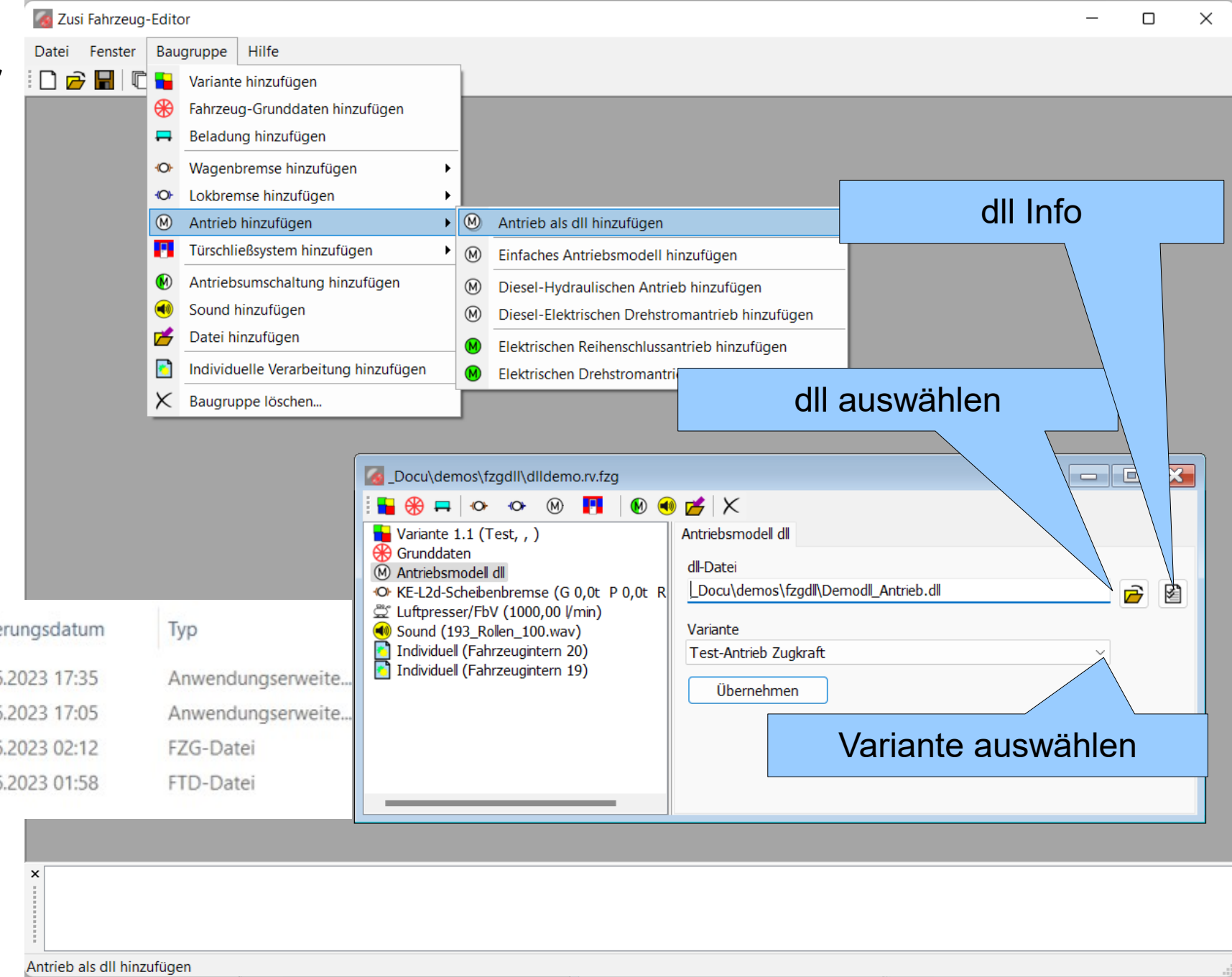
Fahrzeugantrieb-dll

- Ersatz der Zusi-eigenen Antriebsberechnung durch eine dll
- Win32 in 32 und 64 bit (.NET dll ist nicht kompatibel)
- Gleicher Ansatz für dynamische Bremse (ist aus Zusi-Sicht auch ein Antrieb, der rückwärts läuft)
- Pro Fahrzeug(-Familie) eine dll pro Antrieb und ggf. eine pro dyn. Bremse
- dll liegt im Rollingstock-Verzeichnis der Baureihe und wird im Fahrzeugeditor verknüpft

Fahrzeugeditor



Name	Änderungsdatum	Typ
Demodll_Antrieb.64.dll	13.06.2023 17:35	Anwendungserweite...
Demodll_Antrieb.dll	13.06.2023 17:05	Anwendungserweite...
dlldemo.rv.fzg	19.06.2023 02:12	FZG-Datei
dlldemoftd.ftd	13.06.2023 01:58	FTD-Datei

dll-Schnittstellen – was ist eine dll?

- dynamic link library (dll) ist ausführbarer Code
- Quasi eine exe, die nicht selbst starten kann
- Anwendungen können dll laden und nutzen
- Erstellung z.B.mit C++ oder Delphi
- Win32, 64 und .NET dlls nicht kompatibel
- Vorteil der dll: Programmfunktionen können ausgelagert werden
  - Mehrere Programme können eine dll nutzen
  - Dritte können ausgelagerte Funktionen ändern
  - dll kann mit anderen Entwicklungstools erstellt werden

dll-Schnittstellen - Beispiel-dll

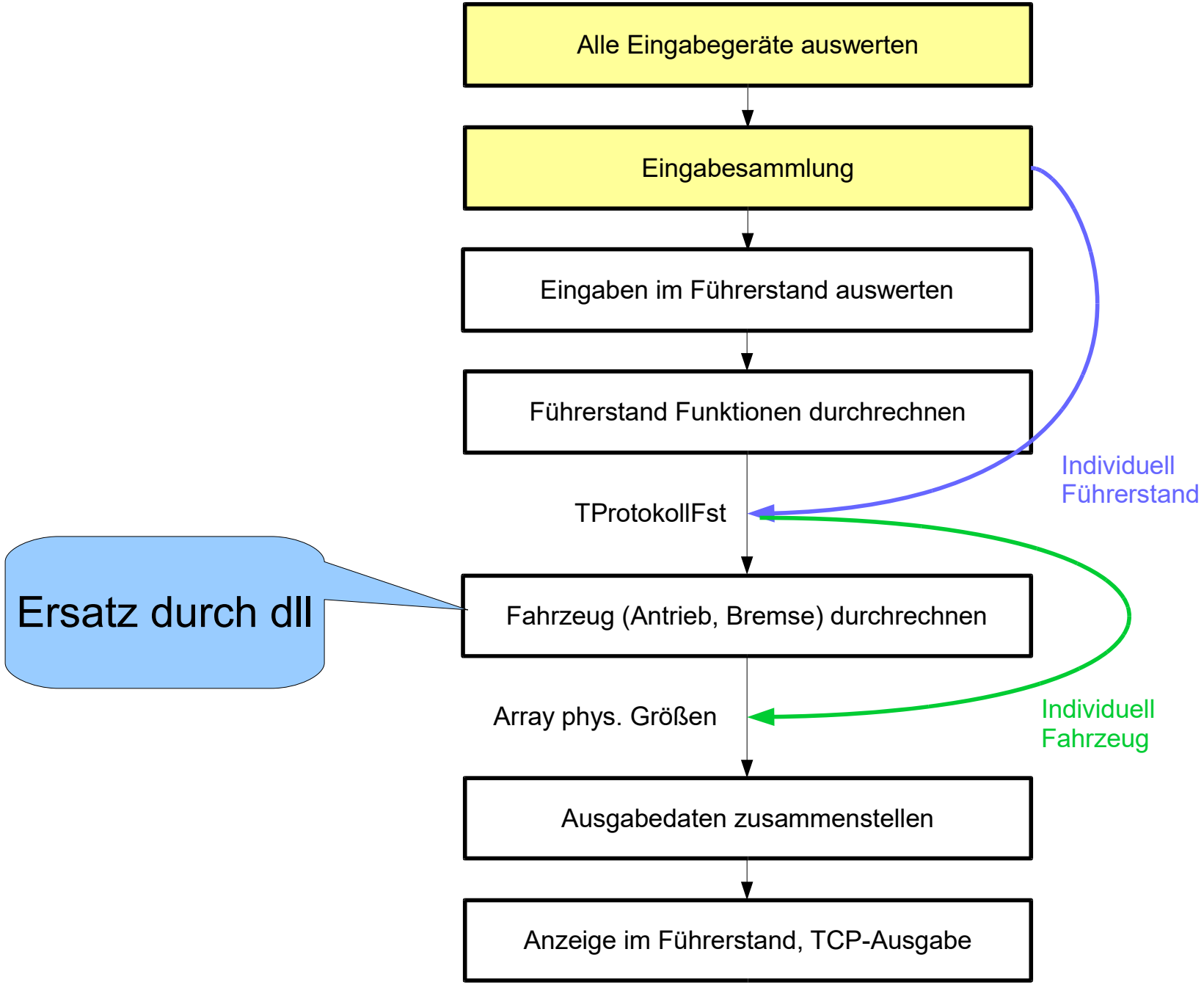
```
library Rechendemo;
```
<- das wird dann Rechendemo.dll

```
interface

function Autor:PChar; stdcall;
function Rechne(a, b:longint):longint; stdcall;

exports Autor,
        Rechne;

implementation

function Autor:PChar; stdcall;
begin
  Result:='Carsten Hölscher';
end;

function Rechne(a, b:longint):longint; stdcall;
begin
  Result:=a + b;
end;

end.
```
<- Diese Funktionen stellt die dll extern bereit

dll-Schnittstellen – Beispiel Aufruf

```
program Rechner;
```
<- das wird dann Rechner.exe

```
var

dllHandle:THandle;
Autoraufruf: function:PChar; stdcall;
Rechenaufruf: function(a, b:longint):longint; stdcall;

begin

  // dll laden
  dllHandle:=LoadLibrary('Rechendemo.dll');

  // Adressen der dll-Funktionen ermitteln
  @Autoraufruf:=GetProcAddress(dllHandle, 'Autor');
  @Rechenaufruf:=GetProcAddress(dllHandle, 'Rechne');

  // Funktionen nutzen
  LabelAutor.Caption:=Autoraufruf;
  SpinEditWert.Value:=Rechenaufruf(2, 212);

end;
```


Alle Eingabegeräte auswerten
Eingabesammlung
Eingaben im Führerstand auswerten
Führerstand Funktionen durchrechnen
Individuell
Führerstand
TProtokollFst
Ersatz durch dll
Fahrzeug (Antrieb, Bremse) durchrechnen
Array phys. Größen
Individuell
Fahrzeug
Ausgabedaten zusammenstellen
Anzeige im Führerstand, TCP-Ausgabe

Aufgabe der dll

- Komplette Bedienung (Motor Start+Stop, HS, Fahrschalter)
- Schaltwerk/Nachlaufsteuerung
- Antriebskennfeld
- Fahrdynamikgrößen
- Zwangsbedingungen
- Also der volle Zugriff auf das Antriebssystem
- Dafür ist ein gewisser Programmieraufwand nötig

Führerstand und Sound ansteuern

- dll kann die Antriebswerte und individuelle Größen setzen
- Sounds und Instrumente können diese Werte verarbeiten wie bisher
- Evtl. weitere individuelle Größen nötig?
- Autor muss in dll-Info die Nutzung der individuellen Größen dokumentieren

Die wesentlichen Funktionen

```
 // ruft der Sim einmal pro Rechenschritt für das selbstgefahrene Fahrzeug und
 // für Fahrzeuge im Mehrfachtraktionsmodus im selbstgefahrenen Zug auf,
 // um die Führerstandsbedienung zu übertragen
   procedure Bedienung(AntriebsRenderModus:TAntriebsRenderModus; var
Prot:TProtokollFst; sp:single); stdcall;

 // ruft der Sim 1x pro Rechenschritt auf, dort sollen die Berechnungen
 // erfolgen wird nur für das selbstgefahrene Fahrzeug und für Fahrzeuge im
 // Mehrfachtraktionsmodus im selbstgefahrenen Zug aufgerufen
   procedure Berechnung(dt:double; AntriebsRenderModus:TAntriebsRenderModus;
Mehrfachtraktionsdaten:TMehrfachtraktionsdaten); stdcall;

 // das wird für Fahrzeuge einmal pro Rechenschritt aufgerufen, die im eigenen
 // Zug im Modus "eigener Tf" fahren und für alle Fahrzeuge im Autopilotmodus
 // es muss hier eine vereinfachte, schnelle Berechnung durchgeführt werden
 // komplexe Rechenmanöver können die Performance beeinflussen, da sehe viele
 // Züge mit diesem Modus im Fahrplan unterwegs sein können
   procedure VorspanntraktionsdatenSetzen(sp, SollLeistungNormiert:single);
stdcall;
```

Ein paar weitere Funktionen beispielhaft

```
// wird vom Sim aufgerufen, wenn ein Nullstellungszwang durch das
// Antriebssystem erfolgen muss
  procedure Nullstellungszwang; stdcall;

// wird vom Sim aufgerufen, wenn sich der Modus des HS ändern soll
// modus:  0: HS ein, 1: HS aus, 2: Spannung frisch an, 3: HS frisch ein
  procedure SetzeHauptschalterzustand(modus:Byte); stdcall;

// div. Fahrgrößen
  function LeseAntriebskraft:single; stdcall;
  function LeseAntriebskraftSollNormiert:single; stdcall;
  function LeseOberstrom:single; stdcall;

// muss true zurückgeben, wenn aus Sicht des Antriebsmodells ein
// Nullstellungszwang aktiv ist
  function NullstellungszwangAktiv:Boolean; stdcall;

// muss true zurückgeben, wenn aus Sicht des Antriebsmodells eine
// Hauptschalterauslösung nötig ist wegen unpassender Fahrleitungsspannung
  function HauptschalterWgOLEnde(FahrlTyp:TFahrleitungstyp;
Spannungsfaktor:single):Boolean; stdcall;
```

Demo-dll

- Demo-dll wird mit Quellcode und Fahrzeug mitgeliefert
- Ist gleichzeitig die wesentliche Dokumentation der Schnittstellen
- Einfaches Demofahrzeug mit einem Sound und einem Melder und ein paar Funktionsbeispielen

Weiteres Vorgehen

- In den nächsten Tagen Beta-Version für Antriebs-dll
- Antrieb mit interessierten Anwendern erproben
- Wenn serienreif, Erweiterung auf dll für dynamische Bremsen
- Überlegenswert wäre ein opensource-Framework für dll-Basisfunktionen
- Wer hat Interesse, dlls zu erstellen?