VB6 und die WebView2-Runtime

unter Verwendung des RC6-Frameworks

Exposee

Dieses Dokument beschreibt die Entwicklung einer hybriden Anwendung von der ersten leeren Klasse bis zum Deployment-fertigen Produkt. Das zentrale Element dieses Projektes ist die von Microsoft Ende 2020 veröffentlichte WebView2 Runtime.

Wolfgang Wolf
w.wolf@ww-a.de

Inhaltsverzeichnis

Einleitung

Ende 2020 hat Microsoft WebView2 für Win32 C/C++ veröffentlicht, ein Steuerelement zum Einbetten von Webtechnologien wie HTML, JavaScript und CSS in eigene Desktop-Anwendungen. Laut der damaligen Microsoft-Ankündigung im Windows-Blog, kann WebView2 in jeglicher Win32-Anwendung zum Einsatz kommen. Alle gängigen Windows-Versionen sollen die Runtime unterstützen. Dies bedeutet, dass Windows-App-Entwickler damit einen Zugriff auf die neueste Web-Technologie bekommen und diese in eigene native Desktop-Anwendungen nutzen können. Microsoft nennt das „Hybride Web- und App-Entwicklung“. Gemeint ist damit ein Programm, das auf dem Anwender-PC ausgeführt wird und damit auf alle Funktionen der systemeigenen Plattform Zugriff hat. Die GUI der Anwendung basiert hingegen auf Web-Technologie und kann mit ubiquitären Daten befüllt werden. Desktop und Web verschmelzen und die Bedienung der GUI wird einheitlich. Die konsistente Erfahrung hilft den Anwendern, neue Software oder Programmfeatures schnell zu erlernen und zu verwenden.

Dieses Prinzip wendet Microsoft inzwischen selbst in den eigenen 365 Apps an. Diese stellen neue oder verbesserte Features zur Verfügung, die auf WebView2 aufbauen. Als Beispiel nennt Microsoft die Raum-Suche und die Besprechungseinblicke-Funktionen in Outlook. Wenn Sie diesen Text lesen, wird es mit Sicherheit viele weitere Beispiele geben. Und die werden nicht ausschließlich von Microsoft stammen.

Auf diesem Konzept basierend, wollen wir in diesem Projekt eine eigene native Anwendung von 0 bis 100% programmieren. Auf der letzten Seite angekommen, haben Sie nicht nur eine spannende Technologie kennengelernt, sondern verfügen auch über eine Open-Source-Anwendung, die Sie beliebig erweitern, umbauen oder in eigene Projekte integrieren können. Um das zu ermöglichen, wird die Anwendung in unterschiedliche Schichten aufgeteilt, die eine strenge Trennung der einzelnen Komponenten vorsieht. Diese Schichten werden nach dem MVP (Model View Presenter) miteinander verbunden. Der Presenter ist das Bindeglied zwischen Model und View, also den Daten und dem User-Interface. Letztere erfahren so eine strikte Trennung voneinander. Sie lassen sich getrennt testen und können durch kompatible Module ersetzt oder ergänzt werden. Sollten Sie diese Anwendung in ein eigenes Projekt einbauen wollen, müssen Sie nicht alles neu programmieren. Über die CSS-Definition der GUI passen Sie die Anwendung an das Look and Feel Ihrer eigenen Anwendungen an. Auf der Model-Seite können Sie Ihre eigenen Datenquellen anbinden oder mittels Import- und Export-Funktionen Daten mit anderer Software austauschen. Zu guter Letzt kann das Programm auf einen Intranet-Server installiert werden und liefert von dort den Urlaubsplan als kleiner Web-Server an alle angeschlossenen Clients (Desktops & Smartphones) aus.

Der Aufbau

Das Buch untergliedert sich in mehrere Abschnitte, die aufeinander aufbauen. Somit macht es Sinn den Text chronologisch aufzuarbeiten. Davon ausgenommen sind die hinteren Referenz-Kapitel, in denen Klassen und deren Member sortiert nach Eigenschaften, Methoden und Ereignisse aufgeführt sind. Die Referenzen können Sie auch als Nachschlagtext verwenden.

Wenn es sich um eine Auftragsarbeit handelt, steht am Anfang eines Software-Projektes meistens ein Pflichten- und ein Lastenheft. Unter anderem wird hier in einer, für alle Beteiligten gemeinsam verständlichen Sprache, die Anwendung beschrieben. Wir machen das hier im Buch vereinfacht in der Form einer Anwendungsbeschreibung. Wir fangen damit an womit viele Projekte enden: Mit dem

Programm-Benutzerhandbuch. Damit wird dem Leser das Zielvorhaben vorgestellt und er weiß zu jedem Zeitpunkt, wo der Weg hingeht. Später kann dieser Text als Vorlage in ein eigenes Benutzerhandbuch einfließen.

Damit der Tatendrang des Entwicklers bald bedient wird, steigen wir mit einem Minimalbeispiel in die WebView2-Komponente ein. In Nullkommanichts entwickeln wir unseren ersten Browser auf der Chromium-Basis.

Jetzt noch mal ein bisschen Theorie: Das Entwurfsmuster MVP bietet uns einen Leitfaden für das Skelet unserer Anwendung. Wer ist was? Und wie geht was zusammen? Kurz und knackig, weil das auch nur eine mögliche Vorgehensweise ist. Sie können später auch was Anderes verwenden. Programmarchitekturen gibt es viele und es ist mühselig darüber zu streiten, welche besser ist.

Nun kommen wir zu den einzelnen Komponenten. Wir starten mit dem Presenter, weil diese Komponente auch das Bindeglied zu den anderen Komponenten ist. Auch hier gilt: Man könnte es auch anders machen. Zum Beispiel zuerst das Model entwickeln und testen. Danach die View entwickeln und testen. Am Schluss alles mit dem Presenter zusammenfügen. Ich entscheide mich hier für den ersten Weg, weil der Programmstart zuerst den Presenter initialisiert. Dieser bindet die anderen Komponenten danach an sich. Somit haben wir von Anfang an, wenn der Ausführen-Button der IDE gedrückt wird, was zum „Anfassen“.

Ohne Daten ist eine Anwendung wie ein Haus ohne Bewohner. Das Model sorgt für die nötige Belebung, angetrieben im Herz durch SQLite, ein kleines, aber feines relationales Datenbanksystem. Die im RC6-Framework integrierte Programmbibliothek des DBMS kann optional die Daten auch persistieren oder an irgendeinen großen Bruder (ERP, PPS, HR-Software) weiterreichen. Wir verwenden hier eigenständige Dateien, weil die Software in sich autonom und ohne externe Daten-Abhängigkeiten lauffähig sein soll.

Nun kommen die Views, also das was der Anwender sieht, wo er seine Daten eingibt und auswertet. Wir werden mehrere davon implementieren. Alle werden sich an gemeinsamen Interfaces orientieren, im Fachjargon: sie implementieren die gleichen Schnittstellen. Die Präsentationsmöglichkeiten sind vielfältig. Neben einer Desktop-Anzeige sämtlicher Fenster für Daten und Konfiguration gibt es Export-Views für diverse Formate (HTML, PDF, CSV, PNG) und Endgeräte (Drucker, Browser, Smartphone). Wir können es auf die Spitze treiben und unser Programm als kleinen Web-Server im lokalen Intranet laufen lassen. Alles ohne weitere Abhängigkeiten. Damit bedienen wir Browser- und Smartphone-User, die die Software noch nicht mal installiert haben – ein Web-Link genügt!

Letzten Endes muss die fertige Anwendung noch zum Anwender. Wir zeigen hier, wie das ohne sperrige Installationsroutine geht. Einfach nur ein Copy & Paste auf den Anwenderrechner durchführen – fertig!

Den Schluss machen die diversen Referenz-Kapitel, in denen wir die hier verwendeten Klassen und deren Member aus der WebView2-Runtime und dem RC6-Framework auflisten und dokumentieren. In der Regel gibt es für jede Eigenschaft, Methode und jedes Ereignis eine kurze Beschreibung und ein Quellcode-Beispiel.

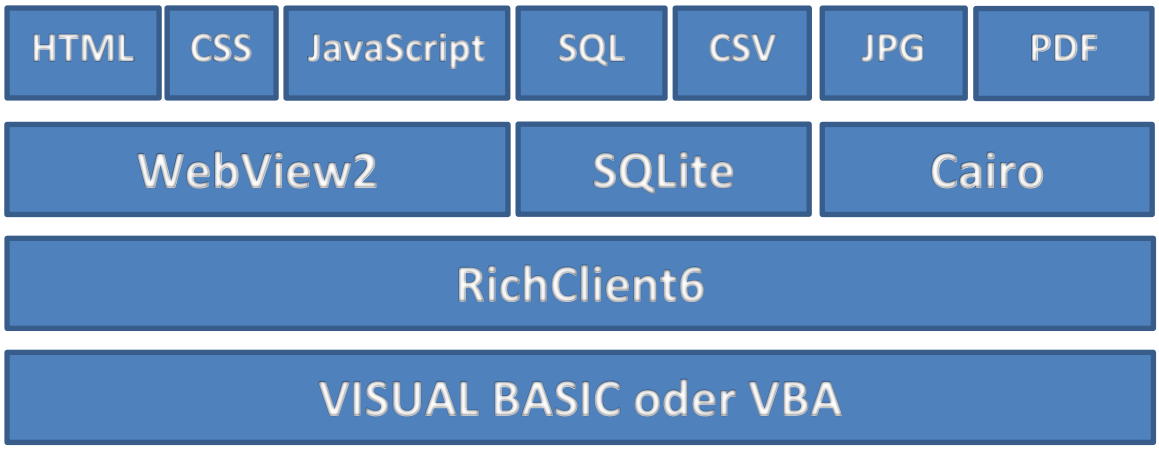
Die Zielgruppe

Dieser Text ist nicht für Einsteiger gedacht. Er setzt aber auch kein fundiertes Profi-Wissen voraus. Bestimmte technologische Basis-Skills sollte der Leser schon mitbringen. Wenn das Grundwissen von Programmierung (VB oder VBA), Web-Design (HTML, CSS, JavaScript) und Datenbanken (SQL) vorhanden ist, wird Sie diese Lektüre nicht überfordern. Als Ausbilder für Fachinformatiker würde ich sagen: Jede und jeder Auszubildende in diesem Fach wird damit klarkommen.

Auch wenn wir über alle verwendeten Technologien sprechen werden, ist das kein Lehrbuch über eine bestimmte Programmiersprache oder Architektur. Wir steigen so tief wie nötig in die Themen ein, beschränken uns aber auf das Wesentliche. Andererseits bietet der Text auch Hintergrundwissen und Best Practices aus der Praxis des Autors. Und manchmal mag etwas auch übererklärt erscheinen, das ist der Tatsache geschuldet, dass der Autor unter anderem auch als Ausbilder tätig ist… 😉.

Die Technologie

In der Einleitung haben wir bereits einige technologische Fachbegriffe verwendet. Nun wollen wir Ordnung in die Begriffe bringen und die Zusammenhänge visualisieren. Folgender technologischer Turm erwartet Sie:



Urlaub?!

Wie planen Sie in Ihrer Abteilung Ihren Urlaub? Auf welche Weise stimmen Sie sich mit Kollegen ab? Auf einem Werbe-Wandkalender mit unterschiedlich farbigen Textmarkern? Oder haben Sie sich eine Kalendervorlage für Excel aus dem Internet besorgt?

Selbst in kleinen Teams oder Abteilungen ist eine Planung des Jahresurlaubs sinnvoll, ja meistens sogar von der Personalabteilung oder Geschäftsleitung erwünscht. Dabei soll in der Regel zumindest der Jahresurlaub verplant werden. Aber bitte so, dass die Abteilung weiterhin reibungslos arbeitsfähig bleibt. Und es versteht sich von selbst, dass gesetzliche Bestimmungen aus dem Urlaubsrecht (BurlG) eingehalten werden. Darüber hinaus sollen die Wünsche und Bedürfnisse der Kolleginnen und Kollegen berücksichtigt werden – und auch die des Betriebes. Ein Beispiel dafür: Wenn der einzige Arzt einer Praxis verreist ist, haben die medizinischen Fachangestellten (Arzthelferinnen) nichts zu tun. Arztpraxen regeln das über einen gemeinsamen Betriebsurlaub. Ein weiteres Beispiel: Da in christlich geprägten Kirchen manche Feiertage immer an einem Donnerstag liegen und andere ganz zufällig auch auf die unmittelbare Nähe des Wochenendes fallen, entstehen sogenannte Brückentage, an denen auch gerne der Betrieb einvernehmlich ruht. Nicht zu vergessen sind Schulferien, die bei der Urlaubsplanung der Angestellten eine entscheidende Rolle spielen. Puh, das braucht viele unterschiedliche Textmarker und ohne VBA-Programmierung macht Excel hier auch wenig Spaß.

Schluss mit Papierkram und langen Excel-Listen! Wir programmieren uns einen einfach zu bedienenden, übersichtlichen und flexiblen Urlaubsplaner. Der soll in der Anwendung intuitiv sein, sich übersichtlich präsentieren wie ein großer Wandkalender und die Feiertage und Ferien unterschiedlicher Bundesländer berücksichtigen. Zwar ist hier bereits das Tun-Wort „Programmieren“ gefallen, vorerst schauen wir uns das zu entwickelnde Programm lediglich aus der Anwenderperspektive an. Wir beschreiben hier das Werk, wie in einem Benutzerhandbuch oder einem Pflichtenheft eines Auftraggebers. Was soll also unser Urlaubsplaner alles können? Fassen wir erst mal die primären Anforderungen zusammen.

Ein Urlaubsplaner ist ein Kalender, in dem bestimmte Tage unterschiedlichen Personen und Ereignissen zugeordnet werden. Wir brauchen also:

- Einen Kalender, genaugenommen einen Jahreskalender
- Eine Personenverwaltung
- Funktionen für die Festlegung von Ereignissen wie Feiertage, Brückentage, Ferientage und einen optionalen Betriebsurlaub

Eine Software ohne Standardfunktionen wie Speichern und Lesen von Daten oder ohne Druckfunktion wäre ziemlich nutzlos. Es sei denn, wir wollen explizit eine in der Funktion eingeschränkte Demo programmieren. Also erweitern wir die Liste mit diesen Klassikern und ein paar weiteren nützlichen Dingen:

- Speichern, Laden und Drucken
- Import und Export von Daten

Schauen wir uns diese Punkte einzeln an, um sie genauer zu analysieren und spezifizieren.

Der Kalender

Ein Jahreskalender besteht aus zwölf Monaten, Wochen und Tagen. Das Urlaubsrecht unterscheidet die Wochentage in Werktage und den Sonntag. Zwar basiert diese Regelung auf der Basis einer Sechs-Tage-Woche, jedoch gilt für die meisten Arbeitnehmer eine fünftägige Arbeitswoche. Machen wir es uns einfach und sprechen von fünf Arbeitstagen und dem Wochenende bestehend aus Samstag und Sonntag. Diese gängige Praxis bestätigt Ihnen einen Blick auf einen beliebigen Wand- oder Tischkalender, in denen die Wochenenden hervorgehoben sind. In der Regel sind Tischkalender mehrseitig, je eine Seite pro Monat. Die Wandkalender umfassen ein ganzes Jahr auf einem Papierbogen. Gerade wenn es um die Jahresurlaubsplanung geht, bieten letztere die bessere Übersicht, weil man das ganze Jahr im Blickfeld hat. Da kommt was auf uns zu: 365 Tage und eine Abteilung, die zum Beispiel aus 20 Kolleginnen und Kollegen besteht, ergäbe eine Matrix aus 7.300 Feldern. Und das soll übersichtlich auf einem 22-Zoll Bildschirm dargestellt werden?!

Die Personenverwaltung

Wenn wir die Urlaubsplanung für eine einzelne Person machen würden, könnten wir uns das Programm sparen. Aber so ist es ja nicht. Wir wollen zumindest eine kleine Abteilung bedienen. Auf dem Kalender sollen die Urlaubstage dieser Personen unterscheidbar sein. Der vollständige Name scheidet aus Platzgründen aus. Also werden wir uns zumindest im Kalender auf die Initialen der Personen beschränken. Eine noch bessere Abgrenzung erreichen wir, wenn zusätzlich individuelle Hintergrundfarben für die einzelnen Personen zum Einsatz kommen. Unsere minimalistische Personenverwaltung wird aus den folgenden Attributen bestehen:

- Name und Vorname
- Initialen
- Hintergrundfarbe
- Schulpflichtige Kinder
- Jahres-Urlaubsanspruch

Durch die Festlegung des Urlaubsanspruchs kann unsere Software erkennen, wie viele Urlaubstage bereits verplant sind. Somit kann uns das Programm warnen, wenn wir mehr verplanen als der Person zusteht.

Ereignisse

Neben arbeitsfreien Wochenenden kennt das Urlaubsgesetz auch noch gesetzliche Feiertage, an denen Arbeitnehmer für gewöhnlich nicht zur Arbeit müssen. Diese werden nicht vom Urlaubsanspruch abgezogen. Allerdings kommt es darauf an in welchem Bundesland Sie, beziehungsweise Ihr Arbeitsort beheimatet ist. Wir werden uns auf je ein Bundesland pro Kalenderdatei beschränken. Somit gelten für alle Personen eines Urlaubsplaners einheitliche Feiertage.

Und weil wir schon bei den Bundesländern sind, hier gibt es nicht nur unterschiedliche Feiertage, sondern auch unterschiedliche Schulferien. Zwar haben diese keinen direkten Einfluss auf den Urlaub der Arbeitnehmer (abgesehen von Auszubildenden), sie sollten dennoch im Auge behalten werden. Eltern sind auf Urlaub zu bestimmten Zeiten angewiesen. Arbeitgeber müssen das berücksichtigen, auch wenn es keine Garantie für alle individuellen Wünsche gibt.

Als Brückentag bezeichnet man in der Arbeitswelt Arbeitstage, die zwischen einem Feiertag und einem ohnehin arbeitsfreien Tag (Wochenende) liegen. Das sind gewöhnlich Montage und Freitage oder die Tage zwischen Weihnachten und Neujahr. Deshalb nutzen sie viele Menschen, um sich von der Arbeit frei zu nehmen. Auch in Betrieben werden solche Tage gerne als Betriebsferien genutzt. Es gibt noch weitere Gründe, die einen Betriebsurlaub rechtfertigen. Auf jeden Fall werden diese Tage vom Urlaubsanspruch abgezogen, sie sind Teil des Jahresurlaubs.

Zu diesen allgemeinen Ereignistagen gesellen sich nun noch die individuellen Urlaubstage der jeweiligen Abteilungsmitglieder. All das zusammen, wollen wir in unserem Kalender (übersichtlich) visualisieren.

Speichern, Laden und Drucken

Auf diese Standard-Funktionen wollen wir hier nicht großartig eingehen. Das sind Funktionen, wie es sie in jeder Software gibt. Vielleicht ein paar kleine Präzisierungen, die speziell für unseren Urlaubskalender gelten:

- Wir wollen in Einzeldateien speichern. Eine Anbindung an eine große Datenbank wäre jederzeit möglich, wird hier in diesem Kontext jedoch nicht berücksichtigt.
- Eine Datei enthält einen Jahreskalender eines bestimmten Jahres. Darin enthalten sind alle Daten, die zum Urlaubsplaner gehören: Kalender, Personen und Ereignistage.
- Es kann immer nur eine Datei geöffnet werden. Wenn gleichzeitig mehrere Dateien zu bearbeiten sind, ist das Programm entsprechend der Anzahl mehrfach zu starten.

Import und Export

Die Idee beim Import ist die, dass in einem Folgejahr bestimmte Daten wie z. B. die Personen-Stammdaten aus einem vergangenen Jahr übernommen werden. Das erspart dem Anwender die wiederholte Erfassung dieser Daten.

Beim Export sollen die Personen und deren geplanten Urlaubstage in ein universelles und maschinenlesbares Format gebracht werden. Diese Daten können von einer übergeordneten HR-Software dann gelesen und importiert werden.

Let's go?

Nun haben wir bereits grob umrissen, welche Daten die Software verwalten soll. Und von dem Wie haben wir auch eine Vorstellung. Der eine oder andere eifrige Leser hat gedanklich schon ein Datenbankmodel oder eine Softwarearchitektur angelegt. Das sind Entwickler mit Erfahrung in verschiedenen Software-Technologien. Dazu gehören verschiedene Programmiersprachen, Frameworks und Runtimes, Softwarearchitekturen und Vorgehensmodelle und die Arbeit mit einer oder mehreren IDEs. Die Unterschiede in diesen Technologien können erheblich sein, ein VB-Entwickler wird zum Beispiel JavaScript wie ein Spanier die portugiesische Sprache sehen und verstehen. Dennoch gibt es Gemeinsamkeiten. Um Software zu erstellen und zu warten, braucht man einen Plan, ein Konzept. In der Praxis wird der Entwicklungsprozess in überschaubare Phasen unterteilt. Die Software wird somit Schritt für Schritt fertiggestellt. Durch die Festlegung der groben Anforderungen im vorherigen Kapitel, haben wir bereits so was wie die Anforderungsphase aus einem Wasserfallmodel eingeleitet. In diesem Modell würden wir die Anforderungen jetzt nach und nach präzisieren und verfeinern. Uns also Gedanken machen, wie die Software aussieht und sich verhalten soll. Wenn diese Phase abgeschlossen ist, würden wir mit dem Entwurf beginnen, auch diese Phase erst grob skizzieren und danach verfeinern. Es folgen weitere Phasen wie Implementation, Test und Wartung. Dieses Modell ist unflexibel und seit einigen Jahren veraltet. Heute wird mit wesentlich agileren Methoden, wie zum Beispiel mit dem Spiralmodell entwickelt. Hier wird in einem wiederkehrenden Zyklus gearbeitet. Auch die Software zu diesem Buch ist (ungefähr) so entstanden.

Wie aber schreibt man ein Buch? Kann man hier auch so was wie Vorgehensmodelle anwenden? Softwarearchitekten und -entwickler verfügen oft über einen akademischen Hintergrund und tendieren zum wissenschaftlichen Arbeiten. Bei Programmhandbüchern (die jüngeren Leser werden sich fragen: Hä? Was ist das?), die für gewöhnlich als Nachschlagwerke verwendet werden ist das ok. Hier wird der Zustand und das Wirken einer Software beschrieben, nachdem die Software fertiggestellt ist. Der Autor weiß also schon präzise was die Software kann und wie man sie bedient. Ich habe schon einige solcher Texte geschrieben und kenne mich damit aus. Zugegeben: bevor ich den ersten Satz in diesem Werk geschrieben habe, war die Software dazu schon fast fertig. Hier so zu tun, als ob ich bei bestimmten Fragen noch Alternativen abwägen würde (Wasserfall- oder Spiralmodell?), ist also geschummelt. Die meisten Entscheidungen sind schon gefallen. Was spricht also dagegen, hier ein Kapitel einzubauen, das wie ein Handbuch aufgebaut ist? Ein Kapitel in dem sich der Leser einen präzisen Eindruck von der Software und einem browserbasiertem Bedienkonzept machen kann. Dabei interessiert uns nicht, wo und wie die Daten gespeichert werden, ebenso wenig mit welcher Programmiersprache die UI entsteht.

Ein Urlaubsplaner aus Anwendersicht

Wie in Windows üblich, startet der Urlaubsplaner per Doppelklick auf die Programmdatei, also direkt auf die Exe oder eine Programmverknüpfung. Letztere hat den Vorteil, dass wir komfortabel Befehlszeilenargumente an den Programmpfad anhängen können. Als mögliches Argument wäre zum Beispiel eine Urlaubsplaner-Datei möglich, die sofort beim Programmstart geöffnet wird.

Die GUI des Urlaubsplaners basiert auf der Microsoft WebWiev2-Runtime, die auf Ihrem System vorhanden sein muss. Zwar bietet Microsoft inzwischen automatische Installationen für Windows 10. Das muss aber nicht heißen, dass die Runtime schon überall verfügbar ist. Deshalb wird beim Programmstart die Verfügbarkeit der Runtime geprüft. Sollte diese Prüfung negativ ausfallen, dann

bietet das Programm an, die Runtime von der offiziellen Microsoft-Seite herunterzuladen und zu installieren. Wenn Sie das nicht wollen, bricht das Programm an dieser Stelle ab und wird beendet. Wenn Sie der Installation zustimmen, werden die Installationsdateien heruntergeladen und ausgeführt. Der Urlaubsplaner wartet im Hintergrund auf das Installationsende. Sobald das erreicht ist, wird der Urlaubsplaner fortgeführt.

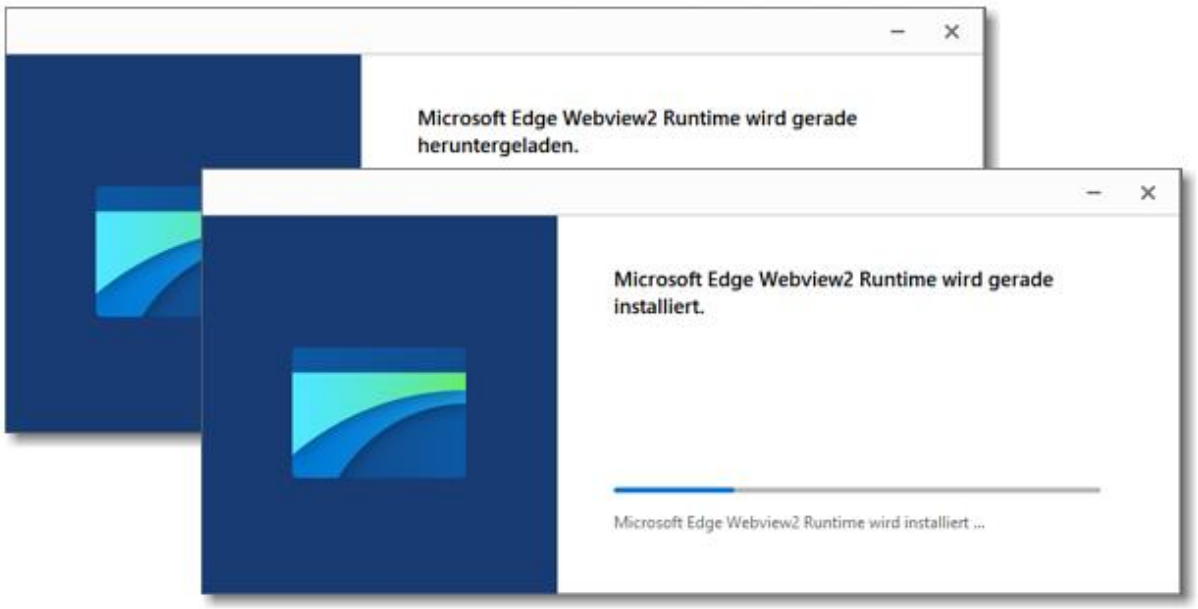


Abbildung 1: Webview2Installation

Das Programm bietet Ihnen beim ersten Start die Möglichkeit, einen neuen Urlaubskalender anzulegen. Dafür braucht es zwei Dinge: das Kalenderjahr und das Bundesland, für das der Urlaubsplaner gelten soll. Ein Urlaubskalender gilt immer für ein einzelnes Jahr und ein bestimmtes Bundesland. Wenn Sie mehrere Jahre berücksichtigen wollen, müssen Sie unterschiedliche Dateien anlegen. Das Bundesland dient lediglich zum Vorbelegen der gesetzlichen Feiertage im Kalender. Dabei werden nur die deutschen Bundesländer berücksichtigt. Sollten Sie das Programm in einem anderen Land, z. B. in Österreich oder in der Schweiz verwenden wollen, ist das auch kein Problem. Viele Feiertage sind im christlich geprägten Europa gleich. Andererseits können Sie die vorab eingetragenen Feiertage beliebig ändern oder ganz darauf verzichten. Einstellbar ist das in den Programm-Optionen.

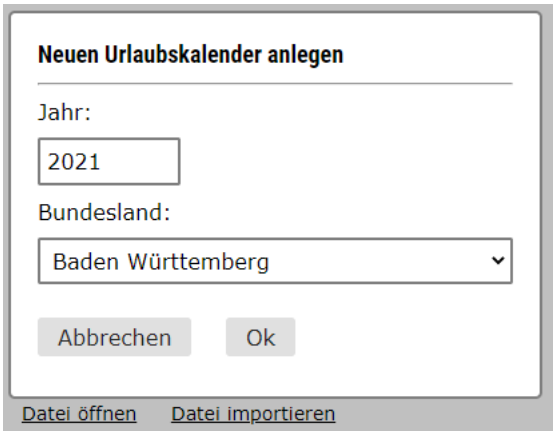


Abbildung 2: Neuen Urlaubskalender anlegen

Alternativ zur Neuanlage können Sie eine bereits vorhandene Datei öffnen. Klicken Sie auf *Datei öffnen* und wählen Sie eine up-Datei von einem Datenträger. Es empfiehlt sich diese Dateien an einen zentralen Ort zu speichern, weil sich das Programm den Pfad zu diesem Ordner merkt und beim nächsten Mal wieder vorschlägt.

Eine weitere Möglichkeit, wäre eine Neuanlage mit Datenimport. Das ist eine Kombination aus den zuvor beschriebenen Fällen. Es wird ein neuer Urlaubskalender angelegt, einige Daten wie z. B. Personen und Einstellungen werden von einer vorhandenen Datei übernommen. Das ist immer dann günstig, wenn Sie eine Folgedatei für ein neues Jahr anlegen wollen. Dann sparen Sie sich schon die Eingabe der Personendaten und die Schulferien werden auch aus dem Vorjahr übernommen. Sie müssen später nur noch die jeweiligen kalendarischen Daten, gemäß den Bestimmungen der Kultusministerkonferenz eintragen. Diese finden Sie im Internet z. B. auf https://www.kmk.org/service/ferien.html für Deutschland. Für Österreich und die Schweiz gelten natürlich deren Regelungen. Nach einem abschließenden Klick auf den Schalter Ok, startet der Urlaubsplaner mit einem vorkonfigurierten Jahreskalender, in dem die Feiertage gemäß dem gewählten Bundesland bereits eingetragen sind.

An den Feiertagsregelungen kann sich eventuell mal was ändern. Deshalb ist es sinnvoll diese zu prüfen und ggf. zu korrigieren. Das können Sie über die Programm-Optionen einstellen. Weil weitere Einstellungen beim Start eines neuen Kalenders sinnvoll sind, widmen wir uns sofort dieser Funktion. Klicken Sie dafür auf den Menü-Schalter , das Menü wird eingeblendet. Wählen Sie hier den Menüpunkt *Optionen*.

Optionen

Die Programm-Optionen sind in folgende Kategorien eingeteilt:

Slots

Hier sehen Sie die Anzahl der aktuell verwendeten Slots und können diese einstellen. Slots sind Spalten im Kalender, in die die Urlaubstage eingetragen werden. Sie brauchen in der Regel nicht so viele Slots wie Personen in im Urlaubsplaner verwaltet werden, weil nie alle an einem Tag Urlaub haben. Es sei denn, es gibt im Unternehmen einen Betriebsurlaub oder Brückentage, an denen der gesamte Betrieb ruht. Diese Tage gelten aber für alle Beteiligten und werden deshalb zentral verwaltet. Sie können ruhig mit wenigen Slots starten, und diese nach Bedarf erhöhen.

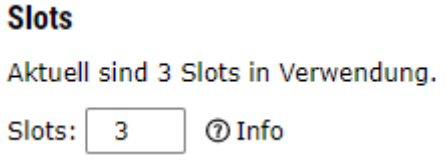


Abbildung 3: Anzahl der Slots festlegen

Farben

Über die Skin-Liste können Sie das Erscheinungsbild der Benutzeroberfläche (GUI) festlegen. Es stehen die Optionen *Hell* und *Dunkel* zur Verfügung. Anmerkung: Die Einstellung greift erst nach dem Speichern.

Hier können Sie eine Hintergrundfarbe für Feiertage, Ferien und den Betriebsurlaub einstellen. Klicken Sie dafür auf eines der farbigen Rechtecke. Es öffnet sich ein RGB-Farbfenster, in dem Sie mit der Maus

eine beliebige Farbe auswählen können. Alternativ dazu können Sie Farbwerte für die drei Grundfarben Rot, Grün und Blau getrennt festlegen. Sie dürfen sich getrost auch dunkle Farbtöne aussuchen. Das Programm erkennt diese Töne und wählt entsprechend eine kontrastreiche Vordergrundfarbe für den Text.

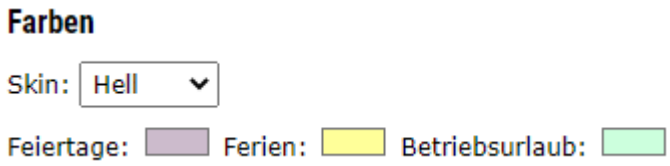


Abbildung 4: Farben einstellen

Feiertage

Bei der Anlage des Urlaubsplaners haben Sie sich für ein deutsches Bundesland entschieden. Damit hat das Programm bereits die gesetzlichen Feiertage berechnet. Nun kann es sein, dass sich kurzfristige Änderungen in Ihrem Bundesland ergeben, die in der aktuellen Programmversion nicht berücksichtigt sind. Oder Sie planen für ein anderes europäisches Land, in dem andere Regeln gelten. Deshalb können Sie in diesem Abschnitt die Feiertage beliebig ändern, löschen oder neue hinzufügen Natürlich dürfen Sie die Feiertage auch beliebig umbenennen. *Import* ermöglicht Ihnen das Einlesen von Feiertagen aus einer anderen Ulaubsplaner-Datei.

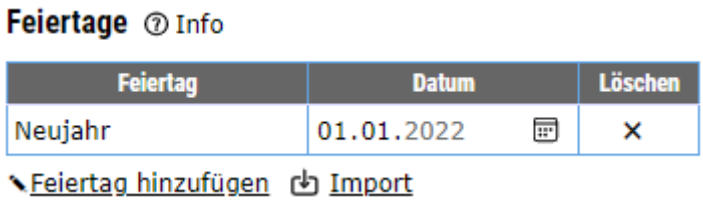


Abbildung 5: Feiertage festlegen

Schulferien

Bei kollidierenden Urlaubswünschen mehrerer Beschäftigten müssen Arbeitgeber unter sozialen Gesichtspunkten entscheiden. Dafür sind unter anderem auch die Schulferien der Kinder relevant. In dieser Tabelle hat das Programm die üblichen Ferien für Deutschland vorbelegt, allerdings ohne Datum. Beginn und Ende der Ferien bestimmt in Deutschland die Kultusministerkonferenz. In Österreich und in der Schweiz das Bundesministerium für Bildung bzw. die Konferenz der kantonalen Erziehungsdirektoren. Die Links zu den offiziellen Stellen finden Sie hier:

- Deutschland: https://www.kmk.org/service/ferien.html
- Österreich: https://www.bmbwf.gv.at/Themen/schule/schulpraxis/termine.html
- Schweiz: https://www.edk.ch/de/bildungssystem/ides

Ergänzen Sie die Von- und Bis-Spalte für die Ferientage in Ihrer Region. Einige Bundesländer haben neben den zusammenhängenden Ferientagen auch noch einzelne schulfreie Tage im Kalender. Diese können Sie nach Belieben nachtragen. Sie müssen dabei nicht auf eine chronologische Reihenfolge achten. Fügen Sie zusätzliche Ferientage einfach unten an. Nach dem Speichern ordnet das Programm diese Tage nach dem Von-Datum neu. Bei einzelnen Ferientagen tragen Sie bei Von und Bis jeweils das gleiche Datum ein. Verwenden Sie dafür Ihre Tastatur oder über das Kalender-Icon den eingeblendeten Kalender. Bitte beachten Sie, dass alle verwendeten Felder ein gültiges Datum brauchen, weil sonst

ein Speichern nicht möglich ist. Wenn Sie auf bestimmte oder alle Ferien verzichten wollen, müssen Sie alle nicht verwendeten Zeilen löschen. *Import* ermöglicht Ihnen das Einlesen von Schulferien aus einer anderen Ulaubsplaner-Datei.

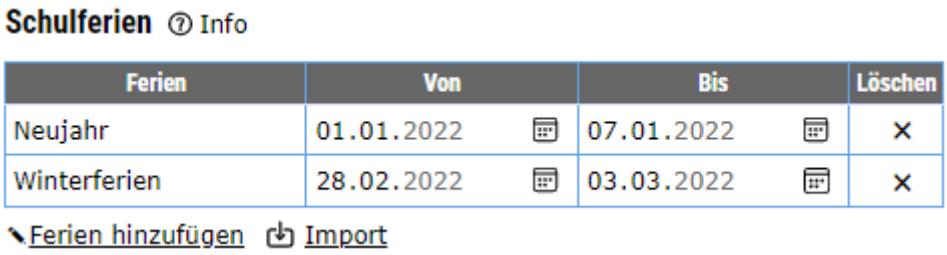


Abbildung 6: Schulferien erfassen

Betriebsurlaub und Brückentage

Arbeitstage, die zwischen einem Feiertag und einem ohnehin arbeitsfreien Tag (Wochenende) liegen, werden als Brückentage bezeichnet. Solche Tage werden in Betrieben gerne als Betriebsferien genutzt. Um nicht für alle Personen diese Tage einzeln im Urlaubsplaner eintragen zu müssen, können Sie diese hier erfassen. Dabei gelten die gleichen Regeln wie bei den Schulferien: Jede angelegte Zeile braucht ein Von- und ein Bis-Datum. Handelt es sich dabei um einen einzelnen Tag, dann ist in beide Felder das gleiche Datum einzutragen. Auch hier können Sie wahlweise mit der Tastatur oder mit dem Hilfs-Kalender neben dem Datumsfeld arbeiten. *Import* ermöglicht Ihnen das Einlesen von Betriebsurlaub und Brückentage aus einer anderen Ulaubsplaner-Datei.

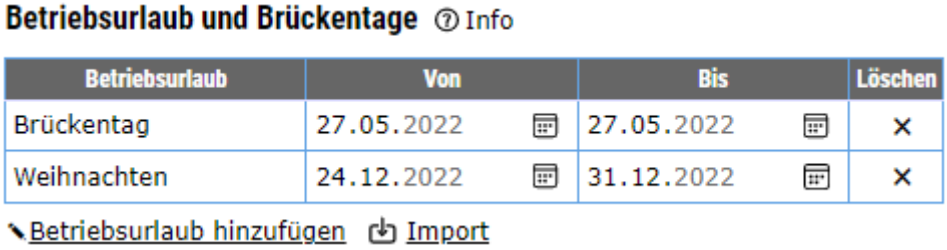


Abbildung 7: Betriebsurlaub festlegen

Export

Sollen die Daten aus dem Urlaubsplaner in ein anderes Programm überführt werden oder auf einem Datenträger in einem allgemein lesbaren Format abgespeichert werden, bietet die Software folgende Optionen:

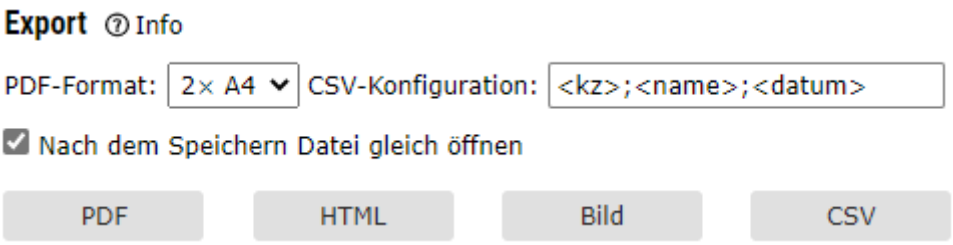


Abbildung 8: Daten exportieren

PDF

Das PDF (*Portable Document Format*) ist als plattformunabhängiges Dateiformat auf allen Betriebs-systemen lesbar, sofern dort ein PDF-Reader verfügbar ist. Darüber hinaus eignet sich dieses Format,

um den Kalender komfortabel aufs Papier zu bringen. Zwar hat die Software auch eine eigene Druckfunktion, über die verschiedenen PDF-Formate erhalten Sie jedoch mehr Gestaltungsmöglichkeiten. So können Sie ein PDF auch im DIN-A3-Format erzeugen, selbst wenn Sie keinen eigenen A3-Drucker haben. Das erzeugte PDF ist später leicht auf einem entfernten A3-Drucker ausdruckbar. Eine weitere Option ist das Aufteilen des Kalenders auf zwei Seiten, wahlweise 2x DIN-A4-Seiten im Querformat oder 2x DIN-A3-Seiten im Hochformat. Bei diesen Varianten können Sie nach dem Ausdruck die beiden Seiten zusammenkleben und bekommen so einen übersichtlichen (Wand-) Kalender. Diese Funktion ist vor allem dann nützlich, wenn Sie mehrere Slots verwenden.

HTML
Damit erzeugen Sie eine HTML-Seite, die in jedem Browser angezeigt werden kann. Optional können Sie diese Datei über einen Web-Server z. B. im Intranet anbieten. Die Seite ist nach responsiven Techniken aufgebaut. Damit kann sie auf unterschiedlichen Endgeräten, unter anderen auch auf Smartphones und Tablet-Computern komfortabel angezeigt werden.

Bild
Diese Option erzeugt ein Bild. Als Ausgabeformat können Sie zwischen JPG (JPEG) und PNG (Portable Network Graphics) auswählen.

CSV/TSV
Das C(T)SV-Format ist eine Textdatei mit strukturierten Daten, die sich zum Import in andere EDV-Systeme (HR- oder ERP-Programme) eignet. In der Konfiguration können Sie die Feldanordnung und das Feldtrennzeichen anpassen. Als Feldtrennzeichen wird meistens das Tabulatorzeichen oder ein Semikolon verwendet.

Beispiel für TSV (Tab-separated values): `<datum>{tab}<kz>{tab}<name>`
Beispiel für Semikolon (direkt in Excel zu öffnen): `<kz>;<name>;<datum>`

Für alle Export-Varianten sorgt die ausgewählte Checkbox „*Nach dem Speichern Datei gleich öffnen*“ dafür, dass die exportierte Datei sofort in einem zum Format passenden Programm angezeigt wird.

Integrierte Hilfe
In jeder Kategorie können Sie einen integrierten Hilfe-Text einblenden. Klicken Sie dazu auf den Fragezeichen-Schalter der jeweiligen Kategorie. Ein Hilfetext erscheint im oberen Fensterbereich. Diesen können Sie über das kleine x wieder ausblenden. An gleicher Stelle erscheint auch eine Fehlermeldung, wenn die eingegebenen Daten nicht vollständig oder nicht konsistent sind. Beachten Sie, es gilt: Entweder Alles oder Nichts speichern. So lange noch Fehler in den Daten enthalten sind werden Änderungen nicht übernommen.

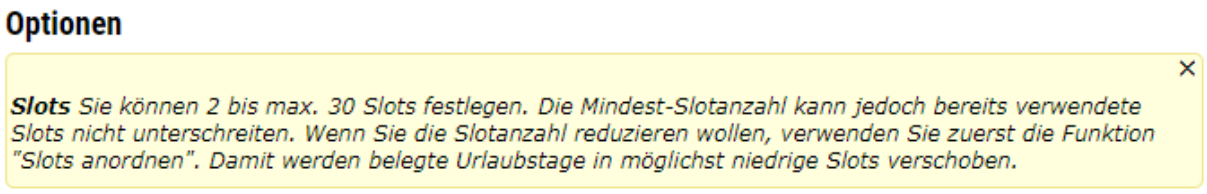


Abbildung 9: Der Hilfetext erscheint da, wo man ihn braucht: Im aktuellen Fenster

Abbrechen & Übernehmen

Abbrechen verwirft alle Änderungen, *Übernehmen* übergibt die neuen Einstellungen an den Urlaubsplaner. Das Optionen-Fenster wird damit geschlossen. Beachten Sie, dass diese Einstellungen nicht automatisch in die aktuelle Datei gespeichert werden. Einstellungen werden erst dauerhaft zusammen mit den Nutzdaten gespeichert, wenn Sie im Hauptmenü das Speichern ausführen.

Personen anlegen und verwalten

Sofern Sie keine Personen aus einer anderen Datei importiert haben, ist die Personenliste noch leer. Neben der leeren Tabelle steht ein Hilfetext, der Ihnen für den Anfang die Bedeutung der Felder erläutert. Klicken Sie in das erste Namensfeld und geben Sie hier Name und Vorname der zu erfassenden Person ein. Es empfiehlt sich diese Reihenfolge, weil die Zeilen nach dem Übernehmen neu sortiert werden. Vorneweg können Sie auch eine Farbe und ein Namenskürzel eintragen, das müssen Sie aber nicht. Eine Zufallsfarbe wird vom Programm vorgeschlagen und vorbelegt, das Namenskürzel ergibt sich aus den Initialen. Sie können beides vorab oder nachträglich ändern. Bei den Farben ist eine Änderung dann sinnvoll, wenn sich zwei Farben zu ähnlich sind. Gleiche Namenskürzel können natürlich auch vorkommen und sollten geändert werden. Dabei können Sie kreative Kombinationen aus Klein- und Großbuchstaben wählen.

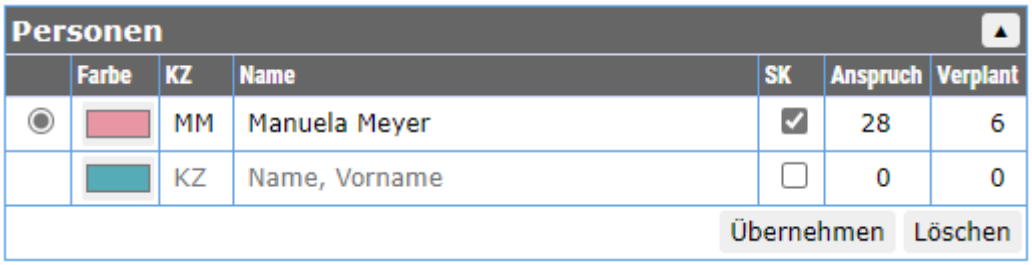


Abbildung 10: Personen anlegen (und ändern)

Die Spalte SK steht für schulpflichtige Kinder. Dieses Feld soll Sie daran erinnern, dass die Person bei der Urlaubsplanung auf Ferienwochen angewiesen ist. Funktional hat dieses Feld keine Bedeutung. Der Betrieb ist gemäß deutschem Arbeitsrecht nicht verpflichtet allen Urlaubswünschen der Angestellten nachzukommen.

Bei Anspruch tragen Sie die verfügbaren Urlaubstage ein. Zum Jahresanspruch sollten Sie auch noch die nicht genommenen Urlaubstage aus dem Vorjahr dazurechnen. Aus dieser Summe ergibt sich der Anspruch für das aktuelle Jahr.

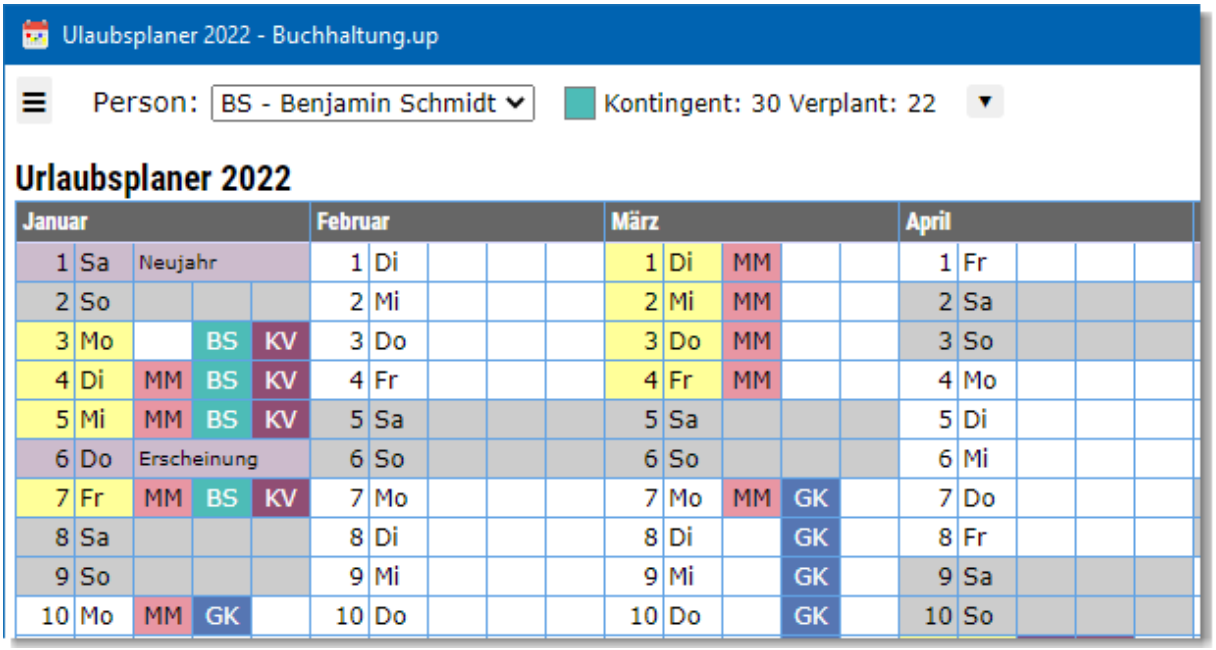
Die letzte Spalte ist schreibgeschützt. Diese Spalte wird vom Programm befüllt, sobald Sie Urlaubstage in den Kalender eintragen. Im Screenshot sehen Sie hier 6 bereits verplante Urlaubstage. Diese ergeben sich aus zwei Brückentagen und vier Tage Betriebsurlaub, die in den Optionen bereits eingetragen sind.

Klicken Sie auf *Übernehmen*, um die Person dem Planer hinzuzufügen. Bevor das passiert, prüft das Programm Ihre Eingaben auf Vollständigkeit und Plausibilität. Nach der Übernahme steht in der ersten Spalte ein selektiertes Optionsfeld, das anzeigt, dass diese Person ausgewählt ist für das Erfassen von Urlaubstagen im Kalender. Sie können sofort mit der Urlaubsplanung beginnen oder weitere Personen anlegen. Ab drei Personen verbreitert sich die Tabelle auf zwei Spalten. Das sorgt dafür, dass der Bildschirm besser ausgenutzt wird.

Wenn alle Personen angelegt sind, können Sie die Tabelle auch wahlweise einklappen. Das geht mit dem Pfeil-Schalter im Tabellenkopf. Dabei schrumpft die Tabelle auf eine einzelne Zeile zusammen und es wird nur noch die aktive Person in einer aufklappbaren Liste angezeigt. Über diese Liste können Sie jederzeit eine andere Person aktivieren.

Urlaubstage planen

Aktivieren Sie in der Personen-Tabelle den Optionsschalter für die Person deren Urlaub Sie planen wollen. Nun können Sie im Kalender beliebige Arbeitstage anklicken. In der angeklickten Zelle erscheint das Namenskürzel. Die Hintergrundfarbe der Zelle ändert sich in die voreingestellte Farbe der aktiven Person. Sollte es sich bei der Farbe um einen dunklen Farbton handeln, dann ändert das Programm die Schriftfarbe nach weiß, ansonsten bleibt sie schwarz. Das gilt unabhängig vom voreingestellten Skin. Die Farben der Personen als auch die dazu passende Schriftfarbe sind davon unabhängig. Das gilt auch für die Farben der Feiertage, der Ferien und dem Betriebsurlaub, die Sie in den Optionen festgelegt haben.



Januar					Februar					März					April				
1	Sa	Neujahr			1	Di				1	Di	MM			1	Fr			
2	So				2	Mi				2	Mi	MM			2	Sa			
3	Mo		BS	KV	3	Do				3	Do	MM			3	So			
4	Di	MM	BS	KV	4	Fr				4	Fr	MM			4	Mo			
5	Mi	MM	BS	KV	5	Sa				5	Sa				5	Di			
6	Do	Erscheinung			6	So				6	So				6	Mi			
7	Fr	MM	BS	KV	7	Mo				7	Mo	MM	GK		7	Do			
8	Sa				8	Di				8	Di		GK		8	Fr			
9	So				9	Mi				9	Mi		GK		9	Sa			
10	Mo	MM	GK		10	Do				10	Do		GK		10	So			

Abbildung 11: Die eingeklappte Personentabelle mit Urlaubskalender

Sie werden feststellen, dass Sie mit den voreingestellten Slots unabhängig von der Personenanzahl eine Zeit lang gut planen können. Irgendwann aber wird der Zeitpunkt kommen, wo eine dritte Person am gleichen Tag Urlaub beansprucht wie zwei bereits verplante Personen. Wechseln Sie dann in die Programm-Optionen und erhöhen Sie dort die Slot-Anzahl.

Jeder hinzugefügte Urlaubstag erhöht die Zahl in der Spalte *Verplant* der Personentabelle. Sollten Sie mehr Tage verplanen als der betreffenden Person zustehen, wechselt die Schriftfarbe in dieser Zelle nach rot. Sie wird wieder schwarz, wenn Sie Urlaubstage entfernen oder den Anspruch vergrößern. Beachten Sie, dass eine Änderung in der Personentabelle übernommen werden muss (Schalter *Übernehmen*), damit sie wirksam wird.

Das Entfernen eines Urlaubstages gestaltet sich genauso einfach wie das Hinzufügen. Klicken Sie einfach auf die entsprechende Zelle. Voraussetzung ist, dass in dieser Zelle auch die selektierte Person steht. Wenn das nicht der Fall ist, müssen Sie die gewünschte Person davor auswählen.

Für das Programm und die verplanten Urlaubstage spielt es keine Rolle in welchem Slot die Personen eingetragen sind. Dennoch ist der Urlaubsplaner übersichtlicher und aufgeräumter, wenn die aufeinanderfolgenden Urlaubstage einer Person in der gleichen Spalte angeordnet sind. Deshalb ist es möglich eine belegte Zelle in eine andere Spalte zu verlegen. Klicken Sie dazu einfach die Ziel-Zelle an und der Eintrag wird verlegt. Das können Sie sogar dann machen, wenn die Ziel-Zelle bereits durch eine andere Person belegt ist. Das hat zur Folge, dass die beiden Personen die Spalte tauschen. Damit können Sie mit wenig Klicks im Urlaubsplaner für Ordnung sorgen.

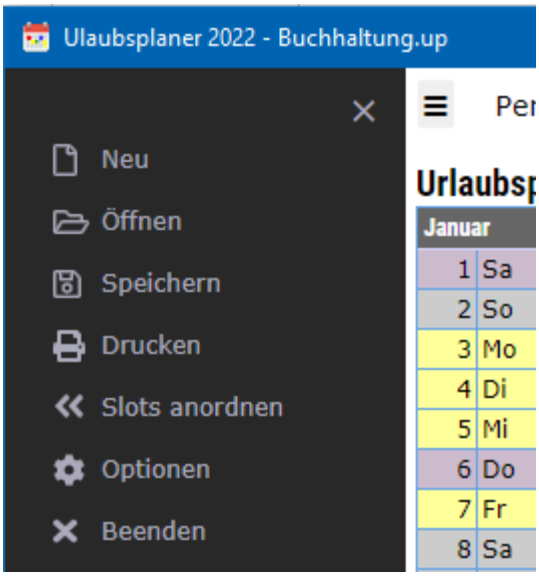


Abbildung 12: Eingeblendetes Hauptmenü

Kein Urlaubsplan ist am ersten Tag fertig. Die Kolleginnen und Kollegen reichen erst nach und nach ihre Urlaubswünsche ein und neue Daten werden so eingetragen, wie sie zur Verfügung stehen. Also müssen Sie den Plan zwischenspeichern, um ihn später wieder bearbeiten zu können. Dafür dienen die Befehle *Speichern* und *Öffnen* im Hauptmenü. Dabei merkt sich der Urlaubsplaner die zuletzt geöffnete, bzw. die zuletzt gespeicherte Datei und öffnet sie automatisch wieder beim nächsten Start. Sollten Sie mal eine andere Datei bearbeiten wollen als die zuletzt gespeicherte, dann verwenden Sie das Menü *Öffnen* aus dem Hauptmenü. Auch hier hat sich das Programm den zuletzt verwendeten Speicherpfad gemerkt und bietet diesen wieder zum Öffnen einer Datei an.

Einen neuen Urlaubsplaner legen Sie mit der Funktion *Neu* im Hauptmenü an. Hier erscheint wieder das bereits bekannte Fenster zur Neuanlage von Urlaubsplanern. Eines haben die Befehle *Neu* und *Öffnen* gemeinsam: Bevor Sie eine andere Datei bearbeiten, sollten Sie die Änderungen der aktuellen Datei speichern. Damit Sie das nicht vergessen, werden Änderungen von der Software überwacht. Falls diese noch nicht gespeichert sind, werden Sie dazu aufgefordert. Gleiches gilt beim Beenden des Programms.

Während der Planung könnte es vielleicht vorkommen, dass Urlaubstage verlegt werden. Das ist oft der Fall, wenn sich herausstellt, dass zu viele Personen einen Urlaubstag für das gleiche Datum beantragt haben. Damit Ihr Team an solchen Tagen nicht unterbesetzt ist müssen Sie umplanen. Dabei werden vorher belegte Slots unter Umständen wieder frei und blähen nur unnötig den Kalender auf. Unnötige Slots können Sie in den Programm-Optionen wieder entfernen. Das setzt aber voraus, dass diese Slots auch wirklich frei sind. Das können Sie erreichen, indem Sie alle geplanten Urlaubstage möglichst weit in einen niedrigen Slot, also nach links verschieben. Damit werden Slots rechts außen frei und können gelöscht werden. Im Hauptmenü finden Sie die Funktion *Slots anordnen,* die Sie bei dieser Reorganisation unterstützt.

Ein ausgedruckter Urlaubsplaner an der Pinnwand ist dem Abteilungsleiter vielleicht wichtig. Bei einem nicht allzu großen Team spricht nichts dagegen, den Planer auszudrucken. Über die Funktion *Drucken* im Hauptmenü bringen Sie Ihren Urlaubsplan aufs Papier. Die GUI der Software basiert auf der Chromium-Engine. Da verwundert es nicht, dass der Druckerdialog fast genauso aussieht wie in vielen

anderen Chromium-Browsern. Für ein gutes Druck-Ergebnis sollten Sie folgende Einstellungen vornehmen:

- Layout: Querformat
- Farbe: Farbe (wenn möglich, setzt einen Farbdrucker voraus)
- Einseitiger Druck
- Papierformat A3 (wenn möglich)
- Skalierung: passen Sie diese so an, dass der Urlaubsplaner auf eine Seite passt
- Ränder: Minimum
- Kopf- und Fußzeilen ausschalten
- Hintergrundfarben einschalten

Sie werden in der Druckvorschau feststellen, dass die Personentabelle fehlt und dafür unter dem Kalender eine Legende mit den Personen, Namenskürzeln und Farben ausgegeben wird.

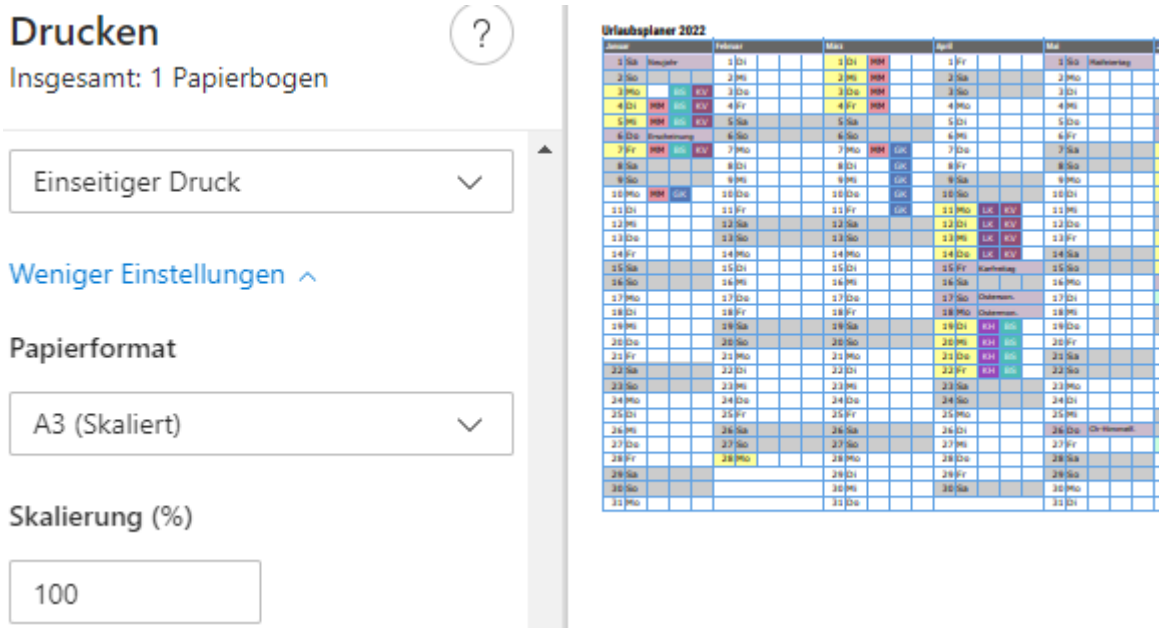


Abbildung 13: Drucken, Einstellungen und Druckvorschau

Let's go (2)?

Ok, lass uns loslegen. Ab jetzt sind wir Softwarearchitekten, Programmierer und Webdesigner in Personalunion. Wir arbeiten mit unterschiedlichen Programmen und Programmiersprachen. Und wir duzten uns! Warum das? Weil's einfacher ist und zum Standard in Programmierer-Newsgroups und einschlägigen -Foren gehört. Und eine Sache müssen wir noch mal präzisieren: In diesem Buch lernen wir keine Programmier- oder Auszeichnungssprache, kein Visual Basic oder VBA, kein JavaScript, kein HTML und kein CSS. Und dennoch ist es ein bisschen von allem, weil das nun mal die Werkzeuge sind, die wir hier brauchen. Es geht darum, ein Programm für den Desktop zu erstellen, dessen GUI auf der WebView2-Runtime von Microsoft basiert. Des Weiteren werden wir hier massiv die Programmbibliotheken (Framework für VB) RC6 von Olaf Schmidt (www.vbrichclient.com) verwenden. In dieser Bibliothek findet sich unter anderem die Klasse cWebView2, die das Bindeglied zwischen Anwendung und der WebView2 Runtime ist. Darüber hinaus bietet uns dieses Framework eine SQLite-Engine, mit der wir unsere Daten verwalten und persistieren. In VB-Kreisen ist die Arbeit von Olaf ein fester Bestandteil für viele Programmierer. An dieser Stelle sei angemerkt, dass in diesem Framework weitere tolle GUI-Werkzeuge stecken, unter anderem die vektorbasierte Cairo-Library. Diese spielt jedoch in diesem Projekt nur am Rande eine Rolle.

Warum die WebView2-Runtime? Die Entwicklung des Internet Explorers ist eingestellt. Andererseits kommen derzeit hybride Anwendungen wieder in Mode, siehe Blazor-Desktop. Ich will hier die WebView2-Runtime und den praktischen Einsatz in einem VB-Programm demonstrieren. Die Möglichkeiten der Engine werden hier nicht ausgeschöpft. Fast alles was ihr jemals in einem Browser gesehen habt, kann damit umgesetzt werden und noch mehr. Noch mehr, weil wir den WebView2 in eine eigene Anwendung packen und somit mit den Klassen und Algorithmen der eigenen Programmiersprache kombinieren werden. Hätte man diese Anwendung auch mit anderen Mitteln programmieren können? Sicher! Vielleicht auch besser, performanter, plattformunabhängiger und vieles mehr. Aber es geht hier nicht um den Vergleich von Technologien oder Sprachen. Es geht einfach nur darum, die Browser-Engine in einer VB-basierten Anwendung zu verwenden.

Und wo sind die Vorteile? Irgendwas muss es ja geben was diese Technik interessant macht. Also ich sehe das so:

- Ich verwende seit vielen Jahren den Internet Explorer, bzw. das IE-Browser-Control in meinen Anwendungen. Damit kann ich Webinhalte (HTML, CSS und JavaScript) in meine native Anwendungen einbetten. Die GUI der Programme bekommen eine gewisse Ähnlichkeit mit dem was viele Anwender vom Browser und Internet kennen. Die Bedienung ist intuitiver und die Optik moderner. Auch ergeben sich so Möglichkeiten, die mit normaler GUI-Programmierung schwer umzusetzen wären. Nun wird aber der Internet Explorer von Microsoft nicht mehr weiterentwickelt. Die letzte verfügbare Version 11 der Engine unterstützt viele Merkmale heutiger HTML/CSS/JavaScript nicht mehr. Im schlimmsten Fall könnte die Engine irgendwann aus dem Betriebssystem verschwinden. Das wäre das Aus für jede Anwendung, die darauf basiert. Die WebView2-Runtime ersetzt die bisherige Technik und bringt alle Neuerungen der Chromium-Engine mit. Das ist modern, das ist nachhaltig - wollen wir's hoffen (UWP hatte einen ähnlichen Ansatz, war aber meines Erachtens ein Rohrkrepierer).
- Durch den Einsatz von WebView2 kann ich meinen Anwendern leichter Funktionen zur Verfügung stellen, die auf allen Geräteplattformen gleich aussehen und sich gleich anfühlen.

Ich als Programmierer kann einiges von dem was ich für den Desktop programmiere auch im Web oder auf mobilen Geräten wiederverwenden.

- Microsoft installiert automatisch die WebView2-Laufzeitumgebung auf Windows 10-Rechnern, um kommende Versionen der eigenen Anwendungen (z. B. Microsoft 365 und Microsoft Office) zu unterstützen. Davon kann man selbst auch profitieren. Microsoft sorgt mit einer sogenannten immergrünen Version automatisch für Aktualisierungen. Sicherheitstechnisch kann das nur von Vorteil sein. Die Runtime wird inzwischen für viele Sprachen angeboten.

Vorbereitungen

Die Runtime kann nur verwendet werden, wenn sie installiert ist. Zwar bietet Microsoft inzwischen automatische Installationen für Windows 10, das muss aber nicht heißen, dass die Runtime tatsächlich überall verfügbar ist. Deshalb auch meine Empfehlung: Liefert den Evergreen Bootstrapper oder ein eigenständiges Installationsprogramm mit eurem Programm aus. Achtet dabei auf die x86-Architektur. Wir brauchen diese Version, selbst wenn auf dem Anwender-Rechner ein x64-Betriebssystem installiert ist. Alternativ kann auch eine unveränderbare Version mit der eigenen Anwendung gekoppelt werden.

Die installierte Runtime findest du unter %Programme%\Microsoft\EdgeWebView\Application\. Hier gibt es einen Ordner, dessen Name sich im Format xx.x.xxx.xx an der Runtime-Version orientiert. Alles was zur Runtime gehört liegt in diesem Ordner, es gibt keine weiteren Abhängigkeiten. Solltest du eine unveränderbare Version verwenden wollen, wird in deinem Programm-Ordner der, von der Microsoft Download-Seite entpackte Versions-Ordner enthalten sein. Microsofts WebView2 Runtime findest du unter: https://developer.microsoft.com/de-de/microsoft-edge/webview2/#download-section

Dann brauchen wir noch das RC6-Framework. Das kannst du von der Seite http://vbrichclient.com kostenlos herunterladen. Als ich diesen Text geschrieben habe, war dort die Version 6.0.8 vom 27.04.2021 aktuell. Die Zip-Datei beinhaltet fünf DLLs: RC6.dll, cairo_sqlite.dll, DirectCOM.dll, WebView2Loader.dll und die RC6Widgets.dll. Im Archiv sind noch Lizenzdateien und vbs-Skripte zum Registrieren der COM-DLLs enthalten. Für die Software in diesem Projekt wird die RC6Widgets.dll nicht benötigt. Eine Registrierung der RC6.dll ist nur auf dem Entwicklungsrechner erforderlich. Auf den Rechnern eurer Anwender reicht das Kopieren der Dateien, eine Registrierung ist dort nicht nötig. Deshalb ist diese Anwendung auch ohne Weiteres direkt von einem USB-Stick oder von einer CD-ROM lauffähig.

Schnellstart

Lege ein neues Exe-Projekt an. Über Verweise kannst du den RC6 einbinden. Füge in das Code-Fenster der Form folgenden Code ein und starte das Projekt mit F5:

```
Option Explicit
Private WithEvents WV As RC6.cWebView2

Private Sub Form_Load()
  Show
  Set WV = New_c.WebView2(Me.hWnd)
  WV.Navigate "https://wiki.selfhtml.org"
  Me.Caption = WV.DocumentTitle
End Sub
```

```
Private Sub Form_Resize()
  If WindowState <> vbMinimized Then
    If Not WV Is Nothing Then WV.SyncSizeToHostWindow
  End If
End Sub
```

Glückwunsch! Du hast soeben deinen ersten Chromium-basierten Browser entwickelt!

Wenn du Einsteiger in VB bist oder VB nur aus der gängigen Literatur kennst, steht für dich jetzt ein großes Fragezeichen im Raum? Wann und wo haben wir im Form-Designer ein Browser-Control auf die Form gesetzt? In der Werkzeugsammlung erscheint noch nicht mal ein zusätzliches Steuerelement. Wie den auch? Wir haben keins eingebunden. Unser Projekt hat lediglich einen Verweis auf den RC6 bekommen. Mehr brauchen wir nicht. Und ich verspreche: Den Form-Designer werden wir in diesem Projekt, an dessen Ende eine vollständige Anwendung steht, kein einziges Mal brauchen. Genaugenommen werden wir im fertigen Programm noch nicht mal die automatisch hinzugefügte erste Form brauchen. Die macht uns lediglich in der IDE das Leben etwas leichter. Somit sind auch sämtliche Inkompatibilitäten früherer VB-Anwendungen oder altgebackenes Aussehen der GUI Vergangenheit. Mit der Kombination RC6 und WebView2-Runtime startet für den VB-Entwickler eine neue Ära. Was aber nicht bedeutet, dass du keine VB-Controls mehr nutzen darfst. Du kannst das weiter tun, brauchst es aber nicht.

Und es kommt noch besser: Neben der WebView2 Komponente kannst du auch ein Set von modernen Vektorbasierenden Steuerelementen aus dem RC6-Framework nutzen. Diese basieren auf der Cairo-Grafikbibliothek, die eine vektorbasierte API als Basis für das GUI-Framework darstellt. Dessen Grafik-Funktionen erlauben neben antialiased Vektor-Output über Pfad-Definitionen auch vielfältige Bild-Manipulation (z.B. alle Arten von alpha-transparentem Blending, Rotationen u. Scaling), das Arbeiten mit modernen Image-Ressource-Formaten wie Alpha-Icons, PNGs, SVGs, JPGs, sowie das direkte Generieren von PDFs. Erwähnenswert sind auch Klassen für die Druckausgabe und Druckvorschau, oder auch der Support für QR-Code (in Lese- und Schreib-Richtung). Zumindest die PDF-Ausgabe werden wir auch in diesem Projekt verwenden. Jedoch muss ich hier warnen: In diesem Text werden wir nur einen kleinen Teil der Klassen und Funktionen aus der RC6-Biliothek verwenden. Nur das was wir brauchen. Für die vollständige RC6-Dokumentation bräuchte es ein eigenes Buch, bei dem eine 3-stellige Seitenanzahl nicht ausreicht.

Lass uns einen Blick auf diesen Code werfen. Diese wenigen Zeilen zeigen schon einige Kernaspekte der WebView2-Runtime. Starten wir mit der Deklaration. Diese zeigt uns, dass die Schnittstelle zur Runtime über eine Klasse im RC6 umgesetzt ist. Es gibt also kein Steuerelement oder OCX, sondern nur der Verweis auf den RC6. Das mag zunächst überraschen, wird uns aber später noch große Vorteile bringen. Ein weiterer Aspekt fällt auf: An die Runtime ist tatsächlich nur diese eine Klasse gebunden. Alles was wir mit dem WebView2 anstellen, läuft ausschließlich über diese Klasse. Das macht die Arbeit überschaubar. Bei meinen früheren Arbeiten mit dem Internet Explorer Control hatte ich mindestens zwei Abhängigkeiten. Das waren das *Microsoft Internet Control* (ieframe.dll) und die *Microsoft HTML Object Library* (mshtml.tlb). Wer sich diese beiden Komponenten im Objektbrowser ansieht, bekommt ein Bild von der enthaltenen Komplexität. In der Deklaration habe ich vorausschauend das Schlüsselwort *WithEvents* verwendet. Das bedeutet, dass unsere WebView2 auch Ereignisse liefert, auch wenn wir diese in dem kurzen Beispiel noch nicht gebraucht haben.

Eine Deklaration liefert noch kein Objekt, es fehlt noch die Instanziierung. Das passiert in der Form_Load-Prozedur, allerdings erst nachdem die Form mittels Show-Methode angezeigt wurde. Das

wird in allen zukünftigen Projekten genauso sein. Die Runtime ist wie ein Kuckuck, der seine Eier in fremde Nester legt. Mit „fremde Nester“ ist ein Elternfenster (*Form*) oder ein Bildsteuerelement (*PictureBox*) gemeint. Auf jeden Fall braucht es einen Container mit *hWnd*-Eigenschaft. Die *hWnd*-Zugriffsnummer wird als Parameter an die Methode WebView2 eines ominösen *New_c*-Objektes übergeben. Daraus entsteht eine Objekt-Instanz des WebView2. Was wir hier sehen ist eine Factory, ein Entwurfsmuster, genaugenommen ein Erzeugungsmuster, bekannt aus den Design Patterns der Softwareentwicklung. Das *New_c*-Objekt wird automatisch im RC-Framework instanziiert und dient zur Erzeugung von weiteren Objekten aus der RC6-Familie. Es kann aber auch für Objekte, die sich aus externen Klassen ableiten verwendet werden. Und zwar so, dass diese ActiveX-Bibliotheken noch nicht mal registriert sein müssen. Erinnerst du dich an die Aussage zum Deployment im ersten Kapitel? Da haben wir schon die Basis für die Copy & Paste Distribution ohne Registrierung.

An dieser Stelle müssen wir später noch mal ran. Das *New_c* Objekt bekommen wir so billig nur auf unserem Entwicklungsrechner geliefert, wo die RC6.dll registriert ist. In unserer fertigen Anwendung werden wir dafür sorgen, dass auch das *New_c* Objekt aus einer nicht registrierten RC6.dll erzeugt werden kann. Wer’s nicht abwarten kann, darf schon mal zum Kapitel Deployment vorblättern.

Wer sich die WebView2-Methode im Objektkatalog anschaut wir feststellen, dass der hWnd-Parameter optional ist. Also könnten wir das WebView2-Objekt auch ohne Parent erzeugen? Jein! In der Tat, die Instanziierung funktioniert auch ohne Parent, spätestens wenn wir irgendwas mit dem WebView2 anzeigen wollen, brauchen wir einen GUI-Container, ein Window im Entwickler-Jargon. Dafür bietet uns die WebView2-Klasse eine eigene Funktion: die *BindTo*-Methode. Diese ist praktisch, weil sie einen Rückgabewert liefert, der anzeigt, ob die BindTo-Aktion erfolgreich war. Den Rückgabewert kann man später dafür verwenden, um zum Beispiel die Runtime nachzuinstallieren, falls sie auf dem Anwender-Rechner nicht vorhanden ist. Auch das ist eine Eigenschaft unseres Deployment-Prinzips, auf das ich später eingehen werde.

Als nächstes navigieren wir schon zur ersten externen Internetseite. Mit der *Navigate*-Methode und einer gültigen URL (darf auch gerne eine https-Seite sein), zeigt uns die WebView2 bereits das Internet. Die darauffolgende Anweisung, in der wir den Seitentitel in den Fenstertitel übertragen zeigt, dass ab diesem Zeitpunkt die aufgerufene Seite schon vollständig geladen ist. Wir können also nach einer erfolgreichen *Navigate*-Methode sofort mit dem HTML-DOM interagieren.

Vollständigkeitshalber habe ich noch das *Form_Resize*-Ereignis des Elternfensters mit dem WebView2-Methodenaufruf *SyncSizeToHostWindow* ergänzt. Das sorgt dafür, dass die WebView2 den verfügbaren Innenbereich des Fensters nutzt. Ich denke, das braucht keine Erklärung. Wenn doch, schlage die Methode in der Referenz zur cWebView2-Klasse nach. Und nun erwartet den Anwender ein weiteres Highlight: Halte die Strg-Taste gedrückt und drehe am Mausrad. Es passiert das gleiche wie im heimischen Browser: Die Anzeige wird skaliert, wobei sich alle Inhalte automatisch anpassen und falls erforderlich neu ausrichten. Eure Programme werden auf dem 4K-Monitor genauso gut oder noch besser aussehen als jemals zuvor.

Eine Sache muss an dieser Stelle noch erwähnt werden: Wenn die WebView2-Komponente den Focus hat, bekommt man im Elternfenster keine Key- und Mouse-Events mehr. Auch dann nicht, wenn die *KeyPreview*-Eigenschaft des Fensters auf *True* eingestellt ist. Warum ist das so? Die Erklärung dafür sehen wir, wenn wir den Task-Manager starten. Ihr werdet dort einige neue *Microsoft Edge WebView2*

Prozesse finden. Die gehören zur Anwendung, obwohl es eigenständige Prozesse sind. Ihr könnt mal spaßeshalber einige der Prozesse abschießen (Task beenden), um zu sehen was passiert.

Fassen wir die neuen Erkenntnisse zusammen: Wenn RC6 eine neue WebView2-Instanz erzeugt, starten im Hintergrund eigenständige WebView2-Prozesse. Einer davon ist direkt mit dem Elternfenster gekoppelt und leitet seine GUI-Ausgabe an die Oberfläche dieses Fensters weiter. Wir können sogar ermitteln, welcher Prozess das genau ist. Schalte dafür die IDE in den Pausenmodus, öffne ein Direktfenster und gib dort folgende Code-Zeile ein:

```
?WV.BrowserProcessId
```

Die Eigenschaft liefert die PID des gekoppelten Prozesses. Den kann man im Taskmanager identifizieren, sofern dort die PID-Spalte angezeigt wird. Wenn nicht, mit der rechten Maustaste in die Kopfzeile klicken und die PID-Spalte aktivieren.

Mit diesem Hintergrundwissen können wir nun einige Konsequenzen daraus ableiten:

- Unsere Anwendung besteht aus mehreren Prozessen. Das heißt, sie kann auch mehrere CPU-Kerne gleichzeitig nutzen. Der VB-Programmierer kennt dieses Prinzip schon von der ActiveX-Exe. Auch wenn die WebView2 keine ActiveX-Exe ist, ist das Prinzip damit am besten vergleichbar.
- Der eigenartige Focus-Wechsel (siehe nächster Text-Absatz) per Tab-Taste lässt sich damit gut erklären. Genauso die fehlenden Key-Ereignisse.
- Der Programmierer sollte diese Prozess-Konstellation immer in seiner Planung berücksichtigen. Sie wirkt sich unter anderem auf den Ressourcen-Verbrauch der eigenen Anwendung aus.

Man kann WebView2 mit anderen VB-Steuerelementen kombinieren. Da WebView2 seine Ausgaben direkt auf die Fläche des Elternfensters zeichnet, werden fensterbasierte Steuerelemente (mit eigener *hWnd*) immer im Vordergrund stehen. Ein Label oder eine Linie hingegen wird gnadenlos übermalt. Es empfiehlt sich deshalb, die WebView2 in ein Bildsteuerelement (*PictureBox*) zu packen. Dennoch wird nicht alles so funktionieren wie erwartet. Zum Beispiel der Fokuswechsel per Tab-Taste. So lange dieser auf einem VB-Steuerelement liegt, ignoriert die Tab-Folge die WebView2. Es sei denn, man überträgt den Focus an die WebView2 im *GotFocus*-Ereignis des Elternfensters. Das funktioniert so weit gut, jedoch ist der Eingabefocus nun im WebView2 gefangen und kommt von hier nicht mehr ohne weiteres raus. Interessanterweise gilt das für Menüs nicht, die funktionieren relativ gut. Nach dem Menüaufruf steht der Focus allerdings auf dem Elternfester und sollte wieder programmgesteuert auf den WebView2 gelegt werden.

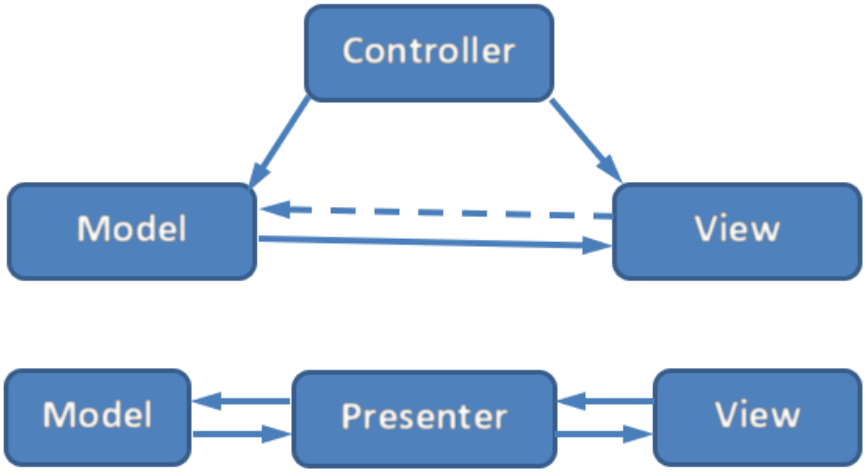
Summa summarum: Lass es besser bleiben! Die Kombination mit VB-Steuerelementen ist eh nicht ratsam, zumal die Optik darunter leidet. Für private Tests, bei denen man schnell einen Schalter braucht, kann man das hingegen machen. In den folgenden Kapiteln werden wir eh die VB-Fenster durch Fenster aus dem RC6-Framework ersetzen. Wir hatten bereits angekündigt, dass wir auf alle VB-Controls verzichten wollen. Das schließt die VB-Formen mit ein.

Model View Presenter

Ganz so populär wie sein Ziehvater, das MVC-Entwurfsmuster (*Model-View-Controller*) ist MVP (*Model-View-Presenter*) nicht, obwohl es lediglich zehn Jahre jünger ist. MVC ist zu Zeiten entstanden, als grafische und ereignisbasierte GUIs noch auf den schwarzweißen (bzw. grünschwarzen) Bildschirmen keine Daseinsgrundlage fanden. Das MVC-Konzept wurde 1979 für die Programmiersprache Smalltalk entwickelt. Die damit erstellten Anwendungen hatten bei Weitem nicht die Betriebssystem-Unterstützung wie die späteren grafischen Nachfolger. Damals musste der Programmierer vieles selbst erledigen, was heute von Steuerelementen oder moderner, von Widgets geboten wird. In heutigen Desktop-Applikationen geht dem Controller sprichwörtlich die Arbeit aus, er fungiert nur noch als Vermittler zwischen den anderen beiden Schichten. Und dabei wird er in der MVC-Architektur auch noch umgangen: Gemäß der Definition kann das Model die View direkt mit Daten versorgen und der Controller schaut nur noch zu. Das wurde in den 90-er Jahren mit der Einführung der RAD-Entwicklungssystemen (*Rapid Application Development*), zu denen auch Visual Basic zählt, noch verstärkt. Steuerelemente haben hier universelle Schnittstellen (*DataSource*) zu Datenquellen bekommen, mit denen sie direkt an die Models gebunden sind. Navigationsbefehle und sonstige CRUD-Operationen (*C-Create, R-Read, U-Update, D-Delete*) wurden plötzlich direkt mit dem DBMS ausgehandelt und der Programmierer wechselte hauptberuflich von der Tastatur zur Maus.

Eine saubere Trennung der Schichten ist aber mit diesen Architekturen kaum noch möglich. Der primäre Zweck einer Schichten-Architektur ist aber genau diese Isolation der Module. Das macht das Model und die View(s) unabhängig und austauschbar. Im MVP-Muster ist der Presenter das Bindeglied zwischen Model und View. Dabei darf man ihn nicht als Controller-Ersatz sehen. Alle Aufrufe aus den Views werden direkt an Presenter delegiert. Er selbst ist von den Views entkoppelt und kommuniziert mit ihnen über Schnittstellen. Die Views sind möglichst „dumm“ was die Logik der Anwendung angeht. Die Models kapseln Geschäftslogik und Daten und liefern letztere in strukturierter Form. Der Presenter ist in dieser Umgebung plötzlich wieder die Nummer Eins, er führt Regie, er ist der King!

Die in der Fachliteratur und im Internet überall verfügbaren Schaubilder zu den beiden Architekturen sehen alle mehr oder weniger gleich einfach aus. Die Praxis gestaltet sich in der Regel etwas komplexer und die Muster werden meistens individuell ausgelegt oder sind vom eingesetzten Framework vorgegeben. Letzten Endes geht es immer um die möglichst saubere Trennung der Module.



Welche Vorteile haben wir in unserem Projekt in Bezug auf diese Architektur zu erwarten? In der Tat ist es so, dass die Schichtentrennung zunächst mehr Aufwand bedeutet. Es geht schon mal beim Planungsaufwand los. Wie gestalte ich meine Module so, dass sie wirklich die nötige Kapselung erfahren und andererseits nicht Performance, Usability und Entwicklungskosten belasten. Machen wir uns nichts vor, eine gute Architektur kostet was. Es handelt sich dabei um vermeintliche Opportunitätskosten. Je nach Projektgröße und Komplexität können sich diese Kosten schnell amortisieren und in Opportunitätserlöse umwandeln.

Ein Beispiel: Zum Projektteam gehört eine kreative Grafikerin, die auch was von Webdesign versteht, aber sonst mit Programmieren eher nichts am Hut hat. Dieser Kollegin kann man die Views anvertrauen und sie wird modern gestaltete Benutzeroberflächen entwerfen, die sogar mit farbig gut abgestimmten Skins arbeiten können. Dabei kann sie ihre Lieblingstools losgelöst von der Datenbasis verwenden. Gleiches Prinzip kann auch auf der Model-Seite angewendet werden, wenn es dort um eine externe Datenanbindung geht.

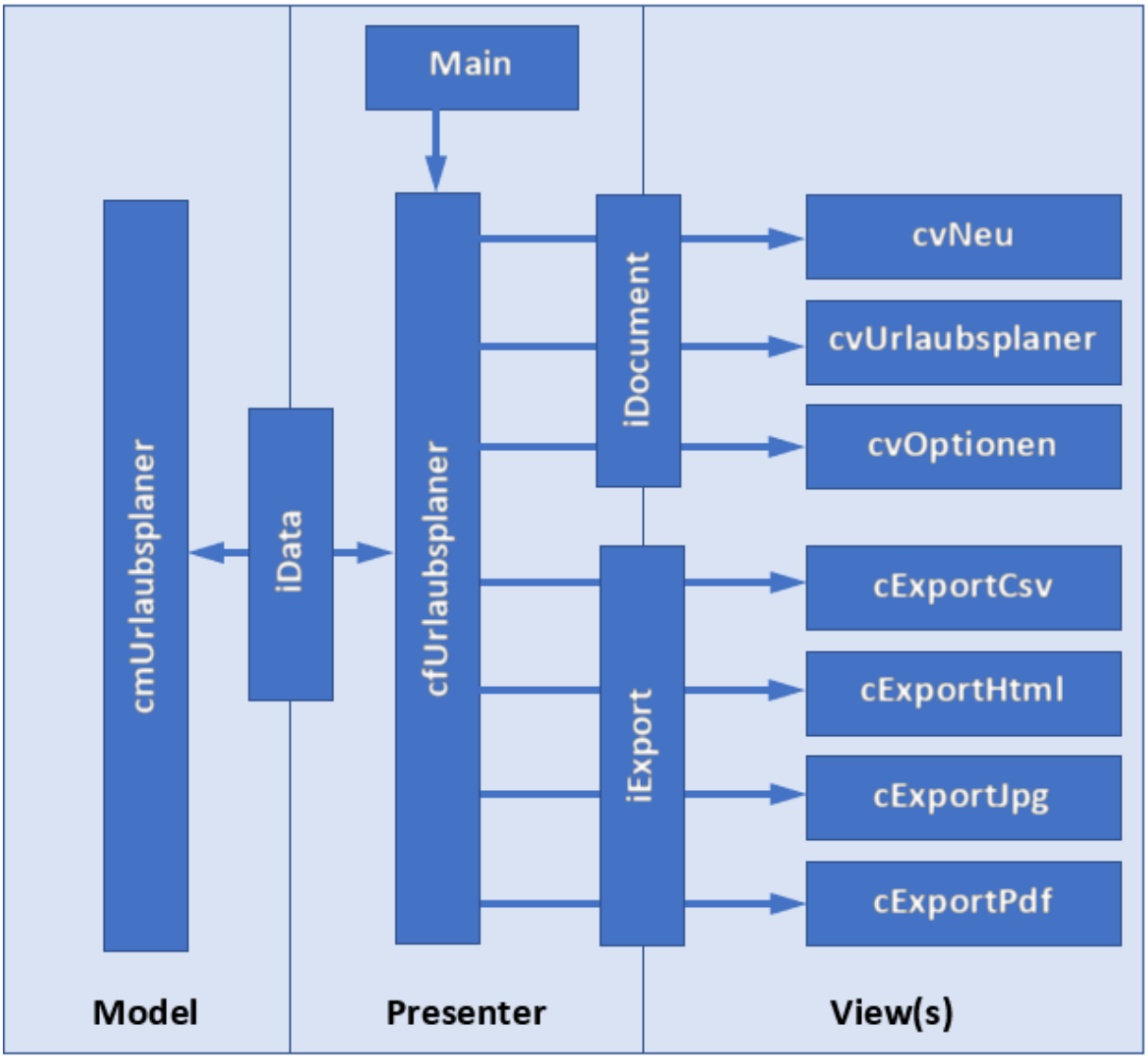
Ein weiteres Beispiel: Die Anwendung ist am Markt gut angekommen. Nun gibt es aber einen Kunden, der beim integrierten Bild-Export das PNG-Format bevorzugt und statt dem CSV lieber mit JSON arbeiten will. Alle Exporte verwenden die gleiche Schnittstelle. Es ist nun ein Klacks einen weiteren Export-Datentyp einzubauen. Das kann so weit gehen, dass dies wie ein Plugin in einer eigenen DLL an die Anwendung gebunden werden kann, ohne dass dabei die Anwendung erweitert oder ausgetauscht werden muss.

Im ersten Kapitel haben wir über die verwendeten Technologien gesprochen. Wenn wir diese und die MVP-Architektur zusammenfügen, ergibt sich folgende Matrix:

Technologie	**Model**	**View(s)**		**Presenter**
Visual Basic (oder VBA)	×		×	×
RC6	×		×	×
WebView2	-	×	-	×
HTML	-	×	-	-
CSS	-	×	-	-
JavaScript	-	×	-	-
SQLite	×	-	-	-
SQL	×	-	-	-
CSV	-	-	×	-
Cairo	-	-	×	-
JPG	-	-	×	-
PDF	-	-	×	-

In einer monolithischen Anwendung müsste man alles in einer Spalte unterbringen. Nicht nur, dass die Arbeit im Team und die Erweiterbarkeit darunter leiden, auch der Test und die Fehlersuche wären unter solchen Umständen schwieriger.

Auch wenn unser Urlaubsplaner aus mehreren Komponenten besteht, ist sein Aufbau, sein Klassendiagramm noch sehr überschaubar. Bezogen auf die Architektur ergibt sich folgendes Schaubild:



Model: Wir bilden alles in einer einzigen Klasse ab. Allerdings binden wir diese Klasse über ein Interface an den Presenter. Damit können wir die Model-Klasse jederzeit austauschen. Das wäre der Fall, wenn wir statt der hier verwendeten SQLite-DB ein anderes DBMS verwenden wollten.

Presenter: Die Klasse cfUrlaubsplaner wird im Start-Modul Main instanziiert. Hier siedeln wir unser Hauptfenster an. Dieses Fenster ist gleichzeitig der Container für die WebView2-Komponente.

View(s): Über zwei verschiedene Interfaces werden die Views angebunden. In den iDocument-Klassen werden statische HTML-Dokumente geladen und über das DOM mit Daten versorgt. Hier sorgt CSS für die Design-Aspekte und JavaScript sorgt für die Dynamik und Interaktion. Die iExport-Klassen bereiten die Daten aus dem Model für unterschiedliche Zwecke auf. Die Daten sind immer die Gleichen, werden aber entsprechend dem Format unterschiedlich dargestellt.

Es gibt im Projekt noch einige Helfer-Klassen, andere werden aus dem RC6-Framework geladen. Diese Klassen kann man nicht immer einer Schicht zuordnen. Die muss man sich wie Werkzeuge vorstellen: Nicht nur der Schmied braucht einen Hammer. Nein, auch der Schreiner, Spengler und Dachdecker verwenden dieses Werkzeug. Als Beispiel sei hier die bereits vorgestellte Factory genannt. Diese Klasse finden Sie im RC6-Framework unter dem Klassennamen *cConstructor*.

Presenter

Im Presenter befindet sich die Schaltzentrale der Anwendung. Und weil wir nun endgültig mit der Praxis durchstarten wollen, spare ich mir weitere theoretische Aspekte zu dieser Schicht. Aus praktischer Sicht ist der Presenter eine Instanz der Klasse *cfUrlaubsplaner*. Dieses Objekt wird beim Programmstart instanziiert und erzeugt selbst zwei weitere zentrale Objekte: Das Hauptfenster der Anwendung und die WebView2-Komponente, die vom Hauptfenster gehostet wird. Der Presenter bindet das Model an sich und je nach Situation eine View. Wenn das Programm das erste Mal ausgeführt wird, gibt es noch keine Daten. Somit wird die *cvNeu*-View erzeugt. Im Gegensatz dazu, wird bei einer geöffneten Datei, direkt nach dem Start, die *cvUrlaubsplaner*-View geladen und zur Anzeige gebracht. Der Daten-Output beider Views landet in der WebView2-Komponente. Weil die Komponente Ereignisse auslösen kann, treffen so sämtliche User-Aktionen im Presenter ein. Dieser delegiert daraus resultierende Aktionen an das Model oder an andere Views.

Wer nun erwartet hat, dass wir den Presenter step-by-step programmieren werden, der irrt. Wie gesagt, das ist kein klassisches Lehrbuch. Wir schauen uns die entscheidenden Stellen im Code an und ich werde die Hintergründe erklären. Im Klassendiagramm sehen wir ganz oben ein Rechteck mit der Beschriftung Main. Richtig vermutet: Das ist ein VB-Modul mit einer *Sub Main()*-Prozedur, mit der unsere Anwendung startet. In dieser Prozedur passieren einige Dinge, über die wir später im Deployment-Kapitel sprechen werden. Zunächst interessiert uns nur die Tatsache, dass hier einige globale Singleton-Objekte erzeugt werden, die wir später brauchen. Dazu zählen vor allem die Factory *New_c* und ein paar weitere Hilfsobjekte. Weiter unten in der Prozedur finden wir den Aufruf:

```
Set cfMain = New cfUrlaubsplaner
If Not cfMain.Show Then Exit Sub
```

Hier wird aus der *cfUrlaubsplaner*-Klasse das *cfMain*-Objekt erzeugt und zur Anzeige gebracht. Ok, *cfUrlaubsplaner* ist aber nur eine normale VB-Klasse und da gibt es eigentlich nichts zum Anzeigen. In VB sind dafür Fenster-Klassen zuständig. Fast richtig, die normale VB-Klasse wird nicht angezeigt, jedoch ein Fenster, dass in der Klasse erzeugt wird. Dazu sehen wir uns die Klasse genauer an. Im Deklarationsteil steht:

```
Private WithEvents mfUrlaubsplaner As cWidgetForm   'Fenster
Private WithEvents mWV As cWebView2                 'WebView2-Komponente
Private mvDocument As iDocument                     'View, irgendeine
Private mUrlaubsplaner As iData                     'Model
```

Was aber ist eine *cWidgetForm*? Im Kommentar steht, es wäre ein Fenster?! Genau, das ist es. Wir deklarieren hier eine Fenstervariable, die auf einer Klasse auf dem RC6-Framework basiert. Hätten wir auch bei der ursprünglichen VB-Form bleiben können? Im Prinzip ja. Durch die Verwendung der RC6-Klasse, verzichten wir zukünftig auf die veralteten VB-Fenster. Mit den neuen Fenstern können wir weitere moderne Features aus dem RC6-Framework nutzen, Stichwort *Cairo*.

Aus diesem Grund macht es Sinn, dass wir an dieser Stelle einen Exkurs in die RC6-Welt und Cairo machen. Cairo (https://www.cairographics.org) ist eine 2D-Grafikbibliothek, die in der Programmiersprache C geschrieben ist, jedoch Bindungen für andere Programmiersprachen bietet. Cairo ist eine freie Software die unter den Bedingungen der LGPL-Version 2.1 (GNU Lesser General Public License) oder der MPL-Version 1.1 (Mozilla Public License) verwendet werden kann. Als solche wurde sie vom Autor des RC6 in dieses Framework integriert. Für Cairo gibt es in RC6 mehrere Klassen, die wichtigste

ist *cCairo*. Bei registrierter RC6.dll können wir *Cairo* genauso wie das *New_c* Objekt ohne explizite Instanziierung verwenden. In der kompilierten Anwendung geht das nicht, was aber nicht weiter schlimm ist. Wir haben dort die Factrory *New_c*, die in der Lage ist uns ein fertig instanziiertes *Cairo* per Methodenaufruf zu liefern. Dazu mehr im Kapitel Deployment.

Zusammen mit der Klasseninstanz *cfUrlaubsplaner* erzeugen wir das Hauptfenster der Anwendung. Der *Create*-Methode können wir unsere Wünsche zu den Fensterdimensionen mitgeben.

```
Private Sub Class_Initialize()
Dim w As Long, h As Long
  '...
  Set mfUrlaubsplaner = Cairo.WidgetForms.Create(Width:=w, Height:=h)
End Sub

Public Function Show(...) As Boolean
  '...
  With mfUrlaubsplaner
    .CenterOn mfUrlaubsplaner.WidgetRoot.CurrentMonitor
    '.CenterOn New_c.Displays.Primary
    .Show
  End With
  Set mWV = New_c.WebView2
  ret = mWV.BindTo(mfUrlaubsplaner.hWnd,...)
  '...
  mWV.IsStatusBarEnabled = False

  file = GetSetting(App.ProductName, "Settings", "LastFile", "")
  If Not fso.FileExists(file) Then file = ""

  Set mUrlaubsplaner = New cmUrlaubsplaner
  If Len(file) = 0 Then
    Call newFile        'Start mit Neu
  Else
    Call openFile(file) 'Start mit Urlaubsplaner
  End If
  Show = True
End Function
```

Vor dem Anzeigen des Fensters sorgen wir, dass es auf dem aktuellen Bildschirm zentriert wird. Die Methode *CurrentMonitor* liefert dafür ein *Display*-Objekt. Im RC6-Framework gibt es die Auflistung *cDisplays,* die alle Bildschirme als *cDisplay*-Objekte enthält. Diese Objekte, als auch die Auflistung sind nützliche Werkzeuge. Sie bieten Funktionen, die in VB schon immer fehlten. Es lohnt sich also einen Blick in den Objekt-Katalog zu werfen, um die Member dieser Klassen zu studieren. Um mit der Auflistung zu arbeiten, kann man sich diese über die Factory erzeugen lassen. So kommt man mit einem Aufruf an den Primärbildschirm, wenn die Anwendung bevorzugt dort starten soll, siehe auskommentierte Zeile.

Die *BindTo*-Methode bindet nun die WebView2-Komponente an das Elternfenster durch die Übergabe der *hWnd*-Eigenschaft im ersten Parameter. Die Methode kann noch weitere Parameter entgegennehmen. Diese sind weiter hinten im Buch in der *cWebView*-Referenz ausführlich beschrieben. Das Zusammenspiel dieser Parameter ist wichtig für den erfolgreichen Start der Anwendung. Sie sind weiter hinten im Buch in der *cWebView*-Referenz / *BindTo*-Methode ausführlich beschrieben. Uns interessiert zunächst nur der Rückgabe-Wert, der in die Variable *ret* gespeichert wird. Wenn die Rückgabe gleich 1 ist, hat die Bindung funktioniert. Hingegen zeigt eine 0, dass irgendwas schiefgelaufen ist. In der Regel wird das eine fehlende Runtime sein. Deshalb sollte neben der

Anwendung auch der Evergreen Bootstrapper von Microsoft mitgeliefert werden. Dieses Programm besorgt und installiert die aktuelle WebView2-Runtime. Dieser Prozess wird vom Installer-Modul überwacht und sorgt dafür, dass nach der Installation sofort mit dem Programmstart fortgefahren wird. Ein Programm-Neustart, schweige denn ein Rechner-Neustart ist nicht erforderlich.

Die Methode *IsStatusBarEnabled* sorgt dafür, dass die WebView-eigene Statusbar unterdrückt wird. Diese mag in einem Browser sinnvoll sein, weil sie immer das Ziel eines Links anzeigt, wenn man ihn mit der Maus überfährt. In einer Desktop-Anwendung ist das eher befremdlich, zumal hier nichts Vernünftiges angezeigt werden kann – wir haben ja keine Links im klassischen Sinne. Deshalb: Ausschalten!

Per *GetSetting*-Methode sehen wir nach, ob in der Registry eine Datei als zuletzt genutzte Datei registriert ist. Bei dem ersten Programm-Start ist das sicher nicht der Fall, also wird sich unser Presenter die Prozedur *newFile* ausführen. Das bedeutet, dass die *cvNeu*-View geladen wird.

Spätestens nachdem die erste Datei gespeichert ist, wird diese als *LastFile* in der Registry hinterlegt und steht beim nächsten Programmstart zur Verfügung. Abgesichert wird diese Aktion durch ein FSO-Objekt (File System Object) aus dem RC6-Framework. Dieses Objekt ist ein Universalwerkzeug für alle möglichen Datei- und Verzeichnis-Operationen. Klar, einiges davon könnten wir auch mit den VB-eigenen Funktionen realisieren. Der FSO-Funktionsumfang geht aber weit über den VB-Standard hinaus. An dieser Stelle prüfen wir lediglich, ob die Datei existiert. Könnte ja sein, dass der Anwender die Datei zuletzt auf einen USB-Stick gespeichert hat. Beim erneuten Programmstart könnte der Stick fehlen. Sollte das zutreffen, wird erneut die Funktion *newFile* angewendet.

newFile

```
Private Sub newFile()
  Set mvDocument = New cvNeu
  mvDocument.LoadPage mWV, mUrlaubsplaner.GetSettings, mSkin
End Sub
```

Wir haben *mvDocument* im Deklarationsteil als *iDocument* deklariert. Da die Klasse *cvNeu* die *iDocument* implementiert, können wir deren Ableger an die *mvDocument*-Variable zuweisen. Aus Sicht des Presenters gibt es nur diese eine Variable. Je nach Bedarf können wir damit beliebige Objekte referenzieren, wenn sie die *iDocument*-Schnittstelle implementieren. *LoadPage* ist eine Methode dieser Schnittstelle, folglich finden wir sie auch in den abgeleiteten Views. Wir werden das gleiche Prinzip in der nächsten Prozedur antreffen. Dort wird entsprechend der Funktion, eine andere schnittstellenkompatible Klasseninstanz zugewiesen.

openFile

Im ersten Teil der Prozedur wird der Windows-Standarddialog zum Öffnen von Dateien aufgerufen. Dafür brauchen wir weder das alte VB-OCX, noch die Windows-API. Wir finden eine entsprechende Methode im bereits kennengelernten *FSO*-Objekt. Genaugenommen kapselt die Methode auch nur die API-Funktion, folglich gibt es hier nichts Besonderes zu erklären.

Interessanter wird es ab der *cmUrlaubsplaner*-Instanziierung. Wir sehen hier zum ersten Mal die Model-Klasse. Genauso wie die Views wird diese Klasse über ein Interface angesprochen, was eine problemlose Austauschoption sicherstellt.

```
'...
Set mUrlaubsplaner = New cmUrlaubsplaner
If mUrlaubsplaner.OpenKalender(file, biv, dbv, updateVer) Then
  FileName = file
  Set mvDocument = New cvUrlaubsplaner
  '...
```

OpenKalender ist eine Methode der Schnittstelle *iData*. Die Methode erwartet den Dateinamen der Urlaubsplaner-Datei und drei weitere optionale Parameter. Mit Hilfe dieser Parameter wird geprüft, ob die Datei in einer älteren Version vorliegt, die zur Software-Version nicht kompatibel ist. Dieser Fall kommt in der Praxis häufig vor. Angenommen wir erweitern die Anwendung in ein paar Monaten zu einer Version 2.0. Diese Version bietet weitere Funktionen, die neue Datenstrukturen im Model voraussetzen, zum Beispiel eine neue Tabelle. Wir liefern diese Version aus und beim Anwender hagelt es Fehlermeldungen, weil in den Alt-Dateien die benötigte neue Tabelle fehlt. Das gilt es zu vermeiden. Um den alten Datenbestand zu retten, müssen wir die Dateien aktualisieren. Das erledigt der dritte Parameter *updateVer*. Vorausgesetzt der Anwender stimmt zu, rufen wir die Methode *OpenKalender* erneut auf, diesmal mit einem *True* im *updateVer*-Parameter. Das sorgt dafür, dass die Datei vom Model ein Upgrade erfährt und danach endgültig geöffnet werden kann.

```
'...
Set mvDocument = New cvUrlaubsplaner
mvDocument.LoadPage mWV, mUrlaubsplaner.GetSettings, mSkin
SaveSetting App.ProductName, "Settings", "LastFile", file
SaveSetting App.ProductName, "Settings", "AppData", _
                    fso.GetPathNameFromFullPath(file)
```

Nun sehen wir den Fall, der in der Prozedur *newFile* bereits angesprochen wurde: Die gleiche *mvDocument* Variable bekommt eine andere View zugewiesen. Der Methoden-Aufruf *LoadPage* ist aber identisch, dafür sorgt die gemeinsame Schnittstelle.

Die *SaveSetting*-Zeilen sind für den VB-Programmierer ein alter Hut. Sie sind hier nur deshalb abgedruckt, weil eine weitere nützliche FSO-Methode zum Einsatz kommt. Die Methode extrahiert aus dem vollständigen Dateipfad den Ordnerpfad. Hat sicher jeder VB-Programmierer schon mal selbst umgesetzt, weil die VB-Runtime diese Funktion nicht bietet. Das FSO kennt einige weitere Methoden für ähnliche Aufgaben, der VB-Objektbrowser zeigt sie euch. Da die Methodennamen recht gut auf die Funktion schließen lassen und die Parameter auch selbsterklärend sind, dürfte der Einsatz in zukünftige Projekte gut gelingen.

createFile

Nachdem sich der Anwender für einen neuen Urlaubsplaner entschieden hat, wird dieser hier angelegt. Für das Model gibt es eine neue *cmUrlaubsplaner*-, auf der View-Seite eine neue *cvUrlaubsplaner*-Instanz. Das Model legt einen neuen Kalender an, dieser wird mit den Parametern Jahr und Bundesland vorkonfiguriert. Die View sorgt anschließend für das Laden der neuen Seite.

```
Private Sub createFile(jahr As Integer, bundesland As eBundeslaender)
  Set mUrlaubsplaner = New cmUrlaubsplaner
  Set mvDocument = New cvUrlaubsplaner
  mUrlaubsplaner.NewKalender jahr, bundesland
  mvDocument.LoadPage mWV, mUrlaubsplaner.GetSettings, mSkin
End Sub
```

importFile

Auch hier wird ein neuer Urlaubsplaner angelegt. Als Basis dienen die Stamm-Daten einer vorhandenen Datei, wenn der Anwender einen Jahreswechsel vornimmt. Personendaten und Konfigurationen werden aus der alten Datei gelesen und in die neue Datei übertragen. Inhaltlich ist diese Prozedur ähnlich aufgebaut wie die bisher vorgestellten Prozeduren. Der Öffnen-Dialog ist fast identisch, es unterscheidet sich lediglich der Dialog-Titel. Danach braucht es nur noch den zusätzlichen Methodenaufruf *ImportPersonenstamm* im Model.

```
If mUrlaubsplaner.ImportPersonenstamm (file, jahr, bundesland) Then ...
```

saveFile

Hier wird ein Urlaubsplaner gespeichert. Falls es dafür noch keinen Dateinamen gibt, sorgt das *FSO*-Objekt wieder für den passenden Speichern-Dialog. Eine eventuell vorhandene Datei wird davor gelöscht und mittels *SaveKalender*-Methode des Models gespeichert. Der neue Dateiname wird per *SaveSetting* in der Registry festgehalten und steht beim nächsten Programmstart wieder zur Verfügung.

exportFile

In dieser Funktion kommt unser zweites View-Interface zum Einsatz. Egal welches Format der Anwender exportieren will, im Presenter erledigt alles diese eine Prozedur. Wir könnten noch einige weitere Export-Formate festlegen, an diesem Code müssen wir nichts ändern. Für jedes Export-Format kommt eine eigene Klasse, die das *iExport*-Interface implementiert, zum Einsatz. Theoretisch können diese Klassen auch in eigene DLLs ausgelagert werden. Im Kapitel Views werden wir die vorhandenen Export-Klassen besprechen und weitere spannende Klassen aus dem RC6-Framework kennenlernen.

Im oberen Codeabschnitt wird der bereits bekannte Speichern-Dialog des *FSO*-Objektes verwendet, um den Pfad und Dateinamen für die Export-Datei festzulegen. Der Dialog wird vorab per *FileExtension* und *Filter* konfiguriert. Diese Parameter unterscheiden sich je nach Export-Format. Die Schnittstellendefinition *iExport* sorgt dafür, dass jede Klasse kompatible Daten liefert.

Aus Sicht des Presenters wird es danach einfach. Alle *iExporter*-Klassen bieten die Methode *GetData*, die ein *Byte-Array* mit Daten im passenden Format liefert. Das gilt für einfache Text-Formate (*CSV*, *HTML*) genauso wie für binäre Daten. Eine weitere Methode des *FSO*, die *WriteByteContent* schreibt diese Daten in eine neue Datei.

Falls der Anwender die Option „Nach dem Speichern gleich öffnen“ gewählt hat, reicht ein einziger Methoden-Aufruf im *FSO*, um die Datei zu öffnen. Dafür muss auf den Anwenderrechner selbstverständlich eine für das Datei-Format zuständige Anwendung installiert sein.

```
Private Sub exportFile(exporter As iExport, _
                       hdSettings As RC6.cHashD, _
                       Optional shellExec As Boolean)
  '...
  fso.WriteByteContent file, exporter.GetData(hdSettings)
  If shellExec Then Call fso.ShellExecute(file)
End Sub
```

showOptionen

Der Optionen-Aufruf ist aus Sicht des Presenters wieder eine sehr entspannte Angelegenheit. Da es sich hierbei ebenfalls um eine View handelt, die die *iDocument*-Schnittstelle unterstützt, braucht es

lediglich die Instanziierung der Klasse und den anschließenden *LoadPage*-Methodenaufruf. Später muss die so geladene View nur noch mit Daten befüllt werden.

```
Private Sub optionen()
  Set mvDocument = New cvOptionen
  mvDocument.LoadPage mWV, mUrlaubsplaner.GetSettings, mSkin
End Sub
```

defragSlots

Auch für diese Funktion erledigt der Presenter nur die Koordination. Die eigentliche Arbeit wird von der Programmlogik, also vom Model erledigt. Das Ergebnis zeigt die geladene View:

```
Private Sub defragSlots()
Dim vUrlaubsplaner As cvUrlaubsplaner
  mUrlaubsplaner.Defrag
  Set vUrlaubsplaner = mvDocument
  vUrlaubsplaner.ShowKalender mUrlaubsplaner.GetKalender
  vUrlaubsplaner.ShowUrlaub mUrlaubsplaner.GetUrlaub
End Sub
```

Es gibt einen kleinen Unterschied: Wir brauchen für die Aktualisierung des Kalenders Methoden, die im Interface *iDocument* nicht enthalten sind. Deshalb deklarieren wir eine temporäre Objektvariable *vUrlaubsplaner* vom Typ *cvUrlaubsplaner*. Dieser Variable können wir problemlos die bestehende *mvDocument* zuweisen, weil die Klasse das *iDocument*-Interface unterstützt. Wohlgemerkt: wir haben nun zwei Objektvariablen, die auf das gleiche Objekt verweisen. Das ist so, als ob wir den gleichen Stern mit zwei unterschiedlichen Teleskopen betrachten. Einmal mit einem optischen, einmal mit einem Radioteleskop. Wir sehen immer das gleiche Objekt, jedoch unterschiedliche Eigenschaften. Die *vUrlaubsplaner* sieht die vollständige Schnittstelle der *cvUrlaubsplaner*-Klasse und kann so Methoden aufrufen, die aus Sicht der *mvDokument*-Variable verborgen sind.

Es gibt noch ein paar weitere Prozeduren in dieser Klasse, auf die ich hier nicht weiter eingehen werde. Das sind Eigenschaften und Methoden, die man überall antrifft, wie z.B. *FileName* zum Speichern des Dateinamens, *Skin* zum Speichern des Erscheinungsbildes oder *checkForChangedData*, damit der Anwender das Speichern der Änderungen nicht vergisst.

Interessanter sind die Ereignis-Prozeduren, die von der WebView2-Komponente ausgelöst werden. Einige davon ergeben sich aus Zustandsänderungen in der Komponente, andere aus Anwender-Aktionen, wie das Klicken auf einen Button. Alle diese Ereignisse schlagen im Presenter auf, in der Klasse also, die den WebView2 hostet. Das verdeutlicht noch mal die Passivität der Views.

Ereignis-Prozeduren

Schauen wir uns noch mal das Konzept der Interaktion des Presenters mit den Views an. Folgender Pseudo-Code zeigt das Wesentliche dieser Zusammenarbeit:

```
Select Case View
  Case Neu:      Set mvDocument = New cvNeu
  Case kalender: Set mvDocument = New cvKalender
  Case optionen: Set mvDocument = New cvOptionen
  End Select
mvDocument.LoadPage mWV, settings, skin

Private Sub mWV_DocumentComplete()
  If TypeOf mvDocument Is cvNeu Then
```

```
    'Lade Daten und übertrage diese in die View
  ElseIf TypeOf mvDocument Is cvUrlaubsplaner Then
    'Lade Daten und übertrage diese in die View
  ElseIf TypeOf mvDocument Is cvOptionen Then
    'Lade Daten und übertrage diese in die View
  End If
End Sub

Private Sub mWV_JSMessage(ByVal sMsg As String, _
                          ByVal sMsgContent As String, _
                          oJSONContent As RC6.cCollection)

  Select Case sMsg
    Case "urlaubsplaner.tdClick"
      'der Anwender hat in eine Tabellenzelle des Urlaubsplaners geklickt
    Case "urlaubsplaner.mnuClick"
      'der Anwender hat einen Menüpunkt im Hauptmenü gewählt
    Case "neu.btnClick"
      'der Anwender hat einen Schalter im Neu-Dialog geklickt
    Case "optionen.btnClick"
      'der Anwender hat einen Schalter im Optionen-Dialog geklickt
  End Select
End Sub
```

Je nach Bedarf wird der *mvDocument*-Objektvariable eine anderes Objekt zugewiesen. Danach wird die Methode *LoadPage* der jeweiligen Klasse aufgerufen. Die Referenz auf die WebView2 wird im ersten Parameter übergeben. Die weiteren Parameter sorgen für Einstellungen und Erscheinungsbild. Intern macht die Methode immer das Gleiche: Es wird ein Template, also eine leere Vorlage für die Seite in die WebView2 geladen und ggf. ein alternativer Skin für das Aussehen angewendet.

Das fertige Laden wird von der WebView2-Komponente im Ereignis *DocumentComplete* signalisiert. Der Präsenter liefert nun, in Abhängigkeit von der geladenen View die Nutzdaten. Die Quellen für die Daten unterscheiden sich, das meiste wird die Model-Klasse liefern. Andere Daten werden direkt aus der Registry per *GetSetting*-Methode gelesen.

Bei den Ereignissen, die der Anwender auslöst, ist es nicht viel anders: Alle Meldungen kommen in der zentralen Ereignisprozedur *mWV_JSMessage* an. Wie der Prozedur-Name schon vermuten lässt, wird dieses Ereignis mittels JavaScript in der WebView2-Komponente ausgelöst. In der Tat, die Klasse *cWebView* erzeugt in jeder geladenen Seite ein globales *vbH*-Objekt. Seine Methode *RaiseMessageEvent* ist der zentrale Rückkanal aus JavaScript in die VB-Welt. Das ist genial gelöst und kommt uns als VB-Entwickler sehr entgegen. Der Grund ist der, dass die mit VB angelegten Interfaces keine Events unterstützen. Die Views können sich also nicht ohne Weiteres beim Presenter melden, zumindest nicht über die Schnittstelle. Somit kann die *mvDocument*-Objektvariable auch keine Ereignisse empfangen. Es gibt in einschlägigen Foren und VB-Seiten immer wieder irgendwelche Tricks, um das zu umgehen. Wirklich schöne Lösungen sind das nicht. Zumindest belasten sie die saubere Trennung der Schichten oder führen früher oder später zu „zirkulären Referenzen“. Letztere sind vom Windows-Programmierer genauso gefürchtet wie DLL-Konflikte (*DLL Hell*).

Schauen wir uns die Argumente der *JSMessage*-Ereignisprozedur an. JavaScript-seitig rufen Sie dazu die Methode *RaiseMessageEvent* auf. Diese Methode erwartet lediglich zwei Parameter. Wärend der erste Parameter ein normaler String ist, können Sie den zweiten Parameter wahlweise mit Strings oder mit JSON-Daten befüllen. Den ersten Fall können wir bei einfachen Nachrichten verwenden, wie das Beispiel der Menüs zeigt:

```
Select Case sMsg
  Case "urlaubsplaner.mnuClick"
    Select Case sMsgContent
      Case "mnuNew":      If checkForChangedData Then Call newFile
      Case "mnuOpen":     If checkForChangedData Then Call openFile
      Case "mnuSave":     Call saveFile: mWV.jsRun "closeNav"
      Case "mnuPrint":    Call printPage
      Case "mnuDefrag":   Call defragSlots
      Case "mnuOpt":      Call optionen
      Case "mnuClose":    mfUrlaubsplaner.Unload
    End Select
  Case ...
End Select
```

Das *sMsg*-Argument zeigt, dass der Anwender ein Menü gewählt hat, im zweiten Argument *sMsgContent* wird präzisiert um welches Menü es sich handelt. Die verschachtelte Select-Case Anweisung ordnet diese Fälle wie eine Weiche den bereits vorgestellten Presenter-Funktionen zu. Die Namen der Argumente suggerieren, dass im ersten Parameter der Name einer Nachricht (Message) und im zweiten Argument der Inhalt (Content) übergeben wird. Frei nach Faust: „Name ist Schall und Rauch“, bleibt es dem Programmierer überlassen, wie und womit er die Parameter befüllt.

Für die Übermittlung komplexerer Daten bietet sich deshalb das JSON-Format an. Hiermit können strukturierte Text-Daten übertragen werden. JavaScript-seitig kann JSON mittels Methode *stringify*() ein beliebiges JavaScript-Objekt dafür aufbereiten, auf der VB-Seite kommt ein weiteres Highlight aus dem RC6-Framework zu Einsatz, die *RC6.cCollection*.

Dieses Collection-Objekt ersetzt die VB-eigene Collection und bietet eine Vielfalt von Zusatz-Eigenschaften und Methoden. Und sie ist auch noch wesentlich schneller als das Original.

Um die Arbeit mit dieser Collection zu verdeutlichen, müssen wir uns zunächst den Aufbau von JSON ansehen. Wer damit bereits vertraut ist, kann diesen Abschnitt überspringen. Ich werde mich aber auch kurzfassen, weil wir hier nicht JavaScript lernen wollen.

Schauen wir uns die Optionen unseres Urlaubsplaners an. Wir bestimmen hier einige Programm-einstellungen und erfassen diverse Daten zu Feiertagen, Ferien und Betriebsurlaub. Um alle Daten in einer Struktur zusammenzufassen, bietet sich eine Baumstruktur (Treeview) an. Diese Strukturen können sowohl Werte-Paare als auch hierarchisch geordnete Listen enthalten. Für solche Daten ist das XML-Format prädestiniert. JSON kann das auch, bietet sogar einige Vorteile:

- Es braucht keine End-Tags
- Es ist kompakter und verwendet die JavaScript-Syntax. Damit kann es viel effizienter gelesen und erzeugt werden.

Die Notation von JSON ist einfach: Die Daten liegen in Name- / Werte-Paaren vor und werden durch Kommas getrennt. Objekte werden in geschweifte Klammern eingebunden, Arrays in eckige Klammern.

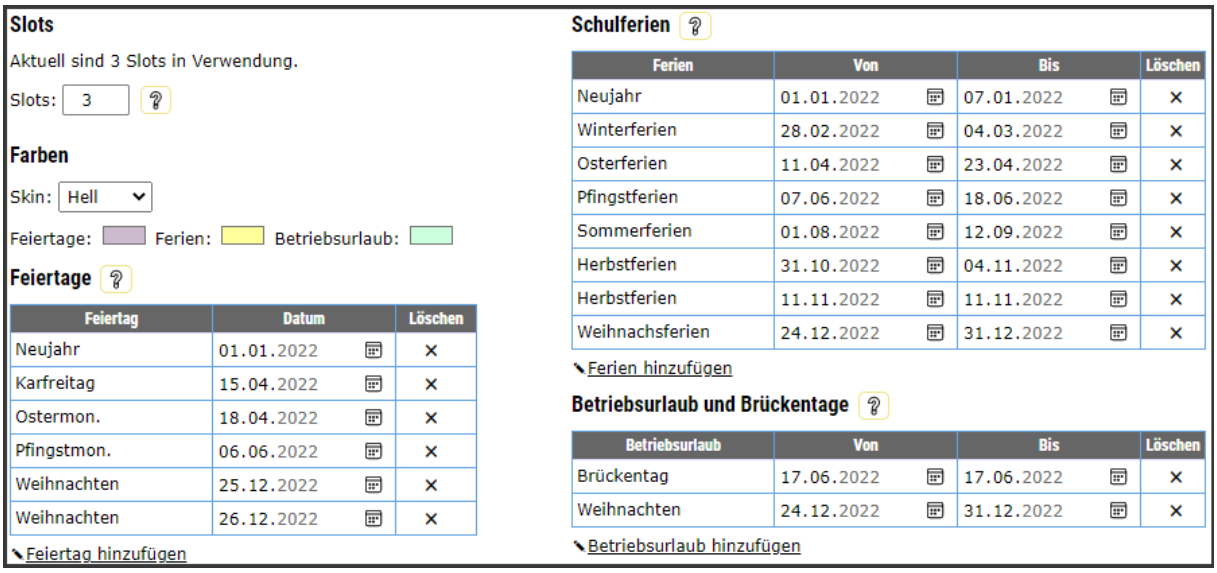


Abbildung 14: Optionen-Dialog

Aus den Optionen-Screenshot ergibt sich folgendes, auch für Menschen gut lesbares und nachvollziehbares JSON:

```
{
 feiertage:[
  {"id":"44562","name":"Neujahr","datum":"2022-01-01"},
  {"id":"44666",name":"Karfreitag","datum":"2022-04-15"},
  {"id":"44669","name":"Ostermon.","datum":"2022-04-18"},
   ...
  ],
urlaub:[
  {"id":"44729","name":"Brückentag","von":"2022-06-17","bis":"2022-06-17"},
  {"id":"44919","name":"Weihnachten","von":"2022-12-24","bis":"2022-12-31"}
 ],
ferien:[
    ...
 ],
 "slots":"3",
 "cFeiertage":"#ccbbcc",
 "cFerien":"#ffff99",
 "cUrlaub":"#ccffdd",
 "skin":"0",
 "pdf":"0",
 "csv":"<kz>;<name>;<datum>",
 "open":true
}
```

Diese Daten (für bessere Lesbarkeit, hier mit Umbrüchen abgedruckt) kommen im zweiten Argument, also in *sMsgContent* an. Als mittelloser VB-Entwickler müssten wir nun einen Parser schreiben, der uns diese Daten zerlegt und in passende VB-Strukturen modelliert. Brauchen wir nicht, die RC6-Runtime erkennt den JSON-Aufbau und befüllt damit das dritte *oJSONContent*-Argument. Was da genau enthalten ist, lassen wir uns von der VB-IDE anzeigen. Mit einem Haltepunkt in der Prozedur, können wir das *oJSONContent*-Objekt im Überwachungsfenster visualisieren:

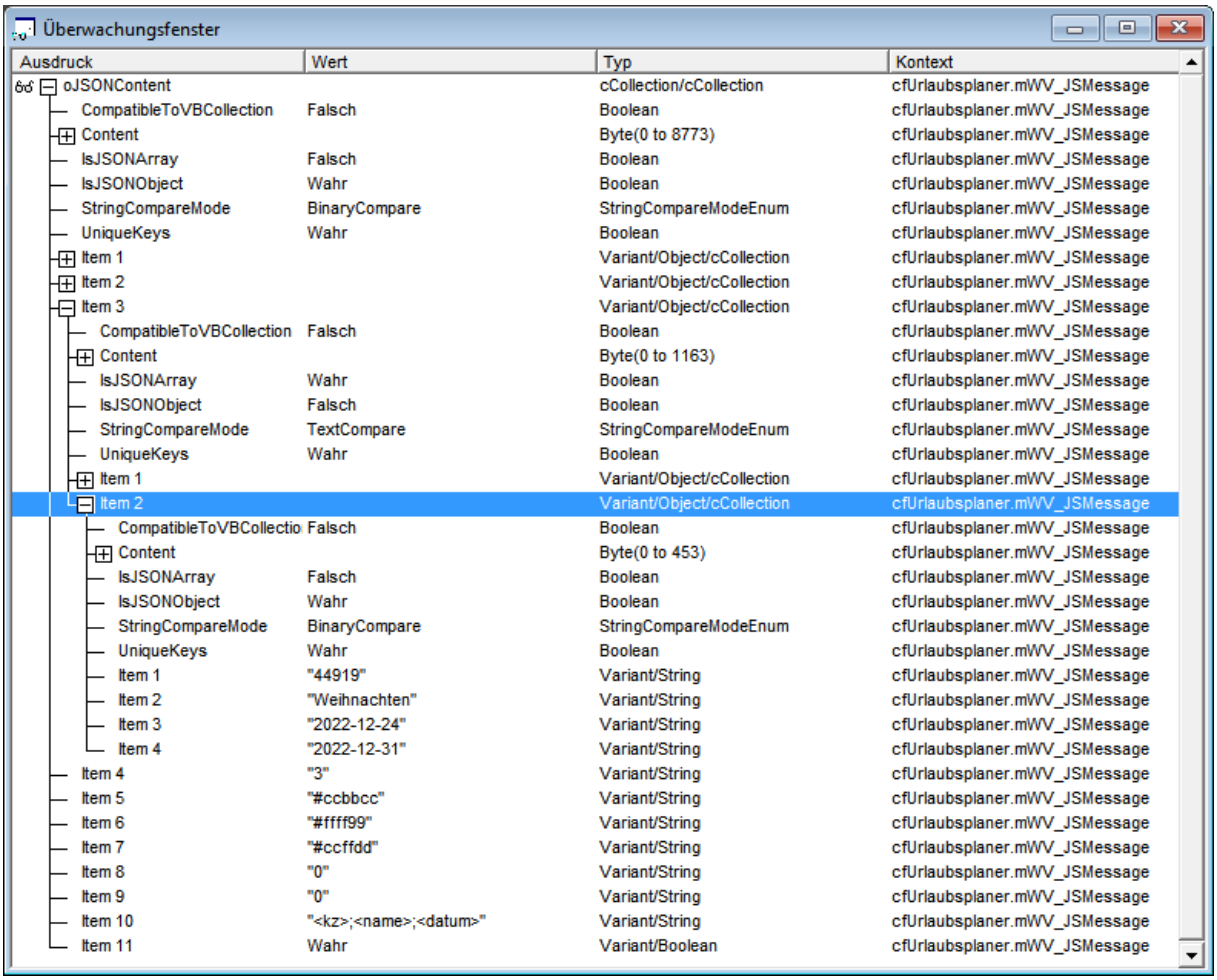


Abbildung 15: Konvertierte JSON-Daten in einem cCollection-Objekt

Wir sehen in der Wurzel der *cCollection* unser Argument, das *oJSONContet*-Objekt. Wir können auch einige Eigenschaften erkennen, die diese Collection beschreiben:

- *CompatibleToVBCollection = Falsch*. Die cCollection kann als VB-Ersatz für dessen Collection dienen. Diese hier wäre allerdings damit nicht kompatibel. Wenn ihr das RC6-Framework im Projekt ohnehin referenziert, bietet es sich an die cCollection statt der VB-Collection zu nutzen. Selbst wenn ihr keine der Zusatzfunktionen braucht, werdet ihr von mehr Speed profitieren.
- *IsJSONArray = Falsch*. Das stimmt! Unser JSON-Datenobjekt besteht nur aus Eigenschaften. *feiertage*, *ferien* und *urlaub* (gemeint ist hier der Betriebsurlaub) sind drei Arrays, alle anderen Eigenschaften sind Werte-Paare.
- *IsJSONObject = Wahr*. Das *oJSONContet*-Objekt ist als JSON-Objekt identifiziert, das ist keine Überraschung.
- *UniqueKeys = Wahr*. Klar, wir können also jedes Element in der Collection über einen eindeutigen Schlüssel ansprechen. Und wie? Ganz einfach: Das Direktfenster starten und dort folgendes eingeben: *?oJSONContent.KeyByIndex(0).* Das Ergebnis lautet: *feiertage*. Das könnt ihr mit einem anderen Index auch versuchen. Es wird sich zeigen, dass die Schlüssel den Namen der Eigenschaften entsprechen, genau wie ihr diese im JavaScript-Code definiert habt.
- *Content* ist ein *Byte-Array*, das die Rohdaten enthält. In der Praxis werden wir das eher selten brauchen.
- *Item 1 – Item 11* Das sind die Elemente der Collection. Alles was in JSON ein Objekt war ist hier wieder ein *cCollection*-Objekt. Item 2 habe ich im Screenshot aufgeklappt, um dessen Inhalt

anzuzeigen. Und wir sehen wieder Items, in denen die beiden Brückentage aufgeführt sind. Diese wiederum bestehen nur noch aus Variant-Variablen.

Warum aber sind die Variants alle vom Typ String? Die Eigenschaft *Slots* sollte zumindest nummerisch sein. Und die Daten der Kalendertage könnten ja auch vom Typ *Date* sein. Das ist richtig. Der RC6-JSON-Parser weiß das aber nicht. Er konvertiert das, was er an Daten zugewiesen bekommt. Und das sind nun mal Strings, weil wir die JSON-Objekte genauso befüllt haben. Um dem vorzugreifen: Im JavaScript-Code steht:

```
data.slots = document.getElementById('txtSlots').value;
```

Hier wird also das Value-Attribut eines Textfeldes ausgelesen. Auch wenn dieses Feld in der HTML-Vorlage als *number* deklariert wurde,

```
<input type="number" id="txtSlots" min="2" max="9" value="2" >
```

ändert das nichts an der Tatsache, dass *value* vom Typ String ist. Hier kann ich nur vermuten, dass *value* aus Gründen der Abwärtskompatibilität noch immer String ist, *type="number"* wurde erst mit HTML-5 eingeführt.

Natürlich könnte man bereits im JavaScript-Code die Slots-Eigenschaft vorab in eine Zahl umwandeln,

```
data.slots = parseInt(document.getElementById('txtSlots').value);
```

das Ergebnis wäre im *oJSONContet* ein in Variant/Currency. Fakt ist, irgendwo müssen wir uns vergewissern, dass der Anwender keine ungültigen Daten eingibt. Spätestens da beginnen wieder die Glaubensfragen über die richtige Stelle, die ich hier nicht diskutieren will.

savePersonen

Die Personenverwaltung ist eine einfache Tabelle mit unterschiedlichen Eingabefeldern. Wenn im Kalender neue Personen angelegt oder die Eigenschaften einer vorhandenen Person geändert werden, müssen die Änderungen zum Model übertragen werden. Auf der View gibt es dafür den Button „*Übernehmen*“.

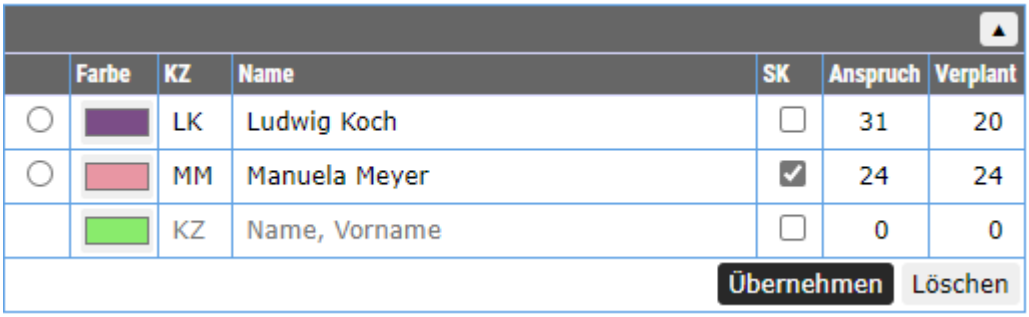


Abbildung 16: Personendaten speichern

Der Klick auf diesen Button schlägt erneut in der universellen *JSMessage*-Prozedur des Presenters auf. Und auch diesmal empfangen wir strukturierte Daten, die jedoch einfacher aufgebaut sind. Das widerspiegelt sich im *oJSONContent*-Aufbau. Statt Hierarchien übertragen wir hier nur alle Feld-IDs und deren Werte. Hierbei können wir von den assoziativen Arrays in JavaScript profitieren. Es ergeben

sich sogenannte Schlüssel-Werte-Paare. Das passt wunderbar zu einer Collection. Die Feld-Id wird zum Element-Key, der Feld-Wert zum Element-Wert. Beispiel:

```
//Daten mit JavaScript einem assoziativen Array speichern
data["txtName"] = item.value;
//Daten übertragen:
vbH().RaiseMessageEvent('personen.save',JSON.stringify(data));

'Daten in VB auslesen:
name = data.item("txtName")
```

Für den Presenter wird damit die Speicherung der Daten wieder sehr kompakt:

```
Case "personen.save"
  If mUrlaubsplaner.SavePersonen(oJSONContent, errors) Then
    vUrlaubsplaner.ShowPersonen mUrlaubsplaner.GetPersonenStatus
    vUrlaubsplaner.ShowUrlaub mUrlaubsplaner.GetUrlaub
  End If
```

Wir übergeben nur noch die *oJSONContent*-Collection an das Model zum Speichern. Danach wird die View aktualisiert.

btnXXXExport

Soeben haben wir in JavaScript assoziative Arrays kennengelernt. Vielleicht sind euch diese auch schon unter dem Begriff Hashtabelle untergekommen. Das RC6-Framework bietet uns die Klasse cHashD, die auch eine Auflistung von Schlüssel-Wert-Paaren darstellt. Wir verwenden hier diese Werte-Sammlung, um die unterschiedlichen Export-Klassen zu konfigurieren. So können wir beim PDF-Export das Seitenformat festlegen oder für den CSV-Export den Aufbau der Felder. Bild-Dateien benötigen weder das eine noch das andere. Die Bild-Abmessungen ergeben sich aus der Anzahl der Slots und das Bildformat (JPG oder SVG) können wir über die Interface-Eigenschaft *FileExtension* übertragen.

Das Verwenden einer Hash-Liste ist vorteilhaft, weil:

- sie einen indizierten Zugriff sowohl auf die Keys als auch auf die Items bietet
- Anders als in VB, die Item-Werte überschrieben werden können
- beim Hinzufügen von Schlüssel-Werte-Paaren die Performance um Größenordnungen schneller als das VB-Original ist. Beim Lesen noch immer doppelt so schnell ist
- mittels *Exists*-Methode geprüft werden kann, ob ein Schlüssel bereits enthalten ist

In unserem Projekt sorgt diese Klasse dafür, dass wir mit alles Mögliche an Einstellungen über eine immer gleichbleibende Schnittstelle transportieren können. Im folgendem Code-Auszug sehen wir die Export-Konfiguration in Abhängigkeit vom Format:

```
Set hdSettings = New_c.HashD

'HTML-Export
With mUrlaubsplaner.GetSettings
  hdSettings.Add "jahr", .fields("jahr").Value
  hdSettings.Add "slots", .fields("slots").Value
  hdSettings.Add "cFeiertag", .fields("cfeiertag").Value
  hdSettings.Add "cFerien", .fields("cferien").Value
  hdSettings.Add "cUrlaub", .fields("curlaub").Value
End With
```

```
'PDF-Export, zusätzlich
hdSettings.Add "format", _
                mWV.jsProp("document.getElementById('selPDF').value")

'CSV-Export, nur
hdSettings.Add "template", _
                mWV.jsProp("document.getElementById('txtCSV').value")
```

An dieser Stelle begegnen wir einer weiteren cWebView-Eigenschaft, die *jsProp*. Damit können wir bequem Attribute oder CSS-Eigenschaften von DOM-Elementen auslesen oder ändern. Dazu übergeben wir der Methode einen JavaScript-Einzeiler, der das DOM-Element identifiziert und anschließend eine Eichenschaft abruft. Ein DOM-Element ist am einfachsten über seine *id* identifizierbar. Das erledigt dann die *getElementById()*-Methode. Achtung: Diese Methode gibt null zurück, wenn keine Elemente mit der angegebenen ID existieren. Aus Sich von VB ist das ein Variant/Empty. Eine ID sollte innerhalb einer Seite immer eindeutig sein. Wenn mehr als ein Element mit der angegebenen ID existiert, gibt die Methode das erste Element im zurück.

Wenn es um Attribute und Formularfelder geht, werden wir meistens das HTML-*value*-Attribut verwenden. Dieses Attribut kann bei folgenden Elementen verwendet werden: *button*, *input*, *meter*, *li*, *option*, *progress* und *param*. Wenn es hingegen um CSS-Eigenschaften geht, kann man das Style-Objekt verwenden, dass dem Element zugeordnet ist. Dieses Objekt kennt eine Reihe von Eigenschaften, die man alle lesen oder ändern kann.

Nun werdet ihr euch fragen: Warum haben wir in den oben vorgestellten Prozeduren statt der JSON-Collection nicht die Werte direkt aus den Steuerelementen gelesen? Im Prinzip hatten wir das so machen können. Wir hätten diese Werte sogar über die Views abrufen können, indem wir dort geeignete Methoden implementiert hätten. Die Antwort lautet: Performance. Man muss überlegen was passiert, wenn die *jsProp*-Methode verwendet wird. Zunächst wird der Parameter als JavaScript-Aufruf an die WebView übergeben. Dieser Aufruf muss erst mal durch den JavaScript-Parser und Interpreter. Die Anweisung muss also übersetzt werden. Danach kann diese Anweisung ausgeführt werden. Das Ergebnis wird für VB aufbereitet und kommt als Methoden-Ergebnis wieder in der aufrufenden Prozedur an. Sollten nach diesem Muster viele Werte abgefragt oder geändert werden, wird sich das bald beim Anwender bemerkbar machen: Die Anwendung wird langsam, spürbar langsam.

Model

In der Model-Schicht werden die Daten verwaltet. Das Model ist an den Presenter über die Schnittstelle iData angebunden. Diese Schnittstellendefinition erlaubt uns das Model jederzeit durch eine alternative Variante mit Anbindung an eine andere Datenquelle auszutauschen. Genauso gut kann das Model erweitert werden, ohne dass die Kompatibilität zum Presenter leidet. Eine mögliche Erweiterung wäre das Hinzufügen einer ODBC-Schnittstelle zu einer beliebigen Datenbank. Statt die Daten wie hier in der Anwendung in eine lokale Datei zu speichern, könnte man sie über ODBC an ein größeres DBMS weiterleiten und später wieder von dort laden.

Intern verwendet das Model eine SQLite-Datenbank, die aber nur zum Sichern der Daten auf einen Datenträger geschrieben wird. Ansonsten arbeitet diese DB nur im Arbeitsspeicher. Egal woher die Daten stammen, ist es sinnvoll diese virtuelle Memory-DB zum Halten und Abfragen zu nutzen. Das

RC6-Framework bietet dafür spezifische Klassen, von denen wir einige hier verwenden werden. Im Deklarationsteil der Klasse *cmUrlaubsplaner*, die das Model repräsentiert, finden wir die Objekt-Variable *mCnn*. Das Objekt ist die eine Instanz der Klasse *cConnection* aus dem RC6-Framework. Die Variable wird zusammen mit der Model-Klasse instanziiert:

```
Option Explicit
Implements iData
Private mCnn As cConnection
'...

Private Sub Class_Initialize()
  Set mCnn = New_c.Connection
  mCnn.CreateNewDB ":memory:"
End Sub
```

Jedes Mal, wenn wir einen neuen Kalender Kalender anlegen, werden in dieser virtuellen Datenbank einige Tabelle angelegt und mit einem Satz von Daten initialisiert. Die Tabellen werden per SQL angelegt, dafür bietet die *cCollection* die Methode *Execute*. Wenn euch das alles schon aus der ADO-Welt bekannt vorkommt, dann liegt ihr richtig: Die Ähnlichkeit dieser beiden Welten ist gewollt. Solltet ihr noch Projekte im Einsatz haben, die z.B. auf MS-Access aufsetzen, dann wäre ein Wechsel zu SQLite nicht all zu schwierig. Ihr würdet auf jeden Fall von einer schnelleren DB-Engine profitieren und die ganzen JET-Voraussetzungen und Versions-Inkompatibilitäten bleiben euch erspart.

Tabellen-Definition

Die Syntax für das Anlegen von Tabellen bietet viele Optionen, wir beschränken uns hier auf die Kommandos, die in der Anwendung verwendet werden. Ihr werdet in den meisten Anwendungen mit diesem Befehlssatz auskommen.

```
CREATE TABLE [IF NOT EXISTS] tabelle1 (
  spalte1 INTEGER [NOT NULL DEFAULT 0],
  spalte2 REAL [NOT NULL DEFAULT 0],
  spalte3 TEXT[(n)] [NOT NULL DEFAULT ''],
  spalte4 NUMERIC [NOT NULL DEFAULT 0],
  spalte6 BLOB,
[PRIMARY KEY (spalte1, spalte3),]
[FOREIGN KEY (spalte3) REFERENCES tabelle2(spalte1)])
```

Hier legen wir eine Tabelle an mit den Hauptdatentypen von SQLite. Im Gegensatz zu den meisten SQL-Datenbanken schränkt SQLite den Typ der Daten nicht aufgrund des deklarierten Typs der Spalte ein. Stattdessen verwendet SQLite eine dynamische Typisierung. Der deklarierte Typ einer Spalte wird nur zur Bestimmung der Affinität der Spalte verwendet. Somit werden viele gängige Datentypnamen aus traditionelleren SQL-Implementierungen unterstützt, die intern in diese fünf Haupttypen umgewandelt werden. Wir werden davon in der Definition unserer Tabelle Gebrauch machen.

Alles was in eckigen Klammern steht ist optional. Das gilt sogar für die TEXT-Spalte, SQLite kennt keine Längenbeschränkungen. Normalerweise kommt es zu einem Fehler, wenn eine neue Tabelle in einer Datenbank erstellt wird, die bereits eine Tabelle, einen Index oder eine View mit demselben Namen enthält. Wenn jedoch die "IF NOT EXISTS"-Klausel als Teil der CREATE TABLE-Anweisung angegeben wird und eine Tabelle oder eine View gleichen Namens bereits existieren, hat der CREATE TABLE-Befehl einfach keine Wirkung (und es wird keine Fehlermeldung zurückgegeben).

Die DEFAULT-Klausel gibt einen Standardwert an, der für die Spalte verwendet werden soll, wenn der Benutzer bei einem INSERT keinen Wert explizit angibt. Ohne DEFAULT-Klausel in der Spaltendefinition wird der Standardwert NULL verwendet.

Jede Tabelle in SQLite darf höchstens einen Primärschlüssel haben. Wenn die Schlüsselwörter PRIMARY KEY zu einer Spaltendefinition hinzugefügt werden, dann besteht der Primärschlüssel der Tabelle aus dieser einen Spalte. Beispiel:

```
CREATE TABLE t1(a PRIMARY KEY, b)
```

Hier wäre die Spalte *a* gleichzeitig Primärschlüssel. Sollte dieser aus mehreren Spalten bestehen, kann man diese Spalten in der PRIMARY KEY-Klausel angeben. Der PRIMARY KEY ist für gewöhnliche Tabellen optional. Außer bei WITHOUT ROWID-Tabellen (Spezial-Fall) haben alle Zeilen innerhalb von SQLite-Tabellen einen 64-Bit-Integer-Schlüssel mit Vorzeichen, der die Zeile innerhalb ihrer Tabelle eindeutig identifiziert. Diese Ganzzahl wird normalerweise als *rowid* bezeichnet. Auf den *rowid*-Wert anstelle eines Spaltennamens zugegriffen werden. Die Daten für *rowid*-Tabellen werden als B-Baum-Struktur gespeichert. Das bedeutet, dass das Abrufen oder Sortieren von Datensätzen nach *rowid* sehr schnell ist (etwa doppelt so schnell wie eine ähnliche Suche durch Angabe eines anderen PRIMARY KEY oder indizierten Werts). Wenn eine Tabelle einen Primärschlüssel hat, der aus einer einzigen Spalte besteht, und der deklarierte Typ dieser Spalte *INTEGER*, wird die Spalte zu einem Alias für die *rowid*.

SQL-Fremdschlüssel (FOREIGN KEY) werden verwendet, um Beziehungen zwischen Tabellen zu erzwingen.

Um Daten schneller auffindbar zu machen oder um sicherzustellen, dass Wertekombinationen von bestimmten Spalten eindeutig sind, können Indexes auf die betroffenen Tabellen und Spalten erzeugt werden. Die Syntax dafür lautet:

```
CREATE [UNIQUE] [INDEX IF NOT EXISTS] index1 ON tabelle2 (spalte1, spalte2)
```

Optional ist auch hier alles was in eckigen Klammern steht. Wenn das Schlüsselwort UNIQUE zwischen CREATE und INDEX steht, sind doppelte Indexeinträge nicht erlaubt. Jeder Versuch, einen doppelten Eintrag einzufügen, führt zu einem Fehler. Achtung bei NULL-Werten: Für die Zwecke von eindeutigen Indizes werden alle NULL-Werte als verschieden von allen anderen NULL-Werten betrachtet und sind somit eindeutig! Dieses Verhalten ist typisch auch für andere Datenbanken wie z.B. PostgreSQL, MySQL, Firebird und Oracle. Andere DBMS-Systeme wie Informix und Microsoft SQL-Server folgen der anderen Interpretation des SQL-92-Standards, die besagt, dass alle NULL-Werte gleichwertig zueinander sind. Wenn das Model an eine externe Datenbank angeschlossen werden soll, wäre das zu beachten.

Wenn wir Tabellen anlegen, können wir für die Spalten Default-Werte angeben. Manchmal werden auch Default-Datensätze gebraucht. Denken wir nur an den ersten Januar. Dieser Tag ist höchstwahrscheinlich weltweit ein Feiertag, also können wir ihn bereits in die Feiertags-Tabelle eintragen. Dafür wird erneut die *Execute*-Methode der *cConnection*-Klasse verwendet, diesmal mit einem INSERT-Kommando:

```
INSERT INTO tabelle1 (spalte1, spalte2) VALUES (1, Neujahr)
```

Damit wäre alles Wichtige zum Anlegen der Tabellen gesagt und wir können loslegen!

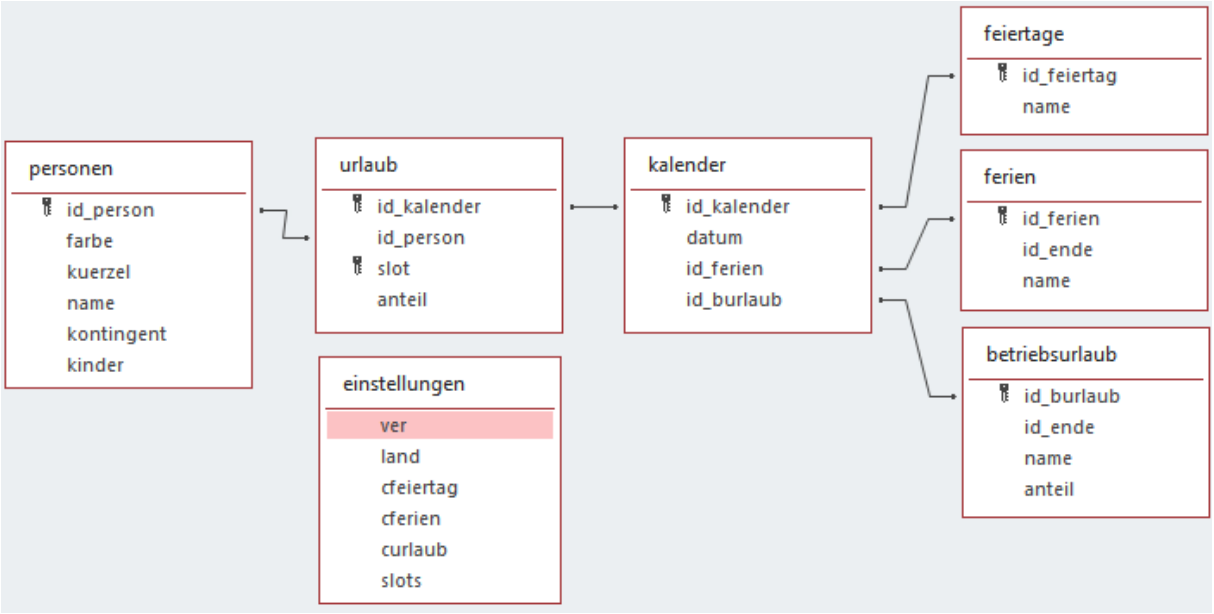


Abbildung 17: Tabellen und Beziehungen

Personen

Spalte	Datentyp	Default	Schlüssel
id_person	INTEGER	0	PK
farbe	TEXT(7)	''	
kuerzel	TEXT(2)	''	
name	TEXT(30)	''	
kontingent	FLOAT	0	
kinder	BOOLEAN	FALSE	

Tabelle 1: personen

Id_personen ist der Primärschlüssel für die Personen-Tabelle. Als einzige Spalte vom Typ INTEGER, erfüllt die Spalte die Bedingungen für die *rowid*. Die Farbe definieren wir direkt als HTML-Farbinformation. Diese wird im Format „#RRGGFF“ als Hex-Werte (00-FF) für die drei Grundfarben RGB gespeichert. Kontingent gibt den verfügbaren Urlaub für das aktuelle Jahr an. Hier als FLOAT-Deklariert, also eine Kommazahl, um ggf. auch halbe Urlaubstage zu berücksichtigen. Kinder ist ein boolescher Wert, der angibt, ob die Person schulpflichtige Kinder hat. FLOAT und BOOLEAN wird SQLite intern in eigene Datentypen konvertieren. Nichtdestotrots sind diese Definitionen sinnvoll, weil die RC6-Implementierung uns später im Recordset tatsächlich Double- und Boolean-Daten liefern wird.

Kalender

Spalte	Datentyp	Default	Schlüssel
id_kalender	INTEGER	0	PK
datum	SHORTDATE	0	
id_ferien	INTEGER	*null*	
id_burlaub	INTEGER	*null*	

Tabelle 2: kalender

Auch hier ist der Primärschlüssel *id_kalender* ein Alias für die *rowid.* Wir werden hier später den Long-Wert auf einem VB-Datum ablegen. Das wäre zwar nicht nötig, macht aber vieles einfacher. Dieser Wert ist per Definition eindeutig und leicht zu ermitteln. Die zweite Spalte *datum* ist vom Typ *SHORTDATE*. Auch das ist ein Pseudo-Datentyp, der auch hier vom RC6-Framework konvertiert wird.

Die beiden Spalten *id_ferien* und *id_burlaub* sind Verweise in die Tabellen *ferien* und *betriebsurlaub*. Damit wird im Kalender gekennzeichnet ob ein Tag ein Ferien- oder Betriebsurlaubstag ist.

Der aufmerksame Datenbankentwickler wird nun vielleicht anmerken, dass wir in dieser Tabelle die zweite Normalform (Datenbank-Normalisierung) verletzen. Diese besagt, dass kein Nichtprimär-attribut funktional von einem Primärschlüssel abhängt. Wenn wir im Primärschlüssel bereits den Longwert eines Datums speichern, bräuchte es das Feld *datum* nicht mehr. Das ist grundsätzlich richtig. Allerdings ist diese Abhängigkeit in diesem Fall rein zufällig. Wir hätten für den Primärschlüssel einen beliebigen anderen Wert nehmen können, z. B. einen sogenannten Surrogatschlüssel (laufende Nummer). Hinzu kommt, dass SQLite Datumswerte anders als VB behandelt. Intern kann SQLite zwar auch ein Datum als Ganzzahl speichern, allerdings beinhaltet diese Ganzzahl die am dem 01.01.1970 00:00:00 vergangenen Sekunden (Unix-Time).

Feiertage, Ferien und Betriebsurlaub

Spalte	Datentyp	Default	Schlüssel
id_feiertag	INTEGER	0	PK
name	TEXT (30)	0	

Tabelle 3: feiertage

Spalte	Datentyp	Default	Schlüssel
id_feiertag	INTEGER	0	PK
id_ende	INTEGER	0	
name	TEXT (30)	''	

Tabelle 4: ferien

Spalte	Datentyp	Default	Schlüssel
id_burlaub	INTEGER	0	PK
id_ende	INTEGER	0	
name	TEXT (30)	''	
anteil	FLOAT	1	

Tabelle 5: betriebsurlaub

Alle id-Felder sind vom Typ Integer. Der Anwender sieht auf der Oberfläche jedoch Datumswerte. Das kommt davon, dass diese Felder auf das Datumsfeld in der Tabelle *kalender* verweisen.

In der Tabelle betriebsurlaub befindet sich noch eine Spalte anteil, vom Datentyp *FLOAT*. RC6 liefert uns hier einen Double-Datentyp. Dieser soll festlegen, ob ein ganzer Tag Urlaub veranschlagt wird oder nur ein halber Tag. In manchen Unternehmen ist es üblich Heiligabend und Silvester je einen halben Tag zu gewähren. Der Mitarbeiter benötigt somit an diesen Tagen nur jeweils einen halben Urlaubstag.

Urlaub

Spalte	Datentyp	Default	Schlüssel
id_kalender	INTEGER	0	PK, FK in die Tabelle kalender, SK
id_person	INTEGER	0	PK, FK in die Tabelle personen
slot	INTEGER	0	SK
anteil	FLOAT	1	

Tabelle 6: urlaub

In dieser Tabelle werden die geplanten Urlaubstage verwaltet. Anders als in den bisherigen Tabellen finden wir hier einen Primärschlüssel, der sich aus zwei Felder zusammensetzt. Gleichzeitig stellen

diese Schlüssel einen Verweis auf die Tabellen *personen* und *kalender* her. Wörtlich ausgedrückt, bedeutet das: Ein Urlaubstag wird durch eine Person und einen Kalendertag definiert. Mehrere Personen können am gleichen Tag Urlaub planen und andererseits kann die gleiche Person für unterschiedlichen Tagen Urlaub beantragen. Die Kombination aus Urlaub und Person muss aber einmalig sein. Gleiches erwarten wir aus der Kombination *id_kalender* und *slot*. Der Slot ist für uns eine Monatsspalte. Um Überschneidungen in der Darstellung zu vermeiden, sollte ein Slot nur einmalig belegbar sein. Die Person spielt in diesem Zusammenhang keine Rolle.

Einstellungen

Spalte	Datentyp	Default	Schlüssel
ver	INTEGER	CURRENT_DB_VER	
land	INTEGER	0	
cfeiertag	TEXT(7)	'#CCBBCC'	
cferien	TEXT(7)	'#FFFF99'	
curlaub	TEXT(7)	'#CCFFDD'	
slots	INTEGER	2	

Die letzte Tabelle ist eine Hilfstabelle, in der wir Programmeinstellungen festhalten. Auf einen Primärschlüssel können wir verzichten, weil diese Tabelle nur aus einem einzelnen Datensatz besteht. Im Feld *ver* (INTEGER) halten wir die Version der DB fest. Das ermöglicht uns später den Abgleich mit der Programmversion und hilft beim Update der Datenbank. Das Bundesland wird über das Feld *land* festgehalten. Wir brauchen dieses Feld später kaum noch, weil die Feiertage für ein bestimmtes Bundesland berechnet werden. Somit dient dieses Feld nur noch zur späteren Information. Die drei c-Felder speichern eine Farbinformation im HTML-Format dar für die Kennzeichnung von Feiertagen, Ferien- und Betriebsurlaubstagen. Den Abschluss mach die Spalte *slots*, die anzeigt wie viele Slots pro Monat verfügbar sind. Dieses Feld wird mit einer 2 initialisiert. Es stehen also von Anfang an zwei Spalten pro Monat zur Verfügung.

Tabellen anlegen

Wenn ein neuer Urlaubsplaner angelegt wird, erzeugt das Model zunächst eine Memory-DB, eine Datenbank also, die nur im Arbeitsspeicher liegt. Dafür werden zuerst per Execute-Methode die oben beschriebenen Tabelle angelegt. Je nach Startkonfiguration (Jahr, Bundesland) werden danach einige Tabellen mit Daten initialisiert. Für die Anlage der Tabellen werden CREATE TABLE Anweisungen per SQL wie folgt definiert:

```
CREATE TABLE IF NOT EXISTS einstellungen (
  ver        INTEGER NOT NULL DEFAULT CURRENT_DB_VER,
  land       INTEGER NOT NULL DEFAULT 0,
  cfeiertag  TEXT(7) NOT NULL DEFAULT '#CCBBCC',
  cferien    TEXT(7) NOT NULL DEFAULT '#FFFF99',
  curlaub    TEXT(7) NOT NULL DEFAULT '#CCFFDD',
  slots      INTEGER NOT NULL DEFAULT 2;

CREATE TABLE IF NOT EXISTS personen (
  id_person  INTEGER NOT NULL DEFAULT 0,
  farbe      TEXT(7) NOT NULL DEFAULT '',
  kuerzel    TEXT(2) NOT NULL DEFAULT '',
  name       TEXT(30) NOT NULL DEFAULT '',
  kontingent FLOAT NOT NULL DEFAULT 0,
  kinder     BOOLEAN NOT NULL DEFAULT FALSE,
  PRIMARY KEY (id_person);
```

```
CREATE TABLE IF NOT EXISTS kalender (
  id_kalender INTEGER NOT NULL DEFAULT 0,
  datum       SHORTDATE NOT NULL DEFAULT 0,
  id_ferien   INTEGER,
  id_burlaub  INTEGER,
PRIMARY KEY (id_kalender);

CREATE TABLE IF NOT EXISTS feiertage (
  id_feiertag INTEGER NOT NULL DEFAULT 0,
  nameTEXT(30) NOT NULL DEFAULT '',
  PRIMARY KEY (id_feiertag);

CREATE TABLE IF NOT EXISTS ferien (
  id_ferien  INTEGER NOT NULL DEFAULT 0,
  id_ende    INTEGER NOT NULL DEFAULT 0,
  name       TEXT(30) NOT NULL DEFAULT '',
  PRIMARY KEY (id_ferien);

CREATE TABLE IF NOT EXISTS betriebsurlaub (
  id_burlaub  INTEGER NOT NULL DEFAULT 0,
  id_ende     INTEGER NOT NULL DEFAULT 0,
  name        TEXT(30) NOT NULL DEFAULT '',
  anteil      FLOAT NOT NULL DEFAULT 1,
  PRIMARY KEY (id_burlaub);

CREATE TABLE IF NOT EXISTS urlaub (
  id_kalender INTEGER NOT NULL DEFAULT 0,
  id_person   INTEGER NOT NULL DEFAULT 0,
  slot        INTEGER NOT NULL DEFAULT 0,
  anteil      FLOAT NOT NULL DEFAULT 1,
  PRIMARY KEY (id_kalender,id_person),
  FOREIGN KEY (id_kalender) REFERENCES kalender(id_kalender),
  FOREIGN KEY (id_person)   REFERENCES personen(id_person);

CREATE UNIQUE INDEX IF NOT EXISTS ixslotbelegung
  ON urlaub (id_kalender,slot)
```

Warum aber immer wieder dieses IF NOT EXISTS? Ganz einfach: Bei der Anlage eines neuen Urlaubsplaners kann die Option Import verwendet werden. Das bedeutet, dass eine vorhandene Datenbank verwendet wird, in der nur noch bestimmte Tabellen bereinigt werden und andere Tabellen mit neuen Konfigurationen vorbelegt werden. Das sehen wir in den folgenden SQL-Anweisungen, die ebenfalls per Execute-Methode durchgeführt werden.

In die Tabelle Einstellungen wird das selektierte Bundesland übernommen:

```
INSERT INTO einstellungen(land) VALUES (BUNDESLAND)
'und für eine importierte DB:
UPDATE einstellungen SET land = BUNDESLAND;
```

In der *personen*-Tabelle wird nichts gelöscht und nichts geändert. Beim Import-Fall bleiben so alle Personen-Daten und Einstellungen erhalten.

In den Tabellen *kalender*, *feiertage, ferien, betrebsurlaub und urlaub* werden alle vorhandenen Daten gelöscht (falls vorhanden / Import-Fall):

```
DELETE FROM kalender;
DELETE FROM feiertage;
DELETE FROM ferien;
```

```
DELETE FROM betriebsurlaub;
DELETE FROM urlaub;
```

Ferien gibt es in allen Bundesländern, allerdings beginnen und enden die an unterschiedlichen Kalendertagen. Es gibt dafür keine Mathematik oder Systematik, mit der sich das berechnen lässt. Dennoch wollen wir unsere Anwender die Arbeit erleichtern und werden zumindest die Ferien-Tabelle mit den gängigsten Ferien-Namen vorbelegen:

```
INSERT INTO ferien (id_ferien,name) VALUES
  (-7,'Neujahr'),
  (-6,'Winter'),
  (-5,'Ostern'),
  (-4,'Pfingsten'),
  (-3,'Sommer'),
  (-2,'Herbst'),
  (-1,'Weihnachten');
```

Die *id_ferien* wird mit negativen Werten belegt, was uns später anzeigt, dass hier noch kein Datum festgelegt ist. Das müssen später die Anwender nachhohlen. Als nächstes wird der Kalender angelegt. Die Datensätze im Kalender sind durch das eingestellte Jahr vorgegeben und werden sich nicht mehr verändern. Für die Anlage neuer Datensätze braucht man nicht zwingend SQL, das geht auch mit den Methoden des cRecordsets, wo wie man das bereits vom ADO-Recordset kennt:

```
minD = DateSerial(jahr, 1, 1)
maxD = DateSerial(jahr, 12, 31)
Set rs = mCnn.OpenRecordset("SELECT * FROM kalender")
For d = minD To maxD
  rs.AddNew
  rs!id_kalender = CLng(d)
  rs!datum = d
Next d
rs.UpdateBatch
```

Da wir das Bundesland bereits kennen, stehen die gesetzlichen Feiertage auch schon fest. Die genauen Kalendertage sind entweder mit festem Datum vorgegeben oder können ausgehend vom Ostersonntag berechnet werden. Hierbei möchte ich gar nicht auf Details eingehen, der Code zum Berechnen der Feiertage sollte keine Fragen aufwerfen.

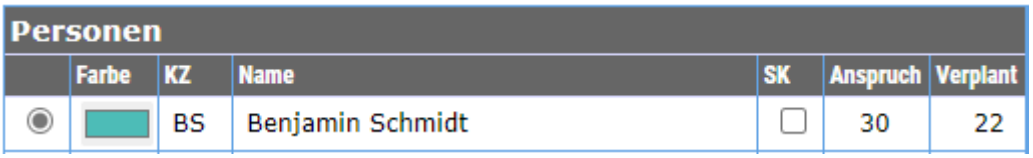
iData – Schnittstelle zum Model

Personen

Um die Personen im Urlaubsplaner zu verwalten, finden wir hier die üblichen CRUD-Methoden (Create, Read, Update und Delete). Wobei in diesem Fall die Create- und Update-Methode zu einer Funktion *SavePersonen* zusammengelegt wurden. Auf die interne Implementation dieser Methoden werde ich nicht eingehen, das ist normales VB-Handwerk.

```
Public Function GetPersonen(Optional id_person As Long) As cRecordset
Public Function DeletePerson(id As Long) As Boolean
Public Function SavePersonen(data As RC6.cCollection, _
                             errors As cArrayList) As Boolean
```

Die Funktion *GetPersonen* liefert die Daten für eine bestimmte oder für alle Personen. Für die Unterscheidung sorgt hier das optionale *id_person* Argument. Der ScreenShot für die Personen-Verwaltung zeigt noch mal den Bedarf:



Personen

	Farbe	KZ	Name	SK	Anspruch	Verplant
◉		BS	Benjamin Schmidt	☐	30	22

Abbildung 18: Personen-Verwaltung

Alle diese Daten rufen wir in einer einzelnen Abfrage ab. Dazu verknüpfen wir alle beteiligten Tabellen (*personen*, *urlaub*, *betriebsurlaub* und *kalender*) mittels JOIN:

```
SELECT
  personen.id_person,
  personen.farbe,
  personen.kuerzel,
  personen.name,
  personen.kontingent,
  personen.kinder,
  COALESCE(SUM(urlaub.anteil),0) + (
    SELECT
      COALESCE(SUM(betriebsurlaub.anteil),0)
    FROM
      betriebsurlaub
      INNER JOIN kalender ON betriebsurlaub.id_burlaub=kalender.id_burlaub
  ) AS verplant
FROM
  personen
  LEFT JOIN urlaub ON personen.id_person = urlaub.id_person
GROUP BY
  personen.id_person,
  personen.kuerzel,
  personen.name,
  personen.kontingent
ORDER BY personen.name
```

Die Daten aus der *personen*-Tabelle sind selbsterklärend. Wir brauchen aber in der letzten Spalte noch die verplanten Tage. Diese ergeben sich aus den individuell verplanten Tagen zuzüglich der Betriebsurlaubstage. Und weil es sich hierbei immer auch um halbe Tage handeln kann, bilden wir Summen. Die *COALESCE*-Funktion sorgt dafür, dass *null*-Werte das Ergebnis nicht verfälschen, statt *null* wir eine nummerische 0 geliefert. Sollten wir nur die Daten einer einzelnen Person benötigen, ergänzen wir die Abfrage mit einer WHERE-Klausel:

```
WHERE personen.id_person=" & id_person
```

Mit DeletePerson löschen wir eine einzelne Person aus dem Personenstamm. Um Datenmüll zu vermeiden, müssen auch eventuell verplante Urlaubstage für diese Person gelöscht werden. Um die referenzielle Integrität sicherzustellen, werden diese Lösch-Kommandos in eine Transaktion eingeschlossen:

```
With mCnn
  .BeginTrans
  .Execute "DELETE FROM urlaub WHERE id_person=" & id_person
  .Execute "DELETE FROM personen WHERE id_person=" & id_person
  .CommitTrans
End With
```

An die SavePersonen-Funktion werden die Daten per JSON-Collection übertragen. In der Prozedur-Definition finden wir ein weiteres Argument, *errors As cArrayList.* Das ist sozusagen der Rückkanal zum aufrufenden Objekt, in dem alle gefundenen Fehler gesammelt werden. *cArrayList* ist eine weitere Klasse im RC6-Framework. Es handelt sich hierbei um eine Liste von Daten. Bei der Initialisierung der Klasse wird festgelegt um welchen Datentyp diese Daten sind:

```
Set errors = New_c.ArrayList(vbString)
```

Da wir hier Fehlermeldungen sammeln wollen, definieren wir die Member der Liste als Strings.

Es wäre verkürzt zu sagen, dass die cArrayList ein besseres Array ist. In der Tat kann eine cArrayList auch als Luxus-Array verwendet werden, intern steckt aber viel mehr in dieser Klasse. Dazu reicht ein Blick in den Objektkatalog. Wir finden hier Funktionen wie: *Add* und *Remove*, *Push* und *Pop*, *Queue* und *DeQueue*, *Sort* und vieles mehr. Wharscheinlich bräuchte es ein ganzes Informatik-Semester, um allein diese Klasse zu erklären.

Optionen

Alles was in den Optionen angezeigt und konfiguriert wird kommt aus der Datenbank über Model-Methoden:

```
Public Function GetSlots() As Integer
Public Function GetSettings() As cRecordset
Public Function GetFeiertage() As cRecordset
Public Function GetFerien() As cRecordset
Public Function GetBetriebsurlaub() As cRecordset
```

Alle diese Funktionen liefern Recordsets, die sich aus SQL-Abfragen ergeben:

```
/*Slots*/
SELECT slots FROM einstellungen

/*Settings*/
SELECT
  (SELECT
     strftime('%Y',min(datum))
     FROM kalender)
  AS jahr,
  einstellungen.slots,
  einstellungen.cfeiertag,
  einstellungen.cferien,
  einstellungen.curlaub,
  (SELECT COALESCE(max(slot),2) FROM urlaub) AS minslots
FROM
  einstellungen

/*Feiertage*/
SELECT
  feiertage.id_feiertag,
  kalender.datum,
  feiertage.name
FROM
  feiertage
  INNER JOIN kalender ON feiertage.id_feiertag = kalender.id_kalender
ORDER BY
  feiertage.id_feiertag
```

```
/*Ferien*/
SELECT
  ferien.id_ferien,
  k1.datum AS ferienstart,
  k2.datum AS ferienende,
  ferien.name
FROM
  ferien
  LEFT JOIN kalender AS k1 ON ferien.id_ferien = k1.id_kalender
  LEFT JOIN kalender AS k2 ON ferien.id_ende = k2.id_kalender
ORDER BY
  ferien.id_ferien

/*Betriebsurlaub*/
SELECT
  betriebsurlaub.id_burlaub,
  k1.datum AS bustart,
  k2.datum AS buende,
  betriebsurlaub.name,
  betriebsurlaub.anteil
FROM
  betriebsurlaub
  INNER JOIN kalender AS k1 ON betriebsurlaub.id_burlaub = k1.id_kalender
  INNER JOIN kalender AS k2 ON betriebsurlaub.id_ende = k2.id_kalender
ORDER BY
  betriebsurlaub.id_burlaub
```

Beim Speichern der Daten erhalten wir vom Presenter wieder eine JSON-Collection mit allen Feld-IDs und den Daten. Auch hier sammeln wir eventuelle Fehler in einer *cArrayList*, die später in der aufrufenden Prozedur des Presenters zur Anzeige gebracht werden.

```
Public Function SaveOptionen(data As RC6.cCollection, _
                             errors As cArrayList) As Boolean
```

Um den Code im Model besser zu strukturieren, verteilen wir das Speichern intern auf kontext-bezogene private Prozeduren wie *saveFerien*, *saveFeiertage* und *saveBetriebsurlaub.* Die Haupt-prozedur kapselt alle Änderungen wieder in eine Transaktion. Es wird also alles korrekt oder gar nichts gespeichert so lange noch Fehler in der Eingabe gefunden werden.

Schauen wir uns exemplarisch das Speichern der Einstellungen an, weil wir dafür ein neues Objekt aus dem RC6-Framework kennenlernen werden, die *cCommand*-Klasse. Das davon abgeleitete Objekt definiert einen SQL-Befehl, der auf der Datenquelle ausführt werden soll.

```
Dim cmd As cCommand
Set cmd = mCnn.CreateCommand("UPDATE einstellungen " _
                             "SET slots=?,cfeiertag=?,cferien=?,curlaub=?")

With cmd
 .SetInt32 1, slots
  .SetText 2, cFeiertage
  .SetText 3, cFerien
  .SetText 4, cUrlaub
End With
cmd.Execute
```

Parametrisierte SQL-Abfragen oder Kommandos können optional auch als mConnection-Methode ausgeführt werden. Beispiel aus der Prozedur *saveBetriebsurlaub*:

```
mCnn.ExecCmd "INSERT INTO betriebsurlaub " & _
             "(id_burlaub,name,id_ende,anteil) VALUES (?,?,?,?)", _
             CLng(datum(0), name, CLng(datum(1), anteil)
```

Beim Betriebsurlaub müssen wir darauf achten, dass bei einem Intervall von-bis nicht auch Wochenenden und Feiertage dabei sind. Das wären freie Tage, für die kein Urlaubskontingent verbraucht wird.

Kalender

Um einen Kalender anzulegen, öffnen, importieren und speichern sin in der *iData*-Schnittstelle folgende Funktionen deklariert:

```
Public Function GetJahr() As Long
Public Sub NewKalender(jahr As Integer, _
              Optional bundesland As eBundeslaender)
Public Function OpenKalender(FileName As String, _
              Optional ByRef biv As Long, _
              Optional ByRef dbv As Long, _
              Optional updateVer As Boolean) As Boolean
Public Function ImportPersonenstamm (FileName As String, _
              jahr As Integer, _
              Optional bundesland As eBundeslaender) As Boolean
Public Function ImportKalender(Tage As String, _
              jahr As Integer, _
              FileName As String) As cRecordset
Public Function GetKalender() As cRecordset
Public Function SaveKalender(FileName As String) As Boolean
Public Function DataChanged() As Boolean
```

Das Jahr für den geladenen Urlaubsplaner wird an unterschiedlichen Stellen gebraucht. Intern wird das Jahr in der Modul-Variable *mJahr* gespeichert und kann somit schnell nach außen gereicht werden:

```
GetJahr = mJahr
```

Ein neuer Kalender wird mittels *NewKalender* angelegt. Pflichtargument ist das Jahr für diesen Kalender. Das Bundesland ist optional für die Berechnung der gesetzlichen Feiertage. In dieser Prozedur werden hauptsächlich die Tabellen angelegt und einige Werte initialisiert, siehe Abschnitt *Tabellen anlegen* in diesem Kapitel.

Beim Öffnen eines vorhandenen Urlaubsplaners brauchen wir den vollständigen Dateipfad. Für das Einlesen der Datei wird zunächst eine Connection zu dieser Datei geöffnet und die Datenbank auf eine neue InMemory-DB kopiert:

```
With New_c.Connection(FileName, DBOpenFromFile)
  Set mCnn = .CopyDatabase(":memory:")
End With
```

Anmerkung: RC6 bietet auch eine alternative Klasse cMemDB, die speziell für InMemory-Datenbanken konzipiert ist. Funktional unterscheiden sich die beiden Objekte kaum. Sollte jedoch irgendwann unser Model eine Anbindung an ein externes DBMS bekommen, bietet dieses Objekt eine nützliche Methode, die *CreateTableFromRs*. Darüber können InMemory-Tabellen direkt aus ADO-Recordsets erstellt werden.

Direkt nach dem Öffnen der Datenbank werden die lokalen Klassenvariablen befüllt. Unter anderem wird so auch die Version der Datei gelesen und mit der Version der Anwendung (Konstante

CURRENT_DB_VER) verglichen. Wenn beide Versionen nicht zueinander passen, wird das Öffnen abgebrochen. Davor die beiden ByRef Argumente *biv* und *dbv* gespeichert. Diese Argumente werden vom Presenter ausgewertet, wobei der Anwender entscheiden kann, ob die Datei aktualiesiert werden soll. Wenn das der Fall ist, wir die Prozedur erneut aufgerufen, diesmal mit einem True im letzten optionalen Argument *updateVer*. Der Vorgang von oben wiederholt sich und in der lokalen Prozedur *updateVersion* wird die DB aktualisiert:

```
Set rs = mCnn.OpenRecordset("SELECT * FROM einstellungen")
biv = CURRENT_DB_VER
dbv = rs!ver
If updateVer Then
  If Not updateVersion Then Exit Function
ElseIf dbv <> CURRENT_DB_VER Then
  Exit Function
End If
```

In einer früheren Version des Urlaubsplaners wurden die Schulferien noch nicht berücksichtigt. Entsprechend fehlten unter anderem noch die Spalte *kinder* in der Personen-Tabelle. Um alle Kompatibilitäts-Kriterien zwischen DB und Anwendung zu prüfen, kann das Datenbankschema analysiert werden. Das Connection-Objekt bietet dazu die Methode *OpenSchema.* Diese Methode liefert ein Recordset in dem alle relevanten Informationen zur DB enthalten sind. Uns interessieren die Felder *type*, *name*, und *sql*. Wenn der aktuelle Datensatz sich auf eine Tabelle bezieht, steht in *type* „table“, in *name* der Tabellenname und in *sql* die CREATE TABLE-Definition.

```
Set rs = mCnn.OpenSchema
Do Until rs.EOF
  If rs!type = "table" Then
    Select Case rs!name
      Case "personen"
        If InStr(20, rs!sql, "kinder", vbTextCompare) = 0 Then
          sql = "ALTER TABLE personen " & _
                "ADD COLUMN kinder BOOLEAN NOT NULL DEFAULT FALSE"
```

Das ist eine einfache Methode zum Abgleich der Kompatibilität. Es geht auch genauer, indem alle Tabellen einzeln analysiert werden:

```
Dim tbl As cTable, col As cColumn
For Each tbl In mCnn.DataBases("filedb").Tables
  If tbl.name = "personen" Then
    For Each col In tbl.Columns
      '...
    Next
  End If
Next
```

Gleiches gilt für Indizes, Views, Constraints und Triggers. All diese DB-Objekte bieten Methoden zur Analyse und Manipulation ihrer Eigenschaften.

Ähnlich wie beim Öffnen einer Datei wird auch beim Import verfahren. Die Datei wird geöffnet und kopiert, ggf. ohne Nachfrage aktualisiert (die Aktualisierung bezieht sich ja nur auf die InMemory-DB). Danach wird wieder die Prozedur *NewKalender* durchlaufen. Deshalb stehen in den Tabellendefinitionen immer wieder dieses IF EXISTS. Hier sind die Tabellen schon da und müssen nur noch bereinigt, bzw. neu initialisiert werden.

Die Funktion *GetKalender* liefert in einer etwas komplexeren Abfrage ein Recordset mit allen Daten, die zum GUI-Aufbau des Kalenders in der View benötigt werden:

```
SELECT
  kalender.id_kalender,
  kalender.datum,
  CAST(strftime('%d',datum) AS INTEGER) AS tag,
  CAST(strftime('%m',datum) AS INTEGER) AS monat,
  CASE strftime('%w',datum)
    WHEN '0' THEN 'So'
    WHEN '1' THEN 'Mo'
    WHEN '2' THEN 'Di'
    WHEN '3' THEN 'Mi'
    WHEN '4' THEN 'Do'
    WHEN '5' THEN 'Fr'
    WHEN '6' THEN 'Sa'
    END AS wtag,
  CASE strftime('%w',datum)
    WHEN '0' THEN 1
    WHEN '6' THEN 1
    ELSE 0
    END AS we,
  feiertage.name AS feiertag,
  ferien.id_ferien,
  betriebsurlaub.name AS burlaub
FROM
  kalender
  LEFT JOIN
    feiertage ON kalender.id_kalender = feiertage.id_feiertag
  LEFT JOIN
    ferien ON kalender.id_ferien = ferien.id_ferien
  LEFT JOIN
    betriebsurlaub ON kalender.id_burlaub = betriebsurlaub.id_burlaub
ORDER BY
  tag,
  monat
```

Diese Abfrage wäre mit Sicherheit wesentlich kompakter, wenn wir auf die Berechnungen für den Tag, Monat, Wochentag und Wochenende verzichten würden. Das ist richtig, schließlich könnten wir diese Informationen mit VB-Code leicht aus dem Datum ableiten. Achtet aber mal auf die Sortier-Reihenfolge. Warum wird hier nach Tag und Monat sortiert? Das ist ein Zugeständnis an die View, die den Kalender per HTML aufbaut. Eine HTML-Tabelle wird sequenziell in der Reihenfolge erste Zeile - alle Spalten, zweite Zeile - alle Spalten, usw. aufgebaut:

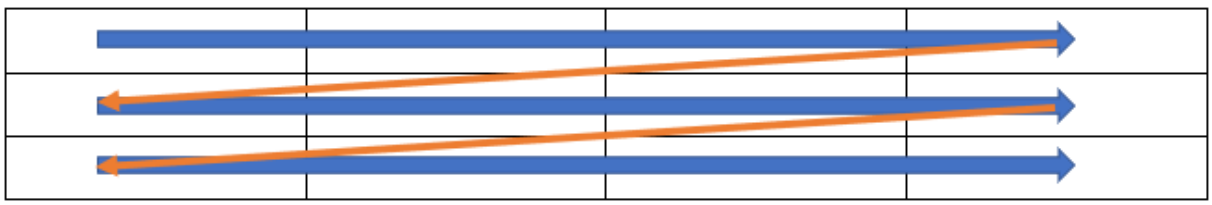


Abbildung 19: Aufbau einer HTML-Tabelle

Das Gegenstück zum Öffnen eines Urlaubsplaners ist das Speichern des Urlaubsplaners. Das ist basiert genauso wie beim Öffnen auf der Methode *.CopyDatabase*. Für das Speicher werden lediglich Quelle und Ziel getauscht. Die Quelle ist nun unsere InMemory-DB und das Ziel eine Datei, bzw. der vollständige Pfad zu dieser Datei:

```
With New_c.fso
    If .FileExists(FileName & "~") Then .DeleteFile FileName & "~"
    mCnn.CopyDatabase FileName & "~", , True
    If .FileExists(FileName & "~") Then
        .MoveFile FileName & "~", FileName, True
        With New_c.Connection(FileName, DBOpenFromFile)
            Set mCnn = .CopyDatabase(":memory:")
        End With
    End If
End With
```

Auch hier verwenden wir wieder diverse Methoden des FSO-Objektes, indem wir die Daten zunächst in eine temporäre Datei speichern, umbenennen und erneut öffnen. Das stellt sicher, dass im Falle eines Fehlers unsere Änderungen nicht verloren sind. Dieses Verfahren bietet einen weiteren Vorteil, den wir in der Funktion *DataChanged* nutzen werden:

```
DataChanged = (mCnn.TotalAffectedRowsInSession > 0)
```

Die Connection bietet nicht nur die aus ADO bekannte Funktion *AffectedRows* nach einer Datenbank-Aktion, sondern die gleiche Funktion als globalen Zähler für alle getätigten DB-Schreibaktionen. Das ist sehr praktisch, weil wir so leicht an die Information kommen, die besagt, ob sich die Daten seit dem letzten Öffnen geändert haben. Wir sparen uns so die aufwendige Überwachung der Änderungen in einer Modul-Variable. Der globale Zähler wird durch Speichern und das erneute Öffnen der Datenbank zurückgesetzt.

Urlaub

```
Public Function GetUrlaubskalender() As cRecordset
Public Function ToggleUrlaub( id_kalender As Long, _
                              id_person As Long, _
                              slot As Integer, _
                         ByRef verplant As Single, _
                         ByRef ueberzogen As Boolean) As Boolean
Public Function GetTag(id_kalender As Long) As cRecordset
Public Function GetUrlaub() As cRecordset
Public Function Defrag() As Integer
```

Die Export-Funktionen des Urlaubsplaners erfordern noch mehr Daten, hier brauchen wir neben dem Kalender auch noch die Personen- und Urlaubsdaten. Das liefert die Funktion *GetUrlaubskalender,* deren Recordset-Rückgabe auf folgende Abfrage basiert:

```
SELECT
  kalender.id_kalender,
  kalender.datum,
  CAST(strftime('%d',kalender.datum) AS INTEGER) AS tag,
  CAST(strftime('%m',kalender.datum) AS INTEGER) AS monat,
  CASE strftime('%w',kalender.datum)
    WHEN '0' THEN 'So'
    WHEN '1' THEN 'Mo'
    WHEN '2' THEN 'Di'
    WHEN '3' THEN 'Mi'
    WHEN '4' THEN 'Do'
    WHEN '5' THEN 'Fr'
    WHEN '6' THEN 'Sa'
  END AS wtag,
  CASE strftime('%w',kalender.datum)
    WHEN '0' THEN 1
    WHEN '6' THEN 1
```

```
    ELSE 0
  END AS we,
  feiertage.name AS feiertag,
  ferien.id_ferien,
  betriebsurlaub.name AS burlaub,
  urlaub.slot,
  urlaub.anteil,
  urlaub.id_person,
  personen.kuerzel,
  personen.name,
  personen.farbe
FROM
  kalender
  LEFT JOIN
    feiertage ON kalender.id_kalender = feiertage.id_feiertag
  LEFT JOIN
    ferien ON kalender.id_ferien = ferien.id_ferien
  LEFT JOIN
    betriebsurlaub ON (kalender.id_burlaub = betriebsurlaub.id_burlaub)
  LEFT JOIN
    urlaub ON kalender.id_kalender = urlaub.id_kalender
  LEFT JOIN
    personen ON urlaub.id_person = personen.id_person
ORDER BY
    kalender.id_kalender,
    urlaub.slot
```

Die hier abgedruckten SQL-Abfragen sind gut lesbar, weil sie in einem SQL-Abfragetool editiert und von dort kopiert wurden. In VB müssen wir einen (langen) String erstellen, Stichwort: String-konkatination. Diese Codes sind schlecht zu editieren, schlecht zu lesen und haben darüber hinaus auch noch schlechtes Laufzeitverhalten. Beispiel:

```
sSQL = "SELECT "
sSQL = sSQL & "kalender.id_kalender, "
sSQL = sSQL & "kalender.datum, "
sSQL = sSQL & "CAST(strftime('%d',kalender.datum) AS INTEGER) AS tag, "
sSQL = sSQL & "CAST(strftime('%m',kalender.datum) AS INTEGER) AS monat, "
sSQL = sSQL & "CASE strftime('%w',kalender.datum) "
'...
sSQL = sSQL & "ORDER BY "
sSQL = sSQL & "kalender.id_kalender, "
sSQL = sSQL & "urlaub.slot "
```

Die Alternative alles in eine Anweisung zu packen, liefert bei kleinen Abfragen optisch etwas bessere Ergebnisse. Bei größeren Abfragen meldet sich VB bald mit folgendem Fehler:

```
sSQL = "SELECT " & _
"kalender.id_kalender, " & _
"kalender.datum, " & _
"CAST(strftime('%d',kalender.datum) AS INTEGER) AS tag, " & _
"CAST(strftime('%m',kalender.datum) AS INTEGER) AS monat, " & _
"CASE strftime('%w',kalender.datum) " & _
"WHEN '0' THEN 'So' " & _
"WHEN '1' THEN 'Mo' " & _
"WHEN '2' THEN 'Di' " & _
"WHEN '3' THEN 'Mi' " & _
"WHEN '4' THEN 'Do' " & _
"WHEN '5' THEN 'Fr' " & _
"WHEN '6' THEN 'Sa' " & _
"END AS wtag, " & _
"CASE strftime('%w',kalender.datum) " & _
"WHEN '0' THEN 1 " & _
```

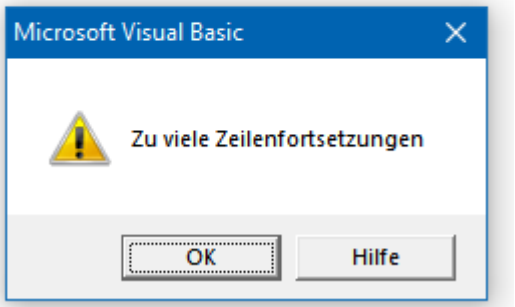


Abbildung 20: Zu viele Zeilenfortsetzungen

Dafür liefert uns das RC6-Framework auch eine Klasse, den *cStringBuilder*. Dieses Objekt beseitigt alle zuvor genannten Nachteile:

```
With New_c.StringBuilder
    .AddNL "SELECT "
    .AddNL "kalender.id_kalender,"
    .AddNL "kalender.datum,"
    .AddNL "CAST(strftime('%d',kalender.datum) AS INTEGER) AS tag,"
    .AddNL "CAST(strftime('%m',kalender.datum) AS INTEGER) AS monat,"
    .AddNL "CASE strftime('%w',kalender.datum)"
    '...
    .AddNL "ORDER BY kalender.id_kalender,urlaub.slot"
    Set iData_GetUrlaubskalender = mCnn.OpenRecordset(.ToString)
End With
```

Das Erstellen der Instanz kostet natürlich auch Ressourcen. Aus Performance-Sicht ist dieses Objekt bei größeren Strings im Vorteil, da wo die String-Konkatenation von VB spürbar wird. Intern arbeitet diese Klasse mit einem größeren, direkt bei der Instanziierung angelegten String (ca. 8KB). Die Add- und AddNL-Methoden (NL steht für NewLine) ersetzen die übergebenen Argumente mit der Mid$-Funktion. Bei Bedarf wird der interne String-Puffer vergrößert. Der *StringBuilder* hat noch weitere Vorteile. So kann er neben normalen Text (*.ToText*-Methode) mittels *.ToUTF8*-Methode auch UTF-8 kodierte Zeichenketten liefern. Das ist recht praktisch, zumal das die De-facto-Standard-Zeichenkodierung des Internets ist. Da unsere Views in der WebView2-Komponente dargestellt werden, wird uns dieses Objekt noch öfter begegnen.

Die Funktion *ToggleUrlaub* ist das Hauptwerkzeug für die Urlaubsplanung. Diese Funktion wird vom Presenter aufgerufen, denn dieser von der WebView2 einen Klick auf eine Zelle des Urlaubsplaners empfängt. Erinnern wir uns an die Definition der Tabelle Urlaub: Ein Urlaubstag wird durch zwei Schlüssel-Felder bestimmt, der Kalendertag und die Person. Um Eindeutigkeit in der Darstellung zu erzielen, brauchen wir darüber hinaus den Slot. Das sind die drei ersten Argumente der *ToggeUrlaub*-Funktion. Die Funktion empfängt zwei weitere *ByRef*-Parameter, die von der erfolgreich ausgeführten Funktion befüllt werden. Über *verplant* wird der Presenter über die inzwischen verplanten Tage informiert. Im Argument *ueberzogen* wird mittgeteilt, dass das verfügbare Urlaubskontingent der betroffenen Person überzogen ist.

Es gibt mehrere mögliche Situationen, die zum Aufruf der Methode führen:

- Mausklick auf leere Zelle. Die Zelle soll von der aktuellen Person belegt werden.
- Mausklick auf von aktueller Person belegte Zelle. Die Zellen-Belegung soll aufgehoben werden.
- Mausklick auf von nicht-aktueller Person belegte Zelle. Funktionsabbruch: die Zellen-Belegung darf nicht geändert werden.
- Mausklick auf von nicht-aktueller Person belegte Zelle, jedoch belegt die aktuelle Peron einen anderen Slot des gleichen Kalendertages. Die beiden Belegungen werden vertauscht.

Diese Bedingungen werden von der Logik des Models erkannt und korrekt umgesetzt. Hier gibt es noch viel Spielraum, um die Bedienung der Anwendung komfortabler und intuitiver zu machen.

All diese Änderungen erfordern Änderungen in der angeklickten Zelle, ggf. aber auch in den Nachbarzellen des aktuellen Kalendertages. Deshalb macht es Sinn nach der Daten-Aktualisierung den kompletten Tag im Urlaubsplaner zu refreshen. Die Daten dafür liefert die Funktion GetTag in Form eines Recordsets. Die SQL-Parameterabfrage dafür lautet:

```
SELECT
  kalender.id_kalender,
  urlaub.id_person,
  urlaub.slot,
  urlaub.anteil,
  personen.kuerzel,
  personen.farbe
FROM
  kalender
  LEFT JOIN urlaub ON kalender.id_kalender = urlaub.id_kalender
  LEFT JOIN personen ON urlaub.id_person = personen.id_person
WHERE
  kalender.id_kalender = ?
ORDER BY
  urlaub.slot
```

Gleiches gilt für die Funktion *GetUrlaub*. Da wir hier das ganze Jahr benötigen entfällt die *WHERE*-Klausel. Die Sortierung muss mit der Kalender-ID ergänzt werden.

```
SELECT
  kalender.id_kalender,
  urlaub.id_person,
  urlaub.slot,
  urlaub.anteil,
  personen.kuerzel,
  personen.farbe
FROM
  kalender
  INNER JOIN urlaub ON kalender.id_kalender = urlaub.id_kalender
  INNER JOIN personen ON urlaub.id_person = personen.id_person
ORDER BY
  kalender.id_kalender,
  urlaub.slot
```

Eine weitere Funktion, die nach Optimierungen schreit, ist die *Defrag*-Methode. Sinn dieser Prozedur ist belegte Zellen möglichst weit nach links zu verschieben, um so die Anzahl der verbrauchten Slots zu minimieren, siehe „*Slots anordnen*“, auf Seite 17. Der aktuelle Algorithmus geht stupide nach der

Regel möglich-alles-nach-links vor. Damit erreicht er das bestmögliche Ziel, jedoch entstehen so auch unschöne Spalten-Anordnungen für die Urlaubsbelegung einzelner Personen.

View(s)

Im Urlaubsplaner verwenden wir zwei Typen von View, die jeweils über ein eigenes Interface mit dem Presenter gekoppelt sind. Über die iDocument-Schnittstelle wir die GUI der Anwendung erstellt, über die iExport die unterschiedlichen Export-Formate.

iDocument

Diese Schnittstelle kennt nur eine einzige Methode, die *LoadPage*-Methode.

```
Public Sub LoadPage(WV2 As cWebView2, _
                    Settings As RC6.cRecordset, _
                    Optional Skin As SkinConstants)
```

Wir übergeben der View über das Argument *WV2* eine Referenz auf die WebView2 Komponente, dort soll die Seite angezeigt werden. Die beiden anderen Argumente liefern Einstellungen und optional einen Skin. Es gibt drei Klassen im Projekt, die über diese Schnittstelle an den Presenter angebunden werden. Das sind die Klassen:

- cvNeu erstellt eine Seite, über die der Anwender einen neuen Urlaubsplaner anlegen oder importieren kann.
- cvOptionen erstellt eine Seite über die Programmeinstellungen vorgenommen und Feiertage, Ferien und der Betriebsurlaub verwaltet wird. Außerdem werden von hier aus, alle Exporte vorgenommen.
- cvUrlaubsplaner erstellt den Kalender, die Personalverwaltung und beinhaltet das Haupt-Menü.

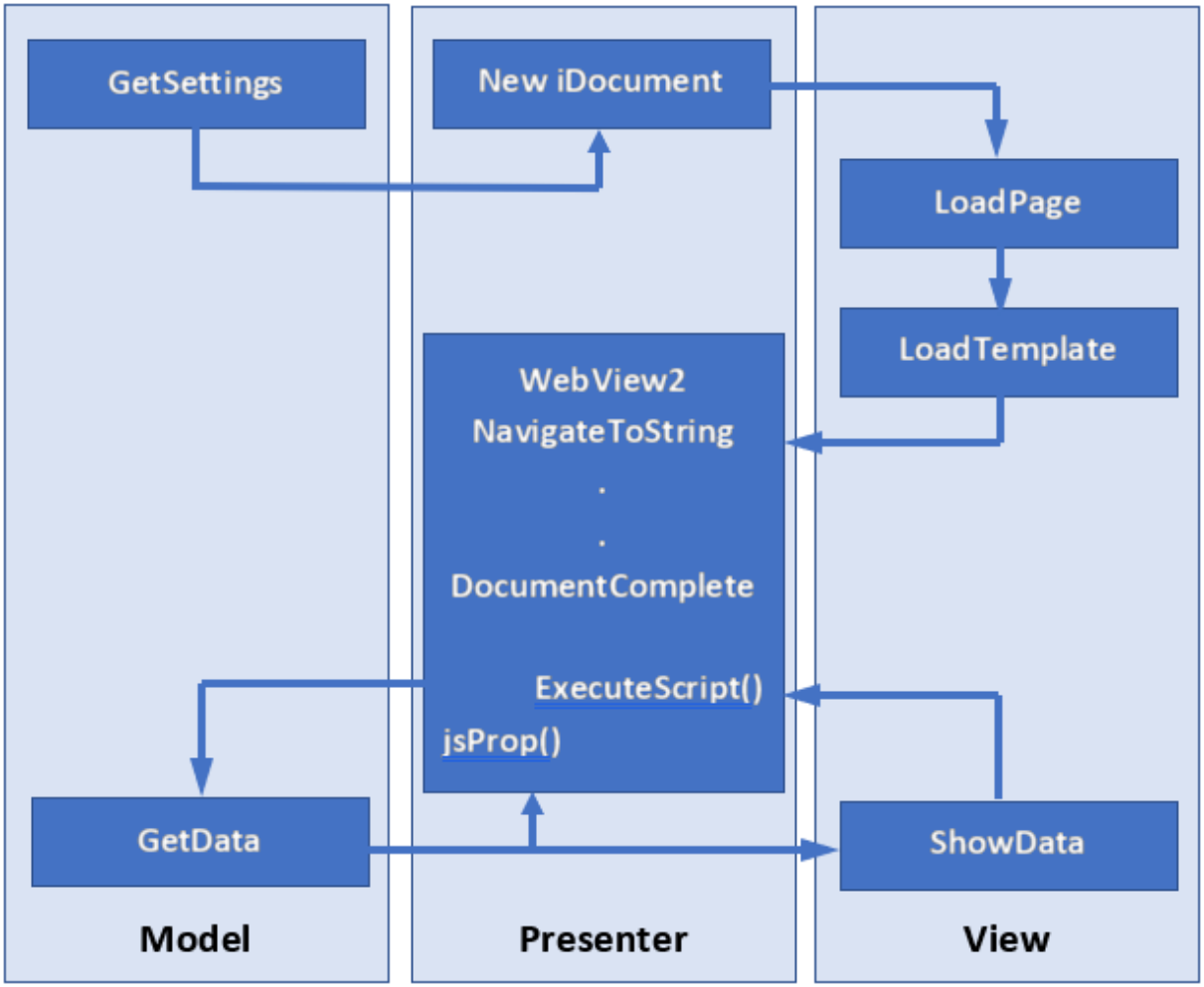
Das Prinzip zum Laden und Anzeigen der Views, im folgendem auch Seiten genannt, ist immer das gleiche. Im Presenter wird eine Instanz der Klasse erstellt und danach deren *LoadPage*-Methode aufgerufen. Zur Erinnerung, der Pseudo-Code im Presenter:

```
Select Case View
  Case Neu:      Set mvDocument = New cvNeu
  Case kalender: Set mvDocument = New cvKalender
  Case optionen: Set mvDocument = New cvOptionen
  End Select
mvDocument.LoadPage mWV, settings, skin
```

Die View lädt in *LoadPage* eine statische HTML-Vorlage in der bereits alle CSS-Definitionen, JavaScripts sofern benötigt und mindestens ein grobes HTML-Gerüst enthalten ist. Diese HTML-Seiten werden mit ganz normalen HTML-Editoren erstellt und getestet. Das muss nicht zwingend ein VB-Programmierer machen. Im Gegenteil: hierbei haben gelernte Medien-Fachleute oft das bessere Auge und Geschmack. Im Entwicklungsmodus (IDE) befinden sich diese Seiten noch als normale Dateien auf dem Entwickler-rechner oder im Intranet. In der fertigen Anwendung werden diese Seiten nicht mehr als editierbare Dateien ausgeliefert, sondern in eine Ressourcen-Datei gepackt. Das gilt nicht nur für die Seiten, sondern auch für Bilder und sonstige Zusätze. Einige Dateien können auch neben der Anwendung als editierbare Ressource ausgeliefert werden. Das ermöglicht dem Anwender eigene Skins zu erstellen.

In einer Intranet-Umgebung wäre es denkbar, die HTML-Vorlagen auch über einen Web-Server bereitzustellen. Im Prinzip könnte dieser sogar die Aufgabe des Models übernehmen. Je nachdem wie weit man diese Auslagerung treibt, mutiert die Desktop-Anwendung zu einer Web-Anwendung, die Grenzen sind fließend.

Die geladene Seite wird in die WebView2-Komponente, deren Referenz der View bekannt ist, über deren Methode *NavigateToString* übertragen.



Nach dem vollständigen Laden der Vorlage erzeugt die WebView2-Komponente das Ereignis *DocumentComlete*. Nun werden weitere Daten vom Model geladen und entweder direkt über *jsProp*-Methoden oder erneut durch die View aufbereitet und anschließend per ExecuteScript in die Seite übertragen.

Ab dieser Stelle übernimmt der Presenter, die View Ihren Dienst beendet. Sowohl das Seiten-Layout als auch die Daten sind in der WebView2 zur Anzeige gebracht worden. Es kann sein, dass später größere Layout- Oder Datenanpassungen im Programmverlauf anstehen, dann kann die View wieder übernehmen. Der Auslöser wird in der Regel ein Event der Webview2 sein, wie zum Beispiel das Navigieren zum nächsten Datensatz in einer DB-Anwendung. Dieses Event wird dem Presenter gemeldet, er ruft die Daten beim Model ab und aktualisiert genauso wie oben im Ereignis DocumentComplete die geladene Seite. Die Dienste der View könnten bei größeren Updates wieder gebraucht werden.

cvNeu

Diese Klasse erzeugt eine Seite, über die der Anwender einen neuen Urlaubsplaner anlegen oder importieren kann. Die Klasse besteht lediglich aus der Interface-Methode *LoadPage*.

```
Private Sub iDocument_LoadPage(WvBrowser As RC6.cWebView2, _
                               Settings As RC6.cRecordset, _
                       Optional Skin As SkinConstants)
Dim template As String
Dim cssSkin As String

  If HtmlHelper.ReadFromRes Then
    mWV.Navigate "file:///" & Replace(AppPath, "\", "/") & "/Res/neu.html"
  Else
    template = HtmlHelper.AppRes.GetContent("neu.html")
    template = Crypt.UTF8ToVBString(template)
    If Skin = skDark Then
      cssSkin = HtmlHelper.Crypt.UTF8ToVBString( _
                                 HtmlHelper.AppRes.GetContent("dark.css"))
      template = Replace(template, _
                         "<!--skin-->", _
                         "<style> " & cssSkin & " </style>", , _
                         1)
    End If
    mWV.NavigateToString template
  End If
```

Es gibt ein paar neue Objekte in dieser Prozedur, die wir kurz besprechen werden. *ReadFromRes* ist eine globale Variable in einem global verfügbaren Modul *HtmlHelper*. Über diese Variable steuern wir woher die Html-Vorlagen geladen werden. Im Entwicklungsmodus (IDE) ist es hilfreich, wenn diese Dateien von einem lokalen Verzeichnis gelesen werden. Damit können die Vorlagen leicht geändert und per F5 (oder Methode Reload) aktualisiert werden, ohne dass die Anwendung erneut gestartet werden muss. Also auch hier echtes *Edit & Continue*, dass die VB-Entwickler so sehr an ihrer Sprache lieben. Wenn die Anwendung ausgeliefert wird und beim Anwender ausgeführt wird, ist dieses Vorgehen kontraproduktiv. Die von jedermann einsehbaren und editierbaren HTML-Dateien müssten mit ausgeliefert werden. Deshalb werden diese Ressourcen, zusammen mit Bildern, JavaScript-Dateien und unveränderbare CSS-Dateien in eine globale Ressourcendatei gepackt. Diese Datei muss aber nicht umständlich vor jedem Deployment von Hand erstellt werden, das erledigt unsere Anwendung automatisch von selbst. Siehe dazu die Ausführungen im Kapitel Deployment.

Halten wir fest: Die HTML-Vorlagen können entweder direkt per *Navigate*-Methode von der Festplatte in die WebView2 gelangen oder per NavigateToString aus einer globalen Ressource gelesen werden. Genaugenommen gibt es noch eine dritte Variante: Die Navigate-Methode kann auch von einer URI, also einem Web-Server lesen. Dieser Server kann im Internet, im Intranet, auf dem eigenen Rechner als eigenständiger Prozess oder sogar Modul in der eigenen Anwendung sein. Auch dafür bietet das RC6-Framework passende Objekte.

Beim Lesen aus dieser Ressource müssen wir die UTF8-kodierten Seiten in normale VB-Strings umwandeln. Dafür nutzen wir ein Crypt-Objekt, dass uns ebenfalls das RC6-Framework zur Verfügung stellt. Dieses Objekt habe wir bereits beim Start der Anwendung in eine globale Objektvariable, im Modul HtmlHelper instanziiert. Der Name der Klasse zeigt, welche Dienste sie liefert: Es handelt sich um kryptografische Methoden, die Daten konvertieren, ver- und entschlüsseln, komprimieren und

entpacken oder Hashes von Strings berechnet. Ein Blick in den VB-Objektkatalog zeigt die vielfältigen Verwendungszwecke dieser Klasse.

Es ist relativ einfach Anwendungen mit Skins auszustatten. Ein Skin ist ein, vom Anwender umschaltbares GUI-Motiv, eine Zusammenstellung von Einstellungen, die das Erscheinungsbild der Anwendung festlegen. Im Urlaubsplaner kann der Anwender das normale helle Theme mit einem dunklen ersetzen. Dafür werden im Template einige CSS-Definitionen ersetzt. Beispiel:

```
body {background-color:#333}
```

Mit dieser CSS-Definition für den Body wird die Hintergrundfarbe der Anwendung auf einen sehr dunklen Farbton eingestellt. Alle Einstellungen werden ein einer CSS-Datei zusammengefasst. Diese Datei wird per VB-Replace-Methode in das Template eingebaut. Das ist ein trivial einfaches Vorgehen, ist aber einfach anzuwenden. Man kann auf diesem Weg dem Anwender sogar gestatten eigene Skins zu erstellen. Damit die erforderlichen CSS-Dateien editierbar bleiben, sollten sie nicht in das Ressource-File integriert werden. Die Gestaltungs-Möglichkeiten sind vielfältig und beschränken sich nicht auf Farb-Konstellationen.

Wichtig bei der Integration von Skins ist der HTML-Kommentar <!--skin-->. Dieser Kommentar dient als Textmarke für das Einfügen der CSS-Definitionen per VB-*Rplace* und muss in der HTML-Vorlage hinter den dort definierten CSS-Anweisungen stehen.

```
<!DOCTYPE html><html lang="de">
<head>
...
  <style>
    *{font-family:Verdana;font size:16px;padding:0px;margin:0px;color:#000}
  </style>
  <!--skin-->
</head>
<body>
```

Die Definitionen im Skin werden die Standard-Definitionen aus der HTML-Vorlage überschreiben.

Der Aufbau der Klasse *cvNeu* ist deshalb einfach, weil die Klasse selbst wenig zum Layout der Seite beiträgt. Die eigentliche Arbeit steckt in der HTML-Vorlage. Spätestens als Microsoft die HTML-basierten chm-Hilfen eingeführt hat, müssen sich auch VB-Programmierer mit dieser Thematik beschäftigen. Es sei denn, der Programmierer kann diese Aufgaben an einen Web- oder Mediendesigner delegieren. Wie dem auch sei, der VB-Programmierer wird sich mit einigen Web-Themen auseinandersetzen müssen, um zu wissen, wie man DOM-Elemente anspricht, welche Eigenschaften diese Elemente haben und wie die Kommunikation aus der WebView2 in Richtung Anwendung funktioniert.

Die HTML-Seite hat folgenden Aufbau:

```
<!DOCTYPE html><html lang="de">
  <head>
  <title>Urlaubsplaner anlegen</title>
  <meta charset="utf-8">
  ...

  <style>
    *{font-family:Verdana;font-size:16px;padding:0px;margin:0px;color:#000}
```

```
    ...
  </style>
  <!--skin-->

  <script type='text/javascript'>
    document.addEventListener('click', function(e){
      if(e.target.tagName=='BUTTON' || e.target.tagName=='A')
        vbH().RaiseMessageEvent('neu.Click', e.target.id);
    });
  </script>
</head>

<body>

<div class="bg" style="position:absolute;width:100%;height:100%">
 <div id="main">
   <div id="new">
     <h1>Neuen Urlaubskalender anlegen</h1>
     <hr>
     <label for="txtJahr">Jahr:</label><br>
     <input type='number' id='txtJahr' min='2021' max='2050'><br>
     <label for="selLand">Bundesland:</label><br>
     <select id='selLand'>
       <option value='1'>Baden Württemberg</option>
       <option value='2'>Bayern</option>
        ...
     </select>
     <button id='btnCancel' class='btn'>Abbrechen</button>
     <button id='btnSave'   class='btn'>Ok</button>
   </div>
   <p class="bg">
     <a id='fileOpen'   class="bg" href="#">Datei öffnen</a>
     <a id='fileImport' class="bg" href="#">Datei importieren</a>
   </p>
  </div>
 </div>
</body>
</html>
```

Die *<!DOCTYPE>*-Deklaration repräsentiert den Dokumenttyp, hier HTML5. Mit dem Tag <html> beginnt das HTML-Dokument. Das Attribut *lang=de* definiert die Seiten-Sprache, hier also Deutsch. *<title>* steht für den Seitentitel. Das <head>-Element ist ein Container für Metadaten, CSS-Deklarationen und JavaScript. Als Meta-Tag sollte zumindest *<meta charset="UTF-8">* angegeben werden, das spezifiziert die Zeichenkodierung für das HTML-Dokument.

Das *<style>*-Element, leitet die Formatvorlage für das HTML-Dokument ein. Hier wird festgelegt, wie HTML-Elemente dargestellt werden sollen. Wenn die WebWiew2-Komponente ein Stylesheet liest, formatiert sie das HTML-Dokument entsprechend den Informationen im Stylesheet. Wenn einige Eigenschaften für denselben Selektor (Element) in verschiedenen Stylesheets definiert wurden, wird der Wert aus dem zuletzt gelesenen Stylesheet verwendet. Das machen wir uns zunutze und platzieren hinter dem Style-Element einen HTML-Kommentar *<!--skin-->*. Unsere Anwendung wird an dieser Stelle den Kommentar mit einem spezifischen Skin-Stylesheet ersetzen.

Das *<script>*-Tag wird verwendet, um ein clientseitiges Skript (JavaScript) einzubetten. An dieser Stelle können wir auch Funktionen für die Kommunikation mit unserer Anwendung einbauen. Das ist jedoch nur eine Möglichkeit von vielen. Man kann genauso gut Scripts über ein src-Attribut aus externen Script-Dateien laden. Weitere Varianten wäre das Platzieren einzelner Script-Anweisungen direkt in

HTML-Tags oder einen Script-Block am Ende des Bodys. Letzte Variante verbessert die Anzeigegeschwindigkeit, da die Skriptinterpretation die Anzeige verlangsamt. Diesen Effekt merkt allerdings man nur bei großen Script-Blöcken.

Im Body wird der sichtbare HTML-Content platziert. In Desktop-Anwendungen, die für die ganze GUI HTML verwenden, werden das diverse Formularelemente sein. Unsere einfache Seite besteht aus Labels, Input- und Select-Felder, Buttons und Links. Wobei Letztere nicht zu den Formularelementen zählen. Alle anderen HTML-Elemente definieren Absätze, Blöcke, Tabellen, Listen und Aufzählungen usw.

Da dieser Text weder ein Lehrbuch für HTML, noch für CSS und JavaScript ist, widmen wir uns wieder der Kommunikation zwischen Seite und Anwendung zu. Die RC6-Klasse *cWebView,* über die die WebView2-Komponente eingebunden ist erzeugt in jeder geladenen Seite ein globales *vbH*-Objekt. Diese Objekt hane wir bereits im Presenter bei den Ereignis-Prozeduren angesprochen, siehe Seite 33. Der Name vbH ergibt sich aus *Visual Basic Handler*. Seine Methode *RaiseMessageEvent* ist der zentrale Rückkanal aus JavaScript in die VB-Welt. Das vbH-Objekt ist genaugenommen ein Adapter oder Wrapper. In der Entwurfsmuster-Theorie ein Strukturmuster, das zur Übersetzung einer Schnittstelle in eine andere dient. Für den VB-Entwickler ist es bedauerlich, dass im Web-Bereich vbScript in die Sackgasse geraten ist. JavaScript hat sich auf allen Ebenen durchgesetzt und ist damit auch die zentrale Script-Sprache im WebView2. Das macht die Kommunikation zwischen VB und WebView2 etwas schwierig. Die Adapter im RC6-Framework machen jedoch vieles einfacher, ja sogar erträglich. Bestes Beispiel ist das hier vorgestellte vbH-Objekt, genaugenommen ein Alias für window.chrome. webview.hostObjects.sync.vb_Host. Die interne Implementation sieht wie folgt aus:

```
function vbH(){
  return window.chrome.webview.hostObjects.sync.vb_Host
}
```

Diese Host-Objekte werden als Proxys von der Runtime dargestellt. JavaScript kann darüber mit der Außenwelt, in diesem Fall synchron kommunizieren.

Damit diese Methode aufgerufen werden kann, muss es einen Auslöser geben. Diesen Auslöser liefert ein DOM-Event, dass global als *click-EventListener* im JavaScript-Block erstellt wurde. Es lösen aber nicht alle Klicks die *vbH.RaiseMessageEvent*-Methode aus, sondern nur Buttons (tagName == 'BUTTON') und Links (tagName == 'A'). Theoretisch wäre es möglich alle Klicks an VB zu übermitteln und dort die Auswertung vorzunehmen. Im Sinne der Performance ist das allerdings ungünstig. Bie VB-Benachrichtigung über das vbH-Objekt kostet Zeit. Meine Messungen ergaben einen Versatz zwischen 2 und 20 ms. Deshalb die Empfehlung an dieser Stelle: Nutzt das vbH-Objekt nur da, wo notwendige Daten und Ereignisse übermittelt werden und sortiert möglichst früh Unnötiges aus. Dabei sollte aber nicht Anwendungslogik in die View verlagert werden. Es ist sinnvoll zum Beispiel Eingabevalidierungen bereits in der View zu vornehmen, wenn dafür die Bedingungen bekannt sind.

Die Methode *RaiseMessageEvent* bekommt zwei Parameter in Form von Strings mitgeliefert. Im ersten Parameter übergeben wir das Ereignis im zweiten Parameter die ID des Auslösers, der oft als Sender bezeichnet wird. Das ist hier kein festes Regelwerk. In den beiden Parametern werden keine vorab definierten Daten erwartet. Es obliegt den Beteiligten, also dem VB-Programmierer und dem Web-Designer festzulegen, welche Daten die Methode transportieren soll. Da im Presenter alle Ereignisse

eintreffen, ist es Sinnvoll eine eigene konsistente Regelung festzulegen. Das dient der Code-Lesbarkeit und so der späteren Wartung. Je größer das Entwickler-Team ist, desto wichtiger sind diese Regeln.

cvOptionen

Für die Verwaltung der Programmeinstellungen benötigen wir eine komplexere Eingabemaske. Wir finden hier weitere Steuerelemente und Tabellen. Die Scripts in der HTML-Vorlage sorgen hier nicht allein für Ereignisse im Presenter, sondern werden aktiv für die Darstellung und Bedienung der Seite ausgeführt. Ein einfaches Beispiel sind die Hilfetexte, die am Ende der HTML-Seite definiert sind. Die Absätze (p-Tags) sind mit der CSS-Klasse *info* ausgezeichnet. Diese Klasse sorgt mit der Einstellung *display:none* dafür, dass alle Hilfe-Absätze ausgeblendet sind. Beim Mausklick auf ein Hilfe-Fragezeichen wird die Funktion *showInfo()* aufgerufen:

```
<span id="tgFeiertage" class="tgBtn" onClick="showInfo('infoFeiertage')">
```

Die Funktion wertet das Argument *id* aus, das der ID des Hilfe-Textes entspricht. Der Hilfetext aus der Vorlage wird in den ebenfalls unsichtbaren info-Container (div-Tag) am Anfang der Seite übertragen. Danach wird der Container sichtbar gemacht. Das Gegenstück dazu ist die *closeInfo*-Funktion, die den Hilfe-Container wieder ausblendet. Dafür braucht es kein Argument mehr, weil sich das Schließen auf den zentralen Hilfe-Container bezieht.

```
function showInfo(id) {
  document.getElementById('infoContent').innerHTML =
                                    document.getElementById(id).innerHTML;
  document.getElementById('info').style.display='block';
}

function closeInfo(){
  document.getElementById('info').style.display='none';
}
```

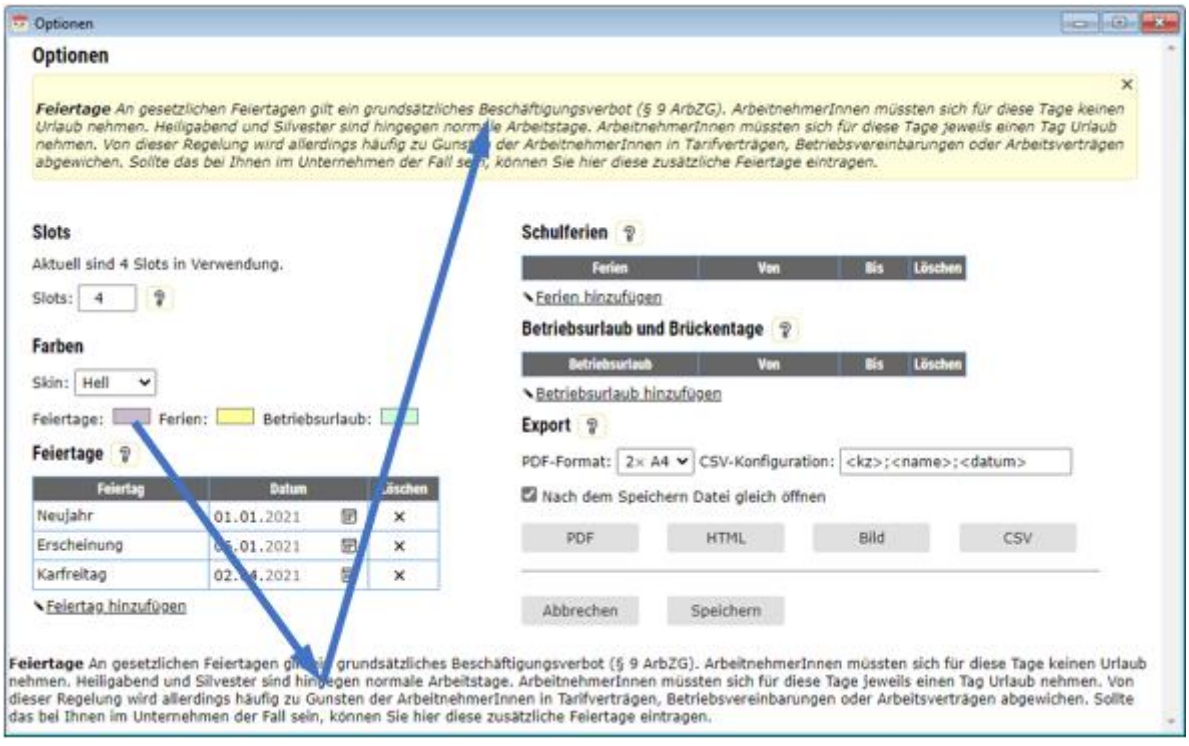


Abbildung 21: Anzeigen der Hilfetexte

Wie man sieht, spielen bei dieser Aktion Presenter und Model keine Rolle. Genaugenommen gilt das für die ganze VB-Programmierung, weil Anzeigen und Ausblenden vollständig vom JavaScript der HTML-Vorlage gesteuert werden. Sollte nun aber der Text dynamisch sein, können wir diesen nicht mehr in der HTML-Vorlage bereithalten. Das wäre zum Beispiel der Fall, wenn die Anwendung mehrsprachig ausgeliefert wird. In diesem Fall würden wir das Klick-Ereignis über das vbH-Objekt an den Presenter weiterleiten. Dieser kann nun die *showInfo*-Funktion selbst aufrufen und statt der ID den fremdsprachigen Text als Parameter übergeben. Die Änderung im Quellcode der HTML Vorlage wäre einfach:

```
function showInfo(infoText) {
  document.getElementById('infoContent').innerHTML = infoText;
  document.getElementById('info').style.display='block';
}

...

<span class="tgBtn" onClick="vbH().RaiseMessageEvent('info','Slots')">
```

Im Presenter müssen wir nur noch das Info-Ereignis abfangen und die showInfo-Funktion mit Hilfetext aufrufen:

```
Private Sub mWV_JSMessage(ByVal sMsg As String, _
                          ByVal sMsgContent As String, _
                          oJSONContent As RC6.cCollection)
Select Case sMsg
  Case "info"
    Select Case sMsgContent
      Case "Slots"
        mWV.jsRun "showInfo", "You can set 2 to max. 30 slots..."
      Case ...
    End Select
  Case ...
End Select
```

Das gleiche Prinzip wenden wir bei Fehlermeldungen an. Beim Speichern der Einstellungen übergibt der Presenter die geänderten Daten an das Model, wo diese geprüft werden:

```
Private Sub mWV_JSMessage(ByVal sMsg As String, _
                          ByVal sMsgContent As String, _
                          oJSONContent As RC6.cCollection)
Dim errors As cArrayList
...
  Select Case sMsg
    Case "optionen.save"
      If mUrlaubsplaner.SaveOptionen(oJSONContent, errors) Then
        ...
      Else
        mWV.ExecuteScript "notification('" & errors.Join("<br>") & "')"
      End If
    Case ...
  End Select
End Sub
```

Anders als der info-Container gibt es den *notification*-Container nicht in der HTML-Vorlage. Deshalb wird dieser in der *notification*-Funktion zuerst angelegt. Das muss nicht so sein. Dieses Vorgehen wurde hier lediglich als alternative Umsetzung implementiert. Es handelt sich dabei um JavaScript-Anwei-

sungen, die zunächst ein neues DOM-Element erzeugen und es direkt an den body-Anfang platzieren, bevor es angezeigt wird:

```
function notification(msg) {
  ...
  ntfc = document.createElement('div');
  ntfc.innerHTML=msg;
  ...
  document.body.insertBefore(ntfc, document.body.firstChild);
  ...
  ntfc.style.display = 'block';
}
```

Gleiches Prinzip wird beim Hinzufügen von Feiertagen, Ferientagen oder Betriebsurlaubstagen. Es werden zuerst neue DOM-Elemente (Tabellenzeilen, Tabellenspalten) erzeugt und an die jeweilige Tabelle angehängt. Die neuen Zellen werden mit dynamisch erzeugten HTML-Code befüllt. Hierbei braucht es keine einzige Zeile VB-Code, alles kann mit JavaScript umgesetzt werden, ohne dass Presenter oder Model bemüht werden.

```
function addFeiertag() {
  let tbl=document.getElementById('tblFeiertage');
  let rows = tbl.rows.length;
  let row = tbl.insertRow(rows);
  let c1 = row.insertCell(0);
  let c2 = row.insertCell(1);
  let c3 = row.insertCell(2);
  c1.innerHTML = "<input ... />";
  c1.innerHTML = "<input ... />";
  c1.innerHTML = "<span ... >&#10006;</span>";
}
```

Das Gegenstück zum Hinzufügen ist das Löschen einer Zeile in der Tabelle:

```
function delUrlaub(r) {
  let i = r.parentNode.parentNode.rowIndex;
  document.getElementById('tblUrlaub').deleteRow(i);
}
```

Beim Speichern der Einstellungen wird für die Übertragung der Daten das vbH-Objekt verwendet. Als wir die Ereignisprozeduren im Presenter besprochen haben (ab Seite 32), konnten wir im dritten *oJSONContent*-Argument der Ereignis-Prozedur die Daten in einer JSON-Collection empfangen. Damit das funktioniert, müssen die Nutzdaten als JSON-Objekt aufbereitet werden. Das Verfahren ist im folgenden Codeabschnitt zu sehen:

```
var data = {feiertage:[],ferien:[],urlaub:[]};
data.slots = parseInt(document.getElementById('txtSlots').value);
data.cFeiertage = document.getElementById('colFeiertage').value;
...
var input = document.getElementsByClassName('feiertag');
var inputList = Array.prototype.slice.call(input);
inputList.forEach(function(item,index) {
  let id = item.id.substr(6);
  id:    id,
    let obj = {
      name:  item.value,
      datum: document.getElementById('txtFtD' + id).value
    };
```

```
  data.feiertage.push(obj);
});
//Gleiches für Ferien und Betriebsurlaub...
vbH().RaiseMessageEvent('optionen.save',JSON.stringify(data));
```

Wir definieren eine Objektvariable *data*, die drei Arrays für Feiertage, Ferien und den Betriebsurlaub enthält. Alle anderen Werte werden als Eigenschaften, also Schlüssel-Werte-Paare in das *data*-Objekt eingebunden. Zum Schluss wird das Objekt per *JSON.stringify*-Methode (konvertiert einen JavaScript-Wert in einen JSON-String) als zweiter Parameter der Methode *RaiseMessageEvent* eingetragen.

Für alle anderen Buttons (Export und Abbrechen) verwenden wir wieder die bereits bekannte Ereignis-Übermittlung mittels vbH-Objekt.

```
document.addEventListener('click', function(e){
  if(e.target.tagName==='BUTTON') {
    if (e.target.id==='btnSave') {
      //siehe oben: JSON-Aufbereitung
      vbH().RaiseMessageEvent('optionen.save',JSON.stringify(data));
    } else {
      vbH().RaiseMessageEvent('optionen.btnClick',e.target.id );
    }
  }
});
```

cvUrlaubsplaner

Debugging

JavaScript-Code, der in der WebView2-Komponente ausgeführt wird, kann in der VB-IDE nicht debuggt werden. Das cWebView-Objekt bietet zwar eine Eigenschaft jsLastError (Seite 75), mit der man den letzten JavaScript-Fehler auslesen kann, das dürfte aber in den meisten Fällen zu wenig sein, um Fehlerfreiheit im Script-Code sicherzustellen. Ein brauchbares Debugging sollte den Inhalt von Variablen ausgeben und ändern. Haltepunkte sollten im Code definierbar sein, die an beliebiger Stelle die Programmausführung unterbrechen und ggf. in Einzelschritten fortführen.

Mit der WebView2-Runtime werden automatisch auch die Chromium-basierten Entwicklertools mitgeliefert. Standardmäßig sind diese Tools sogar eingeschaltet und können über das Kontextmenü, die Taste F12 oder mittels Methodenaufruf OpenDevToolsWindow (s. Seite 85) aufgerufen werden. Damit die Tools nicht auch vom Endanwender aufgerufen werden, können sie über die Eigenschaft AreDevToolsEnabled (s. Seite 71) deaktivieren.

DevTools

DevTools ist eine allgemein verwendete Abkürzung für die *Microsoft Edge Developer Tools*. Beim Aufruf der DevTools über die oben genannten Methoden, öffnet sich ein eigenständiges Fenster, das folgende Informationen und Werkzeuge anbietet:

- Überprüfung und Änderung der aktuellen Webseite live im Browser.
- Überprüfen, Optimieren und Ändern der Formatvorlagen von Elementen.
- JavaScript-Debugging mithilfe des Haltepunktdebuggers und mit der Livekonsole.
- Barrierefreiheits-, Leistungs-, Kompatibilitäts- und Sicherheitsprobleme in der GUI auffinden.,
- Netzwerkdatenverkehr verfolgen und überprüfen.
- Speicher- und Rendering-Probleme identifizieren.
- Vieles mehr…

Das Fenster der DevTools verfügt über eine Hauptsymbolleiste, auf der sich unterschiedliche Schalter und Registerkarten befinden. Bei einigen Registerkarten öffnet sich eine sekundäre Symbolleiste, die für die Funktion spezifische Werkzeuge (Schalter, DropDown-Boxen, Menüs usw.) anbietet.

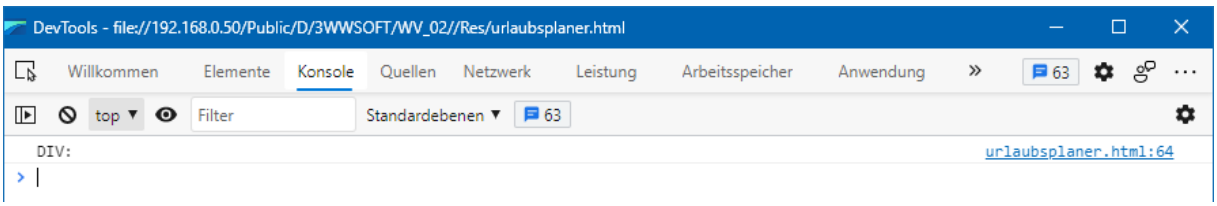


Abbildung 22: DevTools Haupt- und Sekundär-Symbolleiste

Das erste Symbol in der Hauptsymbolleiste ist das sogenannte Inspect-Tool . Damit kann man ein Element auf der aktuellen Seite auswählen. Nachdem das Tool aktiviert wurde, kann man die Maus über verschiedene Teile der Seite bewegen und so detaillierte Informationen tu dem Element unter dem Mauszeiger erhalten. Eine Farbüberlagerung auf der Seite markiert das Element und eine Sprechblase zeigt dessen wichtigste Eigenschaften. Wenn das hervorgehobene Element angeklickt wird, wechselt das DevTool zum Register *Elemente* und markiert dort den HTML-Quellcode für das selektierte Element:

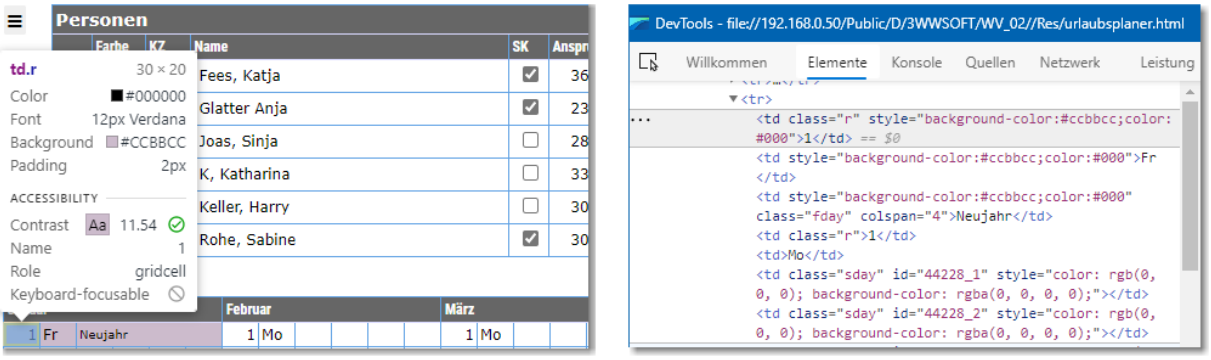


Abbildung 23: Das Inspect-Tool im Einsatz

Der umgekehrte Weg funktioniert genauso: Wenn im HTML-Code ein Element-Tag aus der DOM-Struktur mit dem Mauszeiger überfahren wird, zeigt sich eine kleine Sprechblase über dem Element auf der Seite. Die Aktivierung des Tags im Quellcode erzeugt die bereits erwähnte Farbüberlagerung auf der Seite.

Eine weitere Möglichkeit Elemente zu selektieren, bietet das Kontextmenü des Konsolenfensters:

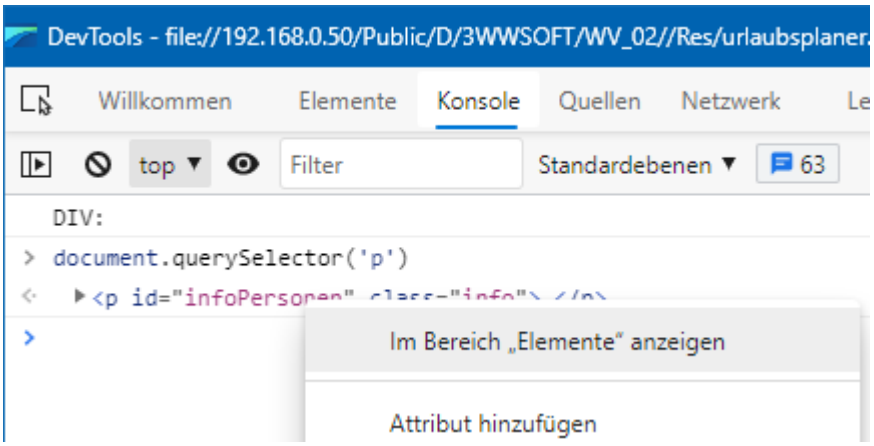


Abbildung 24: Element-Selektion über die Konsole

Die Registerkarte *Elemente* zeigt nicht nur den HTML-Quellcode, sondern im zweigeteilten Fenster weitere Elementeigenschaften. Für die GUI-Gestaltung liefern die *Formatvorlagen* alle CSS-Informationen, die man hier auch temporär deaktivieren, ändern oder ergänzen kann.

Deployment

Referenzen

cWebView Referenz

Eigenschaften

AreDefaultContextMenusEnabled

```
Property AreDefaultContextMenusEnabled As Boolean
```

Legt fest, ob das Standard-Kontextmenü dem Benutzer angezeigt wird. Die Standardeinstellung ist *True*, die Kontextmenüs sind aktiv.

Beispiel:

```
With WV
  .AreDefaultContextMenusEnabled = False 'Browser-Kontextmenü deaktivieren
  .NavigateToString "<!DOCTYPE html><html>...</html>"
  .AreDefaultContextMenusEnabled = True 'wirkt erst mal nicht
  .Reload 'ab hier würde das Kontextmenü wieder erscheinen
End With
```

Anmerkungen:

Aktivieren oder deaktivieren Sie diese Eigenschaft immer nach der *BindTo*-Methode und vor dem Laden einer Seite (*Navigate* oder *NavigateToString*). Ein nachträgliches Umschlaten wirkt sich auf die aktuell geladene Seite nicht aus.

Nach dem Deaktivieren der Kontextmenüs können Sie ein eigenes Kontextmenü optional anzeigen. Dafür können Sie das Ereignis *UserContextMenu* auswerten.

AreDefaultScriptDialogsEnabled

```
Property AreDefaultScriptDialogsEnabled As Boolean
```

Aktiviert oder unterdrückt Window-Dialoge wie *alert*, *print* oder *prompt*. Die Standardeinstellung ist *True*, die Dialoge sind aktiv.

Beispiel:

```
With WV
  .AreDefaultScriptDialogsEnabled = False 'Script-Dialoge unterbinden
  .NavigateToString "<!DOCTYPE html><html>...</html>"
  'das nachfolgende JavaScript wird zwar ausgeführt,
  'der Browser unterdrückt aber das Dialog-Fenster mit der Meldung.
  .ExecuteScript "Window.alert('Meldung in einem Dialogfenster')"
  .AreDefaultScriptDialogsEnabled = True 'wirkt erst mal nicht
  .Reload 'ab hier würde das Dialog-Fenster wieder erscheinen
End With
```

Anmerkungen:

Aktivieren oder deaktivieren Sie diese Eigenschaft immer vor dem Laden einer Seite (Navigate oder NavigateToString). Ein nachträgliches Umschalten wirkt sich auf die aktuell geladene Seite nicht aus.

AreDevToolsEnabled

```
Property AreDevToolsEnabled As Boolean
```

Legt fest, ob der Benutzer das Kontextmenü oder das Tastaturkürzel zum Öffnen des DevTools-Fensters verwenden kann. Die Standardeinstellung ist *True*, die DevTools sind aktiv.

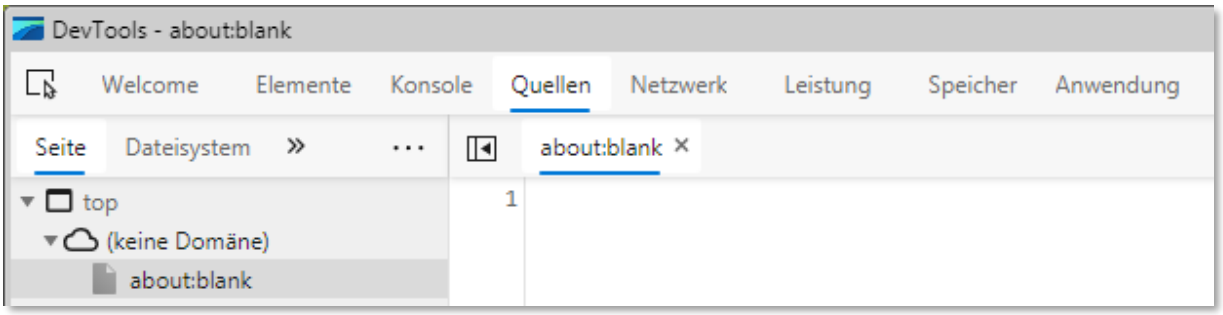
Beispiel:

```
With WV
  .AreDevToolsEnabled = False 'DevTools deaktivieren
  .NavigateToString "<!DOCTYPE html><html>...</html>"
  .AreDevToolsEnabled = True 'wirkt erst mal nicht
  .Reload 'ab hier würden die DevTools wieder aktiv sein
End With
```

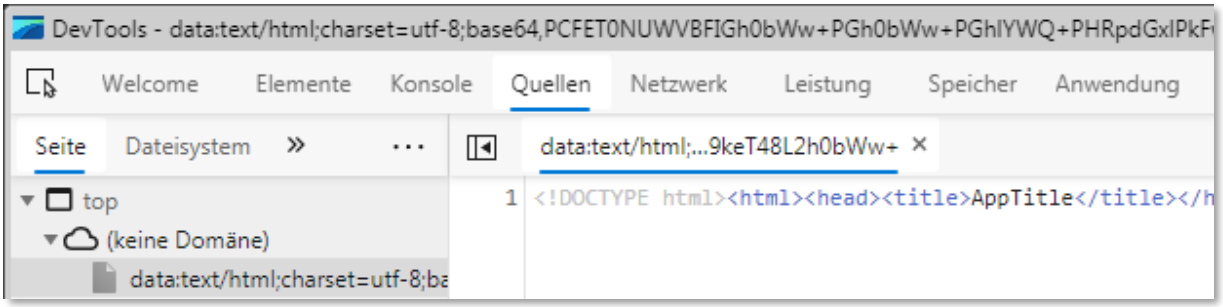
Anmerkungen:

Aktivieren oder deaktivieren Sie diese Eigenschaft immer vor dem Laden einer Seite (Navigate oder NavigateToString). Ein nachträgliches Umschalten wirkt sich auf die aktuell geladene Seite nicht aus. Unabhängig von dieser Einstellung, können Sie die DevTools mittels Methode OpenDevToolsWindow aufrufen.

Es gibt eine Besonderheit beim Verwenden der DevTools, wenn die Seite mittels *NavigateToString* erzeugt wurde. In diesem Fall zeigt der Reiter Quellen die Seite *about:blank* an und Sie können hier keinen HTML-Quelltext sehen. Das macht die Arbeit mit dem JavaScript-Debugger oder das Ändern von CSS-Eigenschaften schwierig.



Wenn Sie allerdings nach dem Start der DevTools die Seite neu laden (per Tastaturkürzel oder die *Reload*-Methode aufrufen), werden die Quellen angezeigt:



AreHostObjectsAllowed

```
Property AreHostObjectsAllowed As Boolean
```

Legt fest, ob Hostobjekte von der Seite im WebView2 zugänglich sind. Die Standardeinstellung ist *True*, die Hostobjekte sind aktiv.

Beispiel:

```
'Hostobjekte (beim nächsten Laden) deaktivieren
WV.AreHostObjectsAllowed = False
```

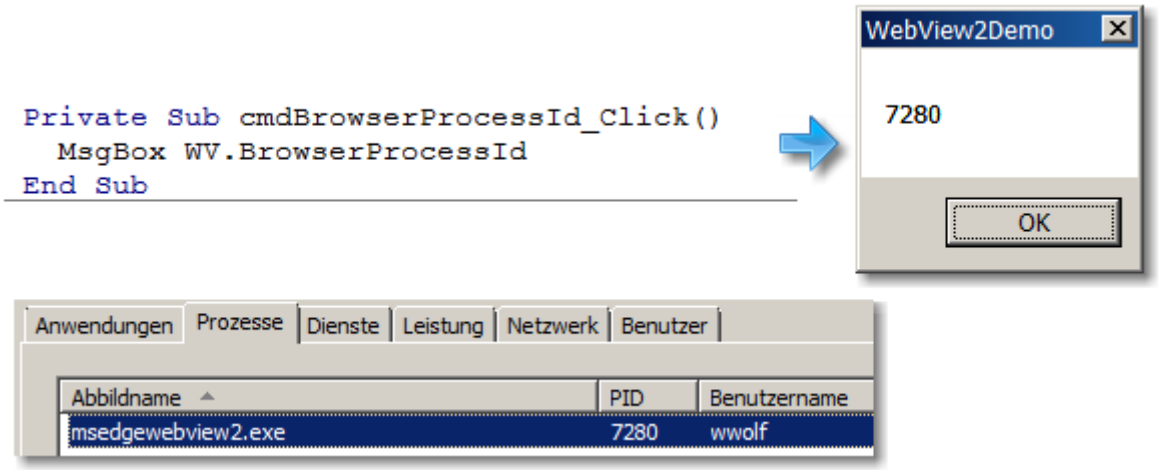
Anmerkungen:

Aktivieren oder deaktivieren Sie diese Eigenschaft immer vor dem Laden einer Seite (Navigate oder NavigateToString). Ein nachträgliches Umschalten wirkt sich auf die aktuell geladene Seite nicht aus. Unabhängig von dieser Einstellung, können Sie die DevTools mittels Methode OpenDevToolsWindow aufrufen.

BrowserProcessId

```
Property BrowserProcessId As Long
```

Beim Aufruf der Bind-Methode wird ein eigenständiger Prozess (msedgewebview2.exe) gestartet. Über diese Eigenschaft kann die PID dieses Prozesses ermittelt werden. Sie finden diesen Prozess und dessen PID auch im Windows-Taskmanager. Die Eigenschaft ist schreibgeschützt.



CanGoBack

```
Property CanGoBack As Boolean
```

Die schreibgeschützte Eigenschaft liefert ein *true*, wenn der WebView in der Lage ist, zu einer vorherigen Seite in der Navigationshistorie zu navigieren.

CanGoForward

```
Property CanGoForward As Boolean
```

Die schreibgeschützte Eigenschaft liefert den Wert *True*, wenn der WebView in der Lage ist, zu einer Folgeseite in der Navigationshistorie zu navigieren.

DocumentTitle

```
Property DocumentTitle As String
```

Ruft den Titel für das aktuelle Top-Level-Dokument ab. Die Eigenschaft ist schreibgeschützt. Über die *jsProp*- oder die *ExecuteScript*-Methode können Sie denoch den Seitentitel ändern.

Beispiel:

```
WV.Navigate "https://wiki.selfhtml.org"
Debug.Print WV.DocumentTitle 'SELFHTML-Wiki
WV.jsProp("document.title") = UCase(WV.DocumentTitle)
Debug.Print WV.DocumentTitle 'SELFHTML-WIKI
```

DocumentURL

```
Property DocumentURL As String
```

Ruft die schreibgeschützte URL des aktuellen Top-Level-Dokuments ab. Verwenden Sie die *Navigate*-Methode, um zu einer anderen Webseite zu navigieren.

Beispiel:

```
Me.Caption = WV.DocumentURL
```

HosthWnd

```
Property HosthWnd As Long
```

Liefert die schreibgeschützte Zugriffsnummer für das Fenster oder Steuerelement (z.B. PictureBox), das die WebView2-Komponente hostet.

Beispiel:

```
Set WV = New_c.WebView2 'erzeugt eine neue WebView2-Instanz
ret = WV.BindTo(picWV.hWnd, , , , "--lang=de")
If ret <> 0 Then
    Debug.Print picWV.hWnd; WV.HosthWnd 'liefert zweimahl die gleiche Zahl
End If
```

IsScriptEnabled

```
Property IsScriptEnabled As Boolean
```

Legt fest, ob die Ausführung von JavaScript im WebView aktiviert ist. Die Standardeinstellung ist *True*, die JavaScripts sind aktiv.

Beispiel:

```
'JavaScript (beim nächsten Laden) deaktivieren
WV.IsScriptEnabled = False
WV.Reload 'JavaScript ist jetzt deaktiviert
```

Anmerkung

Sie müssen diese Option vor dem Laden einer Seite festlegen.

IsStatusBarEnabled

```
Property IsStatusBarEnabled As Boolean
```

Legt fest, ob die Statusleiste angezeigt wird. Die Standardeinstellung ist *True*, die Statusleiste wird bei bestimmten Browser-Aktionen angezeigt.

Beispiel:

```
'Statusleiste aktivieren
WV.IsStatusBarEnabled = True
WV.Navigate https://www.selfhtml.org
```

IsWebMessageEnabled

```
Property IsWebMessageEnabled As Boolean
```

Legt fest, ob die Kommunikation vom Host zum HTML-Dokument der obersten Ebene des WebViews erlaubt ist. Die Standardeinstellung ist *True*, die Kommunikation ist erlaubt.

Beispiel:

```
'Kommunikation vom Host (beim nächsten Laden) deaktivieren
WV.IsWebMessageEnabled = False
WV.Reload 'Die Kommunikation ist jetzt deaktiviert
```

Anmerkung

Sie müssen diese Option vor dem Laden einer Seite festlegen.

IsZoomControlEnabled

```
Property IsZoomControlEnabled As Boolean
```

Legt fest, ob der Benutzer den Zoom des WebView2 beeinflussen kann. Die Standardeinstellung ist *True*, das Zoomen ist erlaubt.

Beispiel:

```
'Das Zoomen mittels Mausrad und Tastaturkürzel
'ist nach dem Aufruf der Seite deaktiviert
WV.IsZoomControlEnabled = False
WV.Navigate https://www.selfhtml.org
```

Anmerkung

Sie müssen diese Option vor dem Laden einer Seite festlegen. Davon unabhängig können Sie den Zoom einer Seite über die Eigenschaft *ZoomFactor* ermitteln oder ändern.

jsCallTimeOutSeconds

```
Property jsCallTimeOutSeconds As Double
```

Legt fest, wie lange ein Aufruf einer JavaScript-Funktion bis zum Auftreten eines TimeOuts laufen darf. Die Standardeinstellung beträgt 3 Sekunden.

Beispiel:

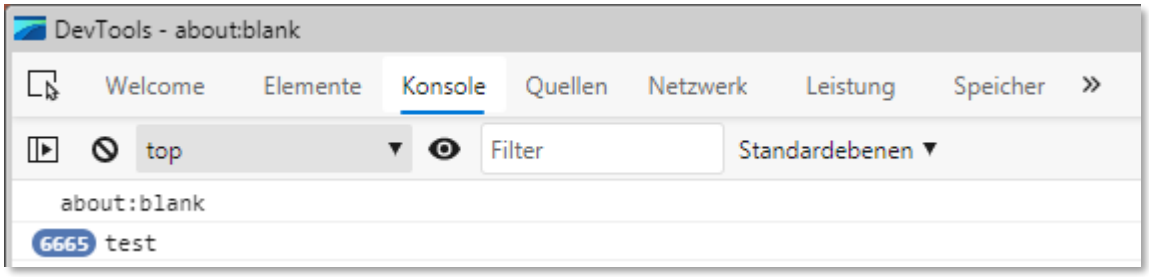
```
WV.AddScriptToExecuteOnDocumentCreated _
   "function endless(){while(true){console.log('test');}}"
...
Debug.Print Second(Now)
Debug.Print WV.jsRun("endless")
```

```
Debug.Print Second(Now)
Debug.Print WV.jsLastError
```

Ausgabe im Direktfenster:

```
45
48
TimeOut, waiting for Result
```

Die JavaScript-Funktion endless beinhaltet eine Endsolschleife, die mittels Funktion jsRun gestartet wird. Wie die Ausgabe zeigt, wartet unser VB-Programm tatsächlich nach dem Start die voreingestellten TimeOut-Sekunden, bevor die nächste Now-Ausgabe erfolgt. Leider ist die Sache jedoch etwas komplexer. Die interne Endlosschleife wird beim Erreichen des TimeOuts nicht beendet, sondern weiter ausgeführt. Wenn Sie vor dem Start der Schleife ein Konsolenfenster starten, können Sie den Verlauf der Ausgabe verfolgen. Sie dürfen sich also nicht bedenkenlos auf diese Eigenschaft verlassen. Vielleicht wird eine spätere Version dieses Problem beheben, aktuell müssen Sie Ihre JavaScript-Funktionen genau prüfen.



Es dürfte auch nicht überraschen, dass diese Konstellation eine hohe CPU-Auslastung im WebView2-Prozess auslöst. Während meiner Tests musste ich sogar nach dem Beenden der Anwendung die msedgewebview2.exe manuell über den Taskmanager beenden.

jsLastError

```
Property jsLastError As String
```

Sollte in einer JavaScript-Anweisung ein Fehler auftreten, kann der Fehlertext über diese schreibgeschützte Eigenschaft abgerufen werden. Bitte beachten Sie, dass die Fehlermeldungen unabhängig von der in der BindTo-Methode angegebenen Sprach-Parameter immer in englischer Sprache erfolgen.

Beispiel:

```
WV.ExecuteScript "document.getElementById('foo').style.color='blue';"
Debug.Print WV.jsLastError
'Ausgabe: Cannot read property 'style' of null (was darauf hindeutet,
'dass das Element 'foo' im Html-DOM nicht gefunden wurde.
```

jsProp

```
Property jsProp(PropName As String)
```

Liest und schreibt DOM-Attribute, die per JavaScript gelesen oder gesetzt werden.

Parameter	Beschreibung
PropName	Eigenschaftsname, die sich aus einem JavaScript-Ausdruck ergibt. Wenn dieser Ausdruck ungültig ist, weil z.B. die Eigenschaft eines

	nicht existierenden Objektes gelesen wird, ist der Rückgabewert ein Variant/Empty. Um dem vorzubeugen können Sie z.B. mit *typeof* vorab testen, ob der JavaScript-Ausdruck ein gültiges Objekt liefert. Achtung: Auch gültige Rückgaben können in VB auffangbare Fehler erzeugen, wenn die Datentypen nicht bei der Zuweisung nicht übereinstimmen.

Beispiel:

```
Dim value As String
If TypeOf WV.jsProp("document.getElementById('txtName')") Is Object Then
  value = WV.jsProp("document.getElementById('txtName').value")
  value = UCase(value)
  WV.jsProp("document.getElementById('txtName').value") = value
  WV.jsProp("document.getElementById('txtName').style.color") = "red"
End If
```

ZoomFactor

```
Property ZoomFactor As Double
```

Liest oder setzt den Zoomfaktor für den WebView2. Der Standardwert ist 1. Beachten Sie, dass unabhängig von diesem Wert der Benutzer den Zoom einer Seite mittels Tastatur und Maus ändern kann, sofern die Eigenschaft *IsZoomControl* das nicht verhindert. Das Angeben eines ZoomFactors kleiner oder gleich 0 ist nicht erlaubt. Eine Änderung des Zoomfaktors kann dazu führen, dass sich *window.innerWidth* oder *window.innerHeight* und das Seitenlayout ändern.

Beispiel:

```
WV.ZoomFactor = 1 'Setzt den Zoom auf die Standardeinstellung zurück.
                  'Gleich mit der Benutzereigabe Strg + 0
```

Methoden

AddObject

```
Sub AddObject(ObjName As String, Object)
```

Beschreibung

AddScriptToExecuteOnDocumentCreated

```
Sub AddScriptToExecuteOnDocumentCreated(jsCode As String)
```

Mit dieser Methode können Sie eine globale JavaScript-Funktion vordefinieren, die in jedes geladene Dokument eingebunden wird.

Parameter	Beschreibung
jsCode	String mit gültigen JavaScript-Code, der eine Funktion darstellt. Diese Funktion können Sie aus sonstigen JavaScript-Code aufrufen oder als Eventhandler für eine beliebige DOM-Komponente verwenden. Außerhalb des Dokumentes können Sie die WebView2-Methoden *jsRun* oder *jsRunAsync* zum Aufruf dieser Funktion verwenden. Sie können mehrere Funktionen gleichzeitig in einem Aufruf definieren oder die Methode mehrfach hintereinander aufrufen.

Anmerkung: Beachten Sie den globalen Charakter dieser Funktionen. Es könnte jederzeit zu Namenskonflikten zwischen den vordefinierten Funktionen und den im Dokument definierten Funktionen kommen.

Beispiel 1: Zwei JavaScript-Funktionen werden in einem Aufruf geladen.

```
WV.AddScriptToExecuteOnDocumentCreated _
    "function add(a,b){return a+b} " & _
    "function sub(a,b){return a-b}"

WV.NavigateToString "<html><head></head><body></body></html>"

Debug.Print WV.jsRun("add", 2, 3) 'Ausgabe: 5
Debug.Print WV.jsRun("sub", 2, 3) 'Ausgabe: -1
```

Beispiel 3: Eine anonyme JavaScript IIFE-Funktion wird geladen. Die Funktion wird unmittelbar bei jedem Laden einer Seite aufgerufen (IIFE: unter Verwendung des Funktions-Scopings wird das Variablen-Hopping innerhalb von Blöcken vermeiden).

```
WV.AddScriptToExecuteOnDocumentCreated "(function(){" & _
    "let x = window.location.href;" & _
    "console.log(x)})();"
```

Wenn Sie Ihre HTML-Dokumente dynamisch aufbauen, den body also über DOM-Methoden nach und nach ergänzen, würde es sich anbieten, an dieser Stelle eine globale Funktion dafür zu implementieren.

Beispiel 4:

```
WV.AddScriptToExecuteOnDocumentCreated _
  function newElement(element,html){" & _
    "let e = document.createElement(element);" & _
    "e.innerHTML=html;"
    "document.body.appendChild(e);}"
```

AddWebResourceRequestedFilter

```
Sub AddWebResourceRequestedFilter(sFilter As String, FilterContext As
eWebView2ResourceFilter)
```

Fügt dem Ereignis *WebResourceRequested* einen URI- und Ressourcenkontextfilter hinzu. Der URI-Parameter kann auf eine Wildcard-Zeichenkette gesetzt werden ('*': kein oder viele Zeichen, '?': genau ein Zeichen).

Parameter	Beschreibung
sFilter	
FilterContext	Gibt die Kontexte der Webressourcenanforderung als eWebView2ResourceFilter Enumeration an. Mögliche Werte:
Filter_ALL	0 - Spezifiziert alle Ressourcen.
Filter_CSP_VIOLATION_REPORT	15 - Gibt einen CSP-Verletzungsbericht an.
Filter_DOCUMENT	1 - Gibt eine Dokument-Ressource an.
Filter_EVENT_SOURCE	10 - Gibt eine EventSource-API-Kommunikation an.
Filter_FETCH	8 - Gibt eine Fetch-API-Kommunikation an.
Filter_FONT	6 - Gibt eine Schriftart-Ressource an.
Filter_IMAGE	2 - Gibt eine Bildressource an.
Filter_MANIFEST	12 - Gibt ein Web-App-Manifest an.

Filter_MEDIA	4 - Gibt eine Medienressource an, z. B. ein Video.
Filter_OTHER	16 - Gibt eine andere Ressource an.
Filter_PING	14 - Gibt eine Ping-Anfrage an.
Filter_SCRIPT	6 - Gibt eine Skript-Ressource an.
Filter_SIGNED_EXCHANGE	13 - Gibt einen signierten HTTP-Austausch an.
Filter_STYLESHEET	2 - Gibt eine CSS-Ressource an.
Filter_TEXT_TRACK	9 - Gibt eine TextTrack-Ressource an.
Filter_WEBSOCKET	11 - Gibt eine WebSocket-API-Kommunikation an.
Filter_XML_HTTP_REQUEST	7 - Gibt eine XML-HTTP-Anfrage an.

Beispiel:

```
WV.Navigate "https://wiki.selfhtml.org"
WV.AddWebResourceRequestedFilter "*", Filter_ALL
WV.Navigate "https://www.w3schools.com"
'im Browser wird kurz die selfhtml-Seite angezeigt
'danach kommt die Meldung „...diese Seite ist leider nicht erreichbar
'in den DevTools sehen Sie folgende Meldung:
'Failed to load resource: the server responded with a status of 403
```

BindTo

```
Function BindTo(HosthWnd As Long, [SecondsToWaitForInitComplete As Double = 8], [browserInstallPath As String], [userDataFolder As String], [additionalBrowserArguments As String]) As Long
```

Bindet die WebView2-Komponente an ein GUI-Objekt (Fenster oder PictureBox). Damit das möglich ist, muss das GUI-Objekt sichtbar sein.

Parameter	**Beschreibung**
HosthWnd	Zugriffsnummer für ein Fenster oder ein Steuerelement (z.B. PictureBox) in dem die WebView2-Komponente gehostet wird.
SecondsToWaitForInitComplete	Optionaler Parameter. Maximale Anzahl der Sekunden für die Initialisierung der WebView2-Komponente. Nach Ablauf dieser Zeit liefert die Methode den Wert 0, die Bindung gilt damit als erfolglos.
browserInstallPath	Optionaler Parameter für einen Pfad in dem die WebView2-Runtime installiert ist.
userDataFolder	Optionaler Parameter für das Arbeitsverzeichnis der WebView2-Komponente. Hier werden temporäre Daten angelegt.
additionalBrowserArguments	Optionale Zusatzparameter, die das Aussehen und Verhalten der WebView2-Komponente beeinflussen, können hier angegeben werden. Die Parameter werden im Format `--name=wert` angegeben. Beispiel: `--lang=de` ändert die Komponenten-Sprache von Standard-Englisch nach Deutsch.

Anmerkungen

Falls Sie mehrere Instanzen der WebWiew2-Komponente verwenden, dürfen sich manche Argumente pro Instanz nicht unterscheiden, sonst scheitert die BindTo-Methode. Achtung: diese Einstellung gilt sogar systemweit. Sie können das unterbinden, indem Sie im Parameter *userDataFolder* einen eigenen Pfad für Ihre Anwendung angeben. In diesem Pfad muss Ihre Anwendung schreibrechte haben. Sie müssen allerdings den Pfad nicht selbst anlegen, WebView2 erledigt das automatisch wärend der

Initialisierung. In den Ordner werden weitere Ordner und Dateien angelegt, in Summe ca. 20 MB. Je nach Rechnergeschwindigkeit kann sich damit der erste Programm-Start leicht verzögern. Falls Ihr Programm eine Install-/Uninstall Funktion verwendet, wäre es sinnvoll bei der Deinstallation auch diese Pfade zu bereinigen.

Die Argumente werden als Teil des Befehls an den Browser-Prozess übergeben. Bestimmte Funktionen sind intern deaktiviert und für die Aktivierung gesperrt. Wenn ein Schalter mehrfach angegeben wird, wird nur die letzte Instanz verwendet. Eine Zusammenführung der verschiedenen Werte desselben Schalters wird nicht versucht, außer bei deaktivierten und aktivierten Funktionen.

Eine einfache Liste der verfügbaren Befehlszeilen-Schalter finden Sie im Internet, z.B. auf folgender Seite: https://peter.sh/experiments/chromium-command-line-switches

Beispiel 1: Die WebView2-Komponente wird an eine PictureBox gebunden.

```
Set WV = New_c.WebView2 'erzeugt eine neue WebView2-Instanz
ret = WV.BindTo(picWV.hWnd, , , , "--lang=de")
If ret = 0 Then
    MsgBox "Das WebView-Binding konnte nicht initialisiert werden!"
End If
```

Anmerkungen: Standardmässig verwendet die Bindung automatisch die aktuelle Runtime (Evergreen). Diese befindet sich standardmäßig auf der lokalen Festplatte, im Programme-Verzeichnis unter \Microsoft\EdgeWebView\Application\xx.x.xxxx.xx\. Dieses Verzeichnis kann mit der Funktion *GetMostRecentInstallPath* ermittelt werden. Wenn Sie eine alternative Runtime verwenden wollen (Microsoft spricht in diesem Fall von einer „fixed version“), können Sie im Parameter *browserInstallPath* den Pfad zu dieser Version angeben. Bitte beachten Sie, dass sich dieser Pfad nicht auf einem Netz-Pfad befinden darf. Lokale Festplatten, CD- oder DVD-Laufwerke oder USB-Sticks funktionieren.

CallDevToolsProtocolMethod

```
Sub CallDevToolsProtocolMethod(MethodName As String, ParamsAsJSON As
String)
```

Führt eine DevToolsProtocol-Methode aus.

Parameter	Beschreibung
MethodName	Der vollständige Name der auszuführenden Methode.
ParamsAsJSON	Eine JSON-formatierte Zeichenfolge, die die Parameter für die entsprechende Methode enthält.

Beispiel:

```
WV.CallDevToolsProtocolMethod _
    "Runtime.evaluate", _
    "{""expression"":""alert('Testausgabe')""}"
'Im WebView erscheint die Meldung: Testausgabe
```

CapturePreview

```
Function CapturePreview(ImageFormat As eWebView2ImageCaptureFormat) As
cCairoSurface
```

Erfasst ein Bild von dem, was im WebView anzeigt wird. Im Parameter gben Sie das Format des zu erfassenden Bildes an. Die Ausgabe erfolgt in einer Cairo-Image-Surface und kann von hier aus weiterverarbeitet werden.

Parameter für ImageFormat	Beschreibung
CaptureAs_PNG	0 - das PNG-Bildformat wird verwendet
CaptureAs_JPG	1 - das JPG-Bildformat wird verwendet

Beispiel:

```
WV.CapturePreview(CaptureAs_PNG).WriteContentToPngFile "C:\WVCapture.png"
```

ExecuteScript

```
Sub ExecuteScript(jsCode As String)
```

Führt den im Parameter übergebenen JavaScript-Code aus. Diese Methode funktioniert auch, wenn die *IsScriptEnabled*-Eigenschaft ausgeschaltet ist. Da die Methode keinen Rückgabewert liefert, können Sie hiermit JavaScript-Rückgaben nicht auswerten. Die Methode bietet sich zum Beispiel an, wenn mehrere DOM-Element-Eigenschaften geändert werden sollen.

Beispiel:

```
'Schlecht:
WV.jsProp("document.getElementById('btn1').style.color") = "red"
WV.jsProp("document.getElementById('btn2').style.color") = "red"
WV.jsProp("document.getElementById('btn3').style.color") = "red"
'Besser:
WV.ExecuteScript _
    "document.getElementById('btn1').style.color='red';" & _
    "document.getElementById('btn2').style.color='red';" & _
    "document.getElementById('btn3').style.color='red';"
```

GetMostRecentInstallPath

```
Function GetMostRecentInstallPath([EdgeChannel As String = "EdgeWebView"],
[VersionString As String]) As String
```

Beschreibung

GoBack

```
Sub GoBack()
```

Navigiert zur vorherigen Seite in der Navigationshistorie.

GoForward

```
Sub GoForward()
```

Navigiert zur nächsten Seite in der Navigationshistorie.

jsRun

```
Function jsRun(FuncName As String, ParamArray P() As Variant)
```

Führt eine JavaScript-Funktion aus. Die Funktion muss auf Dokument-Ebene erreichbar sein. Der Rückgabe-Wert ist ein Variant.

Parameter	Beschreibung
FuncName	JavaScript-Funktion. Auf Groß- und Kleinschreibung achten!
P()	Parameter-Array mit den Parametern für die JavaScript-Funktion

Anmerkungen:
JavaScripts sind viel toleranter was die Funktions-Parameter angeht. Zum einem ist JavaScript nicht so tyisiert wie VB, zum anderem sind das was ein VB-Entwickler als optionalen Parameter explizit definiert, in JavaScript-Funktionen per Default immer dabei.

Beispiel 1:

```
WV.AddScriptToExecuteOnDocumentCreated _
    "function sub(a,b){return a-b} " & _
    "function sum(a,b){return a-b}"

WV.NavigateToString "<html><head></head><body></body></html>"

Debug.Print WV.jsRun("sub", 2)       'Ausgabe:
Debug.Print WV.jsLastError
Debug.Print WV.jsRun("add", 2, 3, 4) 'Ausgabe: 5
```

Im ersten Aufruf übergeben wir einen Parameter zu wenig. Die Funktion wird dennoch ausgeführt, das Ergebnis ist jedoch ein Variant/Empty. Es gibt keinen Laufzeitfehler und auch die Eigenschaft *jsLastError* zeigt nach dem Aufruf keine Auffälligkeiten (Achtung: Fehlerquelle)!

Im zweiten Aufruf haben wir drei statt zwei Parameter übergeben. Die Funktion wird trotzdem (richtig) ausgeführt (Achtung: Fehlerquelle). Dier dritte Parameter, die Zahl 5 wird dabei ignoriert. Was aber nicht heißt, dass dieser Parameter nicht an die Funktion übergeben wird. Die Parameter landen im sogenannten *arguments*-Objekt. Das ist ein Array-ähnliches Objekt, das auf die übergebenen Parameter einer Funktion verweist. Das kann man sich auch zunutze machen, wie im folgeden Beispiel zu sehen ist.

Beispiel 2:

```
WV.AddScriptToExecuteOnDocumentCreated _
  "function concat(){" & _
  "return Array.prototype.slice.call(arguments, 0).join(' ')}"
WV.NavigateToString "<html><head></head><body></body></html>"
Debug.Print WV.jsRun("concat", "Das", "ist", "ein", "Test!")
'Ausgabe: Das ist ein Test!
'Und nicht darauf reinfallen:
Debug.Print WV.jsRun("concat", Array("Das", "ist", "ein", "Test!"))
'Ausgabe: Das,ist,ein,Test!
```

jsRunAsync

```
Function jsRunAsync(FuncName As String, ParamArray P() As Variant) As
Currency
```

Führt eine JavaScript-Funktion asynchron aus. Die Funktion muss auf Dokument-Ebene erreichbar sein. Der Rückgabe-Wert ist eine Token-ID. Was die Parameterübergabe betrifft, gelten die gleichen Bedingungen wie bei der Mehode *jsRun*.

Parameter	Beschreibung
FuncName	JavaScript-Funktion. Auf Groß- und Kleinschreibung achten!
P()	Parameter-Array mit den Parametern für die JavaScript-Funktion

Beispiel 1:

```
WV.AddScriptToExecuteOnDocumentCreated _
    "function sub(a,b){return a-b} " & _
    "function sum(a,b){return a-b}"

WV.NavigateToString "<html><head></head><body></body></html>"
Debug.Print "RunAsync:" & WV.jsRunAsync("sum", 3, 4)
Debug.Print "NavigateToString"

Private Sub WV_JSAsyncResult(Result As Variant, _
  ByVal Token As Currency, ByVal ErrString As String)
  Debug.Print Token & ":" & Result
End Sub

Ausgabe:
RunAsync:3
NavigateToString
3:7
```

Beachten Sie, dass die Rückgabe der asynchronen Funktion in einer eigenen VB-Ereignisprozedur *JSAsyncResult* erfolgt. Diese Prozedur liefert Ihnen neben dem Ergebnis der Funktion auch den Token, den Sie als Rückgabe beim Aufruf bekommen haben. Hier im Beispiel war das eine 3, kann bei Ihnen natürlich eine ganz andere Zahl sein. Auf jeden Fall hilft Ihnen dieser Token das Ergebnis dem passenden Funktionsaufruf zuzuordnen. In der Ausgabe ist auch der Debugg-Aufruf NavigateToString zu sehen. Dieser erfolgt noch vor dem Ereignis *JSAsyncResult,* was der Asynchronität der Methode geschuldet ist.

Nun sollte man annehmen, dass JavaScript-Funktionen beliebig asynchron aufgerufen und über die Tokens die Ergebnisse zugeordnet werden können. Dem ist leider nicht so. Test mit der Version 90.0.818.42 und älter haben gezeigt, dass diese Aufrufe alles andere als zuverlässig sind. Folgender Code funktioniert zum Beispiel nicht korrekt:

Beispiel 2:

```
WV.AddScriptToExecuteOnDocumentCreated _
    "function sub(a,b){return a-b} " & _
    "function sum(a,b){return a-b}"

WV.NavigateToString "<html><head></head><body></body></html>"
Debug.Print "RunAsync:" & WV.jsRunAsync("sum", 3, 4)
Debug.Print "RunAsync:" & WV.jsRunAsync("sub", 3, 4)

Private Sub WV_JSAsyncResult(Result As Variant, _
  ByVal Token As Currency, ByVal ErrString As String)
  Debug.Print Token & ":" & Result
End Sub

Ausgabe:
RunAsync:3
RunAsync:4
4:-1
```

Beide Funktionen werden mit verschiedenen Tokens ausgeführt, in der JSAsyncResult-Prozedur kommt jedoch nur das Ergebnis des zweiten Tokens an. Und selbst folgender Aufruf bringt kein befriedigendes Ergebnis, die Ausgabe der asynchronen Funktion (Token 3) fehlt:

Beispiel 3:

```
Debug.Print "RunAsync:" & WV.jsRunAsync("sum", 3, 4)
Debug.Print "Run:" & WV.jsRun("sub", 3, 4)

Ausgabe:
RunAsync:3
Run:-1
```

Dennoch kann diese Methode Ihnen gute (asynchrone) Dienste leisten, wie folgendes Beispiel zeigt:

Beispiel 4:

```
Public counter As Integer
...

Debug.Print "RunAsync:" & WV.jsRunAsync("sum", 3, 4)
For counter = 0 To 32000
  DoEvents
Next
...

Private Sub WV_JSAsyncResult(Result As Variant,
  ByVal Token As Currency, ByVal ErrString As String)
  Debug.Print Token & ":" & Result; ErrString
  Debug.Print "Counter:" & counter
End Sub

Ausgabe:
RunAsync:3
Counter 3144
```

Zuerst wird die JavaScript-Funktion *sum* asynchron aufgerufen. Im Anschluss folgt gleich eine For-Schleife, die lediglich einen Counter hochzählt. Bitte beachten Sie, dass dieser Counter zumindest Modulweite Gültigkeit haben muss. *DoEvents* sorgt dafür, dass andere Ereignisse verarbeitet werden, hier in unserem Fall die Rückgabe der asynchronen JavaScript Funktion. Der Wert 3144 ist zwar kein Zufallswert, kann aber bei jeder Ausführung ein anderer sein. Immerhin ist so die Parallelverarbeitung zwei unterschiedlicher Threads hiermit belegt.

MoveRelativeToHostWindow

```
Sub MoveRelativeToHostWindow(x, y, dx, dy)
```

Bestimmt die Positionierung und die Abmessungen des WebViews.

Parameter	Beschreibung
x	Erforderlich. Wert der die horizontale Koordinate (x-Achse) in Pixel für den linken Rand.
y	Erforderlich. Wert der die vertikale Koordinate (y-Achse) in Pixel für den Rand von oben.
dx	Erforderlich. Wert der die Breite in Pixel bestimmt.
dy	Erforderlich. Wert der die Höhe in Pixel bestimmt.

Anmerkungen

Es spricht nichts dagegen, den WebView2 wie ein normales Steuerelement auf einem Parent-Fenster zu verwenden. Man kann ihn auch mit anderen Steuerelementen kombinieren. Dafür muss der

WebView2 nicht unbedingt in einem Bildfeld-Steuerelement und auch nicht in einem Benutzer-Steuerelement gehostet werden. Mit der Methode *MoveRelativeToHostWindow* verschieben Sie seine Koordinaten relativ zum Ursprung (0, 0) des Containers, d.h. zur oberen linken Ecke. Die beiden Argumente dx und dy sorgen für die Breite und Höhe. Es gibt jedoch einen gravierenden Unterschied: Der WebView2 ist kein klassisches Steuerelement. Wenn Sie ihn in einem Form-Objekt oder einem Bildfeld-Steuerelement (*PictureBox*) oder in einer untergeordneten MDI-Form eines *MDIForm*-Objekts verschieben, wird das Koordinatensystem des Container-Objekts NICHT verwendet. Die Argumente werden immer in Pixel angegeben! Es gibt auch keine *ZOrder*-Positionierung und in die Aktivierungsreihenfolge (*TabIndex*) können Sie ihn auch nicht ohne Weiteres integrieren. Das UI des WebWie2 wird auf seinen Host, wie eine Projektion auf eine Leinwand übertragen.

Navigate

```
Function Navigate(URI As String, [SecondsToWaitForDocumentComplete As
Double = 8])
```

Bewirkt, dass eine Navigation des Top-Level-Dokuments zum angegebenen URI eingeläutet wird.

Parameter	Beschreibung
URI	Internet-Ressource (i.d.R. eine http- oder https-Adresse)
SecondsToWaitForDocumentComplete	Optionaler TimeOut für die Funktion. Der Standardwert ist 8 Sekunden. Der Aufruf erfolgt synchron. Das heißt, dass nach dem Aufruf die DOM-Elemente sofort verfügbar sind.

Beispiel:

```
WV.Navigate "https://wiki.selfhtml.org"
Debug.Print WV.DocumentTitle 'SELFHTML-Wiki
```

NavigateToString

```
Function NavigateToString(sHTMLContent As String,
[SecondsToWaitForDocumentComplete As Double = 8])
```

Leitet eine Navigation zu htmlContent als Quell-HTML eines neuen Dokuments ein.

Parameter	Beschreibung
sHTMLContent	HTML-Code, der in der Gesamtgröße nicht größer als 2 MB sein. Der Code darf auch Meta-, Script- und Syle-Abschnitte enthalten. Der Ursprung der neuen Seite lautet *about:blank*.
SecondsToWaitForDocumentComplete	Optionaler TimeOut für die Funktion. Der Standardwert ist 8 Sekunden. Der Aufruf erfolgt synchron. Das heißt, dass nach dem Aufruf die DOM-Elemente sofort verfügbar sind.

Beispiel:

```
WV.NavigateToString _
  "<!DOCTYPE html><html><head></head><body>" & _
  "<div>Hello World...</div>" & _
  "</body></html>",3
Debug.Print "NavigateToString completed"
```

Nach dem Ausführen von *NavigateToString* wird das Ereignise *NavigationCompleted* ausgelöst. Später dann auch *DocumentComplete*. Nach dem Aufruf der Methode können Sie anschließend gleich auf die

DOM-Elemente mit den Methoden *ExecuteScript*, *jsRun* und *jsRunAsync* zugreifen. Sollte jedoch irgendwas in der Runtime die Ausführung von *NavigateToString* verzögern, setzt VB die Ausführung nach den im zweiten Parameter festgelegten Sekunden fort. Das lässt sich schön beobachten, wenn Sie im Ereignis *NavigationCompleted* auch eine Debugg-Ausgabe einbauen:

```
Private Sub WV_NavigationCompleted( _
    ByVal IsSuccess As Boolean, ByVal WebErrorStatus As Long)
      Debug.Print "WV_NavigationCompleted"
End Sub
```

Die Ausgabe im VB-Direktfenster ist wie folgt:

```
NavigationCompleted
NavigateToString completed
```

Nun sorgen wir dafür, dass *NavigateToString* länger als die 3 Sekunden dauert, indem wir ein kleines JavaScipt einbauen:

```
WV.NavigateToString _
  "<!DOCTYPE html><html><head></head><body>" & _
  "<div>Hello World...</div>" & _
  "<script>alert('ScriptExecutedOnNavigateToString')</script>" & _
  "</body></html>",3
Debug.Print "NavigateToString completed"
```

Beim erneuten Ausführen, erscheint zunächst die *Alert*-Meldung. Wenn wir nun etwas länger als die 3 Sekunden warten und danach erst auf *Ok* klicken, erwartet uns folgende Ausgabe im Direktfenster:

```
NavigateToString completed
NavigationCompleted
```

Wenn Sie nach dem Ausführen von *NavigateToString* sofort das DOM ansprechen, kann es sein, dass bei einer kleinen *SecondsToWaitForDocumentComplete* -Einstellung noch nicht alle DOM-Elemente geladen sind. Ein Zugriff auf diese Elemente empfielt sich deshalb erst nach dem Event *NavigationCompleted* oder noch besser nach *DocumentComplete*.

OpenDevToolsWindow

```
Sub OpenDevToolsWindow()
```

Öffnet das DevTools-Fenster für das aktuelle Dokument im WebView. Wenn das DevTools-Fenster bereits geöffnet ist, wird es in den Vordergrund geholt, ggf. aus der Taskleiste wiederhergestellt. Wenn Sie den Container, der den WebView hostet schließen, wird auch das DevTools-Fenster geschlossen. Details über die Verwendung der DevTools finden Sie unter:

https://docs.microsoft.com/de-de/microsoft-edge/devtools-guide-chromium

RaiseMessageEvent

```
Sub RaiseMessageEvent(sMsg, sMsgContent)
```

Beschreibung

RaiseResultEvent

```
Sub RaiseResultEvent(Res, Token, ErrMsg)
```

Beschreibung

Reload

```
Sub Reload()
```

Die aktuelle Seite wird neu geladen. Das gilt auch für HTML-Code, der über die Methode *NavigateToString* in den WebView2 geladen wurde. Dies ist vergleichbar mit der Navigation zum URI des aktuellen Dokuments der obersten Ebene, einschließlich aller Navigationsereignisse, die ausgelöst werden, und unter Berücksichtigung aller Einträge im HTTP-Cache. Der Rück- oder Vorwärtsverlauf wird damit nicht geändert.

RemoveObject

```
Sub RemoveObject(ObjName As String)
```

Beschreibung

RemoveWebResourceRequestedFilter

```
Sub RemoveWebResourceRequestedFilter(sFilter As String, FilterContext As
eWebView2ResourceFilter)
```

Entfernt einen WebResource-Filter, der zuvor für das WebResourceRequested hinzugefügt wurde. Siehe *AddWebResourceRequestedFilter*.

SetFocus

```
Sub SetFocus([Reason As eWebView2FocusReason])
```

Setzt den Focus in die WebView2-Komponente. Über den optionalen Parameter *Reason* können Sie das Focus-Ziel innerhalb der Komponente bestimmen. Sollte das Focus-Ziel nicht erreichbar sein (z.B. Host-Fenster ist minimiert oder ausgeblendet), kommt es zu einen auffangbaren Laufzeitfehler (5) in der Anwendung.

Werte für Reason	**Beschreibung**
FocusReason_PROGRAMMATIC	0 – WebWie2 bekommt den Focus. Sollte bei Focus-Verlust irgendein HTML-Form-Element den Focus haben, dann wird dieses Element wieder den Focus bekommen.
FocusReason_NEXT	1 - WebWie2 bekommt den Focus. Sollten im Dokument HTML-Form-Elemente enthalten sein, dann wird das erste Element in der Tab-Reihenfolge den Focus bekommen, sofern es nicht deaktiviert ist.
FocusReason_PREVIOUS	2 - WebWie2 bekommt den Focus. Sollten im Dokument HTML-Form-Elemente enthalten sein, dann wird das letzte Element in der Tab-Reihenfolge den Focus bekommen, sofern es nicht deaktiviert ist.

SetFuncObj

```
Sub SetFuncObj(FuncObj)
```

Beschreibung

SyncSizeToHostWindow

```
Sub SyncSizeToHostWindow()
```

Synchronisiert die Abmessungen des WebView2 mit seinem Host. Die Links- und Oben-Koordinaten werden dabei auf 0 festgelegt.

Beispiel:

```
Private Sub Form_Resize()
  If Not WV Is Nothing Then WV.SyncSizeToHostWindow
End Sub
```

Ereignisse

AcceleratorKeyPressed

```
Event AcceleratorKeyPressed(KeyState As eWebView2AccKeyState, IsExtendedKey
As Boolean, WasKeyDown As Boolean, IsKeyReleased As Boolean, IsMenuKeyDown
As Boolean, RepeatCount As Long, ScanCode As Long, IsHandled As Boolean)
```

AcceleratorKeyPressed wird ausgelöst, wenn der Anwender eine „Beschleunigungstaste“ gedrückt hat. Wenn die gedrückte Taste länger gehalten wird, wird das Ereignis öfter, mit zum Teil anderen Argument-Werten ausgelöst.

Argumente	**Beschreibung**
KeyState (eWebView2AccKeyState)	Ereignistyp: Key_DOWN (0) – Teste gedrückt Key_UP (1) – Taste loßgelassen SysKey_DOWN (2) – Systemtaste gedrückt (z.B. Alt-Taste) SysKey_UP (3) – Systemtaste loßgelassen (z.B. Alt-Taste)
IsExtendedKey	Wahr bei Sondertasten wie die Win- oder Menü-Taste, Strg-Taste, Pleil- und Bild-Tasten usw.
WasKeyDown	Zeigt ob die Taste vor dem Ereignis gerückt war (True) oder nicht (False). Beim Drücken der Taste ist dieses Argument noch False. Wenn Sie die Taste länger gedrückt halten, wird das Ereignis erneut ausgelöst, diesmal ist dieses Argument True. Beim Loßlassen der Taste ist der Wert Wahr
IsKeyReleased	Nach dem Loslassen einer Taste ist im letztem Ereignis dieses Argument True, ansonsten immer False.
IsMenuKeyDown	
RepeatCount	In der aktuellen Version immer der Wert 1
ScanCode	Tastencode für die gedrückte Taste. Achtung: diese entsprechen NICHT den *KeyCodes* (*KeyCodeConstants*) von Visual Basic.
IsHandled	Gibt an, ob das Ereignis *AcceleratorKeyPressed* behandelt wird. Der Standardwert ist True. Da dieses Argument *ByRef* übergeben wird, können sie es ändern. Das hat zur Folge, dass bestimmte WebWiew2-Funktionstasten von der Komponente nicht mehr ausgewertet werden. Damit können Sie z.B. verhindern, dass mittels F3 der interne WebWiew2-Suchdialog geöffnet wird.

Anmerkungen

Die *SystemKeys* werden immer dann in *KeyState* angezeigt, wenn die die Art des Tastenereignisses der Fenstermeldung *WM_SYSKEYDOWN* bzw. *WM_SYSKEYUP* entspricht. Die anderen Funktionstasten ergeben sich aus den Fenstermeldungen *WM_KEYDOWN* und *WM_KEYUP*.

Beachten Sie, dass die VB-Eigenen Form- und Steuerelement-Ereignisse *KeyUp*, *KeyPress* und *KeyDown* nicht ausgelöst werden, wenn der WebWiew2 den Focus hat. Auch nicht, wenn Sie *IsHandled* auf True setzen und auch nicht, wenn im Fenster die Eigenschaft *KeyPreview* den Wert True hat.

ScanCodes

Taste	ScanCode	Taste	ScanCode	Taste	ScanCode
F1	59	Esc	1	Druck	55
F2	60	FST*	58	Rollen	70
F3	61	Strg	29	Pause	69
F4	62	WinL	91	Einfg	82
F5	63	Alt	56	Entf	83
F6	64	AltGr	29/56	Pos1	71
F7	65	WinR	92	Ende	79
F8	66	Menu	93	Bild O	73
F9	67	Pfeil O	72	Bild U	81
F10	68	Pfeil L	75		
F11	69	Pfeil R	77		
F12	70	Pfeil U	80		

*FST=Feststelltaste

DocumentComplete

```
Event DocumentComplete()
```

Das Ereignis wird ausgelöst, wenn das Dokument vollständig geladen und angezeigt wird. Das interne Ereignis body.onload() wurde zu diesem Zeitpunkt bereits ausgelöst.

Um die Reihenfolge der Ereignisse besser nachzuvollziehen, beachten Sie folgendes Beispiel. Wir definieren eine globale JavaScript log-Funktion, die wir sowohl aus dem WebWiew2-Innerem aus auch von aussen über die Methode *jsRun* aufrufen können. Mit einem hohen Wert im Parameter SecondsToWaitForDocumentComplete der Methode *NavigateToString* sorgen wir dafür, dass trotz Parallelität der Ausführung (eigener Prozess und msedgewebwiew2.exe) die Reihenfolge der Ereignisse nicht verfälscht werden. Die log-Funktion wird nun an unterschiedlichen Stellen aufgerufen, das Ergebnis ist im VB-Direktfenster und in der DevTools/Konsole zu sehen.

Beispiel:

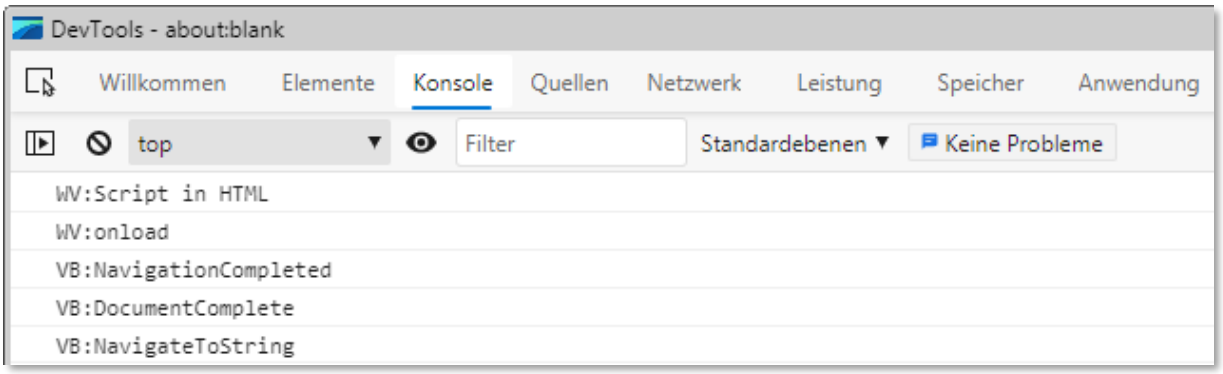
```
Private Sub LocalWebViewInit()
  WV.OpenDevToolsWindow
  DoEvents
  WV.AddScriptToExecuteOnDocumentCreated _
    "function log(msg){console.log(msg)}"
  WV.NavigateToString _
    "<!DOCTYPE html><html><head></head>" & _
    "<body style='border: 1px solid black;' " & _
        "onload='log(""WV:onload"")'>" & _
    "<div id='hello'>Hello World....</div>" & _
    "<script>log('WV:Script in HTML')</script>" & _
    "</body></html>", 2000
  Debug.Print "VB:NavigateToString"
  WV.jsRun "log", "VB:NavigateToString"
End Sub
```

```
Private Sub WV_DocumentComplete()
  Debug.Print "VB:DocumentComplete"
  WV.jsRun "log", "VB:DocumentComplete"
End Sub

Private Sub WV_NavigationCompleted( _
    ByVal IsSuccess As Boolean, ByVal WebErrorStatus As Long)
  Debug.Print "VB:NavigationCompleted"
  WV.jsRun "log", "VB:NavigationCompleted"
End Sub

'Ausgabe im VB-Direktfenster:
VB:NavigationCompleted
VB:DocumentComplete
VB:NavigateToString
```

Ausgabe in der WebWiew2-Konsole:



Beachten Sie die letzte Ausgabe *VB:NavigateToString*. Diese Ausgabe haben wir in der ersten *LocalWebViewInit*-Prozedur definiert, und zwar gleich nach NavigateToString. Zwischendurch wurden jedoch die beiden Ereignisse *NavigationCompleted* und *DocumentComplete* ausgelöst. Darauf dürfen Sie sich aber nicht verlassen. Bei einem kleinen *SecondsToWaitForDocumentComplete*-Parameter, könnte sich diese Reihenfolge verändern. Siehe auch Methode *NavigateToString*.
Jetzt stellt sich noch die Frage: Wann wird das Dokument gezeichnet? Ab wann kann der Anwender was auf dem Bildschirm sehen. Dazu ist Folgendes zu sagen: Auf jedem Fall wird als erstes das leere Fenster zu sehen sein. Wie es witer geht, können wir mit einer kleinen Änderung in der log-Funktion erkennen:

```
  WV.AddScriptToExecuteOnDocumentCreated _
    "function log(msg){alert(msg)}"
```

Statt der Konsole-Ausgabe erzeugen wir eine Meldung. Diese bewirkt, dass die Verarbeitung im WebView2 angehalten wird. In der Beschreibung des JavaScript Events onload steht: „Das Ereignis *onload* tritt ein, wenn ein Objekt geladen wurde.“ Das bestätigt auch unser Versuchsaufbau. Innerhalb des body-Elements wird *onload* verwendet, um ein Skript auszuführen, sobald eine Seite alle Inhalte (einschließlich Bilder, Skriptdateien, CSS-Dateien usw.) vollständig geladen hat.

GotFocus

```
Event GotFocus(Reason As eWebView2FocusReason)
```

Das Event tritt auf, wenn der WebWiew2 den Fokus bekommt.

Parameter für Reason	Beschreibung
FocusReason_PROGRAMMATIC	0 - Gibt an, dass der Code den Fokus in WebView gesetzt hat.

FocusReason_NEXT	1 - Gibt an, dass der Fokus aufgrund von Tabulator-Traversal nach vorne verschoben wird.
FocusReason_PREVIOUS	2 - Gibt an, dass der Fokus aufgrund von Tabulator-Traversal nach hinten verschoben wurde.

InitComplete

```
Event InitComplete()
```

InitComplete wird sofort nach einer erfolgreichen BindTo-Methode ausgelöst. Das soll folgendes Beispiel verdeutlichen. Bevor noch die BindTo-Methode einen erfolgreichen Rückgabewert (1) zurückgibt, wird das Ereignis InitComplete ausgelöst. Hier ist der HosthWnd-Wert aus der BindTo-Methode schon verfügbar.

Beispiel:

```
Private Sub Form_Load()
  Visible = True
  Set WV = New_c.WebView2
  Debug.Print WV.GetMostRecentInstallPath
  Debug.Print WV.BindTo(Me.hWnd)
End Sub

Private Sub WV_InitComplete()
  "WV_InitComplete " & WV.HosthWnd
End Sub

Ausgabe:
C:\Program Files (x86)\Microsoft\EdgeWebView\Application\xx.x.xxx.xx
WV_InitComplete xxxxxx
1
```

JSAsyncResult

```
Event JSAsyncResult(Result, Token As Currency, ErrString As String)
```

Das Ereignis wir in Folge einer asynchron gestarteten JavaScript-Funktion, siehe *jsRunAsync*, ausgelöst.

Argumente	**Beschreibung**
Result	Variant der das Ergebnis der asynchronen Funktion beinhaltet.
Token	Token-ID die von der *jsRunAsync*-Methode zurückgegeben wird. Dieser Token identifiziert so die Aufruf-Methode.
ErrString	Sollte in der JavaScript-Funktion ein Fehler auftreten, erhalten Sie über dieses Argument eine Fehlerbeschreibung. Intern verfügt die Funktion einen *try-catch*-Block, also einen JavaScript-Fehler-behandlung. Der *ErrString* ergibt sich aus der *message* des error-Objektes aus dem *chatch*-Block.

Anmerkungen und Beispiele finden Sie unter der Methode *jsRunAsync*.

JSMessage

```
Event JSMessage(sMsg As String, sMsgContent As String, oJSONContent As
cCollection)
```

Dieses Ereignis können Sie über das interne JavaScript-Objekt vbH auslösen. Die Abkürzung steht für VisualBasic-Host. Dieses Objekt ist sozusagen der Kommunikationskanal aus dem inneren der Seite zum Host, also zu Ihrer Anwendung. Das Objekt bietet die Methode *RaiseMessageEvent*, mit der das Ereignis *JSMessage* ausgelöst werden. Die Ereignisargumente bilden sich über die Aufrufparameter der

JavaScript Methode. Nach diesem Prinzip werden auch die meisten WebView2-Events erzugt, jedoch zunächst vom RC6-WV-Objekt (*cWebView2*-Klasse) aufgefangen und als echte VB-Ereignisse weitergeleitet.

Argumente	**Beschreibung**
sMsg	Variant der das Ergebnis der asynchronen Funktion beinhaltet.
sMsgContent	Wert, der in der JavaScript-Methode *RaiseMessageEvent* als zweiter Parameter übergeben wird.
oJSONContent	Wenn der zweite JavaScript-Parameter (*sMsgContet*-Argument) inhaltlich einem JSON-String entspricht, wird dieser von der cWebWiew2-Klasse in eine cCollection umgewandelt und als Argument im Ereignis übertragen. Wenn nicht, dann ist dieser Parameter *Nothing*.

Beispiel 1

```
WV.NavigateToString _
  "<!DOCTYPE html><html><head></head><body>" & _
  "<button id='btn1' " & _
    "onclick='vbH().RaiseMessageEvent("this.tagName,this.id)'>" & _
    "Button1" & _
  "</button>" & _
"</body></html>"

Private Sub WV_JSMessage( _
    ByVal sMsg As String, _
    ByVal sMsgContent As String, _
    oJSONContent As RC6.cCollection)
  Debug.Print sMsg, sMsgContent
End Sub

'Ausgabe (nach Klick auf Button1):
BUTTON        btn1
```

Ein Klick auf *Button1* löst zunächst mal das JavaScript-Ereignis *onclick* aus. In dessen Ereignishandler wird die Methode *RaiseMessageEvent* des *vbH*-Objektes aufgerufen. Die beiden Parameter können Sie nach Lust und Laune belegen, sie müssen aber angegeben werden. Wenn Sie einen davon nicht brauchen, übergeben Sie eine leere Zeichenfolge. Dieser Methodenaufruf wird vom WV-Objekt anschließend als *JSMessage*-Ereignis an die Anwendung weitergereicht.

Im zweiten Beispiel sehen Sie die interne Implementation des Events *UserContextMenu*.

Beispiel 2:

```
document.addEventListener('contextmenu', function(e) {
  vbH().RaiseMessageEvent('contextmenu', JSON.stringify({
    x:e.screenX,
    y:e.screenY}))
})
```

Auch dieser Methodenaufruf wird vom WV-Objekt empfangen, diesmal aber in das Ereignis *UserContextMenu* umgeleitet. Um beide Maus-Koordinaten weiterleiten zu können, werden diese von der JavaScript-Listener-Funktion zunächst und ein JSON-Objekt verpackt und mittels *stringify* in eine Zeichenkette im JSON-Format umgewandelt.

Die Basis für die *RaiseMessageEvent*-Methode muss aber kein JavaScript-Event sein. Sie können an beliebiger Stelle diese Methode nutzen, um Nachrichten und beliebige Daten an die Host-Anwendung zu senden. Wichtig dabei ist zu verstehen, dass die Initiative zum Nachrichten- und Daten-Transfer von einer JavaScript-Anweisung ausgeht.

LostFocus

```
Event LostFocus(Reason As eWebView2FocusReason)
```

Das Event tritt auf, wenn der WebWiew2 den Fokus verliert. Die *Reason*-Werte finden Sie in der Beschreibung unter *GotFocus*.

NavigationCompleted

```
Event NavigationCompleted(ByVal IsSuccess As Boolean, ByVal WebErrorStatus
As Long)
```

NavigationCompleted wird ausgelöst, wenn die Seite vollständig geladen wurde (*body.onload* wurde ausgelöst) oder das Laden mit Fehler beendet wurde. Beachten Sie zum besseren Verständnis die Erklärungen im Ereignis *DocumentCompete*.

Argumente	**Beschreibung**
IsSuccess	True, wenn die Navigation erfolgreich war. False steht für eine Navigation, die auf einer Fehlerseite gelandet ist (Ausfälle aufgrund von keinem Netzwerk, DNS-Lookup-Fehler, HTTP-Server antwortet mit 4xx).
WebErrorStatus	Ruft den Fehlercode ab, wenn die Navigation fehlgeschlagen ist. Folgende Werte sind möglich: 0 - Unknown 1 - CertificateCommonNameIsIncorrect 2 - CertificateExpired 3 - ClientCertificateContainsErrors 4 - CertificateRevoked 5 - CertificateIsInvalid 6 - ServerUnreachable 7 - Timeout 8 - ErrorHttpInvalidServerResponse 9 - ConnectionAborted 10 - ConnectionReset 11 - Disconnected 12 - CannotConnect 13 - HostNameNotResolved 14 - OperationCanceled 15 - RedirectFailed 16 - UnexpectedError

Eine Beschreibung der Fehlercodes finden Sie unter:
docs.microsoft.com/en-us/dotnet/api/microsoft.web.webview2.core.corewebview2weberrorstatus

NewWindowRequested

```
Event NewWindowRequested(ByVal IsUserInitiated As Boolean, IsHandled As
Boolean, ByVal URI As String, NewWindowFeatures As RC6.cCollection)
```

NewWindowRequested wird ausgelöst, wenn der Inhalt innerhalb des WebView2 das Öffnen eines neuen Fensters anfordert, z. B. durch *window.open*().

Argumente	Beschreibung
IsUserInitiated	True, wenn die Anforderung eines neuen Fensters durch eine Benutzeraktion, z. B. die Auswahl eines Anker-Tags mit Ziel, ausgelöst wurde.
IsHandled	Gibt an, ob das Ereignis behandelt wird. Der Standardwert ist False. Da dieses Argument ByRef übergeben wird, können sie es ändern. Indem Sie hier ein True übergeben, verhindern Sie, dass WebView2 ein eigenes neues Fenster (eigener Windows-Prozes) erzeugt.
URI	Internet-Ressource (i.d.R. eine http- oder https-Adresse)
NewWindowFeatures	Anforderungen an das neue Fenster. Die Felder entsprechen den *windowFeatures*, die an *window.open*() übergeben werden, wie in folgt:
HasPosition	Gibt an, ob die linken und oberen Werte angegeben sind.
HasSize	Gibt an, ob die Werte für Höhe und Breite angegeben sind.
Height	Ermittelt die Höhe des Fensters. Wird ignoriert, wenn *HasSize*() false ist.
Left	Ermittelt die linke Position des Fensters. Wird ignoriert, wenn *HasPosition*() false ist.
ShouldDisplayMenuBar	Zeigt an, dass die Menüleiste angezeigt wird.
ShouldDisplayScrollBars	Zeigt an, dass die Bildlaufleisten angezeigt werden.
ShouldDisplayStatus	Zeigt an, dass die Statusleiste angezeigt wird.
ShouldDisplayToolbar	Zeigt an, dass die Browser-Symbolleiste angezeigt wird.
Top	Ermittelt die obere Position des Fensters. Wird ignoriert, wenn *HasPosition*() false ist.
Width	Ermittelt die Breite des Fensters. Wird ignoriert, wenn *HasSize*() false ist.

Beispiel:

```
WV.NavigateToString _
  "<!DOCTYPE html><html><head></head><body>" & _
  "<button id='btn1' " & _
    "onclick='window.open(""https://selfhtml.org"")'>" & _
    "Button1" & _
  "</button>" & _
"</body></html>"

Private Sub WV_NewWindowRequested(ByVal IsUserInitiated As Long, _
  IsHandled As Boolean, _
  ByVal URI As String, _
  NewWindowFeatures As RC6.cCollection)

  Dim i As Long
  Debug.Print "IsUserInitiated: "; IsUserInitiated
  Debug.Print "IsHandled: "; IsHandled
  Debug.Print "URI: "; URI
  For i = 0 To NewWindowFeatures.Count - 1
    Debug.Print NewWindowFeatures.KeyByIndex(i), _
                NewWindowFeatures.ItemByIndex(i)

  Next
  IsHandled = True
End Sub
```

```
'Ausgabe:
IsUserInitiated: Wahr
IsHandled: Falsch
URI: https://selfhtml.org/
HasPosition     0
HasSize         0
ShouldDisplayMenuBar         1
ShouldDisplayScrollBars      1
ShouldDisplayStatus          1
ShouldDisplayToolbar         1
Left            0
Top             0
Width           0
Height          0
```

ProcessFailed

```
Event ProcessFailed()
```

ProcessFailed wird ausgelöst, wenn ein WebView*2*-Prozess unerwartet beendet oder nicht mehr ansprechbar ist.

SourceChanged

```
Event SourceChanged()
```

SourceChanged wird ausgelöst, wenn zu einer anderen Seite oder zu Fragmentnavigationen navigiert wird. Es wird nicht bei anderen Arten von Navigationen ausgelöst, z. B. bei Seitenaktualisierungen. Dieses Ereignis wird vor *ContentLoading* für die Navigation zu einem neuen Dokument ausgelöst.

UserContextMenu

```
Event UserContextMenu(ScreenX As Long, SreenY As Long)
```

UserContextMenu wird ausgelöst, wenn der Anwender das Kontextmenü anfordert (Klick mit rechter Maustaste). Dafür muss die Eigenschaft *AreDefaultContextMenusEnabled* False sein.

Argumente	**Beschreibung**
ScreenX	X-Koordinate des Mauszeigers
ScreenY	Y-Koordinate des Mauszeigers

Die Koordinate beziehen sich auf das gesamte Screen-Objekt, ausgehend vom Hauptbildschirm. Wenn am Anwender-PC mehrere Bildschirme angeschlossen sind und Sekundär-Bildschirme links oder über dem Hauptbildschirm angeordnet sind, können die Argumente negative Werte aufweisen. Der globale Screen-Bezug ermöglicht es Ihnen ein eigenes Kontextmenü zu implementieren und dieses an den übergebenen Koordinaten auszurichten.