendig usw.
* Die XML-Datei folgt einem definiertes Schema. Dieses ist recht umfangreich, von daher sollte man dem Link da unten folgen.
* Es gibt Eigenschaften, die Pflichtangaben sind, wie z.B. ID, Name, Version, Host usw.
* Für mehrsprachige Add-Ins gibt es die Möglichkeit, die Standardwerte in Abhängigkeit eines Sprachcodes - z.B. de-de für Deutsch - zu überschreiben,

**HTML**

* Definition = Hypertext Markup Language und Hyper hat nix mit Baxxter (Scooter) zu tun.
* Tags, z.B. head, body
* Ein Element ist das was zwischen Anfangs- und Endtag liegt, also sowas wie eine Gruppe.
* Attribute, z.B. Verweis auf CSS, Bild, Style (hardcoded)
* Ein HTML-Dokument hat immer einen <head> und einen <body>
* In den Head kommen z.B. Meta-Angaben rein.
* HTML5 hat neue Tags eingeführt, wie <section>, <article>, <nav>, <footer> oder <aside>

**CSS**

* Cascading Style Sheets
* Das was mit CSS angesprochen bzw. definiert wird, nennen wir Selektor. Zum Beispiel im Bild html, body, ul und Kopfzeile. Es gibt dann verschiedene Arten von Selektoren.
* Zum Beispiel solche, die direkt ein HTML-Tag ansprechen, z.B. hier ul, was unordered list entspricht.
* Oder sogenannte selbst definierte Klassen, die einen Punkt vor dem Namen haben und die dann im HTML-Dokument in den Tags referenziert werden.
* Ich kann auch eine HTML-Hierarchie ansprechen, im CSS würde ul li die Listeneinträge ansprechen oder zu, Beispiel XYZ ul li dann alle Listeneinträge ul li denen im übergeordneten Tag die Klasse XYZ zugewiesen ist.
* CSS vererbt in der Regel Eigenschaften. Angenommen, die definiere die Hintergrundfarbe von Tag header blau, ist alle, was dadrin ist blau.
* Es lassen sich auch eine Art Funktionen in CSS abrufen. Zum Beispiel würde li:first-child oder li:nth-child(2) das erste und zweite li ansprechen.
* Effekte sind dann z.B. abgerundete Elemente (Buttons) oder ein anderes Verhalten beim Hovern.
* Insgesamt ist CSS sehr mächtig.

**JavaScript**

* JavaScript ist plattformübergreifend, heißt also es klappt in Browsern in Windows, Mac, Linux und viele andere.
* JavaScript ist nicht typisiert, das heißt, es muss keine explizite Deklaration von Variablen gemacht werden.
* JavaScript bietet Standardbibliotheken an, zum Beispiel für Datum, Arrays oder Mathematik, die wiederum als Objekte daherkommen.
* JavaScript ist objektorientiert, bietet also z.B. die Möglichkeit, Objekte mit Eigenschaften und Methoden anzulegen. Was vielleicht etwas verwirrend ist, dass auch eine Funktion zum Objekt werden kann.
* Beispiel 1: eine Funktion mit einem Argument, das untypisiert ist. Geht also davon aus, dass auch tatsächlich eine Zahl übergeben wird.
* Beispiel 2: eine weitere Funktion mit zwei Argumenten. Und jetzt wird's interessant, denn ein Argument ist eine Funktion. Das wird euch z.B. bei Office JS öfter vorkommen. Es lässt sich sogar eine Funktion innerhalb der Argumentangabe erstellen. Das ist oft verwirrend.
* Beispiel 3: JavaScript ist eine prototypbasierte Sprache. Ab dem sogenannten Standard ECMAScript 2015 bzw. ES6 steht dann auch das Schlüsselwort class zur Verfügung, was es etwas einfacher macht, Klassen zu deklarieren. Siehe das Beispiel.

**TypeScript**

* TypeScript ist eine Art erweitertes JavaScript und bleibt dabei kompatibel. Das heißt, dass normales JavaScript durchaus verwendet werden kann. Theoretisch gesehen, kann man also mischen.
* TypeScript ist typisiert, das heißt man kann Variablen einen Datentyp zuweisen. Das macht es etwas einfacher zu programmieren. VBA-Entwicklern dürfte das somit entgegenkommen.
* TypsScript wird anhand eines Kompilers (z.B. ein eigener oder Babel) wieder in JavaScript kompiliert

**Unterschied Let - Var**

* Let gilt nur für den Abschnitt innerhalb es verwendet wird.
* Mit Var wird eine Variable initialisiert, egal wo die Deklaration steht. Bei Let nicht.
* Ein erneutes Deklarieren einer Variable über Let führt zu einem Fehler (Variable bereits vorhanden).

**Excel-Add-In Grundgerüst**

* Das zentrale Element jeder Office JS-Anwendung ist die run-Methode. Für Excel ist dies dann Excel.run. Für Word wäre das Word.run. Durch den Aufruf erhält man Zugriff auf den Context, hier dann Excel. Innerhalb der Funktion kann ich dann eigenen Code ausführen.
* In dem Beispiel ist Excel.run in der Funktion TS\_Run gekapselt. Hätte ich jetzt zum Beispiel mehrere Buttons in meinen Add-In, würde ich dann beispielsweise entsprechend viele Funktionen schreiben, die alle Excel.run verwenden.
* Office JS Add-Ins können nicht synchron mit den Office-Anwendungen ausgeführt werden. Von daher werden Anweisungen zum Beispiel zum Ändern von Eigenschaften, wie die Farbe hier im Beispiel in eine Art Batch bzw. Warteschlange geschrieben.
* Erst durch den Aufruf context.sync() werden dann die Operationen tatsächlich ausgeführt. Also dann hier zum Beispiel die Farbe auf Gelb zu setzen.
* Und, ganz wichtig, wenn ich auf Eigenschaften zugreifen möchte, muss ich die erstmal laden. Das macht hier range.Load. Ist also anders als VBA, wo das direkt zur Verfügung steht. Das Range-Objekt nennt man auch Proxyobjekt.
* Proxyobjekte kann man auch als eine Art Platzhalter für echte Objekte verstehen. Es passiert also nichts, solange nicht context.sync verwendet wird.
* Async, Await und Promises kommen aus der Welt der asynchronen Programmierung. Vielleicht am besten mir VBA erklärt. VBA ist synchron, also man muss warten, bis ein Auftrag - die Prozedur - die Arbeit erledigt hat. Asynchron wäre jedoch, wenn die Prozedur das Versprechen gibt, dass es die Aufgabe erfüllen wird und ich in der Zwischenzeit was anderes tun kann. Ist die Prozedur fertig werde ich bzw. ein Hauptthread benachrichtigt, der sich dann wieder darum kümmern kann.
* Async definiert eine Funktion als asynchron und macht daraus automatisch ein Versprechen, also eine Promise. Await kann nur innerhalb von async Funktionen verwendet werden und wartet bis ein Versprechen eingelöst ist: context:sync ist z.B. ein Versprechen.

**Bibliotheken**

* Office JS
* Frameworks JS
* Frameworks CSS

**Sideload**

* Es gibt verschieden Arten, ein eigenes Add-In zu testen. In Visual Studio Code kann man z.B. ein Tool verwenden, was direkt einen lokalen Server und Excel startet.
* Manchmal möchte man das aber vermeiden, sondern das Add-In separat laufen habeb. Zum Beispiel, wenn man das Add-In auf einem echten Server laufen hat. Und es eben halt noch nicht im Store publiziert hat oder möchte.
* Folgendend nur beispielhaft, wie sowas geht.
* Erster Schritt ist die Freigabe eines lokalen Ordners, zum Beispiel hier in meinem Fall der Ordner Addins-Sideload auf Laufwerk C. Der Code des Add-Ins ist immer noch im Ordner Addins.
* Zweiter Schritt ist dann einfach die Manifest-Datei in den freigegebenen Ordner zu kopieren.
* Danach muss man einmal die Excel-Optionen aufrufen und zum Trust Center wechseln. Und dort den UNC-Pfad bei den Katalogen eintragen. Wichtig, UNC-Pfad, nicht ein normaler Pfad. Und wichtig ist das Häkchen. Man kann mehrere UNC-Pfade eintragen, aber immer nur einen mit einem Haken versehen. Excel muss man dann neu starten.
* Nun ist noch sicherzustellen, dass der Server läuft, zum Beispiel falls das ein lokaler Server ist. Ein solcher sollte dann auf das Rootverzeichnis zum Code des Add-Ins zeigen.
* Schließlich kann man das Add-In installieren wie andere Add-Ins auch. Es taucht dann eine neue Kategorie Geteilter Ordner im Dialog auf.
* Mit entprechenden Tools, z.B. Visual Studio Code plus Yeoman kann man sogar im Code Änderungen vornehmen, die dann direkt im Add-In in Excel übernommen werden.

**Objektmodell**

**Aufbau**

* Wie schon bereits erwähnt, stellt die Office JS API zwei Bestandteile zur Verfügung.
* Eine allen Anwendungen gemeinsame Komponente ist die Common API, kann also in Excel, Word oder PowerPoint angesprochen werden. Die Common API stellt eine ganze Reihe an Funktionen, Klassen und Interfaces zur Verfügung. Interfaces muss man sich als eine Art benutzerdefinierter Datentype vorstellen, der Fähigkeiten einer Klasse besitzt, aber nicht instanziiert (new X geht nicht) werden kann.
* In der Dokumentation sind diese Elemente in drei Gruppen angeordnet: Office, OfficeExtension und OfficeRuntime, die dann die Klassen und Interfaces beinhalten. Beispiele: Office.Context, dass eine Eigenschaft displayLanguage zur Verfügung stellt.
* Weitere Klassen und Interfaces sind dann zum Beispiel für das Menüband, Dialoge, Einstellungen, Authentifizierung, Fehler usw. vorhanden.Dei zweite Komponnente wäre dann die Excel API, die Objekte wie Workbook, Worksheet, Range, Chart, PivotTable, Comment, DataValidation usw. zuer Verfügung stellt.
* Im Unterschied zu VBA werden Collections also z.B. für Worksheets mit dem "s" in VBA nicht durch das "s" ausgezeichnet, sondern explizit benannt, also WorksheetCollection oder ChartCollection.
* Ereignisse sind auch verfügbar, die dann den Präfix "on" tragen, wie z.B. onActivate.
* Oft hat die Office JS API mehr Methoden als VBA.

**Common API**

* Ein Beispiel für die Verwendung der Common API.
* Sprache
* Dokumentmodus
* Konfigurationseinstellungen
* Liefern dann folgendes.

**Excel API**

* Beispiel für das Abfragen des Blattnamens, einer Zelladressen und einer Formel.
* Man beachte "Load", wo man ein oder mehrere Argumente angeben kann.
* Leider habe ich bislang keine Dokumentation zu den Textargumenten von "Load" gefunden.

**Excel API (2)**

* Beispiel für das Anlegen und Abfangen von Ereignissen.
* Hier das Ereignis onSingleClicked, was nur einen Klick abfängt, kein Markieren, zum Beispiel.
* Es gibt natürlich noch mehr, wie z.B. onActivate, onChanged oder onDeactivated, onFiltered, onFormatChanged oder onColumnSorted.
* Office JS bietet dann hier etwas mehr als VBA, wo eine Formatierungsänderung nicht abgefangen werden kann.
* Zu beachten ist übrigens, dass das Ereignis wieder in Excel.run gekapselt wird.

**Script Lab**

**Überblick**

* Script Lab ist Open Source Projekt, was selber in TypeScript implementiert wurde.
* Script Lab ist als Add-In für Excel, PowerPoint, Word und Outlook verfügbar.
* Office 365 wird empfohlen. Script Lab läuft auch unter Office 2013 oder Office 2016/2019. Allerdings ist Script Lab dann etwas eingeschränkt. Beispielsweise laufen dann nicht alle Code-Beispiele und das Menüband wird nicht angezeigt.
* Das Motto von Script Lab ist "Code, Run, Share".
* Code steht für das Implementieren, also JavaScript bzw. TypeScript-Code in Script Lab schreiben.
* Run steht für das Ausführen von dem Code in Script Lab ohne die Anwendung zu verlassen, also z.B. Excel.
* Share steht für das Teilen des Codes bei GitHub, was sinnvoll ist.
* Dadurch, dass jetzt das Erstellen und Ausführen von TS-Code in den Office Desktop Apps möglich ist, kann man recht schnell z.B. Prototypen eigener Add-Ins erstellen oder es halt nur als Lernumgebung verwenden.

**Historie**

* Ursprünglich war Script Lab ein Microsoft Garage Projekt. Im Bild die Macher von Script Lab, die das Projekt größtenteils auch in Ihrer Freizeit gemacht haben; jedenfalls woweit ich weiss. Jakob habe ich schon mal persönlich getroffen. In Amsterdam bei einem MS MVP Treffen. Michael Zlatkovski kenne ich zwar jetzt nicht persönlich, aber Michael hat auch ein Buch zu Office JS geschrieben.
* Script Lab ist in der ersten Version 2017 erschienen. Zunächst nur in Englisch und für Excel, PowerPoint und Word. Ich hatte mich 2017 dem Projekt beteiligt und damals Script Lab in vollständig Deutsch übersetzt. Und auch ein paar Artikel zu Script Lab im Excel-Ticker geschrieben.
* 2018 wurde Script Lab allerdings vollständig neu implementiert. Dabei wurden die von Script Lab verwendeten Bibliotheken ausgetauscht, auch z.B. damit eine bessere Barrierfreiheit gewährleistet werden konnte. Leider hat das aber auch dazu geführt, dass das Handling für alle Übersetzungen hätte neu implementiert werden müssen. Das hat das Team damals zeitlich nicht mehr hinbekommen, weshalb in Script Lab jetzt nur noch die Menübandeinträge übersetzt sind und der Rest in Englisch ist.
* Später wurde dann auch Script Lab für Outlook implementiert, welches übrigens als separates Add-In verfügbar ist.
* Das GitHub Projekt ist hier verfügbar.

**Installation**

* Das ist recht einfach: im Menüband im Reiter Einfügen, in der Gruppe Add-Ins, den Befehl Add-Ins abrufen wählen, dann nach Script (reicht schon) suchen, falls das sowieso nicht schon als Empfehlung erscheint und Hinzufügen wählen.
* Hinweise: das muss pro Anwendung gemacht werden, ggf. auch pro User (also Konto) und pro Sprache, also wenn z.B. Excel von Deutsch auf Englisch umgeschaltet wird.

**Menüband**

* Sobald Script Lab installiert ist (erfolgreich), erscheint ein neuer Reiter im Menüband. Die ersten drei Gruppen sind hier am relevantesten.
* Code ruft das Code-Fenster auf, das als Aufgabenbereich erscheint.
* Ausführen ruft ebenfalls einen Aufgabenbereich auf.
* Funktionen ruft einen speziellen Aufgabenbereich auf, über den sich Custom Functions, also benutzerdefinierte JavaScript-Funktionen erstellen lassen.
* Die weiteren Menüpunkte rufen jeweils nur Links auf. Finde ich jetzt nicht so relevant, da man sich das ja auch im Browser merken kann. Community geht übrigens dabei auf Stack Overflow.
* Richtig prickelnd finde ich Stack Overflow allerdings nicht, dann man wird schon etwas gegängelt, wie ein Beitrag auszusehen hat. Aber MS-Devs antworten da.

**Code**

* Das Code-Fenster - Aufgabenbereich - beinhaltet verschiedene Editierfenster - Script, HTML, CSS und Libraries - die jeweils auch eine Syntaxhervorhebung und IntelliSense bieten. Zeige ich nachher mal in der Demo.
* Methoden, Eigenschaften und Objekte sind dann entsprechend mit einem Symbol gekennzeichnet. Ein Rechtsklick liefert übrigens auch weitere Optionen. Der Editor ist der sogenannte Monaco-Editor.
* Oben ist ein Hamburger Menü zu finden, wo man zu den Beispielen von Script Lab oder den eigenen Scripten gelangt. Zu beachten ist, dass eigene Scripte im Browerscache (müsste der Internet Explorer sein) abgelegt werden. Wird der Cache gelöscht, sind auch die eigenen Scripte weg. Daher dann das Feature zum Teilen bei GitHub.
* Oben kann man dann das Script z.B. ausführen, löschen oder teilen und unten gelangt man zu einigen Optionen
* Insgesamt recht einfach zu bedienen, wie eine Website halt.

**Ausführen**

* Das Ausführen eines Scripts ist ebenfalls ziemlich einfach. Den Menüpunkt im Menüband aufrufen und das eigene Script wird wie eine Website angezeigt.
* Somit sollte auch immer ein Button im Code mit drin sein.
* Die Konsole gibt dann Fehlermeldungen oder eigene Meldungen aus.
* Wenn beide Aufgabenbereiche zum Code und Ausführen aktive sind, versucht der für das Ausführen sich immer synchron zum Code zu halten.

**Funktionen**

* Das ist eine sehr interessante Fähigkeit von Script Lab, die auch erst später, mit dem Erscheinen der Custom Functions Preview, hinzugefügt wurde.
* Wenn Funktionen aufgerufen wird und noch keine Custom Function erstellt wurde, erscheint im Aufgabenbereich ein Hinweis, wie weiter zu verfahren ist. Eine Art Assistent.
* Im Code erfolgt das Erstellen einer CF über eine Markierung durch einen Kommentar direkt über der Deklaration der Funktion. Findet der Ausführen-Aufgabenbereich solche Kommentare, wird die CF registriert und steht direkt in Excel zur Verfügung.
* Die Metadaten werden automatisch erstellt. Diese folgen einen bestimmten XML-Schema für Custom JavaScript Functions.
* Das Löschen kann umständlich sein. Also wenn man welche angelegt hat, Excel schließt und wieder öffnet, sind die meist noch drin.
* Die Anleitungen, die man dazu findet, wie das Script oder den Browser-Cache löschen oder ähnliches, klappen nicht, jedenfalls nicht bei mir.
* Einfachste Möglichkeit, ist einfach folgenden Ordner zu löschen:

C:\Users\Mourad\AppData\Local\ Microsoft\Office\16.0\Wef\CustomFunctions

**Visual Studio Code**

**Überblick**

* Visual Sudio Code ist eine kostenlose Entwicklungsumgebung, die auf Windows, Mac und Linux läuft. Zudem ist VS Code Open Source.
* Viele Sprachen werden von VS Code unterstützt: JavaScript, TypeScript, HTML, CSS, MarkDown, JSON, Python, PHP, usw. VS Code hat Syntaxhighlighting und IntelliSense für die Sprachen.
* VS Code lässt sich mit unendlich vielen Plugins erweitern, ebenso wie um Themes. VS Code ist meiner Ansicht nach bislang das flexibelste, was (kostenlos) zu finden ist.
* Der Aufbau ist recht gewohnt, links Baum mit Dateien, in der Mitte die Editoren. Man kann ein Terminal einschalten, was auch gebraucht wird, debuggen und vieles mehr.

**VS Code & Office JS**

* Um Office Add-Ins zu entwickeln, liesse sich rein theoretisch alles manuell einrichten. Einfacher ist allerdings wie folgt vorzugehen, besonders dann, wenn man noch Einsteiger ist.
* Als erstes braucht man NodeJS, eine Laufzeitumgebung, die auch npm installiert. Npm ist ein Paketmanager.
* Danach VS Code installieren. Müsste auch andersherum gehen, habe ich aber nicht ausprobiert.
* Und anschließend Yeoman installieren, was ermöglicht vorlagenbasiert ein Grundgerüst für ein Add-In zu erstellen.

**Kommandozeile**

* Vieles wird in VS Code mit der Kommandozeile erledigt. Ein Terminalfenster lässt sich mit Strg+Ö aufrufen.
* Beispielsweise kann man npm verwenden, um Tools zu installieren oder einen Server zu starten, so denn eine entsprechende Konfiguration vorhanden ist.
* Hier in dem Beispiel habe ich mal die Kurzanleitung für npm mit npm -h aufgerufen.
* VS Code kennt auch sowas wie Kurzbefehle. Diese erreicht man mit Strg + Umschalt + P. Dann wird oben im Fenster eine Liste der verfügbaren Befehle eingeblendet.

**Yeoman**

* Zum Starten der Entwicklung eines Office JS Add-In empfiehlt sich gerade für Einsteiger das Tool Yeoman, mit dann die Grundstruktur eines Projekts eingerichtet werden kann.
* Yeoman ist natürlich kein Muss, man könnte auch alles manuell enrichten. Yeoman ist dann aber doch etwas bequemer.
* Zunächst muss natürlich Yeoman installiert werden. Dazu ist dann in der Kommandozeile der Befehle npm install -g yo generator-office einzugeben + Enter. Das -g steht für Global, was dann heißt, dass Yeoman dann generell zur Verfügung steht.
* Yeoman ist übrigens Open Source und kann da bei GitHub eingesehen werden.
* Ist Yeoman einmal installiert, kann man ein neues Projekt recht einfach anlegen. Dazu im Terminalfenster eingeben: yo office. Man wird dann per Anleitung zu dem Projekt geführt., zum Beispiel, ob ein Taskpane-Projekt sein soll und mit TypeScript verwendt werden soll. Am Ende legt dann Yeoman das Projekt an.
* Damit das Projekt im richtigen Ordner als Unterordner angelegt wird, am besten auch in der Kommandozeile zuerst zu dem Ordner navigieren, also cd .. und cd Ordnername.

**Starten**

* Um das Add-In in VS Code zu starten, also in Excel anschauen und debuggen, einfach im Terminalfenster npm start eingeben. Dann wird ein Server auf Localhost gestartet, standardmäßig auf dem Port 3000, also localhost:3000. Kann man aber ändern.
* Gleichzeitig startet auch Excel und lädt das Add-In. Das ist letztlich wie ein Sideload, nur etwas bequemer.
* Soll der Server wieder heruntergefahren werden, npm stop eingeben.
* Es gibit im Projekt eine Reihe weiterer Befehle, die letztlich nur Skripte sind und in der Konfiguration nachgeschaut werden können. Das ist übrigens eine Welt für sich, bedarf schon etwas Einarbeitung, wenn man z.B. selber welche erstellen möchte.
* Beispielhaft zwei solcher Befehle. Der erste startet nun den Server. Das Sideload des Add-Ins könnte ich dann z.B. manuell machen. Der zweite validiert das Mainfest.

**Zertifikat**

* Manchmal kann es vorkommen, dass ein HTTPS-Zertifikat nicht automatisch angelegt wird und das Add-In nicht startet. Ein solches ist auch für den Localhost notwendig.
* Die drei hier aufgeführten Befehle machen folgendes: prüfen, ob ein Zertifikat installiert und gültig ist, ein Zertifikat installieren und es deinstallieren. Bei einer Installation erscheint eine Windows-Meldung.
* Nebenbei, dieser Befehl ist meines Wissens anscheinend kaum dokumentiert, ich habe das auch nur in einer Bug-Meldung für Script Lab bei GitHub gefunden.
* Ausserdem hatte ich auch schon mal den Fall, dass bei mir das Add-In nur gestartet war, wenn ich VS Code explizit als Administrator ausgeführt habe.

**Office Scripts**

**Überblick**

* Wie der Name es schon sagt, Office Scripts ist dafür, gedacht Abläufe innerhalb von Office mit Type Script zu automatisieren.
* Derzeit ist Office Scripts noch in einer Preview, also letztlich auch noch in Entwicklung. Für wann eine Freigabe vorgesehen ist, kann ich nicht sagen.
* Ausserdem kann Office Scripts nur Online und nur in Excel verwendet werden.
* Office Scripts hat eine modifizierte Office JS API, was es Otto Normaluser einfacher machen soll, es zu verwenden
* Es gibt zudem Beschränkungen, die sich durch die Plattform ergeben. Maximal 5 MB an Abrufen oder Senden von Daten sowie Bereich mit maximal 5 Millionen Zellen.
* Im Quadranten soll sich Office Scripts dann rechts oben positionieren. Office Scripts wäre dann so etwas wie VBA für das Web.

**Voraussetzungen**

* Um Office Scripts verwenden zu können, muss ein kommerzielles M365 vorhanden sein, also dann die Business Versionen (Business und Business Premium), ProPlus, E3, E5 oder A3 und A4 (Edu-Versionen)
* Damit Office Scripts verwendet werden kann, muss es durch einen Admin der Organisation aktiviert werden.
* Und, wie schon gesagt, Excel Online wird dann benötigt.

**Aktivierung**

* Man muss globaler Admin sein oder zumindestens einer, der die entsprechenden Rechte hat, um Organisationseinstellungen zu ändern.
* Im Admin Center dann Einstellungen, Einstellungen der Organisation wählen und dort die Einstellungen für Office Scripts aufrufen
* Es gibt dann die Möglichkeit, Office Scripts sowie das Teilen für alle Personen oder nur für bestimmte Gruppen zu aktivieren.
* Gegebenenfalls muss ein paar Tage gewartet werden, bis die Einstellung aktiv ist.

**Menüband**

* Sobald Office Scripts für den User aktiv ist, erscheint im Menüband der Reiter Automatisieren. Zu beachten ist allerdings, dass Cookies von Drittanbietern erlaubt sind. Speziell in Edge. Das ist noch ein Bug; hatte ich gemeldet und die sind dran.
* Die Einträge im Reiter Automatisieren sind selbsterklärend. Aufzeichnen, Code und eine Minibibliothek zu den eigenen Scripten und Beispielen. Letztere sind übrigens schreibgeschützt, eine Kopie lässt sich sehr einfach erstellen. Zeige ich in der Demo.

**Recorder**

* Der Recorder funktioniert ähnlich wie der in VBA. Aktionen aufzeichnen anklicken und los geht’s. Es erscheint dann ein Aufgabenbereich rechts im Excel-Browser-Fenster. Leider kann man das nicht abdocken. Dort werden dann die aufgezeichneten Aktionen aufgelistet. Wobei, nicht alles ist in Deutsch korrekt übersetzt.
* Der Recorder erstellt Scripte in Type Script.
* Ist man mit der Aufzeichnung fertig, gelangt man zuerst zu einer Übersicht mit den Eigenschaften des Scripts. Von da aus kann man dann direkt zum Code-Fenster wechseln, was hier abgebildet ist.

**Code**

* Bereits erstellte Scripte lassen sich im Code-Fenster editieren. Ruft man Code aus dem Menüband auf, erscheint erstmal eine Liste der Scripte, wo dann das zu editierende Script ausgewählt werden kann.
* Es kann Type Script oder echtes JavaScript für Office Scripts verwendet werden. Empfehlen würde ich eher Type Script, denn Type Script ist typisiert und näher an VBA dran als JavaScript.
* Das Code-Fenster erscheint ebenfalls als Aufgabenbereich. Und leider lässt sich dieses auch nicht vergrößern oder abdocken. Habe ich ebenfalls MS vorgeschlagen, vielleicht kommt das ja noch.
* Falls es jemanden schon aufgefallen ist, die Folie hat dieselben Stichwörter, wie die Folie zu Script Lab. Liegt daran, dass das Code-Fenster ebenfalls der Monaco-Editor ist. Somit stehen dann auch dieselben Funktionen, wie Syntax-Hervorhebung oder IntelliSense zur Verfügung.
* Zum Code-Fenster lassen sich über das Drei-Punkte-Menü auch Einstellungen aufrufen. Ich hab's hier mal auf Dunkel gestellt.

**API**

* Office Scripts verwenden eine spezielle Version der Office JS API. Diese spezielle Version vermeidet, dass man sich mit Load() und Sync() beschäftigen muss. Es ist also etwas näher an der Einfachheit von VBA dran.
* Es gibt nur einen einzigen Einstiegspunkt in Office Scripts, da erzähle ich gleich mehr dazu.
* Die sogenannte Common API kann nicht verwendet werden, was dann z.B. eine Authentifizierung oder Dialoge innerhalb von Scripten in Office Scripts ausschließt.
* Ereignisse, wie z.B. die Aktivierung eines anderen Arbeitsblatts werden von Office Scripts nicht unterstützt.
* Das Objektmodell ist wie bei VBA Workbook - Worksheets - Ranges usw. und die Hierarchie ist ebenfalls ähnlich. Es gibt dann auch Objekte zu Charts, Tabellen, Pivot usw.

**Einstiegspunkt**

* Die Einstiegsfunktion muss immer **main** heißen.
* Die Einstiegsfunktion muss immer ein Argument vom Typ **ExcelScript.Workbook** haben.
* Hilfsfunktionen, die dann in main aufgrufen werden, sind möglich.

**Dateien**

* Die von Office Scripts erzeugten Dateien werden automatisch bei OneDrive im Dokumentenordner abgelegt.
* Die Scripte mit der Endung osts sind letztlich Textdateien, deren Format einem JSON-Format ähnlich ist, ich meine YAML. Theoretisch liessen die sich auch mit einem Editor bearbeiten, ist aber dann doch umständlich, da z.B. doppelte Anführungszeichen maskiert werden müssten.
* Die Textdateien enthalten auch Metadaten, wie z.B. die Versionsnummer oder Eigenschaften. Für die Abbildung hier rechts habe ich die Backslashes zu jedem " weggenommen, um es besser zu visualisieren.

**Power Automate**

* Office Scripte lassen sich auch in Power Automate ausführen.
* Hier ein sehr einfaches Beispiel zu einem Flow. Erstellt habe ich da einen Direktflow und als Trigger ein manuelles Auslösen gewählt. Danach aus der Aktionsgruppe Excel Online (Business) gewählt und als Aktion Office Scripts gewählt. Speicherort ist dann OneDrive und als Excel-Datei habe ich dann die Datei angegeben, wo Office Scripts was reinschreiben soll.
* Das Office Script habe zuvor geschrieben. Sieht dann so aus.
* Wenn der Flow dann ausgeführt wird, dann wird das Datum und die Uhrzeit in die ExcelDatei geschrieben.