APL-Journal

A Programming Language

Fotos: Martin Barghoorn

1-2/2022

APL-Germany e.V.

Doppelnummer

Nr. 1-2 2022

Jahrgang 41

ISSN - 1438-4531

RHOMBOS-VERLAG

Dieter Kilsch
Eine Schnittstelle zu Excel via APL2-COM

Markos Mitsos
Management of Dyalog APL workspaces

Martin Barghoorn
Private PKW in Deutschland

James A. Brown
Acronyms
APL2PLUS™ A NEW MAINFRAME OFFERING

www.apl-germany.de

APL Germany e.V.

Liebe Mitglieder von APL Germany,
liebe APL-Freunde,

heute erhalten Sie die neue Ausgabe des APL-Journals. Es blickt mit seinen Beiträgen auf die beiden Treffen im Mai in Berlin-Adlershof und im November in Stuttgart zurück, die endlich wieder als Präsenzveranstaltung durchgeführt werden konnten und somit wieder der befruchtende Austausch vor allem in den Pausen statt finden konnte. Die Präsentationen dieser Veranstaltungen wurden von einigen virtuellen Teilnehmern online verfolgt und zum Teil auch online gehalten.

Die positive Entwicklung in der Corona-Panmdemie wird leider seit Februar durch den schrecklichen Krieg Russlands gegen die Ukraine überschattet.

Mein Artikel beschreibt eine Schnittstelle zu Excel über die APL2-COM-Schnittstelle, die den Zugriff auf Excel-Blätter sehr vereinfacht.

Martin Barghoorn analysiert die Daten seiner Fahrzeug-Gallerie mit statistischer Akribie und zeigt so die Anderungen verschiedener Parameter im Laufe seiner Auto-Generationen.

James Brown hat in Stuttgart über Acronyme vor allem in der Datenverarbeitung vorgetragen und schreibt hierüber in seinem Artikel, auch über Hintergründe einiger Acronyme. In einem weiteren Artikel stellt er APL2PlusTM, ein Port von Dyalog APL 18.2 zu IBM zLinux vor.

Markos Mitsos berichtet über das Speichern von Funktion, Operatoren, Variablen und namespaces in Textdateien, Versionierung und automatisches Testen und Überprüfen der Anderungen und ihrer Effekte.

Ich wünsche Ihnen viel Freude und Anregungen beim Lesen der Artikel.

Unsere Mitgliederversammlung fand im Mai statt. Zu dem verschickten Protokoll gab es keine Einwände, es ist damit beschlossen. Die nächste Tagung findet in Bingen am Rhein am 24./25. April 2023 mit einer Mitgliederversammlung und Vorstandswahlen statt.

Ich wünsche Ihnen für 2023 viel Erfolg und gute Gesundheit und hoffe auf ein friedlicheres Jahr.

Dieter Kilsch

Kontakt:

Prof. Dr. Dieter Kilsch
E-mail: d.kilsch@th-bingen.de

INHALT

Eine Schnittstelle zu Excel via APL2-COM

Dieter Kilsch

Inhaltsverzeichnis

Die in diesem Artikel beschriebene Schnittstelle erlaubt einen einfachen Zugriff auf Excel mit Hilfe der in [1] beschriebene COM-Schnittstelle von APL2. Sie beinhaltet Routinen zum Lesen, Schreiben und Kreïeren von Excel-Blättern und Dateien und zum Ändern der Eigenschaften und Formatierung einer Zelle. Als Quellen für Methoden und Eigenschaften der zugrunde liegenden VBA-Objekte habe ich [4] und die Internetseite [3] der Firma Microsoft benutzt. Die Funktionen sind nur zum Teil hier abgedruckt, den kompletten Arbeitsbereich finden Sie in [2].

Ich habe die Wirkung der einzelnen Funktionen nicht durch Bildschirmfotos belegt. Das kann aber nach Herunterladen der Software mit APL2 problemlos nachgeholt werden.

1 Lesen eines Blatts mit `XLSles`

1.1 Aufruf der Funktion `XLSles`

Die Funktion `XLSles` erlaubt auf einfache Art das Lesen eines Teils oder des gesamten Excel-Arbeitsblatts. Der Aufruf „`s XLSles dn`“ beinhaltet die Steuervariable `s` und die Informationen zu Datei und Blatt in `dn`, siehe hierzu den Kopf der Funktion in Abb. 1.[1]

Die erste Komponente der Variablen `s` bestimmt, ob die Excel-Datei geöffnet (1), geschlossen (2) oder beides (3) wird. Mit der zweiten Komponente wird festgelegt, ob ein neues Blatt (1) oder das vorherige Blatt (2) gelesen, die Anzahl der Blätter (3) oder die Namen der Blätter (4) zurück gegeben werden soll.

Wird die Datei lediglich geöffnet und nicht wieder geschlossen, so nehmen die globalen Variablen `xlshex, xlshwb, xlshws` die Griffe (handle) auf Excel, die Datei (work book) und das Blatt (work sheet) auf. Die globale Variable `xlssav` wird von denjenigen Funktionen auf 1 gesetzt, die eine Änderung am Inhalt der Datei vornehmen, sie also vor dem Schließen gespeichert werden muss. Dies übernimmt dann die Funktion `XLSscl`.

Die Eingabevariable `dn` enthält in vier Zellen den Dateinamen, den Namen oder die Nummer des Blatts, die linke obere Ecke des zu lesenden Felds und dessen Größe.

Bei fehlerfreiem Bearbeiten enthält die Rückgabevariable `r` in der ersten Komponente eine 0 und die zweite die gelesenen Zellen als geschachtelte Matrix[2]. Im Fehlerfall gibt die erste Komponente ein Fehlernummer zurück, die zweite einen Fehlertext und die dritte die Aufrufsequenz. Letzteres gilt allgemein für alle Funktionen `XLS`....

[1] Lange erste Zeilen sind in den Funktionslisten nicht-sprachkonform umgebrochen!
[2] Geschachtelte Matrix und allgemeine Matrix werden als Synonyme benutzt.

```
r←s XLSles dn;asn;cs;em;hex;hwb;hws;i;isp;ize;nco;nro;ns;off;pac;res;rr;
      tx;cenr
⍝  V7.2  06.01.2022  D.Kilsch  2008 khxls                      (13.07.2005)
⍝ s[1]∊0 1: Das gelesene Blatt bleibt angeschlossen!
⍝ Anbinden und Lesen einer Excel-Tabelle, Schließen der Bindung
⍝ s      V[1]:  S 0: nur lesen                                    [Def.: 3]
⍝                 1/2/3: verbinden, öffnen / schließen, lösen / beides
⍝         [2]:  S 1/2: Lesen einer neuen / der vorherigen Tabelle[Def.: 1]
⍝                 3/4: Anzahl / Namen der Blätter bestimmen
⍝ dn     AV[1] TV    Datei
⍝          [2] TV    Tabelle, leer: aktive Tabelle              [DEF.: '']
⍝               S    Blattnummer
⍝          [3]  V[2] linkes oberes Feld: '' alles lesen         [DEF.: '']
⍝          [4]  V[2] Größe des Datenbereichs
⍝        TV          wie 1⊃dn
⍝ r      AV[1]  S           0: fehlerfrei:
⍝                           1: nicht alle Zellen lesbar
⍝          [2] AM    s[2]∊1 2: gelesene Zellen
⍝               S    s[2]=3  : Anzahl der Blätter
⍝              AV TV s[2]∊4  : Namen der Blätter
⍝          [3] AV V[2]r[1]=1 : nicht lesbare Zellen: Fehler 5 4
⍝        AV[1]  S          <0: Fehlernummer
⍝          [2] AV TV:          Fehlertext
⍝          [3] TV              Aufrufsequenz
⍝
⍝  globale Variable und Parameter:
⍝ xlshex S      Griff auf Excel
⍝ xlshwb S      Griff auf Datei
⍝ xlshws S      Griff auf Tabelle (Blatt)
⍝ xlssav L      muss gespeichert werden?
```

Abb. 1: Kopf der Funktion `XLSles`

1.2 Die Funktion `XLSles`

Ich gehe auf wichtige Programmzeilen ein. Bei jedem Funktionsaufruf wird eine Fehlerüberprüfung durchgeführt, auf deren Beschreibung hier verzichtet wird.

(a) Verbindung zur Datei herstellen, falls `s[1]∊1 3`:

- Verbindung zur COM-Schnittstelle aufbauen: `r←3 11 ⎕NA 'COM'`
- Excel mit Griff `hex` anschließen:
  `(r em hex)←1 COM 'CREATE' 'Excel.Application'`
- Datei öffnen mit Griff `hwb`:
  `(r em hwb)←1 COM 'METHOD' hex 'Workbooks.Open'(1⊃dn)`

(b) Blatt aktivieren:

- Früheres Blatt entlassen, wenn `hws` existiert:
  `→(2≠⎕NC 'hws')/1+⎕LC⋄(r em res)←1 COM 'RELEASE' hws`
- Blatt-Adressierungsart `cs`: Blattnummer (1), Blattname (2) oder aktives Blatt (3) gegeben?
  `cs←↑((0=↑0⍴2⊃dn),(0 1=0=⍴2⊃dn))/⍳3`
- Anbinden des Blatts mit Griff `hws`:
  `r←cs⊃('WORKSHEETS.item()'(2⊃dn))('WORKSHEETS()'(2⌷dn))(⊂'ActiveSheet')`
  `(r em hws)←1 COM 'PROPERTY' hwb,r`

(c) Anzahl `ns` der Blätter ermitteln:
`(r em ns)←1 COM 'PROPERTY' hwb 'WORKSHEETS.COUNT'`

(d) Namen der Blätter ermitteln:

(d1) Nummer des aktiven Blatts ermitteln:
`(r em asn)←1 COM'PROPERTY' hwb 'ActiveSheet.Name'`

(d2) Alle Namen lesen:

```
 i←0⋄tx←0⍴⊂''
DO:→(ns<i←i+1)/UNDO
 (r em hws)←1 COM 'PROPERTY' hwb 'WORKSHEETS.item()' i
 (r em res)←1 COM 'PROPERTY' hws 'Name'⋄tx←tx,⊂res
 (r em res)←1 COM 'RELEASE' hws
 →DO
UNDO:
```

(d3) Ursprünglich aktives Blatt wieder aktivieren:
`i←tx⍳⊂asn`
`(r em hws)←1 COM 'PROPERTY' hwb 'WORKSHEETS.item()' i`

(e) Vom aktiven Blatt lesen:

- Gesamten Inhalt lesen:
  `(r em res)←1 COM 'PROPERTY' hws 'UsedRange.Value'⋄tx←res`
- Ein Feld des Blatts lesen:
  `(r em res)←1 COM 'PROPERTY' hws,(⊂'Cells().Resize().Value'),2↓dn`
  `tx←res`
- Leere Zeilen und Spalten am Ende löschen, nachdem eventuell `tx` von Skalar oder Vektor in Matrix gewandelt wurde: `tx←(+/∨\⌽∨/r)(+/∨\⌽∨≠r←↑¨0≠⍴¨,¨tx)↑tx`

Zum Schließen der Datei wird die Funktion `XLSscl` aufgerufen. Soll sie offen bleiben, so werden die Griffe auf die globalen Variablen gesichert.

2 Schreiben und Kreïeren eines Blatts und einer Datei mit XLSscr

2.1 Aufruf der Funktion XLSscr

Die Funktion XLSscr erlaubt auf einfache Art das Beschreiben eines Teils oder eines gesamten Excel-Arbeitsblatts. Der Aufruf „s XLSscr dn“ beinhaltet die Steuervariable s und die Informationen zu Datei, Blatt, Position und Inhalt in der Variablen dn, siehe hierzu den Kopf der Funktion in Abb. 2.

```
r←s XLSscr dn;cs;em;hex;hwb;hws;i;ldat;ns;res;tx;cr
⍝  V5.8  29.09.2019  D.Kilsch  2008 khxls                    (17.07.2005)
⍝ Anbinden und Schreiben einer Excel-Tabelle, Schließen der Bindung
⍝ s[1]∊0 1: Das geschriebene Blatt bleibt angeschlossen!
⍝ s      V [1] S  0: nur schreiben                              [Def.: 3]
⍝                 1/2/3: verbinden, öffnen / schließen, lösen / beides
⍝          [2] S  0/1: Blatt vor dem Schreiben komplett leeren   [Def.: 0]
⍝          [3] S  -1: Wenn nicht ex., Blatt nicht neu anlegen   [Def.: -1]
⍝                  0: Wenn nicht ex., Blatt am Ende neu anlegen
⍝                 >0: Wenn nicht ex., Blatt vor s[3] neu anlegen
⍝                     Wenn zu groß: Am Ende anlegen
⍝??            TV Wenn nicht ex., Blatt vor Blatt s[3] einfügen
⍝          [4] S  0/1: Wenn Datei nicht existiert, kreieren.     [Def.: 0]
⍝ dn     AV[1]  TV  Datei
⍝          [2]  TV  Tabelle, leer: aktive Tabelle               [DEF.: '']
⍝          [3]  V[2] linkes oberes Feld
⍝          [4]  AM   Data Matrix (legt Größe fest)
⍝ r     (A)V[1] S    0/1/2: fehlerfrei: geschrieben/ hinzugefügt/ kreiert
⍝                   <0   : Fehlernummer:
⍝           [2] AV TV: Fehlertext
⍝           [3] TV     Aufrufsequenz
⍝
⍝  globale Variable und Parameter:
⍝ xlshex S      Griff auf Excel
⍝ xlshwb S      Griff auf Datei
⍝ xlshws S      Griff auf Tabelle
⍝ xlssav L      muss gespeichert werden?
```

Abb. 2: Kopf der Funktion XLSscr

Wie bei XLSles bestimmt die erste Komponente der Variablen s, ob die Excel-Datei geöffnet (1), geschlossen (2) oder beides (3) wird. Auch das Benutzen der globalen Variablen xlshex, xlshwb, xlshws ist gleich. Beim Schreiben wird die globale Variable xlssav immer auf 1 gesetzt. Die Datei muss vor dem Schließen gespeichert werden.

Mit den weiteren Komponenten wird festgelegt,
- ob, das Blatt vor dem Schreiben komplett geleert werden soll, um alten Inhalt zu löschen;
- ob und wo das Blatt neu angelegt werden soll, falls es nicht existiert;
- ob die Datei neu angelegt werden soll, falls sie nicht existiert.

Die Eingabevariable dn enthält in vier Zellen den Dateinamen, den Namen oder die Nummer des Blatts, die linke obere Ecke des zu lesenden Felds und die zu schreibenden Daten als allgemeine Matrix.

Bei fehlerfreiem Bearbeiten enthält die Rückgabevariable r eine Information, ob die Datei lediglich geschrieben (0), ergänzt (1) oder kreiert (2) wurde.

2.2 Die Funktion XLSscr

Ich gehe wieder nur auf wichtige Programmzeilen ein und verzichte auf die Darstellung der Fehlerüberprüfung, die bei jedem Funktionsaufruf durchgeführt wird.

(a) Verbindung zur Datei herstellen, falls s[1]∊1 3:

- Verbindung zur COM-Schnittstelle aufbauen: `r←3 11 ⎕NA 'COM'`
- Excel mit Griff hex anschließen:
  `(r em hex)←1 COM 'CREATE' 'Excel.Application'`

- Datei öffnen mit Griff hwb, ldat: existierende Datei öffnen?:
  ```
  ldat←0 0≡↑(r em hwb)←1 COM 'METHOD' hex 'Workbooks.Open'(1⊃dn)
  ```
  Fehler, falls Datei nicht existiert und keine neue angelegt werden soll:
  ```
  ~ldat∨(4⎕s)∧r≡1 5
  ```

(b) Neue Datei mit Griff hwb und neues Blatt mit Griff hws anlegen und Namen vergeben, falls durch „0<⍴r←2⊃dn“ gefordert:
```
(r em hwb)←1 COM 'METHOD' hex 'Workbooks.Add'
(r em hws)←1 COM 'PROPERTY' hwb 'ActiveSheet'
(r em res)←1 COM 'PROPERTY' hws 'Name' (2⊃dn)
```

(c) Existierendes Blatt aktivieren:

- Früheres Blatt entlassen, wenn hws existiert:
  ```
  →(2≠⎕NC 'hws')/1+⎕LC⋄(r em res)←1 COM 'RELEASE' hws
  ```
- Blatt-Adressierungsart cs: Blattnummer (1), Blattname (2) oder aktives Blatt (3) gegeben?
  ```
  cs←↑((0=↑0⍴2⊃dn),(0 1=0=⍴2⊃dn))/⍳3
  ```
- Anbinden des Blatts mit Griff hws:
  ```
  r←cs⊃('WORKSHEETS.item()'(2⊃dn))('WORKSHEETS()'(2⎕dn))(⊂'ActiveSheet')
  (r em hws)←1 COM 'PROPERTY' hwb,r
  ```

(d) Neues Blatt in existierender Datei mit Griff hws anlegen:

- Neues Blatt vor Blatt mit Namen s[3] einfügen:
  ```
  (r em hws)←1 COM 'PROPERTY' hex 'Worksheets()',(⊂s[3])
  ```
  - Blatt gefunden:
    ```
    (r em hws)←1 COM 'METHOD' hex 'Worksheets.Add[]'('Before' hws)
    ```
  - Blatt nicht gefunden, am Ende anlegen:
    ```
    (r em hws)←1 COM 'PROPERTY' hex 'Worksheets.Item()' ns
    (r em hws)←1 COM 'METHOD' hex 'Worksheets.Add[]'('After' hws)
    ```
- Neues Blatt vor Blatt mit Nummer s[3] einfügen: Ist die Nummer zu groß, also (ns<3⎕s)/'s[3]←0', so wird sie auf 0 gesetzt und das neue Blatt am Ende der Datei angelegt.
  - Griff des Referenzblatts finden:
    ```
    r←(0 1=0=3⎕s)/s[3],ns
    (r em hws)←1 COM 'PROPERTY' hex 'Worksheets.Item()' r
    ```
  - Blatt einfügen:
    ```
    r←∊(0 1=s[3]=0)/'Before' 'After'
    (r em hws) ←1 COM 'METHOD' hex 'Worksheets.Add[]'(r hws)
    ```
- Neues Blatt auf eingegebenem Namen umbenennen:
  ```
  (r em res)←1 COM 'PROPERTY' hws 'Name' (2⊃dn)
  ```

(e) Daten eintragen:

- Blattinhalte und Formate löschen:
  ```
  (r em tx)←1 COM 'PROPERTY' hws 'UsedRange.Value' ''
  (r em tx)←1 COM 'PROPERTY' hws 'UsedRange.ClearFormats'
  ```
- Daten schreiben:
  ```
  r←'PROPERTY' hws 'Cells().Resize().Value',dn[3],(⊂⍴4⊃dn),dn[4]
  (r em res)←1 COM r
  ```
- Spaltenbreite automatisch festlegen:
  ```
  (r em res)←1 COM 'PROPERTY' hws 'UsedRange.Columns.Autofit'
  ```

(f) Datei schließen bei `s[1]∊2 3`:

- Sichern einer neuen Datei mit „SaveAs“, einer existierenden mit „Save“:
  `(r em res)←1 COM 'METHOD' hwb,↑(1 0=ldat)/(⊂'Save')('SaveAs'(1⊃dn))`
  Bei `ldat=1` wurde eine existierende Datei bearbeitet.
- Sichern der Griffe auf die globalen Variablen, lösen der Griffe für Blatt, Datei und Excel, zum Teil mit Hilfe der Funktion `XLSscl`:

```
 ⍎'→CB',⍕↑s
CB0:xlshws←hws⋄→CBend
CB1:xlshex←hex⋄xlshwb←hwb⋄xlshws←hws⋄→CBend
CB2:xlshws←hws⋄→(0≠≡r←XLSscl 0)/FZ⋄→CBend0
CB3:→(0≠≡r←XLSscl hex hwb hws)/FZ
```

3 Schließen der Datei mit `XLSscl`

Diese Funktion übernimmt das Schließen der Datei und das Lösen der Griffe des Blatts, der Datei und des Programms Excel.

3.1 Aufruf der Funktion `XLSscl`

Der Aufruf „`hs XLSscl griff`“ der Funktion `XLSscl` beinhaltet die Steuervariable `hs` und ggf. die Griffe zu Excel, der offenen Datei und dem zuletzt bearbeiteten Blatt, siehe hierzu den Kopf der Funktion in Abb. 3.

```
r←hs XLSscl griff;em;res;ns;sav
⍝  V5.2  21.04.2022  D.Kilsch  2008 khxls                    (25.09.2012)
⍝ Schließen einer Excel-Datei.
⍝ hs      LS      0/1: Hinweis beim Schließen?                  [Def.: 0]
⍝ griff   V[1]    Griff auf Excel
⍝          [2]    Griff auf Arbeitsmappe (Datei)
⍝          [3]    ≠0: Griff auf Blatt, falls angebunden
⍝         S       0: globale Griffe lösen und löschen
⍝ r       S       =0  fehlerfreies Schließen
⍝         AV S    <0: Fehlernummer
⍝            AV   TV   Fehlertext
⍝            TV   Aufrufsequenz
⍝
⍝  globale Variable und Parameter: (kreieren bei s=1)
⍝ xlshex S       Griff auf Excel
⍝ xlshwb S       Griff auf Datei
⍝ xlshws S       Griff auf Tabelle
⍝ xlssav L       muss gespeichert werden?
```

Abb. 3: Kopf der Funktion `XLSscl`

Die linke Eingabevariable `hs` gibt an, ob ein Hinweis zum Schließen am Bildschirm ausgegeben werden soll. Die rechte Eingabevariable `griff` beinhaltet die Griffe oder ist 0. In diesem Fall werden die in den globalen Variablen `xlshex, xlshwb, xlshws` gespeicherten Griffe benutzt. Intern wird `xlssav` auf `griff[4]` gespeichert.

Bei fehlerfreiem Bearbeiten enthält die Rückgabevariable `r` eine Information, ob die Datei lediglich geschrieben (0), ergänzt (1) oder kreiert (2) wurde.

3.2 Die Funktion `XLSscl`

Sind die Griffe auf den globalen Variablen `xlshex, xlshwb, xlshws` gespeichert, so werden diese auf `griff` übernommen.

(a) Lösen des Blatts, falls `3⌷griff` gesetzt ist: `(r em res)←1 COM 'RELEASE' griff[3]`
(b) Datei sichern, falls `xlssav`, also `4⌷griff` gesetzt ist:
`(r em res)←1 COM 'METHOD' griff[2]'Save'`

(c) Datei schließen und lösen. Hierzu muss zunächst sie Sichtbarkeit auf 0 gesetzt werden. Zusätzlich wird ggf. der Hinweis am Bildschirm ermöglicht:

```
(r em res)←1 COM 'PROPERTY' griff[1]'Visible' 0
(r em res)←1 COM 'PROPERTY' griff[1]'DisplayAlerts' hs
(r em res)←1 COM 'METHOD' griff[2]'Close'
COM 'RELEASE' griff[2]
```

(d) Excel lösen: `COM 'RELEASE' griff[1]`

4 Hilfsprogramme

Diese Programme vereinfachen Zugriffe und Berechnungen oder vermeiden Kodewiederholungen. Beim Aufruf all dieser Funktionen muss eine Datei geöffnet sein.

4.1 Setzen von Eigenschaften: `XLSprp`, `XLSpwb` und `XLSpws`

Die erste Zeile enthält jeweils den Aufruf

(a) `XLSprp` unterstützt das Setzen von Eigenschaften einer Datei oder eines Blatts. Der Griff muss übergeben werden:

```
r←griff XLSprp par
r←1 COM 'PROPERTY' griff,par
xlssav←1
```

(b) `XLSpws` unterstützt das Setzen von Eigenschaften eines Blatts:

```
r←XLSpws par
r←xlshws XLSprp par
```

(c) `XLSpwb` unterstützt das Setzen von Eigenschaften einer Datei:

```
r←XLSpwb par
r←xlshwb XLSprp par
```

4.2 Positionen und Bereiche in Excel-Format wandeln

Zeilen- und Spaltennummern einer Matrix werden numerisch angegeben, das Excel-Format benutzt für die Spalten dagegen Großbuchstaben und setzt diese nach „Z“ mit zwei Buchstaben fort: „AA“, „AB“, ... Die wesentliche Umrechnung eines 2-elementigen, numerischen Vektors `num` erfolgt durch die Zeile

```
⎕IO←0⋄r←0 1+[0]27 26⊤¯1+num⋄⎕IO←1⋄r←(⊂[1]⎕AV[65+r])~¨'@'
```

Hierzu gebe ich einige Beispiele an:

(a) Die Funktion `XLSpos`:

```
r←XLSpos num;rr
⍝  V1.1  10.03.2017  D.Kilsch  2017 khxls          (XLSspl  10.03.2017)
⍝ Position in Excel-Format umrechnen.
⍝ num    V        2-elementiger Vektor
⍝        AV V     2-elementige Vektoren
⍝ r      TV       kodierte Position
⍝        AV TV    kodierte Positionen
```

Beispiele:

```
      XLSpos (2 2) (17 77) (1 120)
B2 BY17 DP1
```

(b) Die Funktion `XLSspl`:

```
r←XLSspl num;rr
⍝  V2.1  01.02.2017  D.Kilsch  2017 khxls                    (01.02.2017)
⍝1.— —
⍝ Positionen in Excel-Format berechnen.
⍝ num    S/V      Nummern der Position      (0 1∨.=≡num)
⍝ r      AS/AV TV kodierte Spaltennummern
⍝2.— —
⍝ Bereich in Excel-Format berechnen.
⍝ num    AV V     Bereich von bis           2=∧.=∊(≡num) (⍴num) (⍴¨num)
⍝ r      TV       Bereich in Excel-Format
```

Beispiele:

```
     XLSspl (17 19)(27 117)
S17:DM27
```

5 Weitere Methoden und Eigenschaften

Dieser Abschnitt stellt einige weitere Methoden und Eigenschaften für Excel, eine Datei und ein Blatt zusammen.

5.1 Methoden für Excel

(a) Löschen einer Datei:
`1 COM 'METHOD' xlshex 'Worksheets().Delete' 2`

(b) Ein Fenster öffnen und eine Eingabe erfragen:
`1 COM 'METHOD' xlshex 'InputBox' 'Enter your name' 'APL2 Sample'`

5.2 Eigenschaften einer Datei

(a) Ein Blatt auswählen (aktivieren):

```
xlshws←COM 'PROPERTY' xlshwb 'WORKSHEETS()'(⊂'Tabelle1')
xlshws←COM 'PROPERTY' xlshwb 'Sheets().Select' 1
xlshws←COM 'PROPERTY' xlshwb 'ActiveSheet'
```

(b) Schriftfarbe der Tabelle1 in grün (RGB) ändern:
`1 COM 'PROPERTY' xlshwb 'WORKSHEETS().[B2].Font.Color'(⊂'Tabelle1')65280`

(c) Schriftart, genauer die Schriftserie, in fett ändern:
`1 COM 'PROPERTY' xlshwb 'WORKSHEETS().[B2].Font.Bold'(⊂'Tabelle1')1`

5.3 Eigenschaften eines Blatts

(a) Teile eines Blatts lesen:

- Zeile eins lesen: `XLSpws 'rows().Value' 1`
- Spalte eins lesen: `XLSpws 'columns().Value' 1`
- Drei Zellen von Zeile fünf lesen: `XLSpws 'rows().Resize().Value' 5(1 3)`
- Drei Zellen von Zeile fünf lesen: `XLSpws 'Cells().Resize().Value' (5 1)(1 3)`
- Das komplette Blatt lesen: `XLSpws 'UsedRange.Value'`

(b) In ein Blatt schreiben

- Eine 4,4-Matrix, zeilenweise mit `⍳16` gefüllt, ab D2 schreiben:
  `XLSpws 'Cells().Resize().Value',(2 4),(4 4),⊂4 4⍴⍳16`
- Wirkungslose Aufrufe, die keinen Fehler hervorrufen. Die übergebenen Daten müssen geschachtelt sein:
  `XLSpws 'rows().Value' 1(1 5⍴12)`
  `XLSpws 'rows().Resize().Value' 5(1 3)(1 3⍴⍳3)`

(c) Die Schriftfarbe (font) oder Hintergrundfarbe (interior) im aktuellen Blatt ändern:

```
XLSpws 'UsedRange.Font.Color' 65280
XLSpws 'Range().Font.Color'(⊂'A1:B6')65280
XLSpws 'Range().Interior.Color'(⊂'A1:B6')65280
XLSpws 'Range().Columns.Interior.Color'(⊂'A:F')65280
XLSpws 'Range().Columns.Font.Color'(⊂'7:7')65280
XLSpws 'Range().Columns.Font.Color'(⊂'H:H')65280
XLSpws 'Range().Rows.Font.Color'(⊂'6:6')65280
```

Tab. 1 enthält die RGB-Werte einiger wichtiger Farben.

(d) Die Schriftart im aktuellen Blatt ändern:

- Fett einschalten (Schriftserie):
  `XLSpws 'UsedRange.Font.Bold' 1`
  `XLSpws 'Range().Font.Bold'(⊂'A1:B6')1`
- Neigung anschalten (Schriftform):
  `XLSpws 'Range().Font.Italic'(⊂'A1:B6')1`
- Unterstreichen:
  einfach: `XLSpws 'Range().Font.Underline'(⊂'A1:B6')2`
  doppelt: `XLSpws 'Range().Font.Underline'(⊂'A1:B6')5`
  doppelt fett: `XLSpws 'Range().Font.Underline'(⊂'A1:B6')-4119`
  ausschalten: `XLSpws 'Range().Font.Underline'(⊂'A1:B6')-4142`
- Schriftgröße ändern: `XLSpws 'Range().Font.Size'(⊂'A1:B6')28`

(e) Zellen Zusammenfassen und Zusammenfassung aufheben:
Zusammenfassen: `XLSpws 'Range().merge'(⊂'A1:B2')`
Zusammenfassung aufheben: `XLSpws 'Range().unmerge'(⊂'A1:B2')`

(f) Zahlenformat der Zellen ändern:

- Zahlen mit einer Nachkommastelle:
  `XLSpws 'Range().NumberFormat'(⊂'F1:G1')(,⊂'???,0')`
- Standard-Format:
  `XLSpws 'Range().NumberFormat'(⊂'F1:G1')(,⊂'Standard')`
- Datum-Format:
  `XLSpws 'Range().NumberFormat'(⊂'F:G')(,⊂'TT.MM.JJJJ')`

Hierbei muss beachtet werden, dass auch die Formatangaben nach deutschen Regeln (Dezimalkomma und „TT.MM.JJJJ“ für Datum) angegeben werden müssen. Ferner darf das Format kein geschachtelter Skalar sondern muss ein Vektor oder eine Matrix sein.

Tab. 1: RGB-Werte wichtiger Farben

Farbe	rot	grün	blau	RGB-Wert
rot	255	0	0	255
grün	0	255	0	65280
blau	0	0	255	16711680
cyan	0	255	255	16776960
magenta	255	0	255	16711935
gelb	255	255	0	65535
weiß	255	255	255	16777215
grau	128	128	128	8421504
schwarz	0	0	0	0

6 Eine Anwendung: Organisieren einer Akademie

Der Alumni-Verein der Studienstiftung des deutschen Volkes organisiert eine Winterakademie, eine Kombination aus Skifahren und akademischem Programm für Stipendiat:Innen und Alumni/Alumnae. Unterkunft und Verpflegung wird über einen Reiseveranstalter gebucht, die Anmeldung, Auswahl und Organisation der akademischen Kurse erfolgt über ein Organisationsteam, dem ich angehöre. Verwaltung der Teilnehmer:Innen einschließlich einer Warteliste und dem Stornierungs- und Nachrückverfahen und der Kommunikation zu dem Reiseveranstalter habe ich übernommen. An der Akademie nehmen 60 Teilnehmer:Innen einschließlich Organisationsteam und 5 Dozent:Innen teil. Alumni/Alumnae unterstützen die Teilnehmer:Innen durch eine erhöhten Beitrag (Querfinanzierung).

6.1 Verarbeitungsprozesse Anmeldung und Nachrückverfahren

6.2 Der normale Verarbeitungsprozess

(a) Stipendiat:Innen und Alumni/Alumnae melden sich an. Ein Losverfahren entscheidet über die Teilnahme. Die ausgewählten Teilnehmer:Innen werden in einer Excel-Tabelle geführt und erhalten den Status 0. Die nicht Gewählten werden in einer Warteliste geführt.

(b) Wir fragen die Teilnehmer:Innen nach ihren Kontodaten für die Querfinanzierung (neuer Status 1).

(c) Wir erhalten diese Informationen (neuer Status 2).

(d) Die Teilnehmer:Innen erhalten den Link zur Anmeldung beim Reiseveranstalter (neuer Status 3).

(e) Der Reiseveranstalter bestätigt die Buchung (neuer Status 4).

6.3 Das Nachrückverfahren

(a) Bei einer Stornierung wird der Status n zu -n geändert.
(b) Diese werden dann per Programm von der Teilnehmer:Innenliste gestrichen und durch eine:n Nachrücker:In ersetzt.
(c) Für neue Teilnehmer:Innen beginnt das normale Verfahren ab Punkt (b).

6.4 Unterstützende Programme

Alle unterstützenden Programm verwenden zum Lesen aus und Schreiben in Excel-Tabellen die Funktionen der in Abschnitt 1 - 4 beschriebenen Schnittstelle, z.B. auch das Formatieren der Datumzellen. Die Schnittstelle vereinfacht den Zugriff auf Excel-Dateien deutlich.

Zum Versenden der E-Mails an die Teilnehmer:Innen erstellen die Funktionen Makro-Dateien für den Editor WinEdt. Beim Aufruf der Makrodateien wird das E-Mail Thunderbird gestartet und die E-Mails angezeigt. Diese können, ggf. nach Ändern der Absenderadresse und Ändern der Adresseinträge von „AN“ zu „BCC“, verschickt werden.

`Mail1G` erstellt eine Makrodatei mit acht Gruppen-E-Mails getrennt nach den Kriterien „Stipendiat:In/Alumn.“, „Vereinsmitglied ja/nein“ und „m/w“.

Die Funktion liest alle Informationen aus den Excel-Dateien und verarbeitet diese. Es wird für Punkt drei des Verabrbeitungsprozess benutzt und setzt den Status auf 1.

`Mail1` bereitet die vergleichbaren E-Mails wie Mail1G im Nachrückverfahren und damit für einzelne Teilnehmer:Innen vor und ändert den Status von 0 auf 1.

`Mail2G` erstellt die E-Mail-Makrodateien zum Versenden des Links des Reiseveranstalters an die Teilnehmer:Innen mit Status 2 für vier Zielgruppen (m/w, Stipendiat:In/Alumn.) und setzt den Status auf 3.

`Mail2` erstellt die E-Mail-Makrodateien zum Versenden des Links des Reiseveranstalters an die Teilnehmer:Innen mit Status 2 im Nachrückverfahren für einzelne Teilnehmer:Innen und setzt den Status auf 3.

`Mail3` erstellt E-Mail-Makrodateien für Teilnehmer:Innen mit Status 3, die noch nicht bei unserem Reiseveranstalter gebucht haben. Es ist also eine Erinnerungsnachricht.

6.5 IO-Aufrufe in `Mail1G`

Beispielhaft stelle ich die IO-Aufrufe der Funktion `Mail1G` vor. Zunächst werden die Dateinamen in den Variablen `datin` und `datmail` gespeichert (s. Abb. 4), wobei das Datum und die Uhrzeit Bestandteil der Namen der WinEdt-Makro-Datei „`Sendmailjjjjmmtthhmm.edt`“ und der Logdatei -„`.mlg`“ sind. Das Datum wird in der vierten Zeile formatiert. In der vorletzten Zeile wird die Teilnehmere:Innen-Datei gesichert, in der letzten geöffnet und gelesen.

```
pfad←'F:\Dieter\Vereine\Studienstiftung\Winterakademie\2023\'
datin←pfad,'TeilnehmerInnen.xlsx'
datmail←(⊂pfad,'EMail\EMail1'),¨('Alum' 'Stip',¨⊂'.txt')
txl←r←((⍕(4,4⍴2)⍴¨'0')⍕5↑⎕TS)~' '
datmail←datmail,(⊂pfad,'EMail\SendMail',r,'.'),¨'edt' 'mlg'
⍝═══════  Teilnehmer:Innen: zuerst Backup
→(0>↑r←Pipe 'Copy ',datin,' ',(¯4↓datin),'bak')/FZ
→(0>↑r←1 XLSles datin 'TN')/FZ⋄dtn←2⊃r
```

Abb. 4: Einlesen der Tabelle TN der Teilnehmer:Innen-Datei

In Abb. 5 wird zunächst der Inhalt der E-Mail für Stipendiat:Innen bzw. Alumni/Alumnae aus einer Textdatei gelesen. Diese enthält Variablen, die entsprechend der Bedeutung der acht Gruppen je E-Mail unterschiedlich gesetzt werden. Nach dem nicht dargestellten Aufbereiten der E-Mail-Makros werden diese als Textdatei geschrieben (Zeile nach ¨⋄). `TDTles` und `TDTscr` sind Schnittstellenfunktionen für den `AP210`. Die folgende Zeile ändert den Status der Teilnehmer:Innen, die erste Spalte des Blatts „TN“ der offenen Datei (der linke Parameter ist null) wird ab Zeile 11 auf 1 gesetzt.

Die drittletzte Zeile entfernt abschließende Leerzeichen der E-Mailadressen. Die Vorletzte schreibt diese Adressen gruppenweise in das Blatt „EMail“ der Teilnehmer:Innen-Datei und schließt die Datei. Die letzte Zeile schreibt einen E-Mail-Log ohne Sicherung der alten Datei als Textdatei.

```
 →(0>↑r←¯4 TDTles∊datmail['AS'⍳1⌷sta])/FZ⋄tx←2⊃r
⋄
 ⋄
  ⋄
 →(0≠r←txa TDTscr 0,⊂'0',3⊃datmail)/FZ  ⍝ SendMail schreiben
 →(0>↑r←0 XLSscr datin 'TN'(11 1)(((⍴ind),1)⍴1))/FZ
⍝────  EMail-Liste je Status Anredegeschlecht
 r←stg,[1](⌽¨∨\¨⌽¨r≠' ')Reduce r←⍉⊃email
 →(0>↑r←2 1 0 XLSscr datin 'EMail'(11 1)r)/FZ
 →(0≠r←txl TDTscr 0,⊂'0',4⊃datmail)/FZ  ⍝ Log schreiben
```

Abb. 5: Schreiben der Teilnehmer:Innen-Datei und der WinEdt-Makrodateien

Literaturverzeichnis

[1] IBM. APL2 User's Guide. International Business Machines Corporation. 1994, 2012.

[2] D. Kilsch. COM-interface to Excel. https://seafile.rlp.net/d/9a29c9397c/files/?p=/arbeit/APLWS/comxls.zip&dl=1 (besucht am 21. 11. 2022).

[3] Microsoft. Excel VBA Reference. https://learn.microsoft.com/en-us/office/vba/api/overview/excel (besucht am 30. 04. 2022).

[4] T. Theis. Einstieg in VBA mit Excel. Bonn: Galileo Press, 2009.

Private PKW in Deutschland

Persönliche Datenauswertung von 1958 - 2022

Martin Barghoorn

Warum gerade PKW?

Schon das einst renommierte Meyers Lexikon von 1907 schreibt bezüglich der Motorwagen:
"Infolge der wohltuenden Wirkung auf die Nerven finden wir gerade unter den Gehirnarbeitern enthusiastische Anhänger des Motorwagens."

Allgemein hat die Leidenschaft für den eigenen Wagen, wie sie noch zu Beginn meines Auswertungszeitraums 1958 in Deutschland nicht nur bei Gehirnarbeitern vorherrschte, etwas nachgelassen. Hierfür gibt es eine Vielzahl von Gründen. Aber auch andere Verkehrsmittel haben an Faszination eingebüßt, die Individualreise scheint immer mehr zur Utopie zu werden bzw. wird schon nostalgisch verklärt.

In dieser Studie will ich die Beziehung zu meinen Individualfahrzeugen vom Beginn im Jahr 1958 an - so schwer es mir fällt - möglichst wenig emotional, vornehmlich unter technischen und wirtschaftlichen Aspekten untersuchen.

Datenquellen

Abb. 1: Belege für Auswertung

Als primäre Datenquellen für mein bisher 64 Jahre langes Leben mit Autos dienten mir die Wartburg-Autobücher von Gertrud B. und ein Belegordner der Citroen 2CV-Ente meiner Frau, der von Gertrud K. 1970 angelegt und 5 ½ Jahre akribisch geführt wurde. Dieser Ordner, genannt Auto, hat mich

wegen seiner Vollständigkeit fasziniert, so dass ich mich hinfort verpflichtet fühlte, auch die Belege der nächsten 9 Wagen abzuheften und aufzuheben. Als Datengrundlage für meine kleine statistische Auswertung dienen mir somit: Kaufverträge, Quittungen, Werkstatt- und Tankstellenrechnungen, Fotos, Steuer- und Versicherungsbescheide, Prospekte, Servicehefte sowie eigene Erinnerungen für insgesamt 13 eigene oder fast eigene PKW der Marken Wartburg, VW, Citroen, Mercedes und BMW.

Erstes Fahrzeug, Wartburg 311, Baujahr 1958

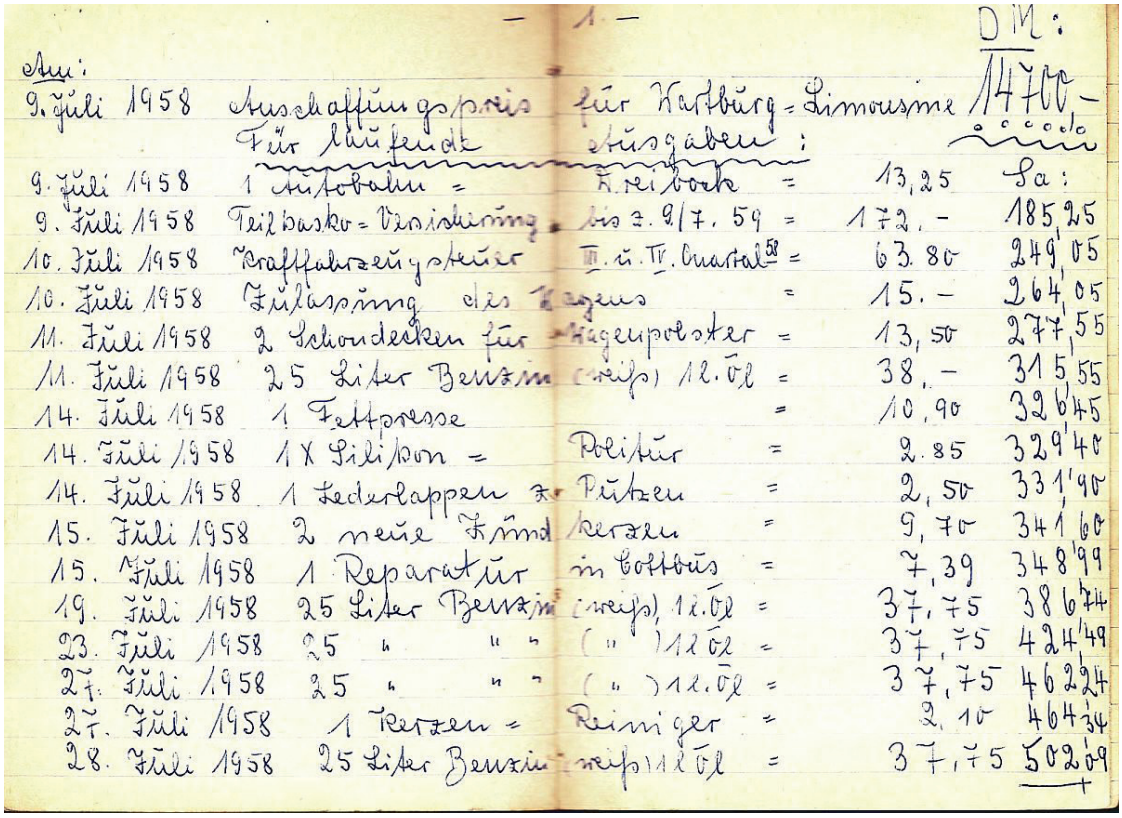
Wartburg 311, Baujahr 1958, technische Daten:

- 2-Takt, 3 Zylinder, Hubraum 900 ccm, wassergekühlt
- Leistung 37 PS bei 4000 U/min, 5 Sitzplätze, Frontantrieb
- Lenkradschaltung, Trommelbremsen überall
- Länge 4,30 m, Gewicht 1010 kg, Höchstgeschwindigkeit 115 km/h
- Neupreis 14.700 DM (Ost), Verbrauch 9 l, ermittelt 8,5

Abb. 2: **Wartburg 311, Baujahr 1958**

In mehreren Autobüchern wurde der Kauf des ersten Familienfahrzeugs sowie alle Fahrten und Reisen mit Fotos, Beschreibungen und kaufmännischen Aufzeichnungen über 10 Jahre lang festgehalten. Ab Kaufpreis wurden alle einzelnen Posten addiert und am Jahresende die Gesamtsumme gebildet. Da Abschreibungen nicht vorgenommen wurden, ergab es sich, dass der Wartburg im Laufe der Zeit *buchmäßig* immer wertvoller wurde. In Wirklichkeit wurde der Wagen nach 16 Jahren für immerhin die Hälfte des Kaufpreises verkauft. Eine derartig lange, DDR-typische Nutzungsdauer habe ich bisher trotz bester Vorsätze nicht mehr erreicht, der älteste Wagen war bisher ein 19 Jahre alter BMW, der leider auf Mallorca einem Unwetter zum Opfer fiel.

Man kann in diesem Autobuch unter anderem erkennen, dass beim Tanken für den 2-Takt-Motor des Wartburgs auch immer Öl im Verhältnis 1:25 mitgekauft wurde. Dieses musste man vor dem Einfüllen in den Tank mit dem Benzin gut durchmischen. Das Benzin wurde damals noch oft mit der Hand in eine große Blechkanne gezapft, gemischt und in den Tank gekippt.



Am:				DM:
9. Juli 1958	Anschaffungspreis für Wartburg-Limousine			14700,–
	Für laufende Ausgaben:			
9. Juli 1958	1 Autobahn =	Freibahn =	13,25	Sa:
9. Juli 1958	Teilbasko-Versicherung	bis 2. 9/7. 59 =	172,–	185,25
10. Juli 1958	Kraftfahrzeugsteuer	III. u. IV. Quartal 58 =	63,80	249,05
10. Juli 1958	Zulassung des Wagens	=	15,–	264,05
11. Juli 1958	2 Schondecken für	Wagenpolster =	13,50	277,55
11. Juli 1958	25 Liter Benzin	(weiß) 12. Öl =	38,–	315,55
14. Juli 1958	1 Fettpresse	=	10,90	326,45
14. Juli 1958	1 X Silikon =	Politur =	2,85	329,40
14. Juli 1958	1 Lederlappen zum	Putzen =	2,50	331,90
15. Juli 1958	2 neue Fund	kerzen =	9,70	341,60
15. Juli 1958	1 Reparatur	in Cottbus =	7,39	348,99
19. Juli 1958	25 Liter Benzin	(weiß) 12. Öl =	37,75	386,74
23. Juli 1958	25 " " "	(") 12 Öl =	37,75	424,49
27. Juli 1958	25 " " "	(") 12. Öl =	37,75	462,24
27. Juli 1958	1 Kerzen =	Reiniger =	2,10	464,34
28. Juli 1958	25 Liter Benzin	weiß) 12 Öl =	37,75	502,09

Abb. 3: ***Buchführung Wartburg im 1. Autobuch***

Abb. 4: ***Erster großer Familienausflug von Dresden nach Berlin, Stalinallee 1958***

Mein 2. und 3. Fahrzeug, VW Rechteckkäfer Baujahre 1958 und 1962

Noch vorhanden sind KFZ-Brief, Kaufvertrag, Versicherungspolicen, wenige Quittungen und die Verschrottungsbescheinigung für 50 DM. Es gibt deshalb fast keine Rechnungen, da in einem Jahr und mit 20 000 gefahrenen Kilometern nur 3 Verschleißteile zu ersetzen waren: Tachowelle für 7,50 DM, Batterie und Reparatur der Benzinleitung in Sevilla für umgerechnet 20 DM. Nachzufüllen waren nur Normalbenzin und etwas Öl und dann lief der Käfer fast bis Afrika.

Abb. 5: ***KFZ-Brief, erster Käfer 1972***

Volkswagen 1200, Baujahre 1958 und 1962, technische Daten:

- 4-Takt, 4 Zylinder, Boxer, Hubraum 1200 ccm, luftgekühlt
- Leistung 30 PS bei 3400 U/min, später 34 PS, 5 Sitzplätze, Heckantrieb
- Knüppelschaltung, Trommelbremsen überall
- Länge 4,09 m Gewicht 730 kg, Höchstgeschwindigkeit 110 km/h
- Neupreis 4.600 DM, Verbrauch 10 l

Abb. 6: Volkswagen 1200, Baujahre 1958 und 1962

Abb. 7: ***2. Käfer, Baujahr 1962***

Abb. 8: ***Robuste, wartungsfreundliche und langlebige Technik, VW Käfermotor:***

Unser 4. Fahrzeug, Citroen 2 CV 4, genannt Ente

Citroen 2 CV 4, technische Daten:

- In Frankreich meistverkauftes Auto, von 1948 - 1990 Stückzahl 5 Millionen
- der Ausdruck Ente geht auf ein wunderschönes Märchen des dänischen Dichters Hans Christian Andersen zurück, *das hässliche Entlein*
- 4-Takt Boxer, 2-Zylinder, Hubraum 431 ccm, luftgekühlt,
- Leistung 23 PS bei 7000 U/min (Geräusch, Schnattern), 4-5 Sitzplätze, Frontantrieb, Farbe weiß
- Länge 3,83 m, Gewicht 650 kg, Höchstgeschwindigkeit 110 km/h
- Neupreis 3.800 DM, Verbrauch 6 l

Abb. 9: 4. Fahrzeug, Citroen 2 CV 4

Über die gesamte Nutzungsdauer der französischen Ente wurde von Gertud K. ein Aktenordner geführt, in dem alle Belege chronologisch gesammelt wurden. Dieser Ordner wurde von mir 47 Jahre später nach verschiedenen Gesichtspunkten ausgewertet und diente als Vorbild für die Langzeit Sammlung der Belege für unsere weiteren 9 Wagen.

Weil einer der beiden Zylinder des luftgekühlten Boxermotors nach 5 ½ Jahren keine ausreichende Kompression mehr hatte, wollten wir keinen Zylinderkopf wechseln und so musste die Ente leider auch den Besitzer wechseln. Ein modernes, leiseres und sicheres Auto sollte es hinfort sein.

Abb. 10: Rechnungsordner und Auswertung

5. Fahrzeug, Neuwagen VW Audi 50, Baujahr 1975, eine moderne Konstruktion

Technische Daten:

- Eine der schnellsten Entwicklungen in der Automobilgeschichte, nur 21 Monate
- 4-Taktmotor Reihenmotor, quer, Hubraum 1093 ccm, wassergekühlt
- Leistung 50 PS bei 5800 U/min, 5 Sitzplätze, Frontantrieb
- Länge 351 cm, 685 kg Gewicht, Farbe Lofotengrün, Höchstgeschwindigkeit 142 k/h
- Neupreis 8.400 DM, Verbrauch 7 l, Sicherheitsgurte, in Wolfsburg gebaut.

Abb. 11: VW Audi 50, im Tal der Rosen in Bulgarien

Der Audi 50 war als Kleinwagen für die Stadt gut geeignet, als Fahrzeug für die große Reise und für südliche Länder wohl eher nicht. Die Freude an meinem ersten Neuwagen wurden bald schon getrübt. Von der Zeitschrift *Auto Motor Sport* war der Audi 50 damals hochgelobt, angeblich noch deutlich besser als der etwas größere Golf, dessen Entwicklung 4 Jahr gedauert hat. Erstmalig sind serienmäßig Sicherheitsgurte an Bord und die Ergebnisse beim Crashtest sind um Klassen besser als jene der Ente, die bei diesem damals noch wenig gebräuchlichen Test sehr schlecht abgeschnitten hatte. Da der Audi nur kleine Räder hatte, war der Reifenverschleiß ungewöhnlich hoch. Außerdem waren Lichtmaschine (noch in der Garantie), Zylinderkopf und Kupplung schnell verschlissen. Der quer eingebaute Motor erwies sich als nicht ausreichend hitzefest.

6. Fahrzeug VW Passat LS Variant, der letzte Neuwagen

Teschnische Daten:

- 4-Takt Reihenmotor, längs, Hubraum 1600 ccm, wassergekühlt
- Leistung 85 PS bei 5600 U/min, 5 Sitzplätze, Frontantrieb
- Länge 426 cm, Gewicht 920 kg, Höchstgeschwindigkeit 173 km/h
- Neupreis 13.800 DM, Verbrauch 7 l, Farbe alpinweiß
- Starke Korrosion, dünnes Blech. kupferhaltiger Recyclingstahl oder DDR- Stahl

Abb. 12: Ein Passat Array[1]

1 Entnommen dem VW-Katalog Passat, Seite 25

Leider mussten wir uns nach knapp 7 Jahren und 133.700 km schon von unserem Passat LS Variant verabschieden, da der Wagen die TÜV-Prüfung wegen starker Durchrostung nicht bestanden hat. Wie man erst viel später erfuhr, litten nicht nur VW- Fahrzeuge der 70er Jahre mehr oder weniger unter interkristalliner Korrosion[2] von verunreinigtem Recyclingstahl. Außerdem war wohl der Passat Variant im Jahr 1978 mit 920 kg mit Sicherheit zu leicht gebaut, unser aktueller BMW wiegt fast das Doppelte, aber hält auch schon jetzt mehr als doppelt so lange. Von den traurigen Erlebnissen frustriert, suchten wir nun nach einem solideren Gefährt.

7. – 9. Fahrzeug Mercedes 123er Diesel Limousine

Technische Daten:

- Der meistgebaute Mercedes PKW aller Zeiten
- 4-Takt Reihenmotor, längs, Hubraum 2000 bzw. 2400 ccm, wassergekühlt
- Leistung 60 bzw. 72 PS bei 4400 U/min, 5 Sitzplätze, Heckantrieb
- Länge 472 cm, Gewicht 1400 kg, Höchstgeschwindigkeit 143 km/h
- Preis gebraucht 4.000 bis 11.000 DM, Verbrauch 9 l, Farbe weiß
- 2 Jahre Lieferfrist

Ab 1975 liefen 2,7 Millionen dieses Fahrzeugtyps vom Band. Das Schließen der Türen verglich man mit dem Klang eines Tresors. Und so bekam man auch beim Fahren das Gefühl der Sicherheit eines Tresors. Andererseits nannte man den Diesel auch Wanderdüne. Unser erster Mercedes 240 D war aus dem Jahr 1978 und damit genauso alt wie unser ohne TÜV und nur mit viel Überredung verkaufter Passat, der Kaufpreis war aber 1985 noch stolze 11000 DM. In unserer Wohnanlage hatten wir aber seitdem mit diesem weißen Wagen deutlich an Ansehen gewonnen. Die Wartungsfreundlichkeit dieses Typs war sehr hoch, so dass ich nach etwas Einarbeitungszeit mit einem Handbuch den Service fast komplett selbst machen konnte. Der Service bestand hauptsächlich aus drei Positionen: Motoröl und Bremsklötzer wechseln sowie Ventile einstellen. Der Verschleiß war insgesamt erfreulich gering. Leider blieben auch die Mercedes Wagen aus dem Baujahr 1978 nicht von der typischen Korrosion der 70er Jahre verschont, sodass wir es leider auch bei unserem ersten 123er Mercedes noch erleben mussten, dass der TÜV bei einem Fahrzeugalter von 14 Jahren die Weiterfahrt in Deutschland untersagte. Heute sind Wagen dieses Typs mehr denn je gefragt. Zitat *„ Die halbe Welt lechzt nach gebrauchten W123, am liebsten als Diesel.“*[3] Einen Vorgeschmack dieser Nachfrage konnten wir schon 2002 erleben, als wir unseren letzten Mercedes nach 6 Jahren und 120.000 km auf dem Automarkt fast zum ursprünglichen Kaufpreis losgeworden wären. Doch mein Sohn fand das sittenwidrig, sodass wir uns großzügig mit der Hälfte zufriedengaben.

2 Zeitschrift Oldtimer Markt, Januar 2023, Seite 14

3 Zeitschrift Oldtimer Markt Heft 1/2023, Seite 25

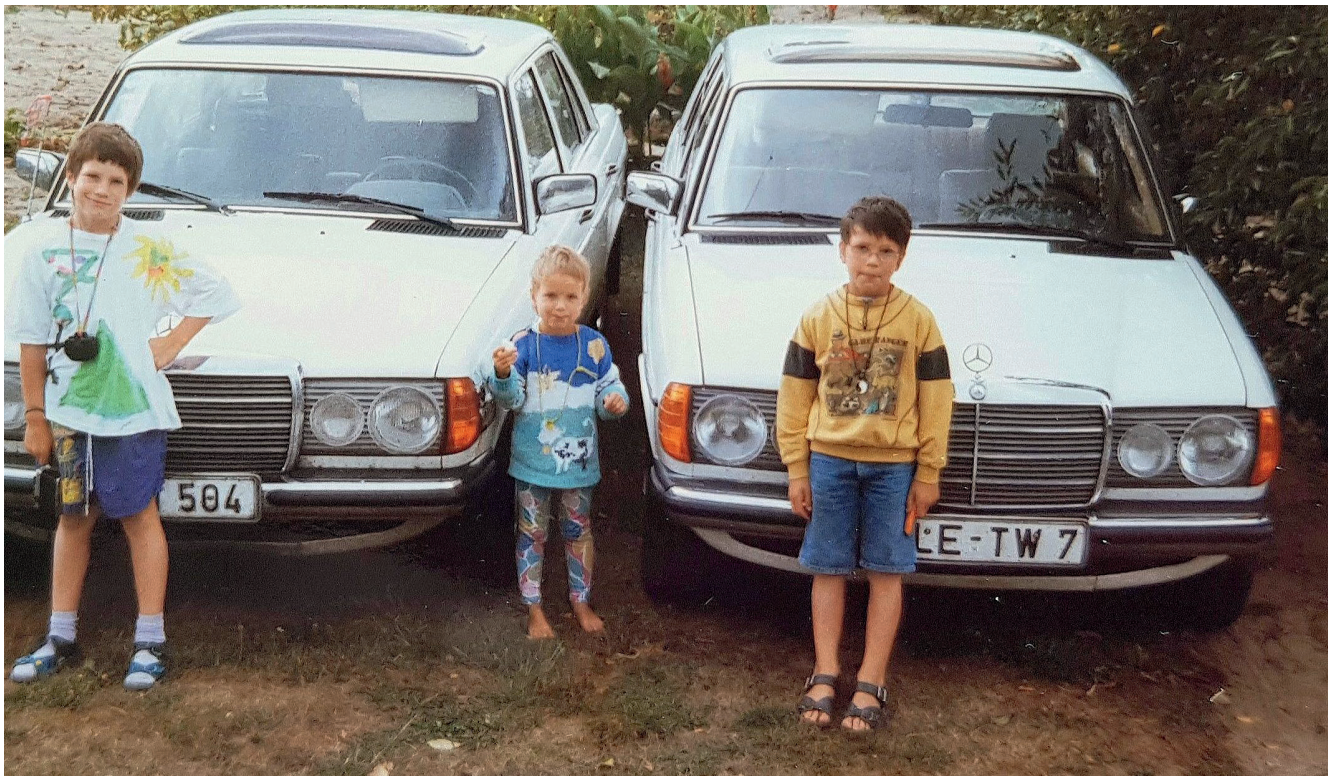


Abb. 13: Mercedes Wechsel im Jahr 1992, Baujahr 1978 gegen 1981

Von diesem letzten Mercedes wollten wir uns – zur Abwechslung mal ohne amtliche TÜV-Zwangsmaßnahmen - trennen, da die Kfz-Steuer für Diesel PKW mittlerweile in schwindelnde Höhen gestiegen war. Außerdem waren in Berlin im Frühjahr 2002 gebrauchte Mercedes PKW kaum zu bekommen, so dass sich ein Marken und Kraftstoffwechsel empfahl. Für Benziner gab es im Interesse der Umwelt eingeführt einen deutlich reduzierten Steuersatz.

Das 10. – 12. Fahrzeug BMW 5er Reihe E34

Technische Daten:

- Beginn der Entwicklung 1981, Produktion seit 1988, bessere Aerodynamik
- 4-Takt Reihenmotor, längs, Benzin, Hubraum 2000 ccm, wassergekühlt
- Leistung 150 PS bei 4000 U/min, 5 Sitzplätze, Heckantrieb
- Länge 472 cm, Gewicht 1445 kg, Höchstgeschwindigkeit 211 km/h
- Neupreis 45.000 – 71.00DM, Verbrauch 9 l, Sicherheitsgurte auch hinten und viele Airbags

Die Umgewöhnung von einem mit etwas LKW-Feeling zu fahrenden Diesel-Daimler auf einen behände und fast spielerisch zu bedienenden BMW-Benziner fiel sehr leicht, auch der Verbrauch war bei höherer Geschwindigkeit geringer. Auch Wartungsaufwand und Reparaturbedarf waren bei diesem Modell erfreulich gering. Dank der modernen Hydrostößel musste ich auch die Ventile nie mehr einstellen.

Abb. 14: 10. – 12. Fahrzeug BMW 5er Reihe E34

Von unserem letzten E34 Touring mussten wir uns sehr seines Unwetter-Totalschadens wegen auf Mallorca 2015 schmerzvoll trennen und von einer bis dahin wunderschönen PKW-Campingreise, die eigentlich bis nach Südspanien geplant war, mit dem Flugzeug zurückkehren. Zunächst bis Süddeutschland und dort musste unter großem Zeitdruck ein Ersatzwagen gefunden werden. Weder der der bewährte Typ E34 noch das Nachfolgemodell E39 waren wegen starker Konkurrenz aus der Schweiz zu bekommen und so sprangen wir mit dem Baujahr von 1996 bis 2004, immerhin ein Neupreisspung von 71.000 DM auf 61.000 € innerhalb von 8 Jahren.

Der ultimative 13. Wagen BMW 5er Reihe E61 Touring

Technische Daten:

- Eine Besonderheit des E61 ist der gewichtsreduzierte Aluminium-Vorderwagen, wodurch eine Achslastverteilung von 50 : 50 erreicht wurde, außerdem Korrosionsfestigkeit
- 4-Takt Reihenmotor, längs, Diesel, Hubraum 2500 ccm, wassergekühlt
- Leistung 177 PS bei 4000 U/min, 5 Sitzplätze, Heckantrieb
- Länge 484 cm, Gewicht 1750 kg, Höchstgeschwindigkeit 225 km/h
- Neupreis 45.000 €, Verbrauch 7 l, Bordcomputer, Navigation und sehr viele Airbags,

Abb. 15: 13. Wagen: BMW 5er Reihe E61 Touring

Abb. 16: Übersicht der Fahrzeuge von 1957 bis 2022

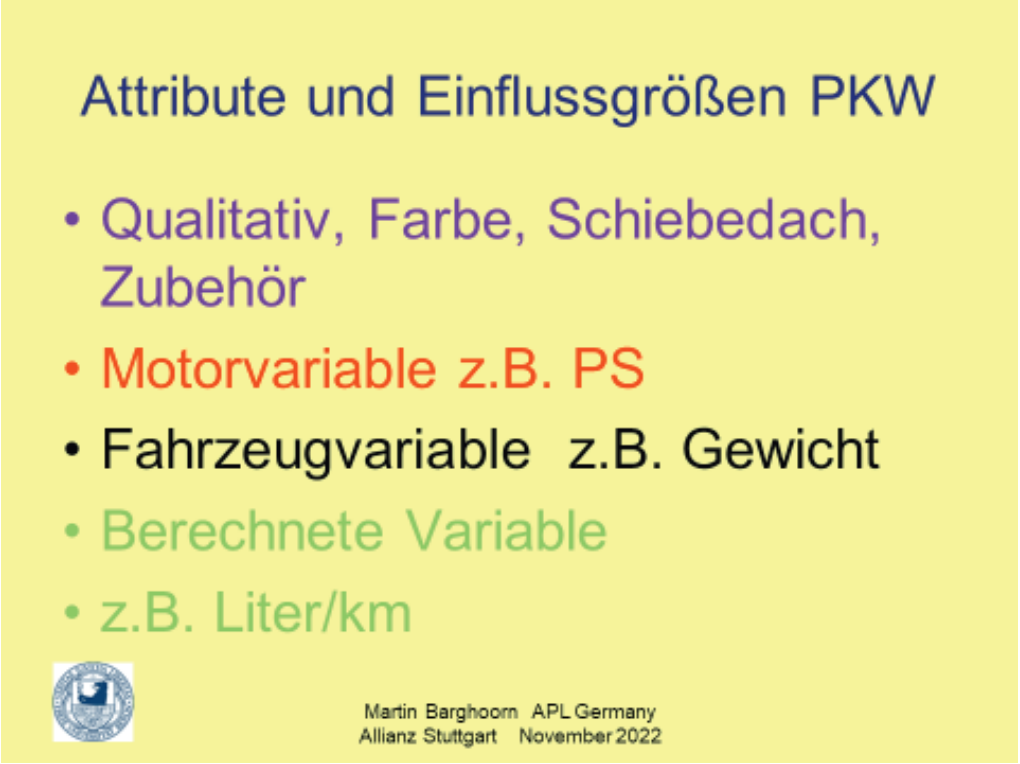


Abb. 17: Auswertung der Fahrzeugdaten

Die Auswertung (siehe Tabellen) wurde mit APL2 gemacht und die Graphiken mit R erstellt.

APL2 1001 - Object Editor - autos

Object Edit Signals Options Windows Help

0	1	2	3	4	5	6	7	8	9	10	11	12
	nick_name	Sprit	Baujahr	Farbe	drive	safety	Zylinder	Hubraum	power_PS	Drehzahl	top_speed	Verbrauch
1 Wartburg 311	Wartburg	Benzin	1958	elfenbein	Front	no	3	900	37	4000	115	8.5
2 VW 1/11 blau	Kaefer1	Benzin	1958	capriblau	Heck	no	4	1200	30	3400	110	10
3 VW weiss	Kaefer2	Benzin	1962	perlweiss	Heck	Gurte	4	1200	34	3400	115	10
4 Citroen 2CV4	Ente2CV	Benzin	1970	weisser Meije	Front	no	2	431	23	7000	111	6
5 Audi 50 Kombi	Audi50	Benzin	1975	lofotengruen	Front	Gurte	4	1100	50	5800	142	7
6 VW Passsat Variant	Passat	Benzin	1978	alpinweiss	Front	Gurte	4	1600	85	5600	173	8
7 Mercedes 240D weiss	W123	Diesel	1978	classicweiss	Heck	Gurte	4	2400	72	4400	143	9
8 Mercedes 200D	W123	Diesel	1981	classicweiss	Heck	Gurte	4	2000	60	4400	135	9
9 Mercedes 240D blau	W123	Diesel	1984	surfblau	Heck	Gurte	4	2400	72	4400	143	9
10 BMW 520i gruen	E34	Benzin	1992	barbadosgruen	Heck	airbag	6	2000	150	5900	211	9
11 BMW 520i weiss	E34	Benzin	1993	alpinweiss	Heck	airbag	6	2000	150	5900	211	9
12 BMW 520i Touring	E34_Tour	Benzin	1996	technoviolett	Heck	airbag	6	2000	150	5900	205	9
13 BMW 525D Touring	E61_Tour	Diesel	2004	orientblau	Heck	airbag	6	2500	177	4000	225	7
14 Gesamt							57	21731	1090	64100	2039	110
15 Mittelwert							4	1672	84	4931	157	8

APL2 1001 - Object Editor - autos

Object Edit Signals Options Windows Help

0	13	14	15	16	17	18	19	20	21	22	23	24
	Gewicht	Laenge	mileage	mileage_total	use_mo	Steuer_Vers	Service	extraord	Wertverlust	KM_pro_Mo	Sprit_Preis	Sprit_Kost
1	1010	430	121000	121000	192	150	283	0	225	631	76	489
2	730	407	20000	115000	10	286	58	10	125	2000	30	720
3	740	407	40000	125000	12	250	450	0	375	3333	30	1200
4	650	383	110000	110000	66	154	132	88	320	1667	36	432
5	685	351	60000	60000	30	300	155	160	740	2000	43	722
6	920	426	133700	133700	83	415	205	100	875	1611	60	928
7	1400	472	151000	224252	79	522	266	100	730	1911	51	1053
8	1400	472	81500	193500	48	470	250	75	538	1698	56	1027
9	1400	472	120136	344136	68	500	230	85	166	1767	67	1278
10	1445	472	125000	278000	76	800	400	115	465	1645	120	2132
11	1445	472	95000	195000	57	750	360	130	242	1667	144	2592
12	1570	480	41000	132000	29	730	400	77	931	1414	146	2229
13	1750	484	101000	234000	86	1186	1100	326	850	1174	194	1914
14	15145	5728	1199336	2265588	836	6513	4289	1266	6582	22518	1053	16716
15	1165	441	92257	174276	64	501	330	97	506	1732	81	1286

Tab. 1: Technische und wirtschaftliche Daten von 13 Fahrzeugen im Vergleich

Für die Auswertung wichtiger Einflussgrößen wurde die Daten entsprechend aufbereitet.

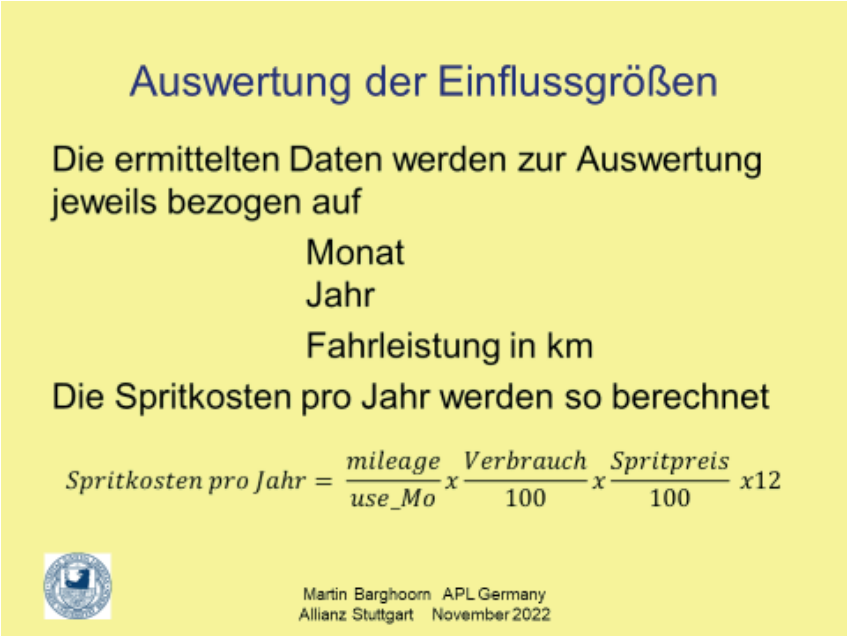


Abb. 18: Auswertung wichtiger Einflussgrößen

Die berechneten und in der Tabelle genannten Mittelwerte bzw. Summen sind jeweils nicht immer beide sinnvoll.

Um die Wirtschaftlichkeit und Qualität der Fahrzeuge besser beurteilen zu können, wurde aus allen Rechnungen die Positionen der Teile und Lohnkosten herausgezogen, die nicht in den Katalog der Verschleißteile fallen und so die Variable *extraordinary* definiert. Die Werte sollten hier möglichst klein sein, an besten 0, wie es beim Wartburg und dem 2. Käfer der Fall war.

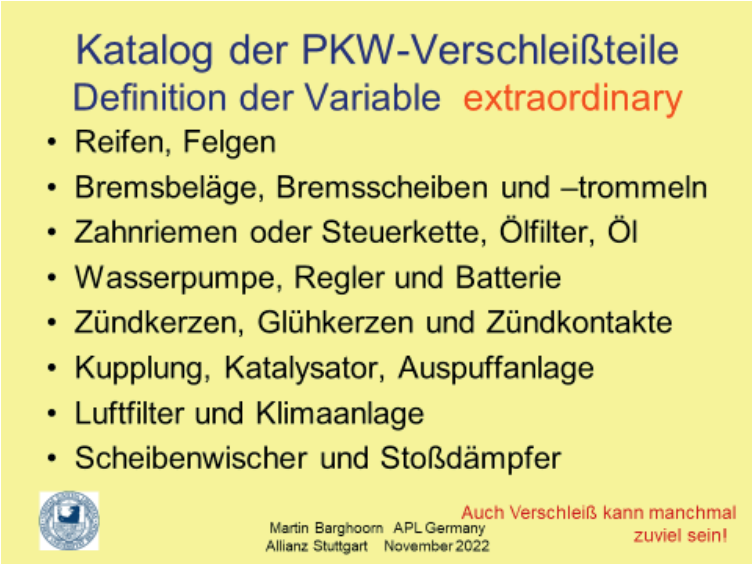


Abb. 19: Katalog der PKW-Verschleißteile

Alle wirtschaftlichen Daten vor 2002 wurden in Euro umgerechnet und die für den Wartburg gültige DDR-Mark der DM gleichgesetzt. Die Gesamtkosten pro Fahrzeug und Jahr wurden als Summe von Service, extraordinary, Wertverlust und Sprit_Kost berechnet. Der Sprit Preis bezieht sich auf den Zeitraum (Mittelwert), in dem das Fahrzeug genutzt wurde. Eine Ausnahme ist der Dieselpreis bei E61, wo ich den Durchschnitt des Jahres 2022 genommen habe.

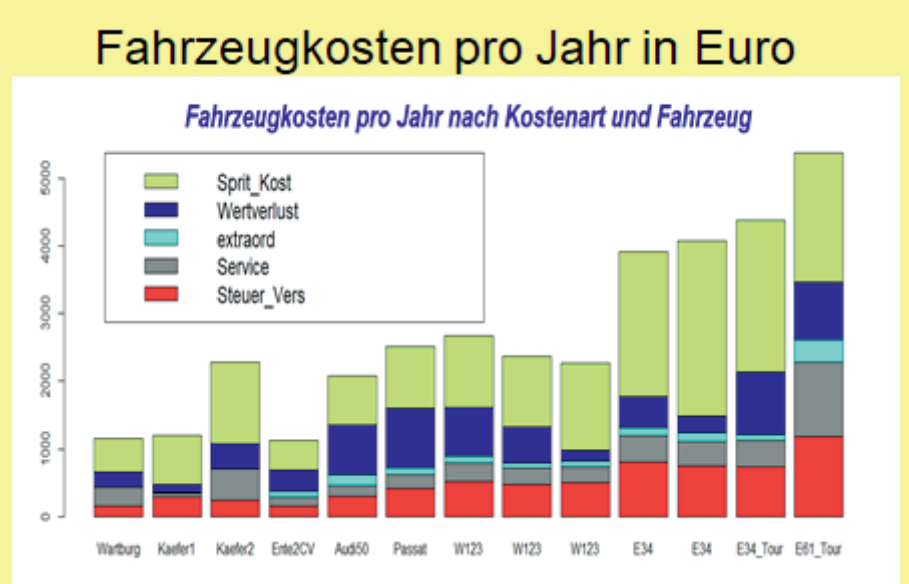


Abb. 20: Fahrzeugkosten pro Jahr in Euro

Beim Vergleich der Fahrzeugkosten pro Jahr ist zu beachten, dass hier die pro Jahr gefahrenen km von einfließen. Im Durchschnitt waren es der Mittelwert der Variable KM_pro_Mo aus der Tabelle mal 12, also 20.785 km. Beim 2. Käfer waren es aber besonders viel, nämlich 40.000 km. Das beim einem durchschnittlichen Benzinpreis von 30 Cent pro Liter. Die Variable Sprit_Kost enthält die absoluten Kosten der Fahrzeuge pro Jahr in €, hängt ab von den gefahrenen km und vom Spritpreis. In der nächsten Graphik sind die Kostenarten anteilig in Prozent wiedergegeben.

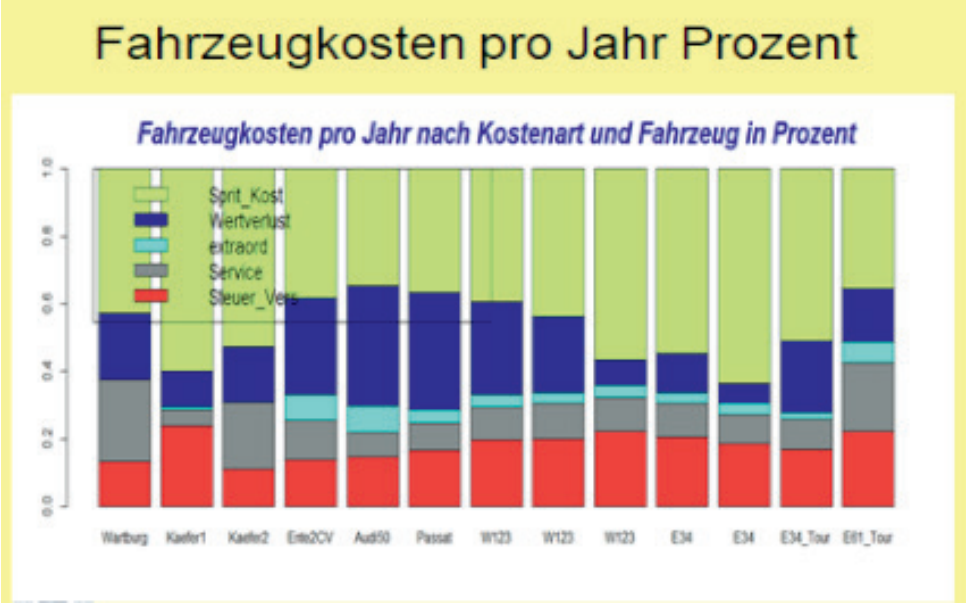


Abb. 21: Fahrzeugkosten pro Jahr in Prozent

Besonders auffällig ist der relativ höhere Anteil des Wertverlustes bei den Neuwagen Citroen Ente, Audi50 und Passat. Absolut der höchste Wertverlust war leider beim BMW 520i Touring mit 931 € zu verzeichnen (Totalschaden). Aktuell liegt der Wertverlust eines Autos in Deutschland gemäß Daten von Check24 nach 10 Jahren bereits bei etwa 80 Prozent des Neupreises.

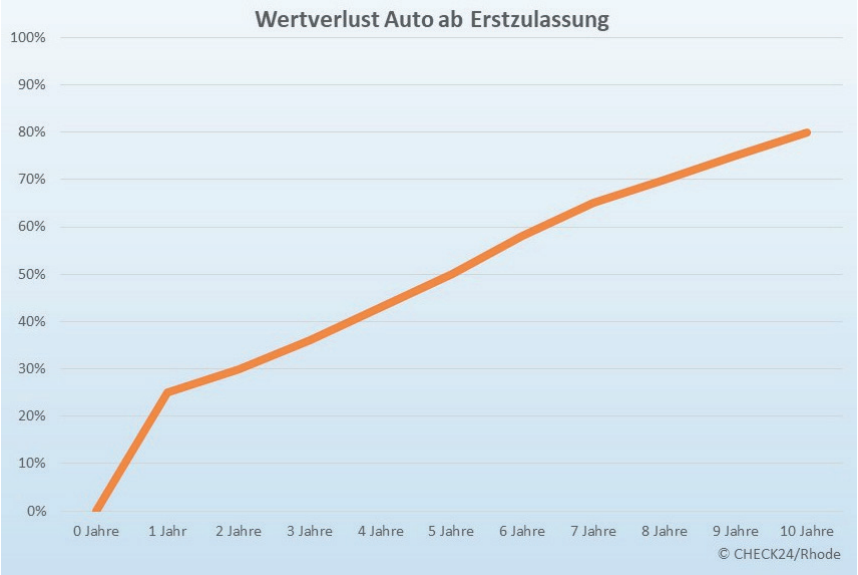


Abb. 22: Wertverlust Auto ab Erstzulassung (Quelle: DAT)

Somit beträgt der durchschnittliche jährliche Wertverlust bei modernen Mittelklassewagen, bei denen man leicht einen Neuwagenpreis von 60.000 € erreicht, nach 10 Jahren und dem genannten Wertverlust von 80 Prozent 4.800 €. Bei der vergleichenden Betrachtung der spezifischen Fahrzeugkosten pro km ergibt sich folgendes Diagramm.

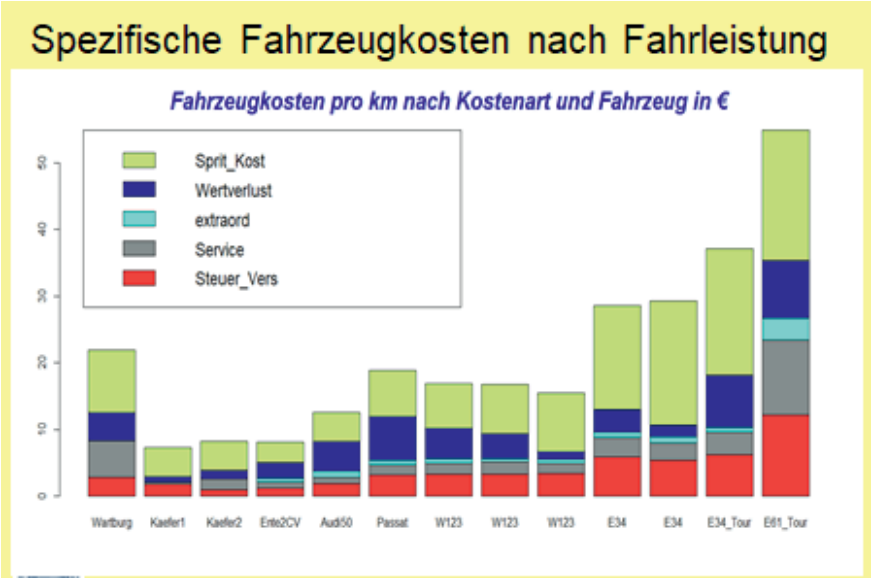


Abb. 23: Spezifische Fahrzeugkosten nach Fahrleistung

Es lässt sich ein Anstieg der Kfz-Gesamtkosten in Deutschland in den letzten 65 Jahren von unter 7 Cent bis über 55 Cent pro km beobachten, also ein Anstieg um mehr als das 7-fache. Diesen Anstieg

maßgeblich beeinflusst hat der Anstieg der Kraftstoffpreise in diesem Zeitraum um mehr als das 6-fache. Dabei ist zu bedenken, dass mein erster eigener Wagen in preiswerter Studenten-Käfer war und unser aktueller Wagen ein 18 Jahre alter Mittelklasse BMW ist, der im November 2022 problemlos den TÜB absolviert hat. Weiterhin erkennt man in der Graphik, dass das Autofahren mit einem Wartburg in der DDR damals schon relativ teuer war, der Benzinpreis war im Zeitraum immer konstant 1,52 Mark.

Abschließend wird das Leistungsgewicht der 13 Fahrzeuge untersucht werden. Das Leistungsgewicht ist definiert als der Quotient der Masse und Leistung eines Fahrzeugs, eines Motors, eines Akkus oder einer Maschine. Bei Antrieben wird im Allgemeinen ein geringes Leistungs-gewicht angestrebt. So erlaubt ein geringeres Leistungsgewicht eines Fahrzeuges eine bessere Beschleunigung oder eine höhere Geschwindigkeit an Steigungen.

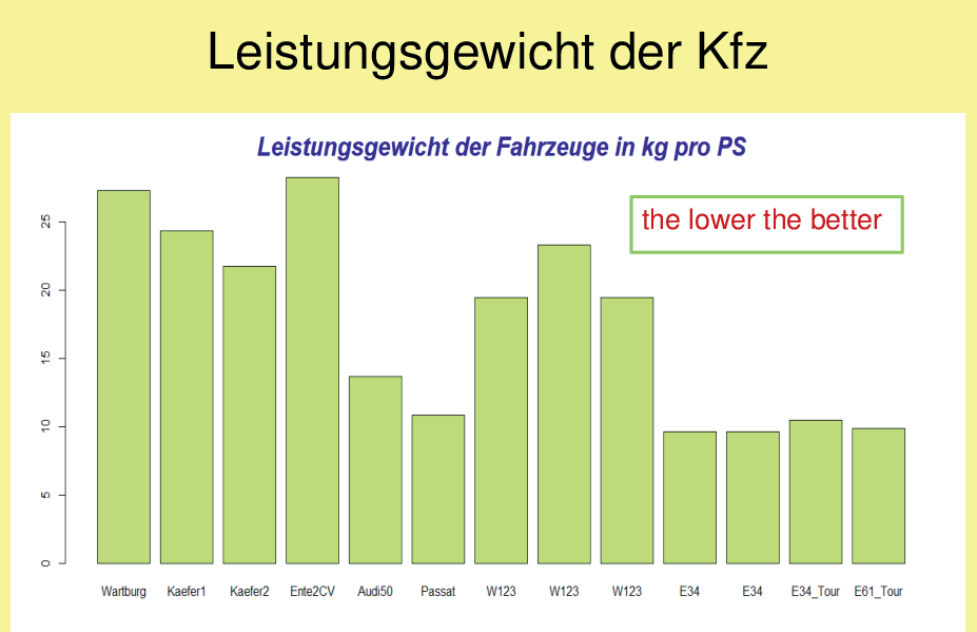


Abb. 24: Leistungsgewicht der Kfz

Wie die Graphik zeigt, hat sich das Leistungsgewicht über den Zeitraum deutlich verbessert. Obwohl das Gewicht der Autos stark zugenommen hat, konnten gleichzeitig die Motoren so verbessert werden, dass sie mehr Leistung bei geringerem Verbrauch erzielen.

Mit der Gewichtszunahme konnte die Lebensdauer erhöht sowie Komfort, Sicherheit und Umweltschutz der PKW in hohem Maße verbessert werden.

Resümee

Die Bearbeitung des Themas ist zu einem etwas größeren Projekt geworden. Die aufwendige Datenerfassung und –aufbereitung mündete in eine spannende Auswertung, die für mich zu überraschenden und spannenden Resultaten geführt hat. Die Haltung und Nutzung von privaten PKW ist deutlich teurer geworden, doch bei Berücksichtigung diverser Einflussfaktoren lässt sich eine Optimierungsstrategie entwickeln. Diese muss natürlich immer wieder überprüft und angepasst werden. Hierfür die Grundlage ist die Datenerfassung, zu welcher ich Sie ausdrücklich ermuntern möchte.

Acronyms

Dr. James A. Brown
with contributions from Bob Smith, Roger Hui, Google, Wikipedia, and LOOP

Why do we care about Acronyms?

Names are really important. If you want to be famous, don't give names to your results. In mathematics, there is a procedure call the "Simplex Method". Do you know who invented it? (I believe it's George Dantzig.) Have you ever heard of him? Probably not because he gave his result a name.

If I invent a new method, I'll call it Theorem 1. Then people will refer to it as Brown's Theorem and I'll be famous.

I got interested in the topic of acronyms because a current project I'm on which needed a name and we wanted an Acronym. I'll discuss this topic at the end of this paper.

What is an Acronym?

I make the claim that Acronym is an Acronym for *"Alphabetical COde for Remembering Odd Names You Make up".* I'm not sure anyone believes this.

Styles of Acronyms

Not everyone agrees that all the examples I give should be called Acronyms. You be the judge. Also, the categories I list here are not strict or mutually exclusive. Some abbreviations might be put in several different categories. I made a choice.

► **Pronounceable as a real word**

NASA: The National Aeronautics and Space Administration
Scuba: Self-Contained Underwater Breathing Apparatus
Sonar: Sound Navigation And Ranging
Radar: Radio Detection And Ranging
Laser: Light Amplification by Stimulated Emission of Radiation
DARPA: Defense Advanced Research Projects Agency
SNAFU: Situation Normal, All F*cked up
Gestapo: Geheime Staatspolizei
Posh: Port Outward, Starboard home
AWOL: Absent WithOut Leave

Note: LOOP is an acronym for "Lots Of Other People"

► Pronounceable as a real word – Computer / APL Related

APL: A Programming Language (if you pronounce it “apple”)

(Is it a coincidence that “APL Plastics Limited” is near the Basingstoke, UK office of Dyalog APL? I don’t think so.)

Source: http://www.aplplastics.com/

Coyote: Call Off Your Old Tired Ethics

In southern San Jose, California is an area called the Coyote Valley which was to be developed as a high-tech area. When IBM decided to build there, they initially called the lab “IBM Coyote”. Then they found out that the organization of prostitutes in San Francisco used the Coyote acronym listed above. This organization held a big party for its members every year on the night of the summer solstice. It’s the shortest night of the year and they would lose the least business. IBM changed the name to the “IBM Santa Teresa Labs” then more recently “IBM Silicon Valley.

GUIDE: Guidance for Users of Integrated Dataprocessing Equipment

GUIDE was a user’s group for users of IBM computer systems.

SHARE: Society to Help Avoid Redundant Effort

SHARE was a user’s group for users of IBM computer systems.

RISC: Reduced Instruction Set Computing

Because RISC is pronounced RISK and Risk is considered bad, a lot of people objected to this acronym.

GIF: Graphics Interchange Format

SPOOL: Simultaneous Peripheral Operations OnLine

► Initialism - not pronounceable

EU - European Union

FBI - Federal Bureau of Investigation

HTML - HyperText Markup Language

URL - Uniform Resource Locator

APL - A Programming Language (if you don’t pronounce it “apple”)

PS - PostScript

- **Initialism – pronounceable (sort of)**

  IOU - I Owe You
  K9 - Canine
  OK – Okay

- **Some people pronounce the word some say the letters**

  FAQ: Frequently Asked Questions
  SQL: Structured Query Language

- **Mixture of saying letter and word**

  JPEG: Joint Photographic Experts Group
  CD-ROM: Compact Disc Read-Only Memory
  MS-DOS: Microsoft Disc Operating System

  These last two could be considered Second Level Acronyms since both ROM and DOS are already acronyms.

- **Second (third?) Level Acronyms (acronyms that refer to other acronyms) Almost always computer related.**

  HASP: Houston Automatic Spooling Priority
  ARM: Advanced RISC Machine
  Power PC: Performance Optimization With Enhanced RISC
  GIMP: GNU Image Manipulation Program
  ATFUSTOLC: IBM APLers show their age: APL Transfer Form UnderScore TO Lower Case

  IBM APL2 on the mainframe can have capital letters and underscored capital letters. To move a mainframe APL2 workspace to a workstation you use)OUT which produces an APL Transfer File (ATF). The workstations use Upper and Lower case so the functions in ATFUSTOLC convert the underscored letters to lower case.

- **Recursive Level Acronyms - Almost always computer related.**

  SYBIL: Sybil Your Burroughs Implementation Language
  WINE: Wine is not an Emulator
  GNU: GNU's Not Unix

- **Used as mnemonics**

  ROY G. Biv: red-orange-yellow-green-blue-indigo-violet Colors of the rainbow

- **USe more than iniTIal letter**

INTERPOL: INTERnational criminal POLice organization
AIDS: Acquired ImmunoDeficiency Syndrome

- **Initials but last word repeated**

ATM Machine
LCD Display
PIN Number
AC & DC Current
RAM Memory
HIV Virus

- **Backronyms - Word came first then acronym later**

Apple Lisa: Local Integrated Software Architecture
BOOK: Bound Offline Organized Knowledge
I think this is my favorite. It's a wonderful app. You don't need to charge it ever – it never runs out of power. You can manually change the page you're looking at. You can add annotations without clicking any buttons
TWAIN: Technology without an interesting name
This was not in my original presentation but came up in discussions. It is really not an acronym. TWAIN manages communication between software and digital imaging devices. The word is from Kipling's "The Ballad of East and West" — '...and never the twain shall meet...'. But people thought it was an acronym because it was always spelled with uppercase letters so one had to be made up.

- **Notable Acronyms**

At Minnowbrook: Manugistics project for new APL
INCA: It's Not Called APL
Longest:
ADCOMSUBORDCOMPHIBSPAC: Administrative Command, Amphibious Forces, Pacific Fleet Subordinate Commandf
Really a word?:
WYSIWYG: What You See Is What You Get
Common:
IDSFA: It doesn't stand for anything

► Poorly Chosen Acronyms

SHIT: System for Highspeed Integrated Test

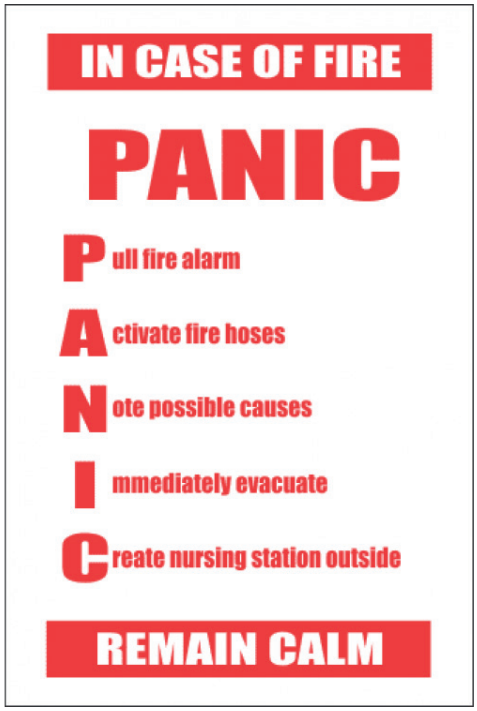
IBM Federal Systems, Owego, New York

My first job at IBM in 1965 was writing software for wiring integrated circuit boards. The device used to test the circuit boards was initially called SHIT and the owners of the system couldn't see any other meaning. We had to point it out to them. Then they changed the name to Highspeed Integrated Test System (HITS).

S.L.U.T: South Lake Union Trolley (since 2007: Officially, it's the South Lake Union Streetcar.)

They did not change this *(Source: http://ridetheslut.com/).*

PANIC: In case of fire, PANIC



If the purpose is to tell you to "REMAIN CALM". PANIC is a terrible name. This is pretty common though. Many times I've been in elevators where there's a sign saying "In case of malfunction, do not become alarmed. Press the button labeled ALARM!

© Safety Signs & Equipment (Pty) Ltd.

S.H.I.T.: Servicio de Hosteleria Industrial de Terrassa

In all fairness, this sign is not in English so it is forgivable, Also, since they use periods after the letters, maybe it's not an acronym at all.

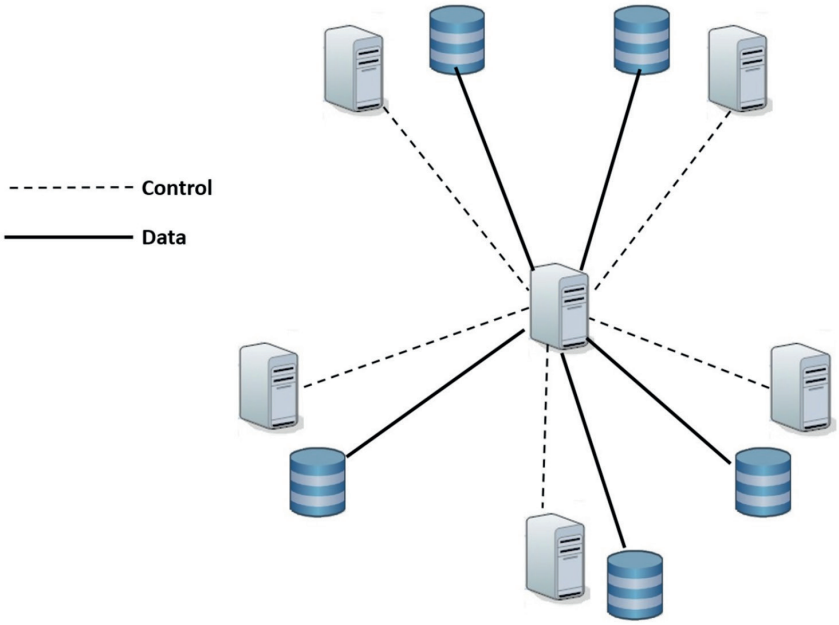
Conclusion

- If you're doing something new, you need a new word
- That doesn't mean anything currently
- And stands for a phrase that describes what you do

"You can't take over the world without a good acronym"

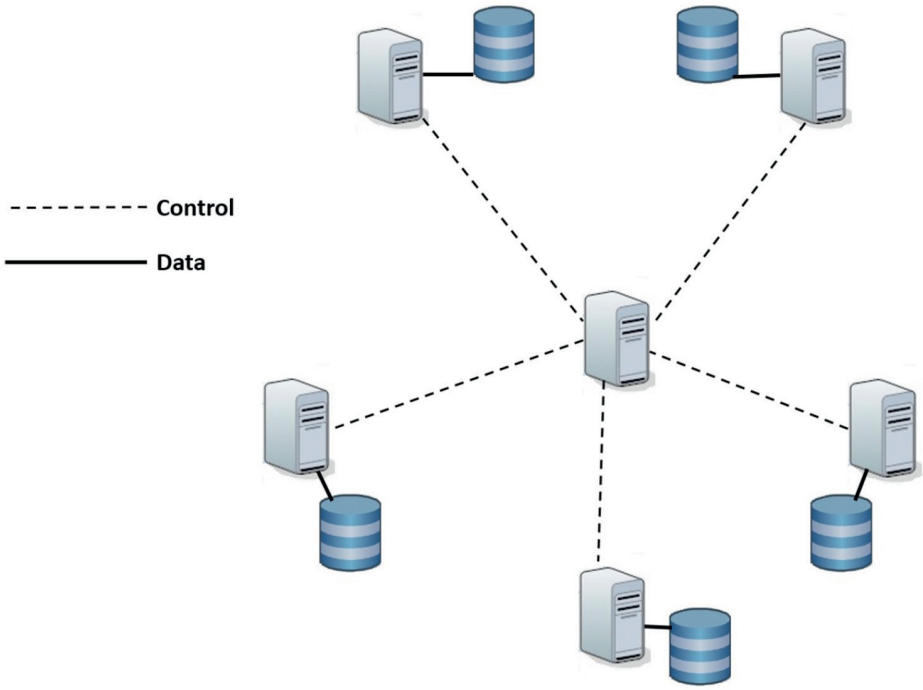
International Microsystems and NestedComputing have designed a new machine and it needed an acronym. The machine has a manager machine and a set of independent processors called nodes which can be viewed two ways.

View 1: Storage for all nodes attached to manager:



In this view, data on the storage devices look like a local database.

View 2: Storage for nodes attached to the node:



In this view, data on the storage devices look like a distributed database. A hardware switch between each node and the manager changes the view in an instant.

We needed a word to describe data which is sometimes local and sometimes distributed. We call such data “Disjoint Data”. The machine is then called the “Disjoint Array Machine” or **DAC.**

This machine with this acronym has the possibility of changing the way computers do parallel processing.

Contact:

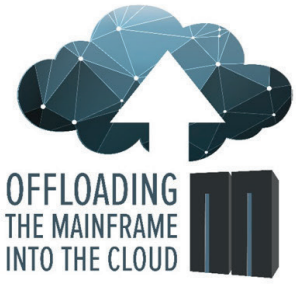
Dr. James A. Brown
e-mail: Jim.Brown@branr.com

APL2PLUS ™

A NEW MAINFRAME OFFERING

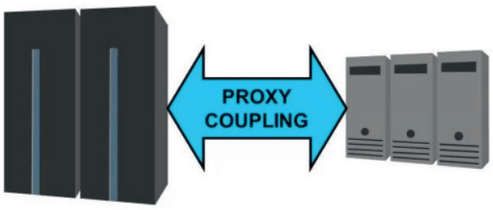
Dr. James A. Brown

Proximal Systems Corporation



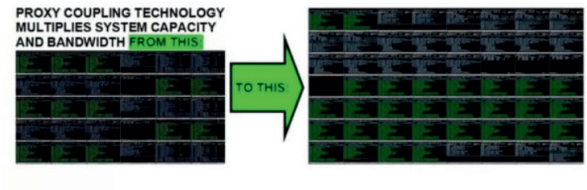
Move CPU resources to lower-cost platforms

Proxy Coupling Technology

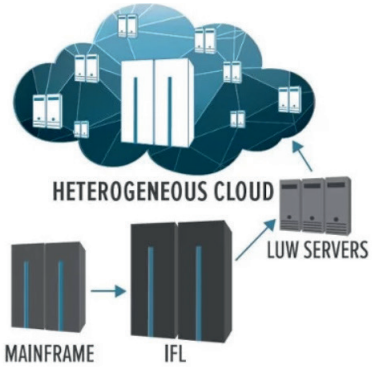


Couple zSeries computing to lower-cost hardware

PSCsort ™



Offloads DFSORT or Syncsort to zLinux.
An example of coupling:



The Proxy Coupling Facility is an environment for monitoring, managing and transparently offloading other mainframe applications.
Currently applies to Sort, Cobol, and soon APL.

APL2PLUS ™ - What is it?

A port of Dyalog APL 18.2 to IBM zLinux
Currently under test.

APL2PLUS ™ - How do you use it?

You can use it directly.

- - Line mode on zLinux
- - Full screen terminal window on zLinux
- - Remote Integrated Development Environment - RIDE
- - Line mode from TSO

You can use it coupling mode.

- - Offload computation from APL2 Mainframe
- - Managed by ProxCF ™
- - Keep existing User Interface
- - Data stays on z/OS
- - Data accessed using FCCA
  - --- Fiber Channel-to-Channel Adapter
  - --- Network File System (NFS) OK but slower

For more Information

Contact: Proximal Systems Corporation

Website: www.proximalsystems.com
Ron Hilton, PSC Founder & CEO, e-mail: rhilton@proximalsystems.com
phone: +1 801-272-0058

Management of Dyalog APL workspaces

Dr. Markos Mitsos — ERGO Group AG

December 5, 2022

Abstract

Dyalog APL offers the the ability to exchange the definition of objects, as functions, operators or namespaces, as well as the content of variables with UTF-8 text files via the **Link** namespace.

This allows the disentanglement of the workspace and its functionality from the management of the code itself. By using some standard tool it is easy to facilitate code versioning as well as code sharing. In ERGO TortoiseSVN is used for this purpose.

Another aspect of workspace management is checking its functionality and documenting the check outcomes. Using Serialise and DB2 a kind of Unit tests can be implemented. These are run automatically when the workspace is deployed, but can also be used to check the effects of external changes to its functionality.

Contents

List of Tables

List of Figures

1 Targets and requirements

There are some common targets and requirements for an APL workspace management system. They pertain to different aspects of the process, like coding, cooperation, debugging, documentation and audits.

coding With respect to coding and the usage of code the system should allow:

- cooperation of a team of developers on one workspace
- cooperation of (teams of) developers through the exchange of functionalities
- incorporation of foreign code in a workspace
- changelog of each object and the ability to retract or even revert changes

code quality Concerning code quality the system should facilitate:

- easy separation of coding and debugging
- distinction between test and production
- an (extensive) array of functionality checks
- the ability to easily run through this array or parts of it on demand
- automated quality checks as part of the deployment process

formalities Finally the system should respect audits requirements and Individual Data Processing guidelines:

- easy accessibility of changelogs
- the ability to recreate the version of the workspace used at some point in time
- documentation of quality checks
- documentation of the deployment process

Note also, that requirements like the recreation of old workspaces versions may be of small practical value. For example other changes in the IT environment may render such a workspace useless. In other cases an old version may be usable for blaming a problem on somebody but not for fixing it or ameliorating the error consequences.

Still there are internal and/or external requirements of the sort. That fore a code management system should fulfil those without creating a huge overhead or otherwise hinder the development process.

2 Ideas and framework

2.1 Code in text files via Link

It is obvious that the code a workspace is easier to handle as a collection of objects than as a whole. On the most trivial level different developers can work on separate objects independently from the rest.

It is equally obvious that a collection of (UTF-8) text files can be better managed than a proprietary workspace. This can mean something as simple as saving and comparing versions of a function.

For these reasons many development teams have used various ways to exchange the code of APL functions and other objects with external systems such as transfer files or DB2 tables.

Dyalog offers the namespace **Link** and its functionalities for this purpose. This is of course the canonical way to manage code in UTF-8 text files and used in the ERGO setting discussed here. A workspace may still be the end product, but it is never saved or loaded during the development process.

To fulfil the requirement of separation between coding and debugging two sub-settings are implemented and used. The one uses a bi-directional link between workspace an text files, while the second uses only the direction from text files to the workspace.

The latter is used to debug safely, without the risk of unintentionally introducing changes into the code. **Link** allows both sub-settings to be used simultaneously. Figure 1 shows the relation of the workspaces to the working copy of the code and the corresponding repository as “back-end”.

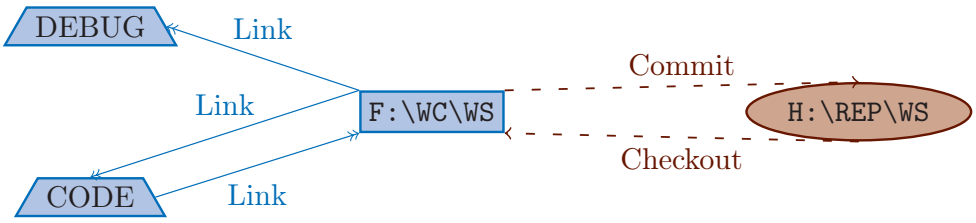


Figure 1: usage of two workspaces for simultaneously coding and debugging

2.2 Versioning via TortoiseSVN

There are many tools to version and compare text files, among them Git(Hub) and (Tortoise)SVN. In ERGO variants of both are used in the actuarial departments of the life and health segments. TortoiseSVN is used in the framework discussed here

This is so, mainly because it can be interactively used through the context menu of the Windows Explorer. Still Git has advantages concerning branch and merge. Possibly a future version will accept both versioning tools on an equal footing.

Anyhow the use of a repository creates the common ground necessary for developing simultaneously in a team. The versioning tool of course takes over the job of coordinating the development and sounding the alarm about conflicts.

The practically useful part of versioning is clearly that changes are retractable and even revert-able. The ability to use (or recreate) a specific revision of a workspace or parts of it could also be useful. As a by-product, and without additional trouble, many formal audit requirements about documentation of changes are satisfied.

Furthermore guidelines about Individual Data Processing can be satisfied with the correct wording:

development The working copy of each developer. With the use of proper checks most bugs and problems will be detected at this stage.

acceptance The repository of the workspace. At this stage all available checks or other processes may be used to ensure the application parts are properly integrated into a whole.

production The workspace itself can be distributed to different persons that those with access to the repository. Therefore a clrear distinction between developers and users may be achieved.

2.3 Structure of workspaces

Link naturally operates on namespaces, which are also a natural compartmentalisation of a workspace. A very easy way to achieve good cooperation between workspaces, is to demand that each workspace will feature a "main namespace" containing only further namespaces, which in turn will contain the real objects.

In such a setting it is nearly trivial to combine elements of different workspaces. The only prerequisite is to ensure that main namespaces bear distinct names — that should be feasible. As an example, each workspace can use some basic algorithms from a common toolbox. This leads to the necessity of providing a

building plan for each workspace. A fixed namespace, namely #**.build.prms**, is reserved for this purpose.

Another functionality that could be needed is the ability to declare objects as useful for coding and debugging or interesting as ideas, but not fit for deployment. Another fixed namespace, #**.test**, can be used to do this.

2.4 Functionality checks via DB2 and Serialise

A bunch of requirements clusters around functionality checks and their documentation. Those checks should enhance quality first and fulfil formalities second. A kind of Unit tests is used, with the difference that outcomes (results or errors) are validated and saved, not typed in as part of the check.

The definitions of the checks are to be collected in #**.build.check**. That way they are versioned together with the workspace code proper, but not deployed as part of the end product.

To save the target outcomes of the checks DB2 is used. DB2 functionalities around `SYSTEM_TIME` warrant that any change in the data is documented. That fore any related question during audit should be easy to answer. To save APL code or data it must be transformed into a string. To this end Serialise is used to create the Array Notation of the object.

Comparisons between check targets and current values ("check runs") are saved using the same principles. Again, this should satisfy requirements around test documentation without causing an overhead. The special case "deployment of application" is linked programmatically to a complete check run.

3 Environment and tools

3.1 IT environment

APL is used in ERGO by the actuarial / mathematical departments of the life and health segments, as well as a very small number of other specialists. This means, that Dyalog itself is part of the official ITERGO applications, but programs written in APL belong to Individual Data Processing (IDP).

As a consequence workspace management must be conceived in a way that will work for "users" (in contrast for example to "administrators"). This again means that no special privileges can be taken as granted and tools must be used as available. At the same time the system must fulfil external requirements, meaning those coming from other parts of the corporation.

A part of the IT environment are official releases of software. Those could include a new version of Dyalog or changes to the operating system. A system of automated checks must be usable for such occasions (too, and not only during development), because they can change the behaviour of IDP programs. Due to the IT infrastructure and the network domains involved, the system cannot for example use component files to save check targets.

3.2 Dyalog and provided tools

There is one Dyalog version available for all APL users. This version can only be updated or upgraded in an official corporation release. Only a vanishingly small minority of developers could (theoretically) use a newer version. That

means, that tools like the namespace **Link** must be used as they are for longer periods of time, small bugs must be circumvented.

The following Dyalog functions of **Link** are central to the build-up of workspaces and checks and used to:

Create create the links necessary for coding and debugging

Import complete workspace with "foreign" code and deploy applications

Serialise transform the arguments as well as the outcome (result or error) of check cases into a string in order to save them in a database

Deserialise recreate saved APL structures (and then serialise them again)

The last step is meant to ensure that the saved targets of a check case are comparable to the serialised current data produced during a check run. Because of slight defects in the preliminary versions of Array Notation in cannot be used properly. This will be rectified as soon as a corrected version is available.

3.3 Versioning and network drives

Fixed and/or physical drives like `C:` are seldom used in ERGO for anything expect user profiles and similar data. Most users do not have access to them. On the other hand the following standard network drives are provided for every user:

`F:` Personal domain. It is warranted that only the user himself can access the data.

`W:` Organisational domain. Members of an organisational unit, like a group or department, have equal and symmetric write privileges on a dedicated share on the drive.

`H:` Public domain. Anybody can request a dedicated share on the drive and then manage himself the (asymmetric) privileges on in.

It would also be difficult to obtain dedicated servers for SVN repositories or access to GitHub on the internet. That fore the suboptimal solution of creating repositories on network shares is accepted:

working copy Developers should use their personal drive for working copies, meaning checkouts that are actively used to modify and correct code. Link listeners may miss changes from time to time because the drive is a network one. For all practical purposes the setup works well.

repository SVN repositories are created in shares on the organisational or public domain. There is some risk that locks on the repository will be suboptimal leading to damage if more than one developers try to modify the repository simultaneously. Because of very small developer teams the risk is negligible.

checkout The term, also sometimes "official checkout", is understood to mean one that is exclusively used to provide access to the latest revision and the SVN log to users or developers from other teams. It is placed in the organisational or public domain, usually in the vicinity of the repository.

3.4 Databases and management of checks

The proposed system of automated checks calls for the arguments and outcomes to be controlled manually and locally in a first step ("validation"), saved in a second ("check target") and retrieved latter ("check run"). An external storage place is needed for them. As stated above, component files are not an option.

DB2 tables on the IBM mainframe are used, SQL Server could be an alternative. Both offer a "system time" concept, the option for the system to remember the state of the table in every moment in time, meaning that each `INSERT`, `UPDATE` or `DELETE` is logged. In DB2 a specification along the lines of

```
, USER_ID VARCHAR(128) GENERATED ALWAYS
  AS (SESSION_USER) IMPLICITLY HIDDEN
, ART_BEARB CHAR(1) GENERATED ALWAYS
  AS (DATA CHANGE OPERATION) IMPLICITLY HIDDEN
, TECHN_GUELT_AB TIMESTAMP(12) WITHOUT TIME ZONE
  NOT NULL GENERATED ALWAYS
  AS ROW BEGIN IMPLICITLY HIDDEN
, TECHN_UNGUELT_AB TIMESTAMP(12) WITHOUT TIME
  ZONE NOT NULL GENERATED ALWAYS
  AS ROW END IMPLICITLY HIDDEN
, TRANS_ACT_BEG T IMESTAMP(12) WITHOUT TIME ZONE
  NOT NULL GENERATED ALWAYS
  AS TRANSACTION START ID IMPLICITLY HIDDEN
```

can be used to create invisible (hidden) columns that document each change.

In combination with an activation command like

```
PERIOD SYSTEM_TIME (TECHN_GUELT_AB, TECHN_UNGUELT_AB)
```

and a shadow table associated with the table proper, the system history is recorded. A `SELECT *` will always access the current state of the table and will not even show the columns concerning changes and system time.

Naming a invisible column, like in `SELECT ART_BEARB`, will return it. A clause like

```
FOR SYSTEM_TIME AS OF CAST('8765-43-21' AS TIMESTAMP)
```

will retrieve the state of the table at the specified point in time. Summarizing, the benefit for audits et cetera comes without overhead or work load for the developer.

However there is an unsolved problem on the DB2 side. As arguments or outcomes can be longer than a few characters, the data type Character Large Object (LOB) is used for them. Retrieving UTF-8 content from such a data field through ADO does not return the data properly. That fore `80 ⎕DR` is used on the data before saving to recreate a pseudo-kind of UTF-16, and `160 ⎕DR` recreates the content after reading it. It is hoped that IBM will solve the problem.

3.5 Implemented package

The system for workspace and check management muss be implemented itself (in APL). For reasons of practical usability as well as because a range of options is

needed, the implementation is significantly longer that a few lines of code. This code must also be managed, leading to a hen-egg conundrum. The following solution was chosen:

- the code itself resides in the namespace #**.div.build** of the workspace DIVERSES, which provides basic algorithms and functionalities
  - it is managed and tested as part of this workspace
  - part of it is the function AUFBAUEN (meaning "build" in German), which is kept as lightweight as possible and acts as a kind of script
  - whenever AUFBAUEN is changed, a workspace WS_BUILDER, which contains only this object, is manually created
- any workspace which wishes to use the code to build or deploy itself loads WS_BUILDER and executes AUFBAUEN
- AUFBAUEN imports a temporary copy of DIVERSES into #**.temp** and deletes this again at the end of its execution
- any workspace which wishes to use the code to manage functionality checks incorporates #**.div.build** into its build plan

The code management namespace contains four functions:

AUFBAUEN The "face" of the system, provided in a separate workspace and used to build or deploy workspaces.

BUILD The "back-end" of the first function, called by it in the background to do the actual work.

CHECK The check driver, setting check targets and comparing them to current check outcomes.

CHECK_OBJ The innermost object calling a named object with a given range of arguments in a loop and returning the produced outcomes.

4 Workspace structure and build-up

4.1 Conventions on structure

4.1.1 Principles

As stated, some structural rules are enforced on workspaces in order to

- easily incorporate functionalities from other workspaces
- manage code based on **Link**
- use automated checks and document them

4.1.2 Basic structure

Workspaces should have the following structure at root (#) level. Each workspace has to choose a distinct name <**nsmn**> for its main namespace.

Table 1: list and short description of the objects in namespace #

object	⎕NC	content
		own namespaces
build	9.1	Building plan of the workspace including lists and descriptions of checks. The contained objects are used to build, check and deploy the workspace, but are not contained in the deployed end product.
<**nsmn**>	9.1	The workspace code proper as main namespace. At least for build-up (but not for checks) it can directly contain objects like functions and variables. It should however contain a range of namespaces, that in turn contain the objects proper. The encapsulation ensures the separation of function and functionality checks and facilitates the cooperation with other workspaces.
test	9.1	Playground for testing new ideas and alternatives. It is not always clear what the best solution to a problem is. Sometimes alternatives have to be tried out. In other cases an idea may linger or problems may have to be solved ad hoc and provisionally. Objects of this sort are collected in the namespace. They are completely ignored during deployment.
		reserved namespaces
temp	9.1	Reserved for temporary objects. It is not to be expected that contained objects will be permanent, they may be overwritten or deleted by other processes. During the building of the workspace in particular the namespace is used to harbour the necessary infrastructure temporarily.
globals	9.1	Reserved for global objects. Such may be connections to databases or Excel Workbooks that are meant to endure between function calls. The namespace could also contain data meant to stay around after a function terminates, so it can be used be others.
		foreign namespaces
<**nsfxn**>	9.1	The code proper of other workspaces. Complete workspaces can be integrated via the respective main namespace, or parts via sub-namespaces. For example some basic utilities will probably be useful in every workspace.

The first three namespaces, and only those, belong to the repository associated with the workspace. Foreign namespaces are managed "somewhere else". The system further distinguishes between namespaces belonging to the same team of developers and such of other developers.

4.1.3 Building plan and check parameters

The namespace #**.build** itself must also have a predefined structure, in order to be usable by the functions in #**.div.build**. Only `#.build.prms.NSL` *must* be set. In all other cases a internal standard will be used, if the corresponding variable is missing, in the functions themself.

To deploy the workspace #**.build.check** must also fulfil some requirements. Those add up to the functions listing check cases reflecting the structure of the main namespace of the workspace been checked. However it is not obligatory to provide check cases for every workspace object. For deploying the namespace `#.build.prms.NSMN` *must* also be present, because it unambiguously declares the distinguished namespace to be checked.

Table 2: list and short description of the objects in namespace #**.build**

object	⎕NC	content
	parameters for build-up and check	
prms	9.1	Storage of workspace building plan.
prms.PATH_REP	2.1	Standard path for repositories in the url syntax needed by SVN. It is assumed that in this path one repository will be found for each workspace.
prms.NSL	2.1	List and description of namespaces in the workspace to-be-build. Next to the workspace's own namespaces, foreign ones are to be listed. Each item may be supplemented by a repository path and/or the revision to be used.
prms.WSN	2.1	Explicit, separate name of the deployed workspace. Note that the finished application will in many cases lie in a different share than the repository or the official checkout.
prms.LX	2.1	Latent Expression of the target workspace. The standard is empty, meaning no action. In many cases the latent expression will open the application's main GUI. Note also, that for a runtime something of the sort is needed.
prms.NSMN	2.1	Name of the main namespace of the target workspace. It should be distinct from all corresponding names of workspaces to which there exists a potential connection.
prms.SQLID	2.1	Creator of the DB2 tables documenting check targets and runs. It is advisable to choose a creator controlled (exactly) by the developers of the workspace.
prms.XLN	2.1	Target Excel Workbook for check targets and runs. The generic strings *'<wsn>'* and *'<art>'* may be part of the variable and are correspondingly replaced by the name of the workspace and the modus of the check.

Table 2: (description of namespace #**.build** continued)

object	⎕NC	content
		functionality checks
check	9.1	Storage of workspace checks.
check.NSL	2.1	List than connects sub-namespaces in #.<**nsmn**> to the corresponding check functions.
check.<FNxN>	3.1	List and description of check cases for the namespace #.<**nsmn**>.<**nsxn**>. Of course checks make only sense for (more or less) functional objects like functions. The description encompasses the description proper as (preferably legible) text, as well as the arguments to be used. The outcome, be it a result or an error, is part of the check management by #.div.build.CHECK.

The list of namespaces #**.build.prms.NSL** must contain one line for each namespace to be incorporated into the target namespace. Each one must contain an affiliation value, signifying if the code originates from other developers, from another workspace of the same developers or belongs to the workspace itself. Those cases lead to different treatment during build-up.

It is possible to fix the revision of foreign code to be used. This way changes may be integrated in a controlled manner.

The list of checks #**.build.check.NSL** must contain one line for each sub-namespace in the main namespace of the target workspace and associate these with check functions. In most cases the function will have the same name as the namespace, possibly in uppercase.

4.2 Build-up from text files

4.2.1 Process

The build-up of a workspace is orchestrated by the function AUFBAUEN provided in the (very slim) workspace WS_BUILDER and proceeds in phases:

setup done in AUFBAUEN

- the latest revision of DIVERSES is exported from the repository and imported via Link.Import into #**.temp**
- the build and check instructions for the workspace and imported via Link.Import into #**.temp**

sources done in BUILD

- the sources of the relevant namespace (each a working copy, official checkout or repository) are determined
- an export of each relevant revision of each relevant repository is created

- the sources are linked via Link.Create or imported via Link.Import

check done in CHECK but only in the case or workspace deployment

- a complete workspace check is run

cleanup done in BUILD

- AUFBAUEN and, in the case of deployment, temporary or global objects are expunged
- the workspace is renamed and, in the case of deployment, saved

4.2.2 Primary build-up modi

The three most important development steps of an application are coding, quality check and deployment. Those steps are reflected in the implemented build-up modi. Foreign components are needed for the latter two and are very useful while coding.

As a convention a strict separation between coding and checking is proposed. While debugging one often changes code just to analyse a problem more efficiently. Those changes should never enter into the text files. Similarly one might test alternatives or ad hoc solutions, which have no place in the workspace. The proposed system allows coding and debugging in parallel, in two separate instances of Dyalog.

The used modi are (of course!) in German, meaning respectively "develop", "check" and "distribute" (in the sense of "deploy"):

entwickeln When developing the code of an workspace one must change it simultaneously in the workspace and the underlying text files. A bi-directional link to the files in the working copy is appropriate, and not every change will be directly committed to the repository. The presence of foreign components in the workspace is useful, because it makes it easier to use them with the correct syntax.

prüfen Wenn debugging one wishes the current (in real time) code to work on, but without the danger of accidentally changing the underlying text files or, even worse, the repository. A one-directional link to the files in the working copy is appropriate.

verteilen The finished workspace, program or application must be "published". An import of the code is mandatory. The deployment step is inseparably linked to a full quality check of the functionality. The publication always uses the latest revision in the repository, not a working copy, because there is not warranty that such a stand will be ever committed in exactly the momentary form. The deployment should have a suiting, public share on a network drive as target.

In this setting no personal copies of workspaces occur. The end user then loads the deployed workspace, which ideally can be used interactively.

We represent visually the three discussed, primary and most important builds. In the first, Figure 2, the workspace is build for coding. The workspace's own namespaces are linked to the working copy (which in turn communicates with the repository). The rest is imported from temporary repository exports.

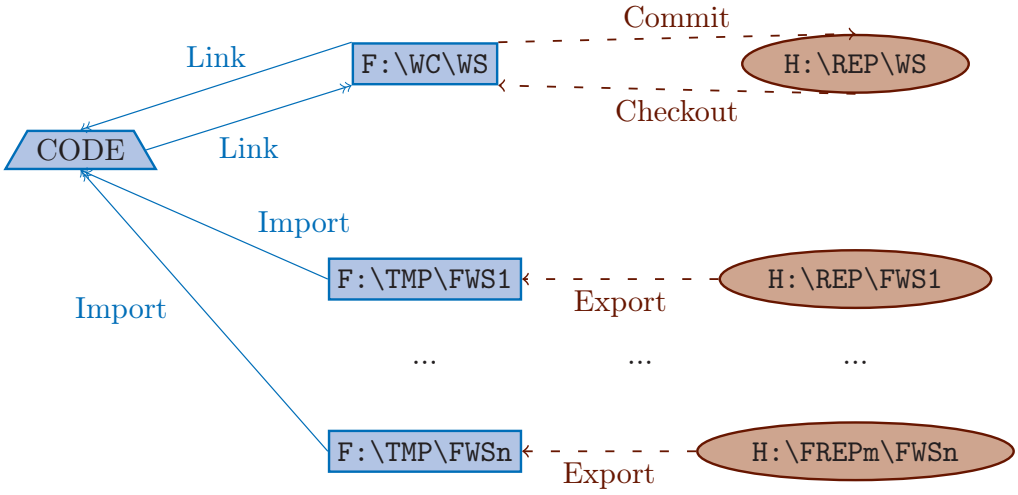


Figure 2: workspace build for coding

In the second, Figure 3, debugging is facilitated. The only difference to coding lies in the one-directional link to the text files of the workspace's own namespaces.

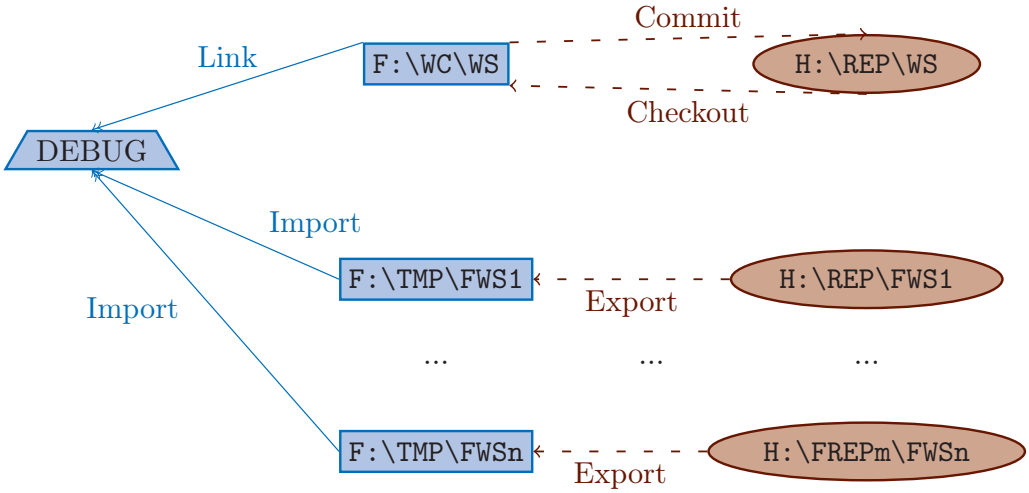


Figure 3: workspace build for debugging

The build for deployment, Figure 4, differs significantly from the rest. Not only is a temporary repository export of the workspace's own code created and imported, but the workspace itself is saved in its destination folder.

4.2.3 Supplementary build-up modi

In many cases further modi may be of interest. Foreign components are needed in each of those. Their meaning in German is respectively "workspace overarching check", "test" and "use without check":

WS-übergreifend prüfen In some cases a developer or a team may work on many associated workspaces simultaneously. In such cases in can happen

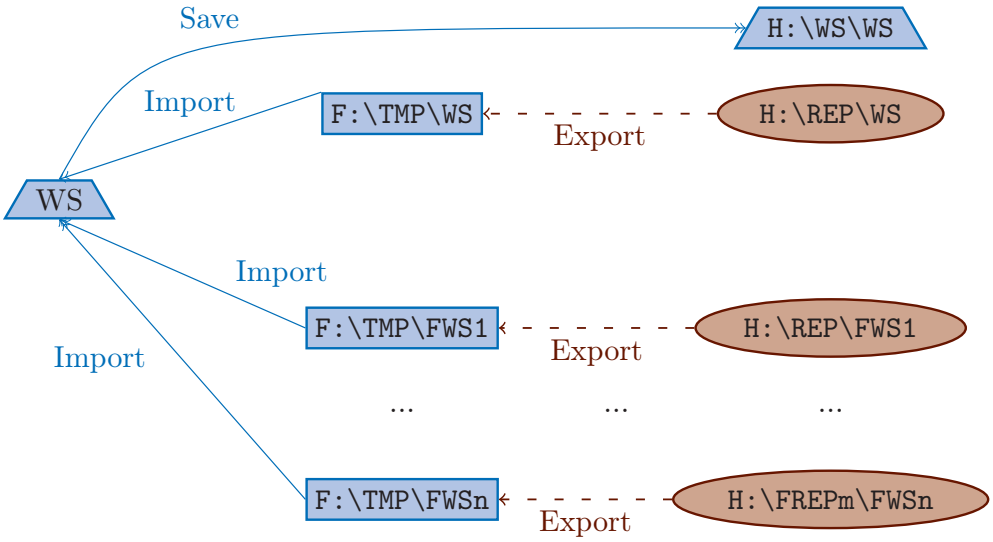


Figure 4: workspace build for deployment

that an error in an object of a leading workspace is first detected in a dependent one. In these cases it is very useful not only to check out corrections via the dependent workspace, but also to do so before and without the necessity of creating a revision in the repository of the leading workspace. To achieve this, foreign code must be linked in the same way as own components.

testen Occasionally the developer may test alternatives or measure performance without worrying about changes occurring parallelly due to coding. Furthermore specialists may choose to use partially developed workspaces with known defects, for example to get estimates on results. In such cases no connection to the workspace proper should exists, the code must be imported.

ungeprüft benutzen The deployment of the workspace encompasses checks that may take considerable time to complete or may be exceedingly sensitive to changes. For example it may be that a check case reports a problem just because external data has changed — but it must be verified that this is the case. For this reasons, deployment may not be possible as quickly as wished for. The alternative would be to use the penultimate step before deployment, aka the last revision in the repository without checks.

4.2.4 Code source

Depending on the build-up modus, there are different necessities or conventions concerning the sources out of which the different code components are taken. As already stated, the workspace's "own" namespaces have to be differentiated from "foreign" ones and the latter must be separated according to their developers.

The following overview clarifies how the different modi are implemented.

Table 3: code source of workspace objects

modus	other developers	foreign namespaces	own namespaces
entwickeln	repository	repository	working copy
prüfen	repository	repository	working copy
verteilen	repository	repository	repository
WS-übergreifend prüfen	repository	working copy	working copy
testen	repository	repository	working copy
ungeprüft benutzen	repository	repository	checkout

4.2.5 Code association type

Depending on the build-up modus, there are different requirements concerning the association of workspace code with the underlying text files. As already stated, when coding one must be able to change the files, while nobody can expect to receive changes from other developers on the fly.

The following overview clarifies how the different modi are implemented. Straight arrows signal an established link, curved ones a one time import.

Table 4: relation of text files to workspace objects

modus	other developers	foreign namespaces	own namespaces
entwickeln	⇝	⇝	↔
prüfen	⇝	⇝	→
verteilen	⇝	⇝	⇝
WS-übergreifend prüfen	⇝	→	→
testen	⇝	⇝	⇝
ungeprüft benutzen	⇝	⇝	⇝

5 Checks and deployment

5.1 Basic check schema

5.1.1 Limits of automated checks

The idea of automated checks is to ensure code quality and functionality without additional (manual) work or overhead. This means that as many workspace functionality aspects as possible should be checked in the background.

But there are some limits. For one thing, interaction cannot be tested properly (at least not with these methods). Performance can also not really be checked, because that would mean large amounts of data and long run times, contradicting the idea of something that happens unobtrusively in the background.

On the practical side, code must be reasonably well written, for example in a functional way, to allow controlled checks without side effects. Still an initial time investment is always necessary, in order to create a meaningful series of checks.

Among other things, in many cases suitable input data must be prepared. Network shares, databases or component files must occasionally be set aside for checks. If a check returns some non-repeatable data, like run times or timestamps, the outcome must be post-processed.

5.1.2 Validation of check outcomes

As already stated, to allow the mechanism of #.div.build.CHECK to assess and process checks, they must obey a certain schema. For the objects in the namespace **goo** of the workspace FOO a check function #.build.check.HOO, or more generally a name dictated by #.build.check.NSL, must be provided.

It must take an right argument `ARG` and contain a `:Select ARG` control structure. Checks for the function goo.HOO are to be collected under `:Case 'HOO'`.

The definition of checks consists of a list. One column is the description, an arbitrary text up to 256 bytes long. Each check should have an unique description that honours its name, meaning that it gives some clues about what it is doing... The definition must also contain the arguments (including operands) to be used.

The check function must then call #.div.build.CHECK_OBJ with the list as an argument. The list will be returned enlarged by the outcomes of the checks. This structure is necessary to allow post-processing of the outcomes (usually the results, but sometimes also the errors), which are highly specific to the objects being checked and the checks used.

The validation is being done "locally" by calling #.build.check.HOO, inspecting the outcomes, correcting goo.HOO and repeating the process as often as necessary.

5.1.3 Check targets and runs

The uniqueness of the description (in fact the description itself) is not needed for local validation. However it is used as a key by #.div.build.CHECK and is that fore demanded and enforced by this function.

In a basic setting fixing check targets is straightforward and trivial. A function call like

```
'setzen' #.div.build.CHECK 'FOO' 'goo' 'HOO'
```

will use #.build.check.HOO to get the checks and their respective outcomes and save everything in DB2 tables. Additionally the check targets will be presented in Excel.

The implemented algorithm uses the described elements and is visualised in Figure 5. In the first step #.div.build.CHECK uses its arguments combined with the global list `#.build.check.NSL` to call the appropriate function, namely #.build.check.GOO. The latter creates a list of arguments for check cases and #.div.build.CHECK_OBJ supplies the corresponding outcomes. After post-processing, the list is passed back as the "targets", meaning the new targets of check cases.

Steps 7 and 8*a* preclude the following discussion about real life settings. #.div.build.CHECK reads in the old check targets from DB2, which are now (just) the "current" ones. The comparison between the two variables could lead

to an error being thrown in the calling environment, or to the last step 9*a*, saving the new targets in DB2. The *a*'s signify that one specific modus of comparison and one possible DB2 table are used. Note also that the underlying DB2 tables sport a shadow system history.

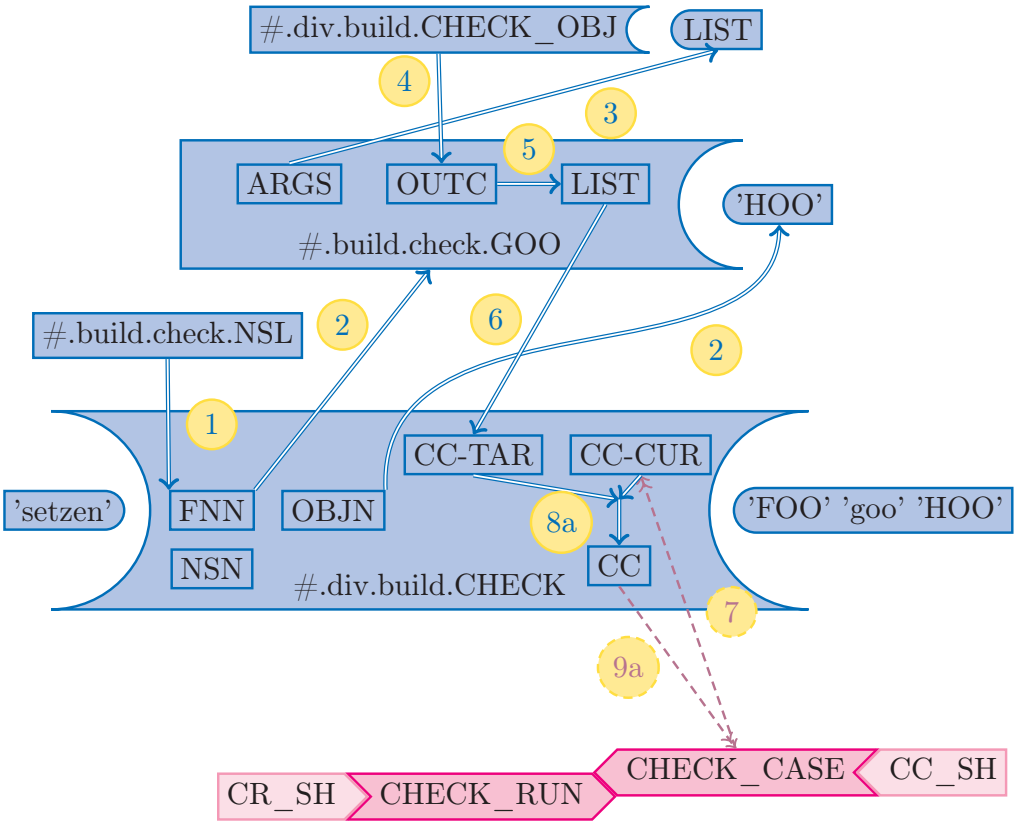


Figure 5: setting check targets after comparison of current with saved state

After that the object goo.HOO may be modified or the environment (Dyalog, interfaces to other programs, operating system) may change. The equally simple call

```
'intern' #.div.build.CHECK 'FOO' 'goo' 'HOO'
```

will retrieve the check targets from DB2, call #.build.check.HOO and compare the two data sets. Deviations will earmarked and everything will be saved in DB2. Additionally the check comparisons will be presented in Excel.

The implemented algorithm uses the described elements and is visualised in Figure 6. It is almost identical to that of setting check targets. However the check states produced in APL are now (just) the "current" ones, whereas those retrieved from DB2 are the "targets". Step 8*b* use a different comparison that creates (error) messages to be presented as a result instead of throwing an error in the calling environment. 9*b* saves the check run in a different DB2 table.

5.2 Comparison hierarchy

5.2.1 Limits of the basic schema

A real life setting cannot be served properly by such a simple schema of checks:

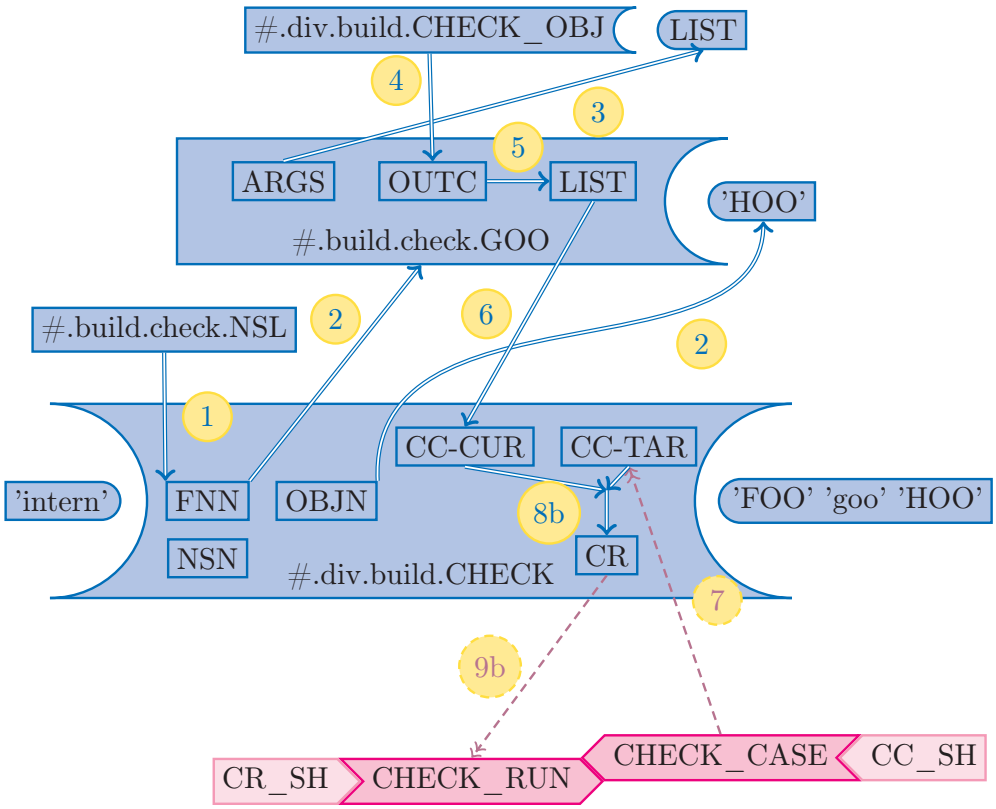


Figure 6: saving a check run after comparison of current with saved targets

length The underlying DB2 tables allow argument and outcome lengths up to 4MB, as well as up to 4KB for operands. Still for any (finite!) length there will be a check at least with one parameter exceeding it. Trying to de-serialise such a string after retrieving it from DB2 will lead to a crash. That fore some additional mechanism is needed that documents unabbreviated lengths as well as completeness or data and differentiates comparison according to those.

changes There are many ways check targets may change. The underlying object may be modified. In this case changes in outcomes are expected (and must be validated), but arguments should remain identical. On other occasions just the order of checks may be changed, or a few descriptions. Here arguments and outcomes should remain fixed.

deviations Similarly to check targets, the comparison done during a check run should be able to differentiate deviations. An error of the form "cannot compare" just because the order of checks has changed is not very useful, and a message "something is different" should differentiate between arguments and outcomes.

5.2.2 Ordering of deviations

A natural way to order deviations of functionality checks uses the description as a key and proceeds in ascending order or importance as follows:

order Checks do not have a natural order. Still it can be helpful to number them in the order they are defined and use this numbers, for example to order them in an Excel Sheet. A deviation should have the least significance and be marked as "purely aesthetic".

outcome A deviation in the outcome signifies that a check was fully repeatable and that something has changed in the underlying object. It can further be specified as resulting from the outcome itself, its completeness or the type of the outcome (result or error). Exchanging a result for an error is probably more significant that an error message slightly differently formulated.

arguments Different arguments imply that a comparison of the outcome is not meaningful. Again further differentiation into content or completeness is possible. Additionally an information about which arguments have changed can be useful.

identification If checks cannot be identified, meaning that their descriptions cannot be matched across the two stands to be compared, they can be only classified as "new, to be added" or "old, to be removed".

An additional problem is that in real life one will occasionally change a few descriptions. For example unspecific descriptions like "error", "problem" and "not usable" may be clarified as "error (right argument rang)", "error (left argument value)" and "error (target component file)" without changing anything else.

In such a case one may not wish to validate *all* checks from scratch. On the other hand one wants to avoid the danger of validating defective results that may stem from changes having been made simultaneously in another function called by the object to be checked.

For such cases an alternative key for check cases in needed. The implemented algorithm can use arguments for this purpose. More specifically it always preferentially matches check cases based on their description. Additionally it tries to match cases using all arguments (including operands) under two conditions, which are minimal requirements for a key

- the case has not been matched using the description
- the arguments are unique among all unmatched cases

5.3 Check case management

5.3.1 Check targets

Check targets are to be set for one (workspace, namespace and) object at a time. The following options for setting check targets are implemented. They are to be specified as a left argument to #.div.build.CHECK. Their meaning in German is respectively "set up to running number", "set up to outcome", "set up to arguments", "set", "set all expect description", "set only description (and running number)" and "set completely":

bis Laufnummer setzen Check cases are identified (only) via their description. Just the current order is accepted as new target. Further differences between TARGET and CURRENT, meaning not matching descriptions, arguments or outcomes, lead to an appropriate error being thrown.

bis Resultat setzen Check cases are identified (only) via their description. The current order as well as the outcomes are accepted as new target. Further differences between TARGET and CURRENT, meaning not matching descriptions or arguments, lead to an appropriate error being thrown.

bis Aufruf setzen Check cases are identified (only) via their description. The current order, the outcomes as well as the arguments are accepted as new target. Further differences between TARGET and CURRENT, meaning not matching descriptions, lead to an appropriate error being thrown.

setzen Set check targets proper. Check cases are identified (only) via their description. The current order, the outcomes as well as the arguments are accepted as new target. Additionally missing cases are added and superfluous removed. In particular an error is never thrown.

außer Beschreibung setzen Synonym with "setzen", clarifies the functionality.

nur Beschreibung (und Laufnummer) setzen Check cases are identified preferentially via their description, additionally via arguments. The latter must be unique among cases without identification via description in order to be used. The current order as well as the descriptions are accepted as new target. Further differences between TARGET and CURRENT, meaning not matching cases, arguments or outcomes, lead to an appropriate error being thrown. Not matching cases are those that could not be identified. In particular check cases that could not be matched via description and do not have unique arguments lead to an error, because it would not be clear how to process them. Furthermore the system does not allow an "exchange" of check descriptions, because the latter are used as the primary identification attribute.

komplett setzen Set including descriptions. Check cases are identified preferentially via their description, additionally via arguments. The latter must be unique among cases without identification via description in order to be used. The current order, the outcomes, the arguments as well as the descriptions are accepted as new target. Additionally missing cases are added and superfluous removed. Cases that could not be matched via description and do not have unique arguments always fall in one of those categories. In particular an error is never thrown.

5.3.2 Check runs

Check runs may encompass one or many objects, several namespaces or an entire workspace. The following options for running checks are implemented. They are to be specified as a left argument to #.div.build.CHECK, which is (also) saved in DB2 as the "run name". Their meaning in German is respectively "internal" and "distribute" (in the sense of "deploy"):

intern If and to what extent any small code change, for example one being part of a larger reorganisation, must be checked, is up to the developer. The control can be on the level of object(s), namespace(s) or workspace. Controls can be "finer" or "coarser" than revisions. This means that one

can go through many cycles of change and control before a commit, or check a whole series of commits in one go.

verteilen The discussed code management stipulates a deployment step, where to workspace is build and saved. As part of this step all checks of the workspace are run (which may take some time). Deployment is always "coarser" than revisions and uses them as code basis.

* In many cases the environment, in which programs run, changes, for example during official corporation releases. Those may change the Dyalog version, interfaces to other programs or the operating system. In many such cases check runs on some workspaces will be necessary. The differentiation to other options is introduced to facilitate the automatic documentation of the control. Names should describe the occasion that necessitated the check run, like "new Windows version x".

The comparison is done in stages, always using identification via arguments in addition to the one via descriptions. Each case is assigned the first true (negative) test result as an (error) message. Their meaning in German is clarified by their description:

Beschreibung anders The test case could not be identified via description, but was identified via (unique) arguments.

nur IST vorhanden The test case is saved in DB2, but does not exist in the current APL workspace.

nur SOLL vorhanden The test case is listed in the current APL workspace, but is not saved in DB2.

Aufrufvollständigkeit anders The arguments are complete in either APL or DB2, but not on the other side (too long).

Aufruf anders The arguments are different in APL than in DB2, although being complete on both sides or incomplete (too long) on both.

Resultatart anders The type of outcome is different in APL than in DB2, for example an explicit result on the one side but an error on the other.

Aufrufvollständigkeit anders The outcome is complete in either APL or DB2, but not on the other side (too long).

Resultat anders The outcome is different in APL than in DB2, although being complete on both sides or incomplete (too long) on both.

Laufnummer anders The running number of the test case is different in APL than in DB2, the order of the cases hat changed.

alles OK The test case could be identified and its state in APL is identical to the one saved in DB2.

Contact:

Dr. Markos Mitsos
e-mail: markos.mitsos©ergo.de

Theoretische Physik

Quantenalgorithmen sparen Zeit bei der Berechnung von Elektronendynamik

Berlin. Quantencomputer versprechen erheblich kürzere Rechenzeiten für komplexe Probleme. Aber noch gibt es weltweit nur wenige Quantencomputer mit einer begrenzten Anzahl so genannter Qubits. Quantencomputer-Algorithmen können aber auch auf konventionellen Servern laufen, die einen Quantencomputer simulieren. Ein HZB-Team hat damit nun am Beispiel eines kleinen Moleküls dessen Elektronenorbitale und ihre dynamische Entwicklung nach einer Laserpulsanregung berechnet. Die Methode eignet sich auch, um größere Moleküle zu untersuchen, die mit konventionellen Methoden nicht mehr berechnet werden können.

„Diese Quantencomputer-Algorithmen sind ursprünglich in einem ganz anderen Kontext entwickelt worden. Wir haben sie hier erstmals genutzt, um Elektronendichten von Molekülen zu berechnen, insbesondere auch ihre dynamische Entwicklung nach Anregung durch einen Lichtpuls," sagt Annika Bande, die am HZB eine Gruppe zur theoretischen Chemie leitet. Zusammen mit Fabian Langkabel, der bei Bande promoviert, zeigte sie nun in einer Studie, wie gut dies funktioniert.

„Wir haben einen Algorithmus für einen fiktiven, völlig fehlerfreien Quantencomputer entwickelt, und ihn auf einem klassischen Server laufen lassen, der einen Quantencomputer von zehn Qbits simuliert," sagt Fabian Langkabel. Dabei begrenzten sie ihre Studie auf kleinere Moleküle, um die Rechnungen auch ohne echten Quantencomputer durchführen zu können und mit konventionellen Berechnungen zu vergleichen.

Sie konnten zeigen, dass auch die Quantenalgorithmen die erwarteten Ergebnisse produzierten. Im Unterschied zu konventionellen Berechnungen eignen sich die Quantenalgorithmen jedoch auch, um mit zukünftigen Quantencomputer deutlich größere Moleküle zu berechnen: „Das hat mit den Rechenzeiten zu tun. Sie steigen mit der Anzahl der Atome, aus denen das Molekül besteht", sagt Langkabel. Während die Rechenzeit sich mit jedem zusätzlichen Atom für konventionelle Verfahren vervielfacht ist das für Quantenalgorithmen nicht der Fall, was sie sehr viel schneller macht.

Die Studie zeigt damit einen neuen Weg, um Elektronendichten und ihre „Antwort" auf Anregungen mit Licht mit sehr hoher Orts- und Zeitauflösung vorab zu berechnen. Damit lassen sich beispielsweise ultraschnelle Zerfallsprozesse simulieren und verstehen, die auch bei Quantencomputern aus so genannten Quantenpunkten entscheidend sind. Aber auch Vorhersagen zum physikalischen oder chemischen Verhalten von Molekülen sind möglich, zum Beispiel während der Aufnahme von Licht und dem anschließenden Transfer von elektrischen Ladungen. Dies könnte die Entwicklung von Photokatalysatoren für die Produktion von grünem Wasserstoff mit Sonnenlicht erleichtern oder dabei helfen, Prozesse in den lichtempfindlichen Rezeptormolekülen im Auge zu verstehen.

(DOI: 10.1021/acs.jctc.2c00878)

(Quelle: Helmholtz-Zentrum Berlin für Materialien und Energie GmbH, Dr. Annika Bande; annika.bande@helmholtz-berlin.de)

Theoretische Physik

Mit dem Zufall gerechnet: Weg geebnet für neue Erkenntnisse über Planeten und Sterne

Dresden-Rossendorf. Mit einer auf Zufallszahlen basierenden Simulationsmethode konnten Wissenschaftler die Eigenschaften von warmem dichten Wasserstoff so genau wie nie zuvor beschreiben

Die Eigenschaften von Quantensystemen aus vielen wechselwirkenden Teilchen zu ermitteln ist immer noch eine enorme Herausforderung. Die zugrundeliegenden mathematischen Gleichungen sind seit Langem bekannt. Allerdings sind sie zu komplex, um sie in der Praxis zu lösen. Diese Barriere zu durchbrechen würde höchstwahrscheinlich zu einer Fülle neuer Erkenntnisse und Anwendungen in Physik, Chemie und Materialwissenschaften führen. Forschern des Center for Advanced Systems Understanding (CASUS) am Helmholtz-Zentrum Dresden-Rossendorf (HZDR) ist nun ein bedeutender Entwicklungsschritt gelungen: Sie konnten sogenannten warmen, dichten Wasserstoff – Wasserstoff unter extremen Bedingungen, wie etwa hohem Druck – mit bisher unerreichter Genauigkeit beschreiben. Mit ihrem auf Zufallszahlen basierenden Ansatz konnten die Wissenschaftler erstmals die Quantendynamik von Elektronen lösen, die bei der Wechselwirkung vieler Wasserstoffatome zum Beispiel im Inneren von Planeten oder in Fusionsreaktoren auftritt (Physical Review Letters, DOI: 10.1103/PhysRevLett.129.066402).

Wasserstoff ist das am weitesten verbreitetste Element im Universum. Als Brennstoff befeuert er nicht nur die Sterne und damit auch unsere Sonne, sondern bildet auch das Innere von Planeten wie etwa des Gasriesen Jupiter in unserem Sonnensystem. Die häufigste Form von Wasserstoff im Universum ist weder das farb- und geruchlose Gas noch die auf der Erde bekannten wasserstoffhaltigen Moleküle wie beispielsweise Wasser. Es ist der warme dichte Wasserstoff der Sterne und Planeten – extrem komprimierter Wasserstoff, der in bestimmten Fällen sogar elektrisch leitfähig wird, wie man es von Metallen kennt. Die Forschung im Bereich der warmen dichten Materie basiert auf Untersuchungen unter sehr hohen Temperatur- oder Druckbedingungen, wie sie überall im Universum anzutreffen sind. Auf der Erdoberfläche kommen derartige Umgebungsbedingungen indes nicht vor.

Simulationsmethoden und ihre Grenzen

Bei dem Versuch, die Eigenschaften von Wasserstoff und anderen Stoffen unter extremen Bedingungen zu ergründen, stützt sich die Wissenschaft überwiegend auf Simulationen. Eine weitverbreitete Methode ist die Dichtefunktionaltheorie (DFT). Trotz ihres Erfolgs ist sie für die Beschreibung von warmem dichten Wasserstoff unzureichend. Der Hauptgrund dafür ist, dass korrekte Simulationen genaue Kenntnisse über die Wechselwirkungen von Elektronen in warmem dichten Wasserstoff erfordern. Da dieses Wissen jedoch fehlt, müssen sich die Forschungsteams immer noch auf Näherungswerte für diese Wechselwirkung beschränken. Das führt zu verfälschten Simulationsergebnissen. Aufgrund dieser Wissenslücke ist es zum Beispiel nicht möglich, die Aufheizphase von Trägheitsfusionsreaktionen korrekt zu simulieren. Die Überwindung dieser Hürde könnte die Trägheitsfusion,

einen der beiden Hauptzweige der Fusionsenergieforschung, erheblich voranbringen. Fusionsreaktoren könnten sich perspektivisch zu einer relevanten kohlenstofffreien Energieerzeugungstechnologie entwickeln.

In der neuen Publikation zeigen Erstautor Maximilian Böhme, Dr. Zhandos Moldabekov, CASUS-Nachwuchsgruppenleiter Dr. Tobias Dornheim (alle CASUS-HZDR) und Dr. Jan Vorberger (HZDR-Institut für Strahlenphysik) erstmals, dass sich die Eigenschaften von warmem dichten Wasserstoff mit sogenannten Quantum Monte Carlo (QMC)-Simulationen sehr genau beschreiben lassen. „Wir haben eine QMC-Methode namens Path-Integral Monte-Carlo (PIMC) erweitert, um die statische elektronische Dichteantwort von warmem dichten Wasserstoff zu simulieren“, sagt Böhme, der seine Promotion am CASUS vorantreibt. „Unsere Methode verlässt sich nicht mehr auf die Näherungswerte, die frühere Ansätze limitierten. Stattdessen berechnet sie direkt die fundamentale Quantendynamik und ist daher sehr genau. Aufgrund des enormen Rechenaufwands stößt unser Ansatz in Bezug auf den Umfang allerdings an seine Grenzen. Obwohl wir uns auf die derzeit leistungsfähigsten Supercomputer stützen können, sind wir bisher nur in der Lage, Teilchenzahlen im zweistelligen Bereich zu verarbeiten.“

https://doi.org/10.1103/PhysRevLett.129.066402

(Quelle: Helmholtz-Zentrum Dresden-Rossendorf, Dr. Tobias Dornheim, Young Investigator, Center for Advanced Systems Understanding (CASUS) am HZDR, E-Mail: t.dornheim@hzdr.de)

APL-Journal
41. Jg. 2022, ISSN 1438-4531
Herausgeber: Prof. Dr. Dieter Kilsch, APL-Germany e.V., Mannheim, Homepage: https://apl-germany.de/, E-Mail: d.kilsch@th-bingen.de
Redaktion: Dipl.-Volksw. Martin Barghoorn (verantw.), Lückhoffstr. 8, 14129 Berlin, E-Mail: Martin@Barghoorn.com
Verlag: RHOMBOS-VERLAG, Berlin, Postfach 67 02 17, D-10207 Berlin, Tel. (030) 261 9461, eMail: verlag@rhombos.de, Internet: https://rhombos.de/
Erscheinungsweise: halbjährlich
Erscheinungsort: Berlin
Druck: dbusiness.de GmbH, Berlin

Allgemeine Informationen

(Stand 2022)

APL-Germany e.V. ist ein gemeinnütziger Verein mit Sitz in Düsseldorf. Sein Zweck ist es, die Programmiersprache APL, sowie die Verbreitung des Verständnisses der Mensch-Maschine Kommunikation zu fördern. Für Interessenten, die zum Gedankenaustausch den Kontakt zu anderen APL-Benutzern suchen, sowie für solche, die sich aktiv an der Weiterverbreitung der Sprache APL beteiligen wollen, bietet APL-Germany den adäquaten organisatorischen Rahmen.

Auf Antrag, über den der Vorstand entscheidet, kann jede natürliche oder juristische Person Mitglied werden. Organe des Vereins sind die mindestens einmal jährlich stattfindende Mitgliederversammlung sowie der jeweils auf zwei Jahre gewählte Vorstand.

1. Vorstandsvorsitzender

Prof. Dr. Dieter Kilsch,
Dumontstraße 12, 55313 Mainz,
Tel. 06131 6982200, E-Mail:
d.kilsch@th-bingen.de.

2. Vorstandsvorsitzender:

Martin Barghoorn
Lückhoffstr. 8, 14129 Berlin,
E-Mail: Martin@Barghoorn.com

Schatzmeister

Jürgen Beckmann
Im Freudenheimer Grün 10
68259 Mannheim
Tel. 0621 7 98 08 40,
eMail: JBecki@onlinehome.de

Beitragssätze

Persönliche Mitglieder:

Natürliche Personen	32,- EUR*
Studenten / Schüler	11,- EUR*

Institutionelle Mitglieder:

Jurist./natürl. Pers.	500,- EUR*

* Jahresbeitrag

Bankverbindung
BVB Volksbank eG Bad Vilbel
BLZ 518 613 25
Konto-Nr. 523 2694

Hinweis:
Wir bitten alle Mitglieder, uns Adressänderungen und neue Bankverbindungen immer sofort mitzuteilen. Geben Sie bei Überweisungen den Namen und/oder die Mitgliedsnummer an.

https://apl-germany.de/