

Inhaltsverzeichnis

1	Die SOFiSTiK-Datenbasis	2
1.1	Konzept	2
1.2	Begriffe	2
1.3	Zugriff durch SOFiSTiK-Programme	2
1.4	Datei CDBASE.CHM	3
1.5	Kontrollmöglichkeiten	5
1.5.1	DBINFO	5
1.5.2	WinGraf	5
1.5.3	DBView	5
1.5.4	DBPrin	5
1.5.5	künftig vermehrt auch GRAFIX	5
2	Wichtige CADINP-Befehle	6
2.1	Prog Template	6
2.2	@CDB	6
2.3	@KEY Name	6
2.4	@Name	7
2.5	@KEY Nr	9
2.6	@Nr	9
2.7	Variable #CDB_IER	10
2.8	Variable #CDB_LEN	11
2.9	Funktion LIT()	12
2.10	Befehle let# und sto#	12
2.11	Befehl prt#	13
2.12	Befehl LOOP	13
2.13	Befehl IF	14
2.14	Kennwort TXA, TXB	15
2.15	Formatierte Ausgabe von CADINP-Variablen	15
3	Weiterführende Hinweise	16
3.1	SOFiSTiK-Handbuch (sofistik_0.pdf)	16
3.2	SOFiSTiK-Forum	16
3.3	SOFiSTiK-Support-Datenbank	16
4	Beispiele	17
4.1	Auslesen einer Gesamtlastfallliste	17
4.2	Auslesen der Koordinaten eines Knotens	17
4.3	Auslesen von Stabschnittgrößen an den SLN-Enden	17

1 Die SOFiSTiK-Datenbasis

1.1 Konzept

- Das Datenbankformat CDB ist eine SOFiSTiK-Eigenentwicklung (Dr.Casimir Katz + Sabine Gebhard) zur indexsequentiellen Datenverwaltung (kein SQL)
- Für FE-Anwendungen optimiert, d.h. geändert wird der Inhalt, nicht jedoch die Datenstruktur (feste Recordlängen)

1.2 Begriffe

- *Index bzw. Schlüssel* – ist der Schlüssel unter dem Daten abgelegt werden (**direkter** Zugriff)
- *Record bzw. Satz* – ist der logische Record (Bytestream, mehrere zusammengehörige Werte = 1 Zeile) eines Schreib/Lesevorgangs (**sequentieller** Zugriff = zeilenweise, d.h. einer nach dem anderen)
- *Item bzw. Wert* – ist ein Wert eines Records (bei @KEY **sequentieller** Zugriff, d.h. einer nach dem anderen in der richtigen Reihenfolge)
- CDBASE hat einen eigenen Speicherbereich (CDBASEMEM), der sich das Inhaltsverzeichnis (Liste aller vorhandenen Schlüssel) für jede Datenbasis merkt + einem Puffer für häufig bzw. zuletzt gelesene Records

⇒ Ein Schlüssel kann mit @KEY **direkt** angesprochen werden, Werte innerhalb des Schlüssels werden **sequentiell** gesucht

1.3 Zugriff durch SOFiSTiK-Programme

- Zugriff über Programmierschnittstellen für C bzw. C++ , FORTRAN , VisualBasic (nicht Gegenstand dieses SOFINARS)
- Anlegen der Datenbank: nur einige wenige Programme
 - AQUA, SOFIMSH?
 - Wichtige Sätze: 0/99 , 10/0
 - Systemwerte (Materialwerte, Querschnitte, Knotenkoordinaten, Elementgeometrie und –eigenschaften)
 - z.B. Satz 20/0 – Knotengeometrie , 100/0 – Stabgeometrie , 200/0 – Flächenelemente
- Schreibender Zugriff – mehrere Programme
 - SOFILOAD, ASE, AQB, BEMESS
 - Berechnungs- und Bemessungsergebnisse
 - z.B. Satz 12/LF – Lastfallheader, 24/LF – Knotenverschiebungen , 102/LF Stabschnittkräfte

- Lesender Zugriff – praktisch alle Programme
 - o (WinGraf, GRAFIX, DBView
 - o Multitasking: Mehrere Lesezugriffe auf einen Datenbanksatz gleichzeitig möglich. Gleichzeitiger schreibender und lesender Zugriff **NICHT** möglich (WINDOWS-Meldung über „Dateilocks“ - Hände weg)

è @KEY ist nur **LESEND**

è Lesender Zugriff ist erst **NACH** Anlegen des Satzes in der DB sinnvoll, d.h. nach ENDE-Satz (z.B. Knotenkoordinaten stehen erst nach Ende von SOFIMSH? Zur Verfügung, weil erst dann die Vernetzung beginnt; vorher wird bei den meisten Programmen die Eingabe geprüft und

1.4 Datei CDBASE.CHM

- o Beschreibung des Datenbankinhalts
- o Ist immer aktuell, weil zusammen mit den Strukturen und Headern in den DLLs und der Datei CDBASE.CDB erzeugt
- o Datei CDBASE.CDB enthält die formale Datenbankbeschreibung für den CADINP-Zugriff
- o **Hinweis:** CHM ist ein komprimiertes WINDOWS-Hilfeformat, unter VISTA / WINDOWS7 ist das Ausführen (Öffnen) auf Netzlaufwerken evtl. nicht gestattet (dann auf eine lokale Festplatte kopieren)
- o Öffnen im TEDDY bei offener DAT-Datei unter „Hilfe“-„Spezielle Hilfe“-„CDBASE“
- o Syntax (siehe auch Kapitel „SOFISTiK Definitions“-„Syntax“)
 - @Rec – Beschreibung eines Records

@Rec: *Nummer/Key[:selector1[:selector2]] Name Beschreibung*

<i>Nummer</i>	Nummer des Hauptschlüssel
<i>Key</i>	Nummer des Unterschlüssels, kann auch sein:
	<i>ID</i> 4-buchstabiger Name=Literal in ' ' (Achsen)
	<i>NR</i> irgendeine Nummer (Materialien)
	<i>LC</i> Lastfall (mit Header 12/LC)
	<i>DC</i> Bemessungsfall (wie LC aber ohne 12/LC)
<i>selector1</i>	Der 1.Wert im Record (meist Integer – Ganzzahl), kann in jedem Record unterschiedlich sein und legt dadurch beim Lesen die zu verwendende Struktur fest, kann sein:
	* irgendein Wert
	+ jeder Wert > 0
	- jeder Wert < 0
	Z+ jeder Wert >= 0
	Z- jeder Wert <= 0
	nn Wert mit Wildcards (10?01, 1??? , 1001)
<i>selector2</i>	Der 2.Wert im Record (wie 1.Wert)

Name Name des Records (zur Verwendung mit @KEY)
Beschreibung deutsch|englisch

Beispiel:

@Rec: 009/NR:0 SECT	Sectional Values	:V200501
	Querschnittswerte	

@Rec:: SOFiSTiK-interne Sätze

- @.... – Beschreibung eines Wertes innerhalb des Records
 - Jeder Wert hat normalerweise 4 Byte
 - Integer (Ganzzahl) bis ± 2 Milliarden darstellbar
 - Float/Real (Fließkommazahl) mit einer Genauigkeit von 6-7 Stellen

@Nummer Name Dimension Beschreibung

Nummer (1.Spalte) Reihenfolge der Werte im Record
 getrennt nach (führenden) Integern (Ganzzahl) und
 folgenden Fließkommazahlen, mit:

@Nummer= Ganzzahl zur Identifizierung

@Nummer# Folgende (variable) Ganzzahl

@Nummer: Folgende (variable) Fließkommazahl

Name Name des Wertes,
 {} – evtl. Einschränkung für Platte/Rahmen
 [] – Strukturen -> relativ kompliziert zum Auslesen
 (z.B. HIST 80/LC)

Dimension Art des Wertes in []
 Vorangestellte Zahl (z.B. 3[]) = Feld

int Ganzzahl

chr verpacktes 4-buchstabiges Literal (ANSI, kein Ü,Ä)

str verpackter String (2 Buchstaben, Unicode, Ü,Ä erlaubt)

è **chr** und **str** können nicht direkt gelesen werden -> Funktion **LIT()**

- Fließzahl ohne Dimension (z.B. Faktor)
- * Fließzahl mit abhängiger Dimension (z.B. Lastwert)
- nn* Fließzahl mit Dimension (z.B. 1001 = Längeneinheit)
 < 999 = expliziter Unit (nicht durch SEIT UNIA
 veränderbar)
 > 999 = impliziter Unit (Ausgabe durch Einheitenset
 wie in
 SOFISTIK.DIM definiert beeinflusst, siehe
 SOFISTIK.DIM
 und Kapitel „SOFISTiK Definitions“-„Implicit Units“)

Fließzahlenwerte in der Datenbank sind dimensionsrein,

also **immer** in

- **m** (Längeneinheit)
- **kN** (Kraft)
- **s** (Zeit)
- **rad** (Winkelmaß, $1 \text{ rad} = 180/\pi^\circ$)
- **°C** (Temperatur, auch K möglich)


⇒ Werte vor der Ausgabe umrechnen, Nachkommastellen verwenden

1.5 Kontrollmöglichkeiten

1.5.1 DBINFO

- Aufruf aus TEDDY, rechte Maus, „Datenbasis“-„Information“

1.5.2 WinGraf

- Aufruf aus TEDDY „SOFiSTiK“-„Grafische Ausgabe“ oder 
- Darstellung Liste, URSULA
- Export Werteliste, LST-Datei im TEDDY

1.5.3 DBView

- Aufruf aus TEDDY „SOFiSTiK“-„Datenbank View“ oder 

1.5.4 DBPrin

- Aufruf im TEDDY mit PROG DBPRIN und URSULA 

1.5.5 künftig vermehrt auch GRAFIX

- Aufruf aus TEDDY „SOFiSTiK“-„Grafix“ oder 
- Tabellenbereich „Alle Werte“

2 Wichtige CADINP-Befehle

2.1 Prog Template

- Ablauf innerhalb CADINP, d.h. innerhalb eines **PROG**
- **PROG TEMPLATE** führt keine Berechnungen aus (eigentlich Eingabemaske), ideal geeignet

2.2 @CDB

- Befehl **@CDB** *dateiname* verweist auf beliebige Datenbank
- Normalerweise nicht notwendig, da die aktuelle Datenbank \$(NAME) verwendet wird

2.3 @KEY Name

☞ CDBASE.CHM ist die Referenz !

- Jedes @Key erzeugt ein Rewind (alles von vorn)
- Außerhalb eines LOOP aufrufen (Performance)
- Entspricht einer Selektion (1.Integer), d.h. nicht jede Zeile in der Datenbank wird ausgegeben, sondern nur die, die der Selektion entsprechen
- Ungültige Aufrufe (z.B. nicht vorhandene) führen zu CADINP-Fehlermeldungen

Beispiel (@key ???):

+++++ Fehler Nr. 10143 in Programm CADARI
CDBASE.CDB enthält keine passende Structure für @KEY

Selektoren:

- Zu den Schlüsseln **ID,LC,DC,NR** gehört immer ein passender Selektor **KWL** (Name, Lastfall, Nummer)
- Ein fehlender Selektor KWL führt zu einer CADINP-Fehlermeldung

Beispiel (@key N_DISP ohne Selektor):

+++++ Fehler Nr. 10144 in Programm CADARI
Dieser Index @KEY erfordert eine explizite Nummer bei KWL

- Weitere Selektoren (**SEL1...SEL6**) grenzen die Auswahl der folgenden @Name ein
- Maximal 6 Selektoren möglich

- Es müssen nicht alle Selektoren gesetzt sein, nicht gesetzte werden auch nicht berücksichtigt

Beispiel: Auslesen der charakteristischen elastischen Schnittgrößen (Normalkraft) eines Querschnitts (Nr. 1)

```
@key SECT_PLA kwl 2 sel3 100  
Let#N @WPN
```

ID (=6) ist zwar durch SECT_PLA voreingestellt, zählt aber als Selektor

2.4 @Name

- Kann nur zum passenden „@Key Name“ aufgerufen werden
- Ungültige / unpassende Namen führen zu CADINP-Fehlermeldungen

```
+++++ Fehler Nr. 10132 in Programm CADARI  
Ungültige Zahl/Ausdruck Typ 502 (unzulässiger Operator/CDB-Item-Name)
```

- Nochmaliger Aufruf des **selben** Namens führt zum Lesen der nächsten Zeile !
- Aber auch der Aufruf eines Namens, der in der Zeile **DAVOR** liegt, führt zum Lesen der nächsten Zeile

⇒ Reihenfolge der Namen in der Struktur berücksichtigen

Beispiel für falsche Reihenfolge:

```
@key N_DISP 1  
Let#ux @UX  
Let#nr @Nr
```

#nr gehört schon zum nächsten Knoten und passt nicht zu #ux

Beispiel für richtige Reihenfolge:

```
@key N_DISP 1  
Let#nr @Nr  
Let#ux @UX
```

#nr und #ux gehören zum selben Knoten

Offset:

- Auf Felder (z.B. Knotenkoordinaten) kann durch Setzen eines **Offset** zugegriffen werden
- Man kann mit einem Offset aber auch die nächsten Werte ab der aktuellen Position abrufen
- @Name und @(Name+0) sind gleichwertig

⇒ Beim Arbeiten mit Offset muss der Ausdruck mit () geklammert werden

Beispiel Knotenkoordinaten:

```
@key NODE
Let#nr @Nr
Let#X @XYZ
Let#Y @(XYZ+1)
Let#Z @(XYZ+2)
```

Selektoren:

- Der 1. Integer (zumeist die Elementnummer NR) kann als zusätzlicher (zu @Name) Selektor angegeben werden mit **@(NR,Name)**

⇒ Die Nummern **NR** sollten in aufsteigender Reihenfolge aufgerufen werden

- CADINP macht nur 1x ein internes REWIND (Lesen von vorn), wenn Nr gesetzt ist
- Ansonsten wird der Wert nicht gefunden

Beispiel für falsche Reihenfolge (nicht aufsteigend):

```
@key NODE
let#nr3 @(3,nr)
let#nr2 @(2,nr)
let#nr1 @(1,nr)
```

Führt zur CADINP-Fehlermeldung:

```
+++++ Fehler Nr. 10126 in Programm CADARI
CDB-Record existiert nicht oder Ende erreicht 200/ 0: 1
```

Beispiel für richtige Reihenfolge:

```
@key NODE
let#nr1 @(1,nr)
let#nr2 @(2,nr)
let#nr3 @(3,nr)
```


2.5 @KEY Nr

- Sinngemäß gelten die Regeln wie für **@Name**
- Unterschied: Es wird erst mal keine Selektion vorgenommen, d.h. jede Zeile in der Datenbank wird ausgelesen
- Sinnvoll für Datensätze, bei denen sich verschiedene Strukturen abwechseln, z.B. Stabdefinition
- **Nr** entsprechend Beschreibung CDBASE.CHM (2 Integer)

Selektoren:

⇒ **KWL** muss immer angegeben werden

2.6 @Nr

- Nur in Zusammenhang mit **@KEY Nr**

⇒ Reihenfolge der Nummern in der aktuellen Zeile berücksichtigen

- Die angegebene Position über Nr ist relativ zu der Anzahl der definierten Selektoren in **@Key Nr**
- Die Positionen werden entsprechend der Datenbankbeschreibung CDBASE.CHM durchgezählt (Feldlängen berücksichtigen !)

Beispiel für gleichwertige Ausgabe (Y-Koordinate eines Knotens):

```
@key 20 0  
let#Y @6  
  
@key 20 0 -1  
let#Y @5  
  
@key 20 0 -1 -1 -1 -1  
let#Y @2  
  
@key 20 0 sel4 -1  
let#Y @2
```

- Wenn bei **@Key Nr** Selektoren angegeben sind, kann man auf diese mit Positionen ≤ 0 zugreifen, man zählt aber der aktuellen Position rückwärts

Beispiel für gleichwertige Ausgabe (Nummer eines Knotens):

```
@key 20 0 -1  
let#Nr @0  
  
@key 20 0 -1 -1  
let#Nr @-1  
  
@key 20 0 -1 -1 -1 -1  
let#Nr @-3  
  
@key 20 0 sel4 -1  
let#Nr @-3
```

2.7 Variable #CDB_IER

- Um CADINP-Fehlermeldungen beim Erreichen des Datensatzendes zu vermeiden, kann eine Fehlerbedingung abgefragt werden
- Hierfür wird die CADINP-Variable **#CDB_IER** verwendet
- Bei jedem @-Zugriff wird diese neu gesetzt mit:
 - Ø 0 alles ok, Wert existiert
 - Ø 1 Record konnte gelesen werden, aber Satzlänge passt nicht
(siehe Erläuterung zu **#CDB_LEN**)
 - Ø 2 Ende des Datensatzes erreicht (bzw. überschritten)
 - Ø 3 Datensatz (KWH/KWL) existiert nicht bzw. hat keine Werte
(z.B. nicht definierter Lastfall

Beispiel (Lastfallliste erstellen, #CDB_IER wird auf „3“ gesetzt, wenn der Lastfall nicht existiert):

```
let#cdb_ier 0  
let#lf 0  
loop 999  
  let#lf #lf+1  
  @key LC_CTRL #lf  
  if #cdb_ier<2  
    let#rtex LIT(@RTEX)  
    TXA #(#lf,6.0) #rtex  
  endif  
endloop
```

2.8 Variable #CDB_LEN

- Einige Datensätze können, je nach Berechnung, kürzer sein, als in der Beschreibung CDBASE.CHM angegeben (optionale Werte)
- Um zu festzustellen, wie lang der Datensatz ist, d.h. wie viele Werte er besitzt, kann CADINP-Variable **#CDB_LEN** verwendet werden

Beispiel (Knotenauflagerkräfte ausgeben, **#CDB_LEN** wird auf „15“ gesetzt, wenn welche existieren oder auf „8“, wenn es keine gibt):

```
let#cdb_ier 0
let#cdb_len 0
@key N_Dis 1
loop
  let#nr @NR
  let#py @PY
  if #cdb_len>10
    txa Knoten #(#nr,5.0) PY = #(#py,7.3) kN
  endif
endloop #cdb len>1
```

- Allerdings erhält auch die CADINP-Variable **#CDB_IER** einen anderen Wert (=1), wenn die Satzlänge kürzer wird, nur ist hier nicht spezifiziert, um wie viel kürzer der Satz ist

Gleiches Beispiel (Knotenauflagerkräfte ausgeben, **#CDB_IER** wird auf „0“ gesetzt, wenn welche existieren oder auf „1“, wenn es keine gibt):

```
let#cdb_len 0
let#cdb_ier 0
@key N_Dis 1
Loop
  let#nr @NR $ CDB_IER=0 , weil NR innerhalb der gelesenen Länge
  liegt
  if #cdb_ier<2
    let#py @PY $ CDB_IER=1 , weil PY ausserhalb der gelesenen
    Länge liegt
    if #cdb_ier<1
      txa Knoten #(#nr,5.0) PY = #(#py,7.3) kN
    endif
  endif
```

⇒ Am besten immer beide Variablen definieren, weil so einige lästige CADINP-Fehlermeldungen umgangen werden können

2.9 Funktion LIT()

- Bezeichnungen und Literale sind in der Datenbank nicht direkt zugänglich, da in Integer-Werte verpackt sind (Definition als **[chr]** oder **[str]**)
- Mit der Funktion **LIT()** können diese Texte in CADINP-Variable als lesbare Texte gespeichert werden

Beispiel (Lastfallname):

```
@key LC_CTRL 1  
let#rtex LIT(@RTEX)
```

2.10 Befehle let# und sto#

- CADINP-Variable sind eigentlich Felder von Float/Real (Fließkommazahlen)

⇒ Große CADINP-Felder am Anfang initialisieren, z.B. durch

- Der Unterschied zwischen **LET** und **STO** besteht darin, dass die **LET**-Variablen nur PROG-lokal sind, die **STO**-Variablen dagegen in der Datenbank gespeichert werden
- Können auch Literale speichern, wenn sie in der Form **let#variable 'literal'** angelegt werden
- Längere (als 4 Zeichen) Texte werden in der Variablen als Feld gespeichert **#variable(0) = 'lite'**, **#variable(1) = 'ral'**
- Felder werden als Aufzählung durch Kommas getrennt (**ohne Leerzeichen !**) definiert oder durch Belegung einzelner Indizes

Beispiel (Variablenfeld):

```
Let#variable 1,2,3
```

Ist gleichwertig zu

```
Let#variable(0) 1  
Let#variable(1) 2  
Let#variable(2) 3
```

2.11 Befehl prt#

- Ausgabe des Inhalts einer CADINP-Variable im E-Format in die LST-Datei zur schnellen Kontrolle
- Ausgabe taucht nicht in URSULA auf

2.12 Befehl LOOP

- Schleifenbefehl in CADINP
- Wird mit **ENDLOOP** abgeschlossen
- Mit **LOOP nn** kann die Anzahl der Umläufe definiert werden, max. Anzahl 999, um Endlosläufe zu verhindern, Voreinstellung: 999, wenn **nn** nicht gesetzt

⇒ Wenn mehr als 999 Umläufe erforderlich sind, kann man mehrere (2) LOOPS ineinander schachteln

- Statt einer Zahl **nn** kann auch der Name eines CADINP-Variablenfeldes verwendet werden, dann ist die Anzahl der Durchläufe gleich der Länge des Feldes

Beispiel:

```
Let#feld 3,4,1,5,7,2
Loop#n feld
  @key LC_CTRL #feld(#n)
  ....
Endloop
```

- Mit **LOOP#variable** kann eine Zählvariable, beginnend mit „0“ definiert werden
- Mit **ENDLOOP bedingung** kann eine Abbruchbedingung definiert werden, die Schleife läuft so lange, bis **bedingung** unwahr (=0) wird

2.13 Befehl IF

- Bedingte Ausführung in CADINP
- Wird mit **ENDIF** abgeschlossen
- Erfüllt (=wahr), wenn ungleich 0

è Die **IF**-Bedingung darf keine Leerzeichen enthalten

- Genaue Vergleiche mit == oder <> funktionieren oft nicht wie gewollt, weil die CADINP-Variablen immer als Float-Zahlen gespeichert sind

è Vergleiche besser mit einer kleinen Toleranz ausführen

Beispiel:

```
let#val1 -26.184
let#val2 -26.184
let#erg1 (#val1==#val2)      $ wahr, da exakt gleich

@key N_Displ 2
let#val2 @(4,py)
let#erg2 (#val1==#val2)      $ falsch, da nicht exakt gleich

let#erg3 abs(#val1-#val2)<0.01 $ wahr, da mit toleranz
```

- Mehrere Bedingungen können mit logischen „und“ & bzw. „oder“ | miteinander verknüpft werden, die einzelnen Bedingungen sollten dann geklammert () werden

Beispiel logisches UND & , logisches ODER | :

```
let#erg1 (1>0)&(2>0) $ wahr = 1
let#erg2 (1>0)&(2<0) $ falsch = 0
let#erg3 (1>0)|(2<0) $ wahr = 1
let#erg3 (1<0)|(2<0) $ falsch = 0
```

Beispiel Verknüpfung:

```
@key N_Displ 1
loop
  let#py @PY
  if (#cdb_ier<1)&(abs(#py)>30)
    txa Werte grösser 30 kN , PY = #(#py,7.3) kN
  endif
endloop #cdb_ier<2
```

2.14 Kennwort TXA, TXB

- Textbereiche, also einzelne Zeilen, am Anfang bzw. Ende einer Seite
- Bei PROG TEMPLATE können beide genutzt werden

2.15 Formatierte Ausgabe von CADINP-Variablen

- Variable, die als Literal, also mit ' ' oder der Funktion LIT() angelegt wurden, können genau so in TXA/TXB übernommen werden
- Alle anderen Variablen sollten formatiert werden, ansonsten werden unpassende Nachkommatausgaben, auch bei Ganzzahlen
- Form: **#(#variable,gesamt_spalten_anzahl.nachkommastellen)**

Beispiel Ganzzahl (z.B. Lastfallnummer):

```
let#lf 71  
txa LF = #(#lf,5.0)
```

Beispiel Ganzzahl (z.B. Auflagerkraft):

```
let#py @py  
txa PY = #(#py,7.3) kN
```

3 Weiterführende Hinweise

3.1 SOFiSTiK-Handbuch (sofistik_0.pdf)

- Kapitel 6.2.13. „LET - und STO – Variable“ S 6-10 ff
- Kapitel 6.2.17. „LOOP,ENDLOOP – Schleifen und Sprünge“ S 6-17 f
- Kapitel 6.2.18. „IF – Logische Abfragen“ S 6-19 f
- Kapitel 6.2.20. „@KEY – Zugriff auf CDBASE“ S 6-21
- Kapitel 6.2.20. „@() – Zugriff auf CDBASE“ S 6-21 ff
- Zu finden im TEDDY unter „Hilfe“-„Spezielle Hilfe“-„SOFiSTiK“
- Oder im TEDDY unter unter „Hilfe“-„SOFiSTiK-Handbücher“-„SOFiSTiK“

3.2 SOFiSTiK-Forum

- <http://www.sofistik.de/forum/index.php>
- Suche nach „@KEY“ bringt z.Zt. 73 Treffer
- VBA-Tool zum Auslesen von Werten aus der Datenbank und zum Weiterverarbeiten mit Excel von Hr. Andrussow z.B. unter <http://www.sofistik.de/forum/viewtopic.php?f=3&t=4017&p=11530#p11530>

3.3 SOFiSTiK-Support-Datenbank

- FAQ-Datenbank unter SOFiSTiK-Online
<http://www.sofistik.de/support/sofistik-online-login/>

4 Beispiele

4.1 Auslesen einer Gesamtlastfallliste

Siehe in Beispieldatei sofinar_datbankzugriff.dat unter Kapitel „LOOPS schachteln“

Ausgabe:

Lastfall	Name
1	Eigengewicht
2	Belastung
111	MAX-MY BEAM
112	MIN-MY BEAM

4.2 Auslesen der Koordinaten eines Knotens

Siehe in Beispieldatei sofinar_datbankzugriff.dat unter Kapitel „Finden von Knoten“

Ausgabe:

Knoten	2 gefunden bei x= 0.00 m y= 0.00 m z= 0.00 m
Kein Knoten gefunden bei x= 1.00 m y= 2.00 m z= 3.00 m	
Knoten	4 gefunden bei x= 5.00 m y= 3.00 m z= 0.00 m

4.3 Auslesen von Stabschnittgrößen an den SLN-Enden

Siehe in Beispieldatei sofinar_datbankzugriff.dat unter Kapitel „Auslesen von Stabschnittkräften an den SLN-Enden“

Ausgabe:

SLN	StabX-Stab[m]	N[kN]	Vz[kN]	My[kNm]
1 (Anfang, S= 0.0 m) 700001	0.0	32.88	0.25	-5.63
1 (Ende, S= 0.6 m) 700001	0.6	32.88	0.25	-5.48
2 (Anfang, S= 0.0 m) 700002	0.0	32.88	-2.31	-4.04
2 (Ende, S= 17.5 m) 700019	1.0	32.88	0.00	0.00
3 (Anfang, S= 0.0 m) 700020	0.0	17.79	-0.20	-5.51
3 (Ende, S= 0.6 m) 700020	0.6	17.79	-0.20	-5.63
4 (Anfang, S= 0.0 m) 700021	0.0	17.79	-1.75	-4.78
4 (Ende, S= 17.5 m) 700038	1.0	17.79	0.00	0.00
5 (Anfang, S= 0.0 m) 700039	0.0	2.15	-0.41	-7.43
5 (Ende, S= 0.6 m) 700039	0.6	2.15	-0.41	-7.68
6 (Anfang, S= 0.0 m) 700040	0.0	2.15	-1.64	-5.15