Contents

1 The SOFiSTiK-Database

1.1 Concept

- The database structure CDB was developped by SOFiSTiK, Dr.Casimir Katz + Sabine Gebhard. The database for the SOFiSTiK programs is a proprietary system based on an index sequential philosophy. This stands for an sequential access to data organized in logical records addressed by a key formed by two Integers.
- The system CDBASE is optimized for typical FE-Analysis. Typically there exist large data volumes which are organized in a sequential manner. It is quite more often that data is changed than the data structure itself. That's why some operations are not possible in CDBASE.

1.2 Terms and Definitions

- *Key* – all data will be saved under a specific key (this key may be used for a **direct** access)
- *Record* – ist der logische Record (Bytestream, mehrere zusammengehörige Werte = 1 Zeile) eines Schreib/Lesevorgangs (**sequentieller** Zugriff = zeilenweise, d.h. einer nach dem anderen)
- *Item* – ist ein Wert eines Records (bei @KEY **sequentieller** Zugriff, d.h. einer nach dem anderen in der richtigen Reihenfolge)
- CDBASE has its own area(CDBASEMEM) to save the content (list of all available keys) for every dababase. This is also a cache for the last read records.

➔ With the @KEY command one can get a **direct** access to the values saved there. A **sequential** search will be processed.

1.3 Database Access with SOFiSTiK-Program Modules

- We provide some prgramming interfaces for C or C++ , FORTRAN , VisualBasic programming languages. (this is not shown here)

- Only a few programs are able to create a new data base
    - AQUA, SOFIMSH?
    - Most important records are: 0/99 , 10/0
    - system values (materials cross sections, nodal ccordinates, element geometry and properties))
    - e.g. record 20/0 – nodes , 100/0 – beam elements , 200/0 – area elements

- some program modules can write into the database
    - SOFILOAD, ASE, AQB, BEMESS
    - Writing of analysis and design results, e.g. record 12/LC – header load case, 24/LC – nodal displacements, 102/LC internal forces and moments beam elements
- Nearly all programm can read from the database
    - WinGraf, ResultViewer
    - Multitasking is possible, but reading and writing at the same time is NOT possible (WINDOWS-Message „database locked“)

➔ The @KEY command is read only

➔ Reading content from the database is only reasonable after all information is written.
For example nodal coordinates are available only after the END command

1.4 File CDBASE.CHM

- A description of the dabavase could be found in the CDBASE.CHM over TEDDY menu help >
- This descriptionis always up to date, because it will be produced all over again with new structures, headers inside the DLLs and the file CDBASE.CDB itself.
- The file CDBASE.CDB contains the formal description of the database for direc access via CADINP.
- **Note**: The CDBASE.CHM file is a zipped WINDOWS help file, which cannot be opend on network drives. You may copy it to you local harddrive and open it there.
- You will find the CDBASE.CHM file inside the TEDDY menu. Please open any Dat-file and goto menu “Help” > “Special Help” > “CDBASE”
- Syntax (please see chapter „SOFiSTiK Definitions“-„Syntax“)
    @Rec – description of a particular record

number	is a numeric value defining the fist key
key	is a numeric value defining the second key OR ID indicating a char identifier or NR indicating "any number" or LC indicating "any loadcase" or any other text
selector1	is a selection string indicating the value of the first item of the record to be matched as follows: * = any value + = any positive value - = any negative value Z+ = any positive value or zero Z- = any negative value or zero nn = numeric value including ? as wild

	(e.g. 10?01, 1??? , 1001)
selector2	is a string indicating the value of the second item of the record to be matched as above.

Example:

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

@Rec: SOFiSTiK-internal sentences

- @.... – description of a special value iside the record
    - Every value has a general length of 4 Byte
    - Integer with a value up to ±2 billion
    - Float/Real with an accuracy of 6-7 decimal places

@Number Name Dimension Description

Number	(1.column) sequence of values inside the record separated by (leading) integer and following real values, with:
@Number=	Integer for idetification
@Number#	Following (varialbe) integer
@Number:	Following (varialbe) real
Name	Name of value, {} – possible constraint for slab/frame [] – structure → vey complex to read (e.g. HIST 80/LC)
Dimension	type of value in [] with prefix (e.g. 3[]) = array
int	integer
bhr	lexical packed literal with 4 letters (ANSI, no Ü,Ä)
chr	lexical packed literal with 4 letters (ANSI, no Ü,Ä), difference to *bhr*: reversed sequence
str	packed string (2 letters, unicode, including Ü,Ä)

→ **chr**, **bhr** and **str** cannot be read directly → use function **LIT()**

-	real without dimension (e.g.factor)
*	real with depending dimension (e.g. load value)
Nn	real with dimension (e.g. 1001 = lenght unit) < 999 = explicit unit (cannot be changed with PAGE UNIA) > 999 = implicit unit (output is defined by a unit set, see CDBASE.CHM, chapter „SOFiSTiK Definitions“-„Implicit Units“) All real values inside the database uses a consisten set of units, which are

m	(lenght)
kN	(force)
s	(time)
rad	(angular dimension, 1 rad = 180/π °)
°C	(temperature , alos [K] possible)

1.5 Check Options

1.5.1 DBINFO

- Call from inside TEDDY, right mouse click, „Datenbase“-„Information“

1.5.2 WinGRAF

- Call from inside TEDDY via menu „SOFiSTiK“ > „WinGRAF“ or with button
- Export list of values to LST-file, men „File“ > „ Export values list (.LS)“

1.5.3 ResultViewer

- Call from inside TEDDY via menu „SOFiSTiK“ > „Result Viewer“ or with button

1.5.4 CDB Export to Excel

- Call from inside TEDDY via menu „SOFiSTiK“ > „CDB Export to Excel“ or with button
- This command opens MS EXCEL. Please activate the macros. Use the command ‘SOFiSTiK CDB2Excel’ inside the menu Add-Ins.

2 Important CADINP-Commands

2.1 Prog Template

- To be used inside CADINP input sequence
- **PROG TEMPLATE** does not process any analysis. It is designed to create user defined input masks

2.2 @CDB

- With command **@CDB** *filename* an database can be connetect to the input file
- this command is normally not necessary, because the current dtabase $(NAME) will be used automatically.

2.3 @KEY Name

➔ Please see CDBASE.CHM as reference!

- Every commad @Key creates a rewind
- Call this command only outside a LOOP
- Corresponds to a selection (1. integer), that means only the selected line out of the database will be exported
- Illegal calls result in a CADINP errror message

Example (@key ???):

```
+++++ error no. 10143 in program SOF_VAR
CDBASE.CDB does not contain a suitable structure for @KEY
```

Selectors:

- The keys ***ID,LC,DC,NR*** need always a corresponding selector ***KWL*** (name, load case, number)
- A missing selection strings KWL results in a CADINP errror message

Example (@key N_DISP without selector):

```
+++++ error no. 10144 in program SOF_VAR
This index @KEY requires an explicit definition of a number at KWL
```

- Additional selection strings (***SEL1…SEL6***) can be used as additional filter for the following @Name command
- A maximum of 6 selection strings are possible
- It is not necessary to use all 6 selection strings.

Example: Output of normal force in steel section 2

```
@KEY SECT_PLA KWL 2 SEL3 101
Let#N @WPN
ID (=6) is defined by default via SECT_PLA, but counts also als slections
string (=sel1)
```

2.4 @Name

- Wokrs only with the corresponding „@Key Name“
- Illegal names result in a CADINP error message

```
+++++ error no. 10132 in program SOF_VAR
Improper number/expression type 502 (illegal operator/CDB-Item-Name)
```

- Recall of the same name results in reading the following line!
- Calling a name of the previous line results also in reading the following line.

➔ Very important is the sequence of names inside the structure

Example wrong sequence:

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

#nr belongs to the following node and not to  #ux
```

Example correct sequence:

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

#nr and #ux belong to the same node
```

Offset:

- Using a OFFSET you can get access to arrays,(e.g. nodal coordinates)
- With an OFFSET you may call the following value beginning with the current position.
- The input of @Name and @(Name+0) are the same

➔ Using an OFFSET the command has to be inside brackets **( )**

Example nodals coordinates:

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

Selection Strings:

- The 1. integer (mostly the element number NR) can be used as additional selection string, like @***(NR,Name)***

➔ Ascending numbers ***NR*** can be used only

- Internally CADINP starts only once a REWIND (sequential reading from the beginning)

Examples for a wrong input sequence (not ascending):

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

This results in a CADINP error message:

```
+++++ error no. 10126 in program SOF_VAR
CDB-Record does not exist or end reached 20/ 0: 1
```

Example for a correct input sequence:

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

2.5 @KEY *Nr*

- The same rules like for **@Name** are valid
- The difference is, that no selection takes place and all rows of the database will be read.
- This is a common command for files with changing structures, like beam elements
- ***Nr*** according to CDBASE.CHM description (2 Integer)

Selection String:

➔ **KWL** must be defined anytime

2.6 @Nr

- To bo used only with **@KEY Nr**

➔ the sequence of numbers in the same row is important

- The position selected with Nr is relative to the number of defined slectors in **@Key *Nr***
- The position is numbered according to the description CDBASE.CHM (please take care of the field lenght !)

Example select in different ways a y-coordinate of the same node:

```
@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
```

- In caes a selection string is used with **@Key *Nr*** prior positions can be selected with a valu <= 0

Example equivalent output (nodal number):

```
@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

- To avoid CADINP error messages reaching the end of the record, a special error condition can be used
- A predefined CADINP varialbe **#CDB_IER** can be used
- With every new @ accessthe variable gets a new value, as is:
  - **0** all ok, value exists
  - **1** record could be read, but length is not correct (see Erläuterung zu **#CDB_LEN**)
  - **2** end of record reached
  - 3 data (KWH/KWL) does not exist or does not contain any values(e.g. loadcase not defined)

Example (create list of load cases, #CDB_IER =3 in case load case not existing):

```
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

- Some records can be shorter, that described inside the CDBASE.CHM (optional values)
- To get the information about the record length the CADINP variable **#CDB_LEN** can be used

  Example (select nodal support forces, **#CDB_LEN** „=**15**“ in case ther are results, or „**=8**“ with no results):

```
let#cdb_ier 0
let#cdb_len 0
@key N_Disp 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
```

- Allthough the CADINP variable **#CDB_IER** will be set to **„=1“**, in case the lenght is shorter. There will be no information how much shorter the length realy is.

  Example (select nodal support forces, **#CDB_IER** „**= 0**“ incase there are results, or „**=1**“ with no results):

```
let#cdb_len 0
let#cdb_ier 0
@key N_Disp 1
Loop
    let#nr @NR      $ CDB_IER=0 , because NR is within the read length
    if #cdb_ier<2
        let#py @PY   $ CDB_IER=1 , because PY is outside the read length
        if #cdb_ier<1
             txa node #(#nr,5.0) PY = #(#py,7.3) kN
        endif
    endif
endloop #cdb_ier<2
```

➔ We recommend to define both variable to avoid annoying CADINP error messages

2.9 Function LIT()

- Descriptions and literal names are not directly accessible, because they were saved in the data base as integer values (Definition as **[chr]**, **[bhr]** oder **[str]**)
- With the function **LIT()** the values can be saved as readable text varialbe in CADINP

Example (load case titl):

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

2.10 Commands let# and sto#

- The CADINP variables are arrays of real numbers (DOUBLE)

➔ big arrays should be defined at the beginning, e.g. „let#variable(1000) 0“

- The variable types **LET** and **STO** will be treated differently. The **LET** variable will be unsed locally ony inside one block PROG…END. The **STO** variable will be saved inside the database an can be used anywere inside the DAT input file.
- You may use also assign literal with form **let#variable ’literal’**
- Literals larger than 8 letters will be saved as an array #variable(0) = ’String-L’ , #variable(1) = ’iteral ’
- It is possible to get acces to different parts of the literal using a specific index for beginning and end (similar to FORTRAN). The start index begins with „1“ , e.g. #variable(3:6) ergibt „ring“
- Arrays will be defined as a list, devided be komma, without any blank

Example (array of variables):

```
Let#variable 1,2,3

Is the same as

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

2.11 Command prt#

- This command enables you to see the variable value in the ECHO print out in result viewer.

2.12 Command dbg#

- To trace the assignment of values
- This will toggle test prints and an interactive debug mode.
- DBG# uses the variable #0, which cannot be used for other purpose therefore
- The output can be seen if the ECHO print is activated inside result viewer.

- **DBG#0** No output of intermediate values
- **DBG#1** Output of the generated input records
- **DBG#2** Additional output of all value assignments
- **DBG#3** Additional output of selected structures (CDB access)
- **DBG#4** Printout to console stream/window
- **DBG#8** Input from console stream/window (interactive mode)
- **DBG#** Switch between option 15 and option 0 (=break and continue)
- **DBG# -2** Immediate STOP of total program run, although all outstanding TXE-Lines will be printed after the error message

2.13 Command LOOP

- Loops in CADINP
- To be finished with **ENDLOOP**
- The input **LOOP *nn*** defines the number of repitions.
- The maximum number of loops is limited to 999, which is also the default setting in case no ***nn*** is set.

➔ in case it is necessary to have more than 999 loops, it is possible to nest severals loop’s

- Instead of a number ***nn*** it is also possible to use a CADINP variable. If this variable is defined as an array the number of loops is equal to the length of the array

Example:

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

- With **LOOP#*variable*** a counting variable can be defined. The first calue starts with „0“

- With **ENDLOOP *constraint*** it is possible to create specific break inside the loop. Usually the loop will be used as long ans the contraint condition is true (=0)

2.14 Command IF

- Conditional blocks in CADINP
- Block input will be closed with **ENDIF**
- The conditional block is executed if the expression following the IF is greater than zero (=true), in case of (=false) the value is 0.

> ➔ it is not allowed to have any blancs inside the **IF**-condition

- Exact comparison with == or <> do not work most of the time, because all CADINP varialbes will be saved als float numbers

> ➔ For comparison please use always a little tolerance

Example:

```
let#val1 -26.184
let#val2 -26.184
let#erg1 (#val1==#val2) $ true = 1, because values are exactly the same

@key N_Disp 2
let#val2 @(4,py)
let#erg2 (#val1==#val2  $ false = 0, because values are not exactly the same

let#erg3 abs(#val1-#val2)<0.01  $ true = 1, because of tolerance
```

- Multiple conditions are possible with logical „and“ **&** respectively „or“ **.** Every single condition should be written with brackets ( )

Example logical AND **&** , logical OR **|** :

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

Example logical combinations:

```
@key N_Disp 1
loop
    let#py @PY
    if (#cdb_ier<1)&(abs(#py)>30)
        txa value larger than 30 kN , PY = #(#py,7.3) kN
    endif
endloop #cdb_ier<2
```

2.15 Record TXA, TXB

- To write additional lines of text at the st**a**rt and ath the **e**nd of the output.
- Also in PROG TEMPLATE you may use both

2.16 Formated Output of CADINP-Variables

- 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 Nachkommas ausgegeben, auch bei Ganzzahlen
- Form: **#(*#variable,gesamt_spalten_anzahl.nachkommastellen*)**

  Example integer (e.g. load case number):

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

Example integer (e.g. support force):

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

3 Additionals Information

3.1 SOFiSTiK-Manual (sofistik_1.pdf)

- Chapter 8.2.13. „LET - and STO – Variables“
- Chapter 8.2.17. „LOOP,ENDLOOP – Loops and Jumps“
- Chapter 8.2.18. „IF – Logical Condistions“
- Chapter 8.2.20. „@KEY – Access to the CDBASE“
- Chapter 8.2.21. „@() – Access to the CDBASE“
- The manual can be found via TEDDY/SSD menu „Help“-„Special Help“-„SOFiSTiK“
- Or via TEDDY/SSD menu „Help“-„SOFiSTiK-Documentation“-„SOFiSTiK“

3.2 SOFiSTiK-User Group

- http://www.sofistik.com/forum/
- Search for „@KEY“ give you more than 150 matches

3.3 SOFiSTiK-Support-Datenbank

- FAQ-Database inside SOFiSTiK-Online portal (access for customers with maintenance contract only)
  http://www.sofistik.com/support/sofistik-online-login/

4 Examples

4.1 Exp 1: Get a List with all Load Cases

Please see Chapter “EXP1: …” in data file cdb-access-examples_1.dat

Output:

```
Loadcase   Name
       1   self weight with factor 1.5
       2   additional load
     111   MAX-MY STAB
     112   MIN-MY STAB
```

4.2 Exp 2: Print Nodal Coordinates

Please see Chapter “EXP2: …” in data file cdb-access-examples_1.dat

Output:

```
Node      2 found at x=  0.00 m  y=  0.00 m  z=  0.00 m
Sorry, no node foud at x=  1.00 m  y=  2.00 m  z=  3.00 m
Node      4 found at x=  5.00 m  y=  3.00 m  z=  0.00 m
```

4.3 Exp 3: Read Beam Forces at the End of Structur Lines

Please see Chapter “EXP3: …” in data file cdb-access-examples_1.dat

Output:

```
SLN                    Beam  X-section[m]     N[kN]    Vz[kN]    My[kNm]
  0 (Start, S=  0.0 m)    1       0.0        -21.95     -0.00     -0.00
  0 (End  , S= 11.0 m)    3       3.0        -22.82      0.00     -2.18
```