```
m m
zO
J
S l1;0,0,100,103,120
H 30,0,T,R0,B0
O R,P
B:Barcode1;63.62,64.03,0,code128,10.58,0.25;[U:CODEA]596
T:Text1;73.03,78.28,0,3,4.86,k,q80;5965101721333
G4.108,3.626,0;R:112.9,92.4,1,1
G3.969,21.96,0;L:112.6,0.5,s,s
G3.969,34.92,0;L:112.6,0.5,s,s
G3.969,43.92,0;L:112.6,0.5,s,s
T:Text2;7.938,9.625,0,3,3,k;CAB Produktechnik GmbH & Co
T:Text3;7.938,13.59,0,3,3,k;Wilhelm-Schickard-Str. 14
T:Text4;7.938,17.5,0,3,3,k;D-76131 Karlsruhe
T:Text5;7.938,27.67,0,5,5,b,k;Etikettendrucker
T:Text6;7.829,32.7,0,5,5,b,k;Label Printer
T:Text7;7.938,41.47,0,5,6,b,k;EOS1/300
T:Text8;7.938,48.2,0,3,3,k;Artikel-Nr.
T:Text9;7.938,52.43,0,3,3,k;Article no.
```

```
T:Text22;38.1,84.75,0,5,6,b,k;V 4.0
I:Image2;105,44.98,0;68196983
A [NO]1
```

Edition 04/2022

Products need Labeling

Programming a cab Printer

JScript

Copyright

This documentation as well as translation hereof are property of cab Produkttechnik GmbH & Co. KG. The replication, conversion, duplication or divulgement of the whole manual or parts of it for other intentions than its original intended purpose demand the previous written authorization by cab.

Editor

Regarding questions or comments please contact cab Produkttechnik GmbH & Co. KG.

Topicality

Due to the constant further development of our products discrepancies between documentation and product can occur. Please check www.cab.de for the latest update.

Terms and conditions

Deliveries and performances are effected under the General conditions of sale of cab.

Edition

04/2022

Contents

1 Basics **8**
1.1 Why JScript . . . 8
1.2 How to read this manual . . . 8
1.3 Download the code samples and cabLabel S3 . . . 9
1.4 What else to read . . . 9
1.5 JScript-enabled devices since 1995 . . . 9
1.6 Editing JScript files under Windows . . . 10
1.7 How we support you . . . 11

2 Communication with the printer **12**
2.1 Use USB or SD-card storage media on printer . . . 12
2.2 Appoint a memory type as default memory . . . 12
2.3 Create the appropriate label jobs . . . 12
2.4 Call up label jobs at the printer . . . 14
2.5 Do not use the USB cable . . . 14
2.6 Connection via a TCP/IP network . . . 14
2.7 FTP for file transfer . . . 14
2.8 FTP for immediate printing without saving . . . 15
2.9 Communication in Production . . . 16
2.10 Direct communication on port 9100 . . . 16
2.11 Accessing a file server with WebDAV . . . 17
2.12 Industry 4.0 with OPC UA . . . 18

3 Basic structure of JScript **19**
3.1 JScript as direct communication with the printer . . . 19
3.2 Lower case letters . . . 19
3.3 Capital letters . . . 19
3.4 Special functions . . . 20
3.5 Comments . . . 21
3.6 Make JScript Code Readable . . . 21
3.7 Escape commands . . . 22
3.8 abc Programming . . . 22

4 The typical structure of a label **24**
4.1 A simple minimalistic label . . . 24
4.2 Useful template for a blank label . . . 25
4.2.1 Units of measurement . . . 25
4.2.2 Job start . . . 26
4.2.3 Size of label . . . 26
4.2.4 Speed and heat . . . 27
4.2.5 Using options . . . 28
4.3 Static objects . . . 28
4.3.1 Texts . . . 28
4.3.2 Using your own fonts . . . 30
4.3.3 Textboxes . . . 35
4.3.4 Hyphenation in Text Boxes . . . 36
4.3.5 Barcodes . . . 37
4.3.6 Images . . . 39

4.3.7 Images as embedded ASCII data . . . 41
4.3.8 Graphical elements (circles, lines and rectangles) . . . 43
4.4 Design the label dynamically . . . 45
4.4.1 Referencing content . . . 45
4.4.2 Reusing content . . . 46
4.4.3 Calling templates from storage . . . 47
4.4.4 Fill in new content . . . 48
4.5 Summary or J SHOW BIG A . . . 50

5 Special functions **51**
5.1 Syntax of the special functions . . . 51
5.1.1 Include in square brackets . . . 51
5.1.2 Pass function parameters . . . 51
5.1.3 Use field names as special function . . . 52
5.1.4 No nestings . . . 52
5.2 Hiding elements . . . 53
5.2.1 Conditional visibility . . . 53
5.3 Ask the operator . . . 55
5.4 Dynamic date and time functions . . . 56
5.5 Adding a time offset . . . 56
5.6 Using counter . . . 56
5.7 Remember the counter value . . . 57
5.8 Calculations and comparisons . . . 57
5.9 Format strings or numbers . . . 59
5.10 Avoid rounding errors . . . 60
5.11 Inserting UNICODE characters . . . 61
5.12 Single buffer mode . . . 62

6 Using the different memories **64**
6.1 The useful user memory . . . 64
6.2 The “I have finished!” info memory . . . 64
6.3 Using a file to store a value . . . 64
6.3.1 Using a TMP file . . . 65
6.3.2 Filling a LOG file . . . 65

7 The most common Escape commands **67**
7.1 Interrupt the printing process (Esc *p1* and Esc *p0*) . . . 67
7.2 When nothing works at all (Esc *t* and Esc *!* Esc *!*) . . . 67
7.3 Query status (Esc *s* and Esc *z*) . . . 68
7.4 Reading from the information memory (Esc *i*) . . . 68
7.5 Start signal (Esc *g*) . . . 69
7.6 Trigger the I/O interface (Esc *xin*) . . . 69
7.7 Reading the I/O interface (Esc *xout*) . . . 70
7.8 Outlook: Printer control in the industry 4.0 age with OPC UA . . . 70

8 Access to databases **72**
8.1 Connecting to the cab Windows service . . . 72
8.2 Stand alone sulution: using a local SQLite file . . . 72
8.3 Get a value from a database . . . 73
8.4 Splitting the database response . . . 73

8.5 Write back to a database . . . 75

9 More than just printing **76**
9.1 Dispensing labels . . . 76
9.2 Tear-off mode . . . 76
9.3 Automatic retraction . . . 77
9.4 Ribbonsaver . . . 77
9.5 Cutting Labels . . . 77
9.6 Using an Applicator . . . 78
9.6.1 Set parameters for an applicator . . . 79
9.6.2 Print and apply or apply and print? . . . 79

10 Diagnostic options **80**
10.1 Monitor Mode . . . 80
10.2 Log incoming data into a file . . . 80
10.3 Preview a label without printing . . . 81

11 Barcodes in global trade **82**
11.1 Data integraty for barcodes . . . 82
11.1.1 Error detection by a check digit shown on the GTIN . . . 82
11.1.2 Error correction in QR and Datamatrix codes with the Reed-Solomon method . . . 84
11.2 GS1 Data Structure . . . 85
11.2.1 Floating numbers as a content of an AI . . . 86
11.3 Barcode types for GS1 data . . . 90

12 Best practice examples **91**
12.1 Single digit month, shift identification and daily counter reset . . . 91
12.2 Print vouchers . . . 93
12.3 Printing data from a CSV file . . . 97

13 Advanced BASIC Compiler **100**
13.1 This is not a BASIC manual . . . 100
13.2 Either JScript or abc . . . 100
13.3 Comments in abc . . . 100
13.4 The PRINT command . . . 101
13.5 Conditional tasks and jumps . . . 101
13.5.1 IF–THEN–ELSIF–ELSE–ENDIF . . . 101
13.5.2 GOTO and GOSUB . . . 102
13.6 Loops . . . 103
13.6.1 FOR–NEXT . . . 103
13.6.2 DO–LOOP . . . 106
13.6.3 WHILE–WEND . . . 106
13.6.4 REPEAT–UNTIL . . . 106
13.7 Subroutines (functions) . . . 107
13.7.1 Using a pointer to an array as an argument of a subroutine . . . 111
13.8 String operations . . . 111
13.8.1 *num* = SPLIT(*string$*, *return_array$()*, *dividing_character$*) . . . 112
13.8.2 *num* = TOKEN(*string$*, *array$()*, *separator$*) . . . 112
13.8.3 Handling sub strings with LEFT$, MID$ and RIGHT$ . . . 112
13.8.4 *pos* = INSTR(*string$*, *search_pattern$*) . . . 114
13.8.5 Sweeping blank spaces at the beginning and end of a string . . . 114

13.8.6 More commands to manipulate strings . . . 114
13.9 Read and write on the interfaces . . . 115
13.10 Creating a parser . . . 116
13.11 Writing onto the graphical display . . . 117
13.11.1 Example abc program: global climate warming up . . . 118
13.11.2 Rectangles and lines on the display . . . 120
13.11.3 Example: a transparent onscreen logo . . . 122
13.11.4 Interact with the user by catching touches . . . 124
13.12 A permanent abc loop . . . 125
13.13 The interaction between JScript and abc . . . 125
13.13.1 Just a dream . . . 126
13.13.2 The correct way to interact (JGET$ and JPUT) . . . 126
13.14 Error handling . . . 127

14 Appendix **129**
14.1 A typical JScript label in detail . . . 129
14.2 Short view on special functions . . . 135
14.3 Error codes for the Esc s command . . . 138

15 Solutions to some exercises **139**

List of Tables

4.1 Format text boxes using the **W** command . . . 36
7.1 Trigger I/O signals Esc *xin* . . . 70
9.1 The cut command *C* with its options . . . 78
9.2 Set parameters for an applicator . . . 79
11.1 Error Correction Levels of a QR Code . . . 85
11.2 AI starting with 0, 1 or 2 of the GS1 data structure . . . 86
13.1 Logical operators in abc . . . 101
14.1 **S** = Label Size . . . 130
14.2 **H** = Setting heat level and printing speed . . . 130
14.3 **O** = setting options . . . 131
14.4 **T** = Text . . . 132
14.5 **W** = Textbox . . . 132
14.6 **B** = Barcode . . . 133
14.7 **I** = (auto loaded) Images . . . 133
14.8 **G**. . . **;L:** = Lines . . . 134
14.9 **G**. . . **;R:** = Rectangles . . . 134
14.10 **G**. . . **;C:** = Circles . . . 134
14.11 Date functions . . . 135
14.12 Time functions . . . 136
14.13 Math functions . . . 136
14.14 User query in standalone operation [?:. . .] . . . 137
14.15 Error codes for the Esc *s* command . . . 138

1 Basics

You are just start reading an introduction to the JScript printer language. This document is not a complete textbook, but should people who already have experience in the use of programming languages have a **fast overview of the most important basics and commands of the JScript language**. At the end of the document, you will also see “abc”, the Advanced BASIC Compiler. It is available in nearly all cab printers[1] and allows a high degree of automation without that a separate PC would be required.

1.1 Why JScript

JScript offers very extensive possibilities for direct programming the cab printer.

JScript is system-independent. No matter if your JScript file is stored on an Apple Macintosh, a Linux computer, a Windows PC, an AS400 or a PLC, the cab printers understand all common formats.[2]

JScript allows access to databases. And this both online and also offline. For online access to databases, the printer needs the Software *cab Database Connector*, which is available as a service on a Windows system. This can, but does not have to, be the same PC or server on which the database is running. The actual access is via OLEDB/ODBC. cab Database Connector works as an intermediary between databases and printers. Several printers can be connected simultaneously to one installation of the cab Database Connector. Offline the printer is able to use a SQLite database.[3]

JScript allows extensive calculations By embedding BASIC code, they can also be extended almost indefinitely. This leads to a flexibility that is otherwise rarely found in label printers.

You can secure processes. Entries can be checked by the printer. Data can be transferred directly, signed and encrypted from systems like SAP, so that faulty operator inputs (e.g. – typing errors) are avoided. If labels are printed, this information can securely be fed back to your system in order to guarantee a complete documentation of created labels. All this directly through the printer.

1.2 How to read this manual

This guide is a quick start into JScript programming. **Read everything that helps you and skip parts you do not need.** This is not a comprehensive textbook. Much will only shown shortened.

As manufacturer we offer you an extensive training program for our printers. This also includes a training course for direct programming of our label printer.

This guide summarizes the most important points of the one-day JScript programming training session and can be used without attending the training for self-study.

The dates of the JScript workshops can be found on the Internet:
`https://www.cab.de/en/info/trainings/overview/#jscript`

We are aware that it will not be possible for everyone to work on an individual training, be it in our training center in Karlsruhe or by one of our trainers on site. Therefore, this manual tries to to bring JScript closer to many readers even without much previous experience. Also the specialists find answers to more complex topics, e.g.

[1]Only the two smallest printers Mach 1 and Mach 2 do not have an integrated Advanced BASIC compiler. They also do not process JScript and must be operated via a Windows PC (GDI devices).

[2]JScript works line-oriented and accepts both a line end <CR><LF> (Microsoft Windows), <LF> (Linux) or <CR> (Apple Mac OS). However, the format should not change within the same file.

[3]Prerequisite is a printer with X4 mainboard and a firmware version 5.25 or higher. New firmware can be installed on cab printers without registration or extra costs via the cab website.

"How do I print a hazard symbol contextually?", which is related to the topic "conditional visibility" (it is explained on page 53).

So don't despair if you can't follow every detail in this manual. As the number of pages increases, so does the complexity and the necessary previous experience. If you get stuck in the middle of the manual, don't worry about it. If you make it to the summary on page 50 you have already understood the most important basics of JScript.

1.3 Download the code samples and cabLabel S3

The listings printed here in this book are mostly attached to the PDF file. In the PDF Reader click on the indicator of the embedded attachments (download icon with file name) and download the files from the PDF instead of laboriously typing them. The file name of embedded attachments is displayed below the listings in light blue font. This procedure to access the listing by click (or double-click) to save (or open) was tested with the PDF Readers Adobe Reader DC and Sumatra PDF. Should you find out that the code examples cannot be downloaded directly from this PDF, your reader's JavaScript Action functions may be prohibited.

Many of our customers also use the software cabLabel S3, which comes with our printers and is also available free of charge as *cabLabel S3 Lite* from our website. It can be used to create the JScript label files.

`https://www.cab.de/cablabel`

This book therefore occasionally contains screenshots and hints on how cabLabel S3 can generate certain JScript elements. A typical use case is often the creation of a layout template in cabLabel S3 Lite and the later filling with data from the production process using the replace command (see section 4.4.4 starting with page 48).

1.4 What else to read

The *programming manual for SQUIX, MACH 4S, EOS2/EOS5, HERMES Q, PX Q* should always be within easy reach for additional reading. The programming manual offers a complete reference of all JScript and abc commands on over 650 pages.

You can find the manual on the cab website:
`https://www.cab.de/en/programming`

Both together, this document and the comprehensive reference "Programming Manual", can help you getting started with JScript.

1.5 JScript-enabled devices since 1995

This manual is intended for the newer devices with a mainboard of X4 Series (or newer). You can recognize these devices by their colored touch displays, which are not built into the older models. If the control is still via keys below the display, or is the touch display monochrome, you have a device of an older mainboard generation in front of you.

Older devices do not support some of the commands described here in the manual. Refer to *Programming Manual for A+ and X-Series, MACH4, PX, Hermes+, Hermes C, EOS1/EOS4*, which commands are also used by the older models.

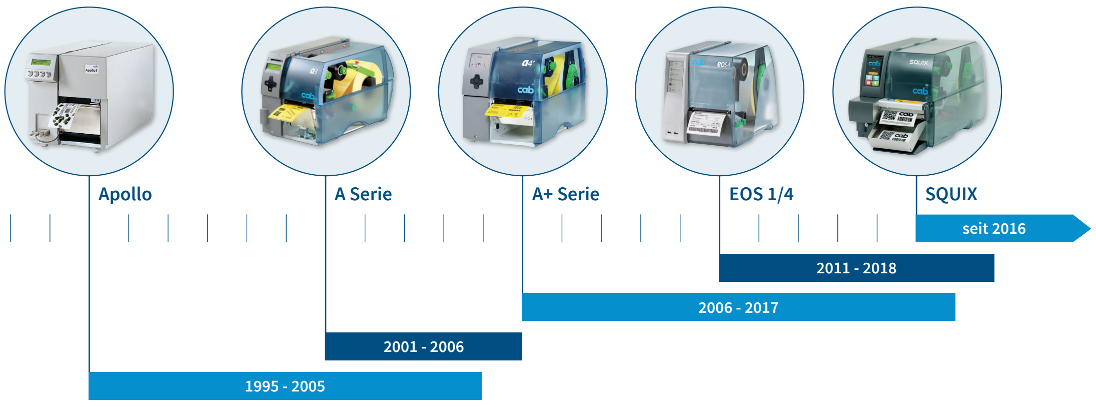


Figure 1.1: cab has been building printers that can be controlled with JScript since 1995

Since January 2020, no devices have been produced that do not have at least a mainboard of the X4 generation.

Figure 1.1 shows an example of the different series each with a 4 inch tabletop printer.

1995 was the birth of the first cab thermal transfer printer and at the same time the first use of JScript as printer language. The Apollo was produced until the middle of 2005. Apollo printers had a two-line text display and four keys for operation.

From 2001 to 2006, the A series supplemented and replaced the Apollo. A series printers, like the Apollo, only had a text display. The four keys of the Apollo were replaced by a navigator pad, the metal housing by a more modern housing made of industrial plastic.

In 2006, the A^+ printers were the first having one identical mainboard for all printers of the series, the ”X2 board”. On its single core CPU runs a monolithic firmware. All printers of the A^+ series have a Navigator Pad (cross-shaped control button) and a graphic display. Not before 2017 the production of the A^+ table printers ended.

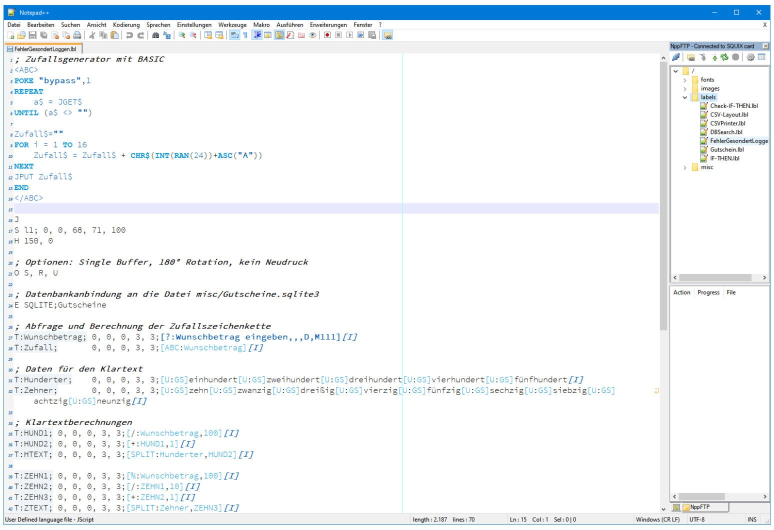
With the EOS we entered new territory in two aspects. The two printers EOS1 and EOS4 were the first ones made completely out of plastic by cab as a new kind of compact industrial printers. On the other hand, the X3 Board first offers a Linux substructure. This opened up a wide field for future enhancements to the printer functions.

As the flagship of the next generation, the SQUIX was first sold in 2016. It’s touch display is colored and the user interface has been completely redesigned. All models of this series, including the HERMES Q and PX Q series have the identical X4 mainboard with a powerful processor and Linux as a substructure.

1.6 Editing JScript files under Windows

To edit the JScript files you can use any text editor, but we recommend the free editor Notepad++. This editor allows you to work with a printer with the help of the extension *NppFTP* – directly to the printer via a local network (see section 2.6 from page 14). You can find extensions in Notepad++ via the menu item “Extensions” and install then via “Plugin administration . . . ”.

https://notepad-plus-plus.org/



```
; Zufallsgenerator mit BASIC
<ABC>
POKE "bypass",1
REPEAT
    a$ = JGET$
UNTIL (a$ <> "")

Zufall$=""
FOR i = 1 TO 16
    Zufall$ = Zufall$ + CHR$(INT(RAN(24))+ASC("A"))
NEXT
JPUT Zufall$
END
</ABC>

J
S l1; 0, 0, 68, 71, 100
H 150, 0

; Optionen: Single Buffer, 180° Rotation, kein Neudruck
O S, R, U

; Datenbankanbindung an die Datei misc/Gutscheine.sqlite3
E SQLITE;Gutscheine

; Abfrage und Berechnung der Zufallszeichenkette
T:Wunschbetrag; 0, 0, 0, 3, 3;[?:Wunschbetrag eingeben,,,D,M111][I]
T:Zufall;       0, 0, 0, 3, 3;[ABC:Wunschbetrag][I]

; Daten für den Klartext
T:Hunderter;    0, 0, 0, 3, 3;[U:GS]einhundert[U:GS]zweihundert[U:GS]dreihundert[U:GS]vierhundert[U:GS]fünfhundert[I]
T:Zehner;       0, 0, 0, 3, 3;[U:GS]zehn[U:GS]zwanzig[U:GS]dreißig[U:GS]vierzig[U:GS]fünfzig[U:GS]sechzig[U:GS]siebzig[U:GS]
    achtzig[U:GS]neunzig[I]

; Klartextberechnungen
T:HUND1; 0, 0, 0, 3, 3;[/:Wunschbetrag,100][I]
T:HUND2; 0, 0, 0, 3, 3;[+:HUND1,1][I]
T:HTEXT; 0, 0, 0, 3, 3;[SPLIT:Hunderter,HUND2][I]

T:ZEHN1; 0, 0, 0, 3, 3;[%:Wunschbetrag,100][I]
T:ZEHN2; 0, 0, 0, 3, 3;[/:ZEHN1,10][I]
T:ZEHN3; 0, 0, 0, 3, 3;[+:ZEHN2,1][I]
T:ZTEXT; 0, 0, 0, 3, 3;[SPLIT:Zehner,ZEHN3][I]
```

Notepad++
Datei Bearbeiten Suchen Ansicht Kodierung Sprachen Einstellungen Werkzeuge Makro Ausführen Erweiterungen Fenster ?
FehlerGesondertLoggen.lbl
NppFTP - Connected to SQUIX card
fonts
images
labels
Check-IF-THEN.lbl
CSV-Layout.lbl
CSVPrinter.lbl
DBSearch.lbl
FehlerGesondertLogge
Gutschein.lbl
IF-THEN.lbl
misc
Action Progress File
NppFTP
User Defined language file - JScript
length : 2.187 lines : 70
Ln : 15 Col : 1 Sel : 0 | 0
Windows (CR LF) UTF-8 INS

Figure 1.2: Notepad++ allows you to colorize the source code individually. Click on "Languages", then on "Own Language define . . . " and "import" the one linked here as a XML file. JScript-Highlighting.xml

1.7 How we support you

The open dialogue with our customers and partners is an important part of our success. You get from cab not only innovative marking technology for the industrial use, we also give you free access to our instructions. Nevertheless, self-study is not always the most efficient way to a labeling solution.

Talk to us. Besides the possibility of an individual solution for training you, we also offer you the possibility to work on your project together to plan and realize them. The direct contact to our developers, support staff and trainers will help you to complete projects faster and more reliably – and thus will end up reducing costs.

2 Communication with the printer

Before we can turn our attention to the printer language, we need to clarify briefly how you can reproduce the mentioned examples on a cab printer.

Here are just a few of the possibilities. There are still significantly more possibilities, but this manual should focus on the pure concentrate printer language. For the interfaces and their optimal use please refer to the interface manual of the respective printer model.

To actively work through this manual you only need one way to get your JScript code into the printer. This can be done with a USB stick, witch can be useful if you still have little or no experience with computer technology and networks. More convenient is the access via the network, but in some cases considerable prior knowledge is required. Decide for yourself how to connect to your printer.

2.1 Use USB or SD-card storage media on printer

The easiest way is to use a storage medium that you can read and write on the printer as well as on your computer, e.g. a USB memory stick. First format the storage medium on the computer, so you still need to add the four main directories "fonts", "images", Create "labels" and "misc" directly in the root directory.

If this seems too complicated, then plug in the storage medium simply freshly formatted into the cab printer. If the printer is switched on it creates the necessary directories independently. You can use the storage medium then after this, just plug it back into your computer and copy the JScript files on it.

2.2 Appoint a memory type as default memory

If the contents of e.g. a USB memory stick are not immediately displayed on the printer you can switch to the USB memory each time you select the menu item "load label" (see figure 2.1 on page 13), or you can set USB as default. Then the printer always searches for files in the USB memory first. The default memory must be selected correctly if you give the printer the command to load of a label, but do not specify the storage location itself (see section 4.4.3 for details on page 47).

2.3 Create the appropriate label jobs

Label jobs are pure text files[4] with a file extension ".lbl". It is important that the extension is actually ".lbl". Are you using a Windows operating system and is the display of the file extension deactivated (for Windows the default state), a created text file would probably have the double extension ".lbl.txt", but which the printer cannot do anything with.

However, before you run to buy an Apple Macintosh or a Linux PC, you can simply activate the display of the file extension under Windows in the folder display options. How this works exactly tells you your favorite search engine on the internet.

The text file, which ends not on ".txt" but correctly on ".lbl" must be placed in the correct subdirectory of the storage medium. It is the "labels" directory.

The two images 2.1 and 2.2 show this behavior of the printer for the two copied files "richtig.lbl" and "falsch.lbl.txt".

[4] "Pure text files" are files that consist of pure 8-bit text, no files from WYSIWYG editors like Microsoft Word or OpenOffice Writer. For professionals: use UTF-8 encoded files without BOM.

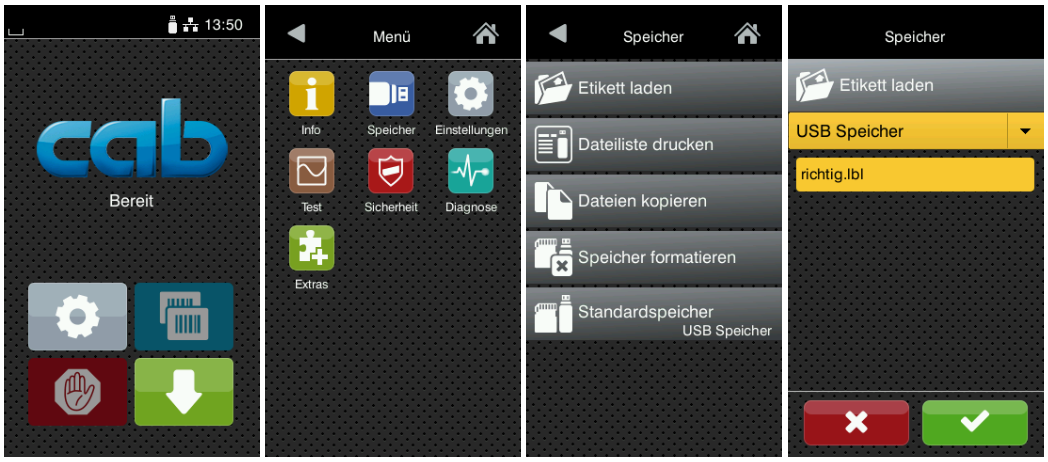
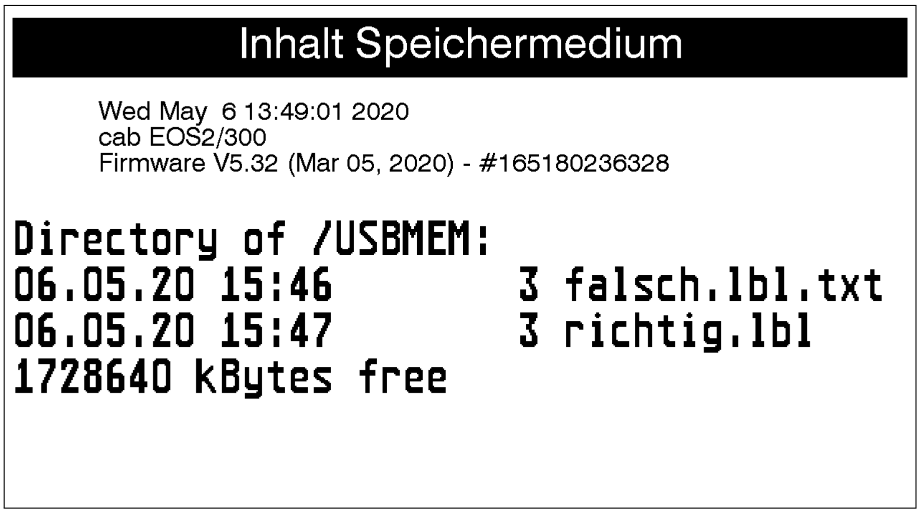


Figure 2.1: Via the gray gear wheel symbol and the blue memory stick button you get to the menu "Storage". Here you can see the data stored in the subfolder "labels" to load copied JScript label files.



Inhalt Speichermedium

```
Wed May  6 13:49:01 2020
cab EOS2/300
Firmware V5.32 (Mar 05, 2020) - #165180236328

Directory of /USBMEM:
06.05.20 15:46          3 falsch.lbl.txt
06.05.20 15:47          3 richtig.lbl
1728640 kBytes free
```

Figure 2.2: There are two files on the disk. The file *richtig.lbl* can also be selected for printing. The file *falsch.lbl.txt* however has the ending *.txt*, which is for a cab printer not accepted as label file (see figure 2.1). Even if the file saved in your Windows operating system is displayed as "falsch.lbl" – do not let Microsoft make fun of you! The printout has been created by the "Print file list" menu item in the "Storage" menu.

2.4 Call up label jobs at the printer

On the printer, press the gray button with the gear wheel (Settings menu). You will get into a menu, where you can now select the blue symbol with the memory stick. In the following menu you can load a label with "Load label", it will display a list of all files located in the directory "labels" and end on ".lbl".

If the file you copied there does not appear, please check the correct file name spelling including the extension. Using the menu item "Print file list" you can also print a listing of all files, output directly to the printer. However, make sure that the material is sufficiently wide in the printer.

2.5 Do not use the USB cable

To transfer pure programming data via USB cable requires additional software. It is therefore not advisable to start with the printer in this way to be addressed. USB connection can be used when you only want to use labeling software or printing with the Windows driver.

2.6 Connection via a TCP/IP network

You are familiar with TCP/IP networks, even if you have never read the term. TCP/IP is the protocol on which the Internet is based. It describes the connection technology between individual devices on the network. This can be your PC, but also hubs, switches, routers, cell towers and of course cab printers are "network devices".

Each network device has a unique name (experts call it MAC address) and an IP address. The IP address allows to address the device in the network.

When you press the gray gear icon on the printer's main screen you get to the menu "Settings" (see Figure 2.1, second illustration from left). There you will find a yellow info icon in the upper left corner that shows you information about the printer. Among many other information you can see the IP address of the printer. Let's assume that the printer shows "192.168.10.1" as the IP address. IP addresses usually consist of four numbers, each with a dot in between. Remember these numbers and enter them in your favorite Internet browser, if this device (means the PC on which the Browser is running) is connected to the printer via the network.

If your browser is connected to the printer, you will see, after entering the IP address of the printer, an overview page of the printer. On the left side of this page you will find a live view of the printer's touch screen, which you can access with the mouse to operate the printer. Just like an "Internet remote control" already installed in the printer. If you are looking for a username and a password, both are "admin", if you have not changed this in the printer settings (which you should do for security reasons).

2.7 FTP for file transfer

A visual remote control is handy – but you can't use it to learn JScript. Instead of the browser, open a file manager (or an FTP software) and enter the following line:

```
ftp://ftpcard:card@192.168.10.1
```

Your file manager should now show you the printer's storage. It will always display the storage that you have selected as default in the "Storage" menu. So leave your USB memory stick in the printer and select "USB" as standard. You can then send your label file via FTP (you can find an explanation of the protocol online in the Wikipedia). This way you can write files to your USB device without constantly having to unplug it.

You can then access the files with the extension ".lbl", which you can find in the folder "labels", via the menu item "Load label" and execute the JScript code.

Advice: Connect a USB keyboard to the second USB plug on the printer. By pressing F2 you will directly get into the *Load label* dialog.

Exercise 2.1 Your first handmade label

Use a plain text editor (e.g. Notepad, but not Microsoft Word or OpenOffice Writer) and enter the following: the lowercase letter f and then the ↵ key. You should add a line with the letters *f* and below in another empty line the Cursor flashes. Now write this file as "formfeed.lbl" via an FTP connection to the "labels" directory. Then select the file in the submenu item "Load label" of the printer.
If you have done everything correctly, the printer will print (move) exactly one empty label. The line with the lower case letter *f* causes a Label feed, or "form feed". If you click the green symbol with the down arrow in the main menu the printer should do exactly the same move.

2.8 FTP for immediate printing without saving

Instead of copying the files via FTP to the printer and then printing them via calling up the printer's menu you can also use a shortcut. The printer allows besides the FTP user name "ftpcard" also the user name "ftpprint" with the password "print".

```
ftp://ftpprint:print@192.168.10.1
```

The big difference is that the printer will print all files you send via the special upload user "ftpprint" to the printer, but executes them immediately without saving them to the storage. So the files are just interpreted as JScript code, and are not placed into the "labels" folder. With this special procedure the file name including the file extension is not relevant. As an exception, you can even use the file "falsch.lbl.txt" to bring it to execution.

However, the FTP protocol was not developed for this purpose. An FTP software therefore assumes that an uploaded file will then be stored in the FTP directory of the printer and exists there. However, you will never be able to download as a user "ftpprint" from the printer, and also folders can neither be called nor created.

Depending on the software you use for the file upload, it can lead to confusion sometimes. If in doubt, use a tolerant software or do not use this shortcut to print the JScript files directly.

Exercise 2.2 Shortcut using the FTP user name *ftpprint*

Take the file created in the previous exercise and load it again via FTP to the printer. This time, however, choose the user name "ftpprint" (and the password "print").

The printer should now immediately perform a feed.

2.9 Communication in Production

In the following three more common methods we will show how to install a cab printer into a production environment. We start with the the most obvious way to send JScript code directly over the network and to receive answers from the printer.

After that we will show how to connect the printer to a file server, which then makes the label jobs available. If there are several printers within production, this is an excellent opportunity for keeping all printers always on the same level by simply setting the file server which is then accessed by all printers.

Finally OPC UA shall be mentioned briefly. We will not discuss this protocol in detail, but here lies a large part of the future, which is to be “Industry 4.0”.

2.10 Direct communication on port 9100

The cab printer allows direct TCP/IP communication on port 9100. If this doesn’t mean anything to you, just skip this section. For readers with a lot of experience in network technology this function of the printer should not be concealed. Otherwise come with the two previously mentioned methods (use a storage medium or transfer via FTP) should be enough to work through this manual.

If you are very familiar with network technology you can open a Telnet session on port 9100 of the printer and enter the JScript lines there directly. Or you copy from the instructions into the session window. Under Windows, a Telnet client is available, but hidden in the Windows feature configuration. You must therefore either activate the Telnet function of Microsoft Windows or use a foreign software like PuTTY.

If you are using Windows 10 and want to feel like a hacker please right-click on the Windows icon in the lower left corner and select “Windows PowerShell (Administrator)”. In the Windows PowerShell window enter the following command:

```
Dism /online /Enable-Feature /FeatureName:TelnetClient /All
```

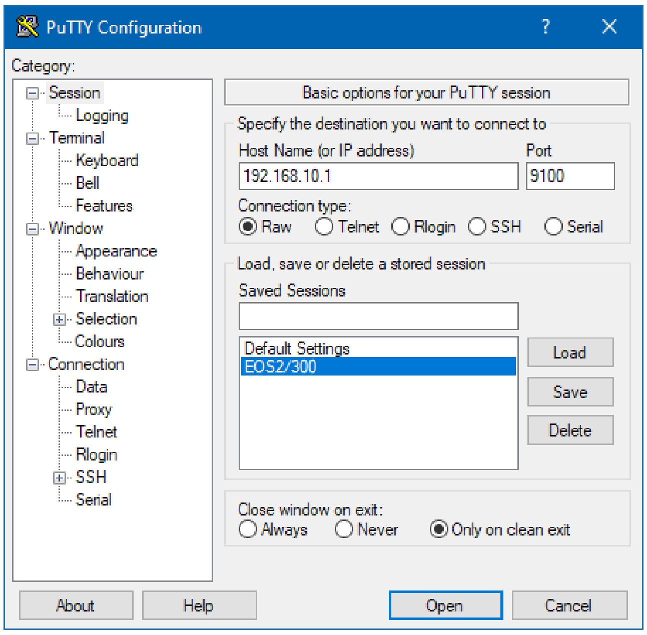
After that you can close the PowerShell window (use the command “exit”) and open it again without administrator rights. Let us guess your printer has the IP address 192.168.10.1, enter the following command to start a telnet session:

```
telnet 192.168.10.1 9100
```

From now on, your keyboard entries are no longer displayed, but directly transfered via the network to port 9100 of your printer. The answers of the printer appear on your window. Try it out and type into the Telnet session the keys Esc and s. The printer should answer with a string that looks something like this: “Y-000000N”.

Exercise 2.3 Only for experts

Connect to port 9100 of your printer by using the Telnet function of your operating system or use a software like PuTTY. In the terminal session with the printer, press the keys Esc and s. Does the printer answer you? Now enter successively the key sequence Esc, !, Esc and !. If you have done everything correctly, the printer should switch off and restart (reboot). Your session will then be terminated inevitably.



Specify the destination you want to connect to
Host Name (or IP address): 192.168.10.1
Port: 9100
Connection type: [✓ Raw] Telnet Rlogin SSH Serial
Load, save or delete a stored session
Saved Sessions: Default Settings, EOS2/300 (selected)
Close window on exit: Always Never [✓ Only on clean exit]

Figure 2.3: The Telnet client PuTTY offers a comfortable way to direct connection to the printer. Select RAW as connection type, the IP address of your printer and port 9100 (not 23). You can use the Save settings also for later use.

2.11 Accessing a file server with WebDAV

The WebDAV protocol has been developed for a long time, the *Web-based Distributed Authoring and Versioning* (see Wikipedia).

This is not intended to explain how to set up a WebDAV server, but is the job of the administrators, who are responsible for the provision of data in Corporate network with security. It shall only be briefly explained what advantage it brings to connect the printers via the WebDAV function.

If your printer has no function to use an external WebDAV server as a data source your firmware is probably not up to the latest version. **First with firmware 5.33 WebDAV was added to the cab printers.** To update the firmware of your printer to the latest, there are two possibilities. The easiest way is to use the web interface by entering the printer's IP address in the browser, and there use the "help" menu. In the help menu you can find a routine to check if the printer's firmware is still up-to-date. If the firmware of the printer does not correspond to the latest version, the routine offers this directly to correct. Just follow the instructions.

The option described above requires an access to the cab servers in the Internet. If this is not possible from the printer, you can download the current firmware also always from the website www.cab.de to your PC and then upload it with the appropriate firmware updater software to the printer (if the printer is not connected to a network, you can load the Update also via a USB memory stick or via a USB cable from a Windows computer).

To use WebDAV as a storage location, you must activate the option in the *Storage* menu (see figure 2.1 on page 13). There is the menu item "default memory", where you must first select WebDAV. If WebDAV is is selected, further menu items appear, with which you can change the address of the WebDAV server. And you can enter the user name and password. If you search for a file and click on "Load label", the label will be loaded by the printer from the WebDAV server and the files will be downloaded from the subdirectory "labels" on the WebDAV store for printing. The WebDAV storage then works as if the files were stored on the internal memory (IFFS) or a storage medium on the printer (USB memory stick or SD memory card). Multiple printers can share one WebDAV server at the same time. A big benefit if you have to distribute label files to a lot of printers, being sure that all will access the same actual version of your files.

2.12 Industry 4.0 with OPC UA

It is still more a marketing slogan, “we are in the industry 4.0 age”, but what does this mean? The very latest industrial revolution connects machines intelligently. In this “Internet of Things”, standards are needed with which the individual devices/things can communicate with each other. A mature standard is the Open Platform Communications standard. Based on this older standard the next generation is OPC Unified Architecture, **OPC UA**.

Since this is a standardized approach to communication, it can be assumed that machine control systems (PLC) of various manufacturers should all have the ability to use OPC UA to communicate with a cab printer.

The big advantage is certainly the standardization, so that the Integration of a device like a cab printer is faster than if you have to create the interface each time new from scratch and implement it in the controller.

But an OPC UA connection has other advantages as well. As a modern standard, OPC UA is secure and scalable. This means that you can communicate between individual devices on a small scale as well as globally with a connection to a cloud - all defined in the standard. All this with the latest encryption technology, OPC UA is secured and controlled in access.

A small overview of the possibilities provided by cab printers (OPC UA speaks here of an information model) can be found on our website:

`https://www.cab.de/en/news/news/opcua/`

The OPC UA standard allows observing a variable, e.g. the remaining amount of thermal transfer foil on the printer, in a simple way from a machine control system. The advantage of monitoring with OPC UA is that the values are subscribed from the printer, correctly the printer’s build-in OPC UA Server. If a value changes the printer automatically informs the subscriber. Man does not have to poll the status of the printer in a continuous loop, but is informed by the printer about status changes. A much lower network load.

In addition, many things can be done via “methods”. This way it is not even necessary to learn JScript anymore. A label layout can be created using the cabLabel software and a method provided for loading and printing a label format file do the rest. Fields on the label are provided automatically as variables in the OPC UA information model and can be specifically changed before a label is printed.

Thus complex tasks such as the use of layout templates and then filling it with data from the production process without JScript knowledge is possible. But please do not put this manual aside to concentrate from now on only on OPC UA. With JScript all cab printers still have a very powerful direct access, mastering it will pay off in any case.

3 Basic structure of JScript

3.1 JScript as direct communication with the printer

With JScript you can communicate directly with a cab printer. The language works line-oriented as a stream of bytes (8-bit data stream). Each line must be terminated with a *<CR>* or *<LF>* or both.[5] Only when a line is finished it is evaluated.

If the JScript interpreter has received a complete line, the first letter of this line is examined more closely. The letters are case sensitive. The letter determines which command should be executed in the printer. Lower and upper case letters have different meanings. The rest of the line are the parameters for the command, e.g. position and size of a graphic or the name of a TrueType font to be loaded.

3.2 Lower case letters

Lower case letters usually refer to global settings that affect the printer per se and are not limited to the currently active label.

However, if the printer is switched off, it loses the data specified by lower case properties or settings. At the next start, the printer settings are taken from the printer's internal default values that can be changed via the menu options on the printer.

Example: The letter "f" triggers a label feed (formfeed). This corresponds to pressing the green symbol on the printer.

```
f
```

As another example, we can use the lowercase letter "t" to create a test print.

```
t
```

Please note that the lines must be terminated with a *<CR>* or *<LF>* or both together. For the sake of easy readability, we have omitted the representation here. You can create a *<CR>* by pressing the ↵ key (line break).

3.3 Capital letters

Capital letters always refer to the label currently in memory. The capital letter J starts a new label (job start). This command must be the first capital letter command, otherwise an error message will be displayed on the printer.

With the "A" command (quantity) the internal creation is stopped and a corresponding number of labels is printed. After that, a label to be newly designed must first be started again with "J" before other capital letters are accepted.

A typical label could look something like this.

[5] *<CR>* and *<LF>* stands for the two ASCII characters *Carriage Return* and *Line Feed*. In most text editors these are normally not directly visible. Special ASCII editors (like Notepad++, Microsoft's Visual Studio Code or Github's Atom Editor) can display them. Make sure to always have an end of line at the end of a JScript file. Otherwise, the last line may not be processed and an error message may appear at the printer if JScript data is sent again (which then mixes with the unfinished line to form a code that is incomprehensible to the printer).

```
J
S 0, 0, 68, 71, 100
T 20, 20, 0, 3, 10;Yummy Joghurt
B 20, 30, 0, EAN13, SC2;456712349876
A 1
```

In the example above you see the commands “J”, “S”, “T”, “B” and “A” in the first position of the lines.

You define a new job	(J = job start),
define the label size	(S = page size),
write a text	(T = text)
and a barcode on the label	(B = barcode).
One copy is printed	(A = number).

Behind the command character, as shown in the example, there are more specifications. These are the parameters of the commands, which will be discussed later in this manual.

3.4 Special functions

Special functions are used within the content of a text. They are put in square brackets.

Example:

```
J
S 0, 0, 68, 71, 100
T 20, 20, 0, 3, 5;Bottled on:
T 20, 30, 0, 5, 8;[DATE]
A 1
```

In the example two texts are placed on the label. The second text contains the special command *[DATE]*. Here the printer inserts the current date.

Within the square brackets is either the function name, which is case-sensitive, or a reference, i.e. a field name. A function name can, but need not, be followed by further parameters, which are separated from the function name by a colon and from each other by commas.

A special function with arguments can look like this:

```
J
S 0, 0, 68, 71, 100
T 20, 20, 0, 3, 5;Best before:
T 20, 30, 0, 5, 8;[MONTH02:0,6]/[YYYY:0,6]
A 1
```

In line 4, arguments are attached to the special command. Their exact meaning is described in a later chapter. In this example, they cause a date offset by 0 days and 6 months.

The chapter 5 starting on page 51 deals with the topic in detail and also gives examples of how you can use special functions to insert, for example, user queries, counters or a calculated date.

3.5 Comments

A comment line starts with a semicolon as command character (first character of the line). After that any content can follow, which is evaluated as comment and is not executed. See listing 3.1 line 2, 4, 6 and 8 as an example.

```
1 J
2 ; define page size
3 S 0, 0, 68, 71, 100
4 ; fixed text
5 T 20, 20, 0, 3, 5;best before:
6 ; the printer calculate a date with 6 month shift
7 T 20, 30, 0, 5, 8;[MONTH02:0,6]/[YYYY:0,6]
8 ; print one label
9 A 1
```

Listing 3.1: Comment lines start with a semicolon Comments.lbl

3.6 Make JScript Code Readable (Source Code Formatting Rules)

JScript is designed to be read and understood quickly by humans. For this reason, JScript has been designed from the beginning to be as tolerant as possible to spaces and blank lines. Use this freedom to make your self-created code readable so that you (or another person) can maintain the code more easily later.

There are no design rules for this. However, JScript is based on the following rules, which you must follow:

1. Each line contains exactly one command as first character of the line.
2. Each line must be terminated with an end-of-line character (*<CR>*, *<LF>* or both).
3. Blank lines are ignored, any number of blank lines can be inserted.
4. After the command (first character of the line) and after a comma, any number of blank lines are allowed. This does not apply to special commands.
5. After the semicolon that follows the parameters, the content begins immediately. Blanks become part of the content here!
6. Comments are allowed as comment lines, starting with a semicolon.

In order to make the code especially readable, the following conventions are followed in this manual wherever possible, in addition to the JScript specifications.

- Comments are always *before* the content to which they refer.
- If a comment refers to several lines (JScript commands), an empty line is inserted after the last command line.
- Each comma separating parameter is followed by at least one space.
- Within a command, all lines are filled up with additional spaces after a comma so that the comma is below the comma in lines with the same command.[6]
- The parameters for the position of the reference point and the angle are aligned with each other across commands (applies to the *T*, *B*, *I* and *G* command lines).
- A command character is followed either directly by the field name (with colon and semicolon) or at least one space.

[6]In your editor, use a non-proportional font in order to arrange the codes according to the conventions.

3.7 Escape commands

JScript was designed so that a simple ASCII text editor is sufficient to program most labels. However, labels do not always consist only of text that can be created with simple editors, but in some cases graphics/logos must be transferred to the printer. In order to be able to communicate quickly with the printer without confusing it, an additional ASCII special character has been allowed, character number 27, also known as *escape character*, abbreviated as Esc.

Tip: If you use the editor Notepad++[7] you can insert an Esc character simply by using the key combination Shift ⇑ and Esc.

It is possible at any time during communication with the printer to send the escape commands, which are immediately processed without waiting for the end of the line.

Escape commands start with the ASCII character Esc, decimal *27*, hexadecimal $1B_{16}$. This is followed by at least one more character, e.g. "t". Together they form the escape command, which is cut out of the rest of the JScript code.

Escape commands do not have to be embedded in JScript, they can also be sent separately to the printer via another interface. For example, JScript data can be transmitted via TCP/IP, while a PLC can query the status via RS232 using Escape commands.

Sending the character string Esc *p1* to one of the printer's interfaces, for example, corresponds to pressing the Pause key on the printer. With Esc *p0* you would end the pause again so that the printer continues its work. A list of the most important ESC commands can be found starting on page 67, all commands are included in the programming manual which you can download from the cab web page.

3.8 abc Programming

If the possibilities of JScript are not sufficient, you can also use the integrated BASIC. The cab printer has an Advanced BASIC compiler ("abc") built into the firmware, which compiles the embedded code and executes it independently from the JScript interpreter.

This manual here is mainly about JScript. Advanced BASIC is covered in a separate chapter starting on page 100. However, you should already have some experience with the BASIC language. For normal printer use, you can work without BASIC, because even calculations can be performed directly in JScript, as will be explained in detail in later chapters.

[7] The editor Notepad++ can be used free of charge under Microsoft Windows. It is a powerful open source editor and can be downloaded from the website `www.notepad-plus-plus.org`.

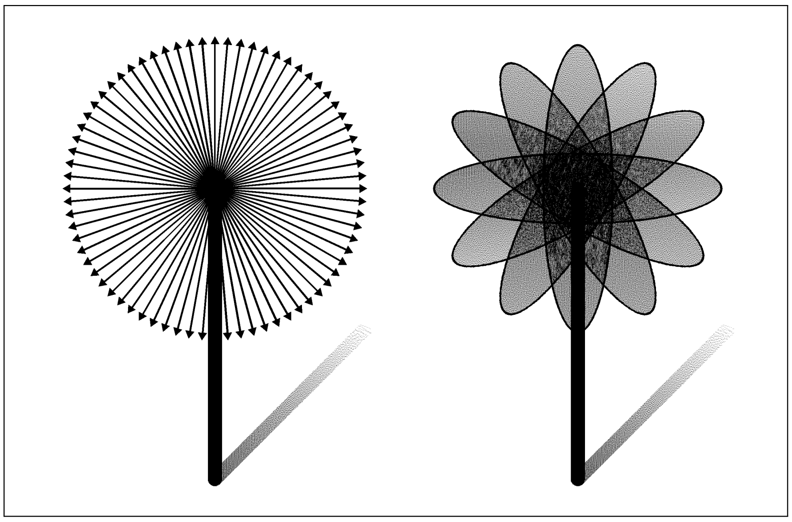


Figure 3.1: The result of the Advanced BASIC program from listing 3.2

```
<ABC>
PRINT "m m"
PRINT "J Flower Power"
PRINT "S l1; 0, 0, 68, 71, 100"
PRINT "H 25"
PRINT "O R"
PRINT "G 25, 24, 270;L: 40, 2, r, r"
PRINT "G 25, 64,  45;L: 30, 2, r, r [S:60,0,45]"
FOR angle = 0 TO 355 STEP 5
    PRINT "G 25, 24, ", angle, ";L: 20, 0.25, r, a"
NEXT
PRINT "G 75, 24, 270;L: 40, 2, r, r"
PRINT "G 75, 64,  45;L: 30, 2, r, r [S:60,0,45]"
FOR angle = 0 TO 330 STEP 30
    PRINT "G 75, 24, ", angle, ";C: 20, 5, 5 [S:25,50,100]"
    PRINT "G 75, 24, ", angle, ";C: 20, 5, 0.25"
NEXT
PRINT "A 1"
</ABC>
```

Listing 3.2: Example for the use of BASIC FlowerPower.lbl

4 The typical structure of a label

4.1 A simple minimalistic label

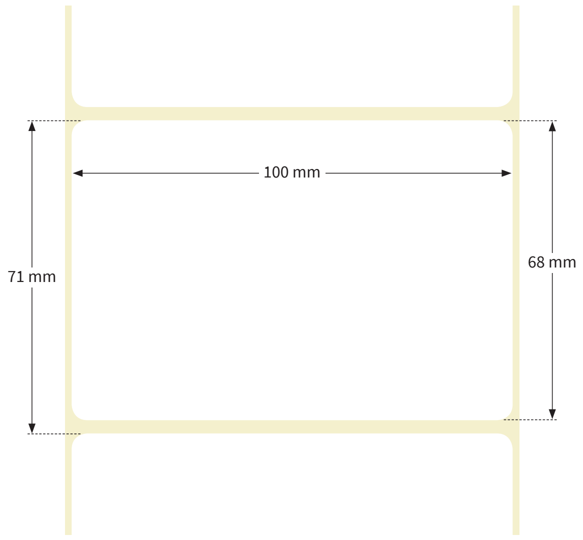


Figure 4.1: Dimensions of the label from listing 4.1

The minimal label consists of three lines (don't forget <*CR*> at the end of the line!) and looks like this:

```
J
S 0, 0, 68, 71, 100
A 1
```

Listing 4.1: Formfeed Deluxe

The label is started with "J" (Job Start). Then "S" is used to determine the label size (Size).

If the label is shifted from 0 mm to the right and 0 mm down, a 68 mm long label is printed. Together with the 3 mm gap, the distance from the beginning of the label to the beginning of the next label is 71 mm. The label is 100 mm wide (see illustration 4.1).

Thus the command has the following structure:
S *displacement to the right, displacement downwards, height, height + gap, width*

A detailed description of the most important commands can be found in the Appendix. All commands can be found in the comprehensive programming guide for JScript.

The printer can use millimeters or inches as the unit of measurement, depending on the selected country setting in the printer settings. Except for the USA country setting, millimeters is assumed.

One copy is printed, "A" stands for number and simultaneously terminates label creation in the printer memory.

Before the "A" command, the contents (texts, barcodes, graphics, etc.) would have to be inserted, because a blank label alone is not very useful.

Exercise 4.1 Three at one stroke S. 139

Measure the size of the labels in your printer. Then create a JScript file “ffdeluxe.lbl” similar to listing 4.1. However, three labels should be created. What happens during execution? Are there three labels each?

Exercise 4.2 Error in label size S. 139

Increase the values for the label height and the distance from the start of the label to the next label start by 20 mm each. What happens now? Can you explain the behavior?

4.2 Useful template for a blank label

However, it is better to use the following extended template for a label in JScript (listing 4.2).

```
m m
J
S l1; 0, 0, 68, 71, 100
H 150, 0
O R
A 1
```

Listing 4.2: A minimal label with the most important information Minimal.lbl

Let’s take a closer look at the individual lines.

4.2.1 Units of measurement

You should use the line

```
m m
```

for the use of metric units (SI base unit Millimeter) or

```
m i
```

for the Anglo-American unit use inches (Imperial or Historical units) to not be surprised by the printer’s country settings. If “USA” is selected as country, the unit of measurement is set to inches by default. All other country settings set millimeters as the unit of measurement. The “m” command permanently overwrites this preset until the next restart of the printer.

Other printer languages often work with the dots of the print bar for position or length specifications. However, JScript always uses the selected measurement system (millimeters or inches), so your data does not depend on the resolution of the print bar. A big advantage of the cab printer language.

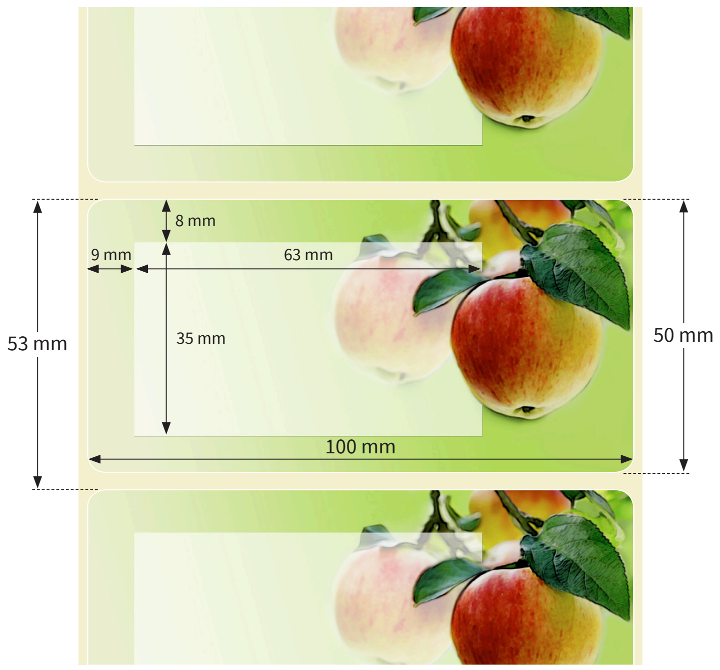


Figure 4.2: A decorative label that should only be printed in the light background area. By cleverly defining the page size in the JScript code, the printer will report an error if, for example, a barcode gets past from the permitted area.

4.2.2 Job start

With “J” the description of the label is started. When a “J” command is received, the printer forgets everything it has stored for the current label and starts again from scratch.

4.2.3 Size of label

The “S” line additionally defines the use of different label sensor types (here the see-through detection by a prefixed parameter “l1”, the lower case letter l followed by the number 1). This additional information must be separated with a semicolon from the following parameters describing the dimensions. The other values are separated from each other by a comma. **If decimal numbers are to be used, a dot must always be used as decimal separator, even if the printer is set to Germany, for example.**

There are a total of four options for recognizing the beginning of the label. “l0” selects the reflex light barrier from below, “l1” selects the transmitted light barrier, “l2” selects the reflex light barrier from above (if available), and “e” does not detect the gap, but uses only the length specification (continuous material).

The remaining parameters are as already mentioned:
S *start detection;shift right, shift down, height, height + gap, width*

Exercise 4.3 Set a print area within the label S. 139

Sometimes it is necessary to print on labels that are already printed and sometimes additionally coated. A mostly rectangular area is let out in order to print data into it during production with the label printer. The figure 4.2 shows such a case. What is the page definition in JScript code for this label?

4.2.4 Speed and heat

Speed and heat are set with the "H" command. The speed is replaced by the value from the JScript command, but the value for the heat is added to the default value in the printer. Maximum up to the limits -20 and +20. Examine the following line

```
H 150, 2
```

it sets the speed to 150 mm/s. However, the heat is only set to +2 if the default value in the printer is 0. If 2 is also set in the printer, the total heat value is 4.

Print speed and heat must be adjusted for the respective label material. To protect the print head, it is recommended to select the lowest possible heat.

The quality of the printout depends strongly on the correct combination of label and ribbon. **If foil (ribbon) and label don't fit together, a print result should not be squeezed out of the printer by excessive heat, it would drastically reduce the lifetime of the print head. Overheated printheads are not covered under warranty!**

The print speed mainly influences the print quality (precisely print image). The lower the speed, the cleaner the print. If the application allows slow printing (there is enough time), you should use this and not set the printing speed too high.

Too little heat will cause the print to fade, while too much heat will cause the print to smear and increase wear on the print head.

Exercise 4.4 A perfect print result

Type the list 4.2 (or just click on the download symbol), adjust the page dimensions to your label size and save the label as "Minimal.lbl". Now insert the following before the last line ("A 1"):

```
T 10, 10, 0, 5, 8;Quality is no accident
```

Change the heat values to -10, -5, 0, 5 and 10 respectively and print one copy. The effect of the temperature setting should be clearly visible.

Exercise 4.5 Fast or pretty?

From the previous example, select the best setting for the heat and now select the speeds 50 mm/s, 100 mm/s, 150 mm/s and 200 mm/s respectively. Can you see a difference in print quality?

4.2.5 Using options

With the command “O” you can activate one or more options. The most frequently used option is probably “R”, which outputs the label rotated by 180° (R = rotated). Other useful options are “S” for single buffer operation or “T” for tear-off mode. “U” can be used to prevent pause reprinting so that no label can be printed twice with the printer’s repeat key (Unique).

See the programming guide for details on the options. There you will find a complete list and more detailed explanations of all options.

If several options are desired, they are separated by a comma.

```
O R, T
```

In this case, the label would be output rotated by 180° in tear-off mode. This combination is common in tear-off mode, since the label is not upside down when the operator removes it.

Exercise 4.6 Headstand of the labels *S. 140*

Take the JScript file from the previous two exercises and select the optimum temperature and speed for a clean print image and protection of the print head. An optimal value can also be between the previously required steps, but you can only enter integers for the heat and preset values for the speed (use the selection in the printer menu as a guide). Now omit the line with the content “O R”. What is noticeable when printing?

4.3 Static objects

In the following, the visible elements of a label layout will be described with their most important parameters. This enumeration is far from being complete, but is not intended to replace the programming instructions.

4.3.1 Texts

Texts are integrated into a label by the command “T”. Following are the parameters:

1. X-position (from left edge to right).
2. Y-position (from top edge to bottom).
3. Rotation angle (integer counterclockwise).
4. Font number (e.g. 3 = Swiss 721, 5 = Swiss 721 bold, 596 = Monotype 821).
5. Font size, as distance between p-line and k-line (length *d* in figure 4.3).
6. Options, separated by commas (s = oblique, i = italic, n = inverse, b = bold, q*n*=compressed[8]).
7. The actual content separated by semicolon.

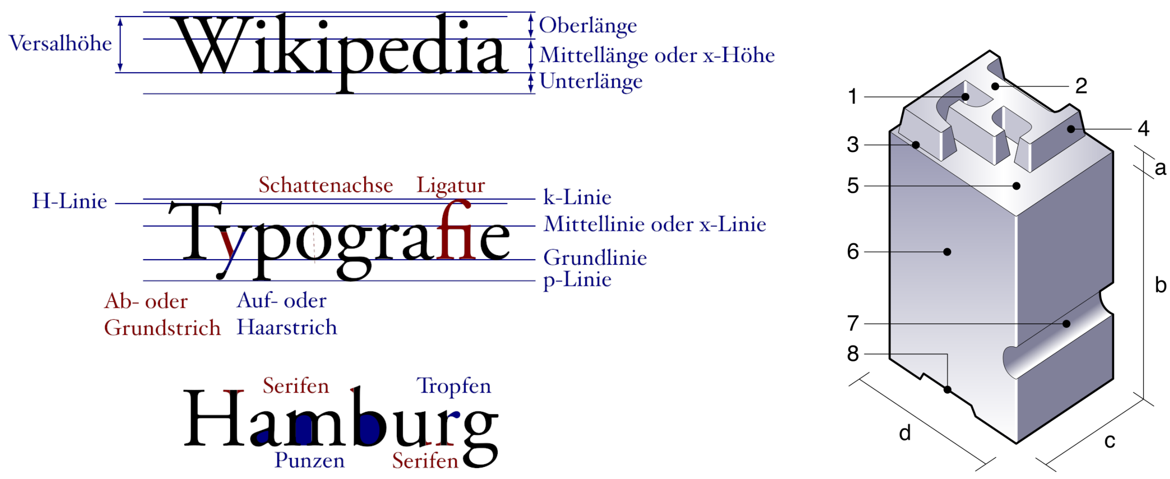


Figure 4.3: Font lines[9] and scheme of a print type. The dimension *d* corresponds to the font height specified in the fifth parameter of the *T* command and is also called cone height.

For texts, the position always refers to the baseline of the font. All other objects (e.g. barcodes or graphics) are described in position by their upper left corner.

You can see in the listing 4.3 that both the specification *pt12* and the specification *4.5* were used for the font size. The line *m m* turns *4.5* into a font size of 4.5 mm. Placing *pt* in front of a font size specification allows JScript to process this size in “DTP point”, a unit of measurement often used in typesetting. For example, common text editors use the unit point for entering font sizes. See Wikipedia for further explanations.

For decimal fractions, always use the period as a separator between the integer part and the decimal part (US notation).

The comma is used in JScript only as a separator for parameters and must not be used for numbers!

```
m m
J
S l1; 0, 0, 68, 71, 100
H 150, -2
T 45, 25,  45, 3,   pt12;Wind
T 45, 25, 315, 3,   pt12;mühlen
T 45, 25, 225, 3,   pt12;mahlen
T 45, 25, 135, 3,   pt12;Mehl
T 44, 42,  90, 3, 4.5, b;für
T 48, 42,  90, 3, 4.5, b;Brote
A 1
```

Listing 4.3: Example for angle specifications and font styles Windmill.lbl

[8]For compression, an integer from 10 to 1000 must be used for the *n*. Text that is not compressed has the option “q100”. If the text is to be compressed to 80 %, you must use “q80” as an option.

[9]By Brian Ammon, CC BY-SA 3.0, `https://commons.wikimedia.org/w/index.php?curid=2349540`

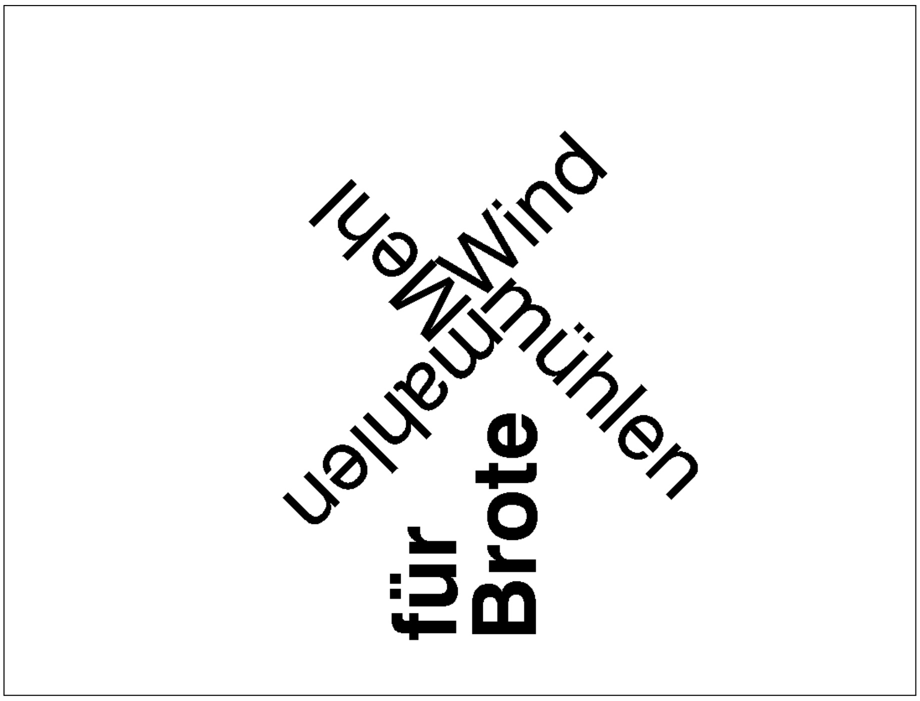
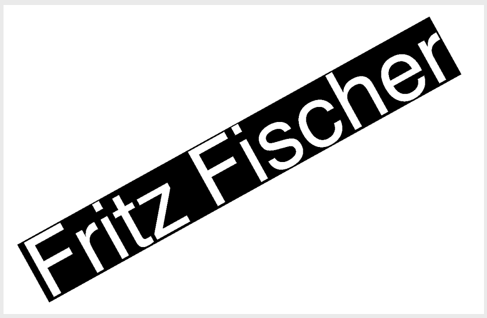


Figure 4.4: Result from listing 4.3

Exercise 4.7 Pretty weird S. 140

Design and print a label on which your name is printed across from the bottom left to the top right, large enough to fill the label to the maximum. Use font no. 3 with the font option “inverse”. Remember that angle specifications are only allowed as whole numbers.



4.3.2 Using your own fonts

In addition to the fonts supplied, it is also possible to use your own fonts. For this purpose, these fonts must be of the type *TrueType*.

Load the font to the printer in the subdirectory *fonts* before you use the font in a label. The printer will look for the font in the default memory. If the default memory is set to *USB*, a font stored in *IFFS*[10] is not

[10]IFFS is the name of the printer’s internal non-volatile memory (Internal Flash File System). This is case-sensitive and file names longer than 8 characters can be used.

found.

To use a TrueType font we first have to load the font into RAM (volatile memory) and assign a font number. This is done with the “M l” and the “F” command.

In listing 4.4 the free fonts *Linux Libertine*[11] and *Linux Biolinum* are used and first loaded into RAM (lines 4 to 10). Then, in line 18, number 11 is assigned to the font family *Linux Libertine* so that these fonts can be used from line 22 on.

When loading the TrueType files, “M l fnt” is followed by the name of the file without file extension, separated by a semicolon. When assigning fonts, after the “F” command use the font number and, separated by a semicolon, the name of the font family. If you do not know the names of the font families, you can also assign the file name of the normal font to the font number. So the two lines 18 and 19 could also be

```
18 F 11;LinLibertine_Rah
19 F 13;LinBiolinum_Rah
```

The special function *[J:r45]* aligns the text right-aligned on an imaginary length of 45 mm. With *[J:c100]* the heading is centered on the label.

It is interesting that we loaded six font styles into memory (lines 4 to 10), but only assigned two font numbers. Nevertheless, we can still use all six fonts. The cab printers search for a bold or italic font that belongs to the same font family with the options *b* and *i*. Just assign a font number to the default (Regular) and use the options for bold or italics.

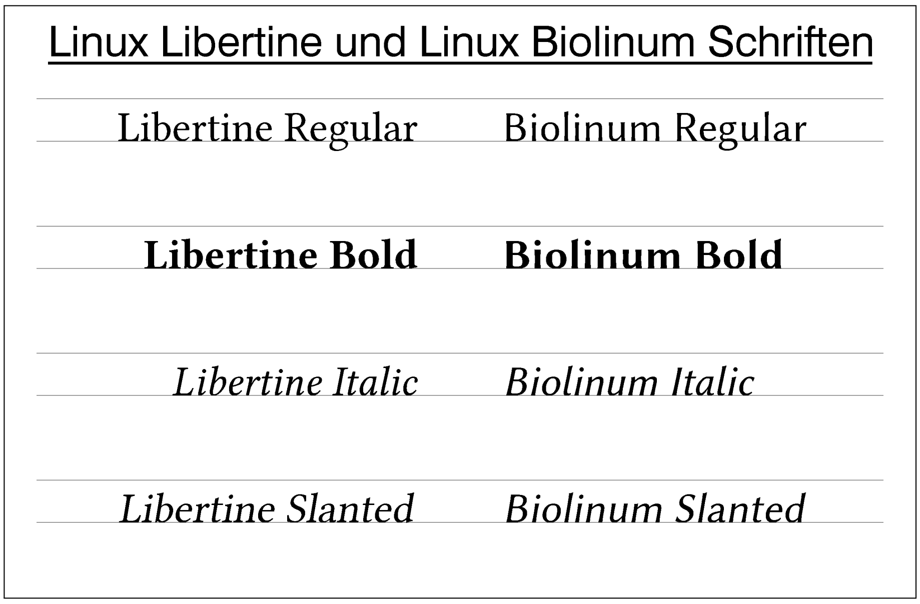


Figure 4.5: Using TrueType fonts in listing 4.4

Notice that all fonts have the same font size of 5 mm? However, some fonts appear to be of different height and strength. The auxiliary lines in the example have a distance of 5 mm to each other. You can immediately see that the font size does not correspond to the visible height, because the p-line is not visible due to the missing descenders (for an explanation see figure 4.3).

One more hint. Some TrueType fonts are of enormous size and usually occupy unnecessarily much memory. The font *Arial UNICODE MS*, for example, has the special characters for many languages of the world on board, but

[11]You can download both fonts here: `https://sourceforge.net/projects/linuxlibertine/`

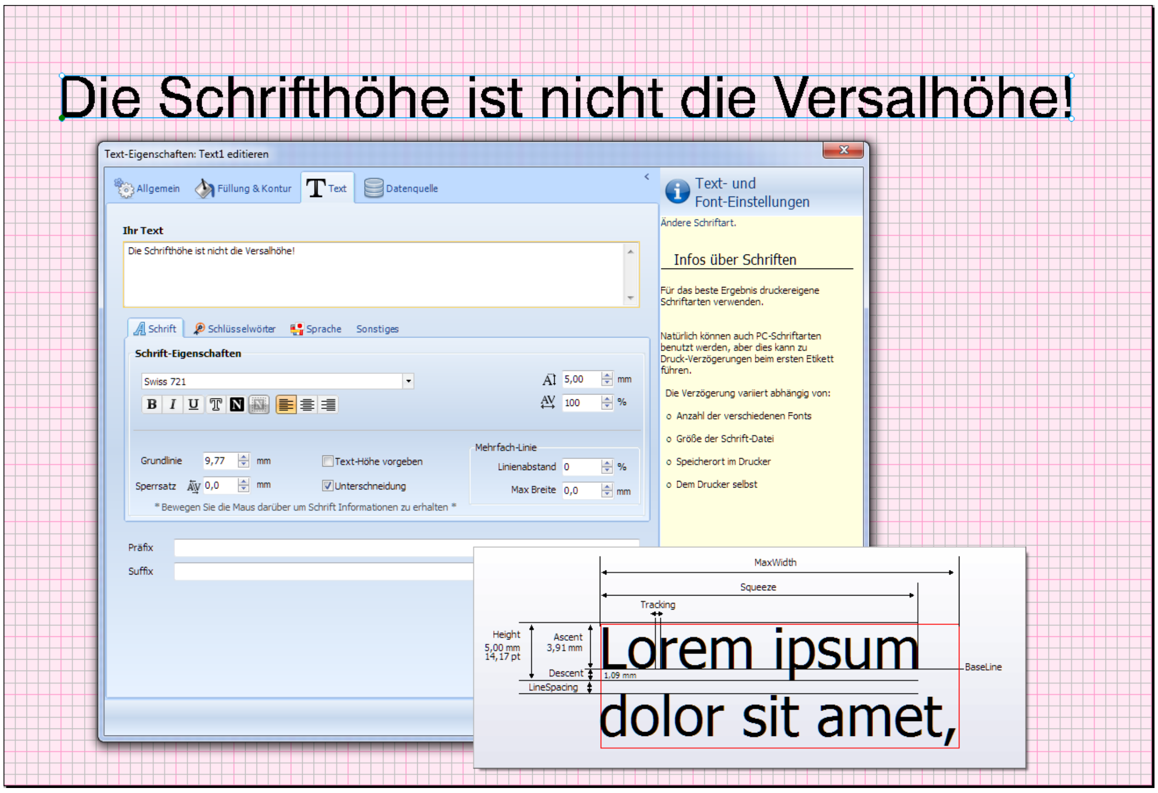
these are usually not used in the actual label. It is better to use a font that is reduced to the specific application in order to save memory and reduce the preparation time in the printer.

Exercise 4.8 **Handwritten name badges** S. 141

You have 30 mm high and 100 mm wide labels on which the name of participants of an event should be printed. To make the name badges look casually friendly, a beautiful handwriting should be used, which the printer does not have yet. Download Kimberly Geswein's "Indie Flower" font and transfer it to your printer.
`https://fonts.google.com/specimen/Indie+Flower`
Design a label with the name of a participant and his organization. For an example, see Figure 4.7.

```
; set measurement to Millimeter
m m

; load fonts (using filenames)
M l fnt;LinLibertine_Rah
M l fnt;LinLibertine_RBah
M l fnt;LinLibertine_RIah
M l fnt;LinBiolinum_Rah
M l fnt;LinBiolinum_RBah
M l fnt;LinBiolinum_RIah

; job start, label size, speed and heat
J
S l1; 0, 0, 68, 71, 100
H 150, -2

; assign font number (using the font family name)
F 11;Linux Libertine
F 13;Linux Biolinum

; insert font examples
T  0, 15, 0, 11, 5;Libertine Regular[J:r45]
T  0, 30, 0, 11, 5, b;Libertine Bold[J:r45]
T  0, 45, 0, 11, 5, i;Libertine Italic[J:r45]
T  0, 60, 0, 11, 5, s;Libertine Slanted[J:r45]

T 55, 15, 0, 13, 5;Biolinum Regular
T 55, 30, 0, 13, 5, b;Biolinum Bold
T 55, 45, 0, 13, 5, i;Biolinum Italic
T 55, 60, 0, 13, 5, s;Biolinum Slanted

; draw guide lines
G  0, 10, 0;L: 100, 0.05
G  0, 15, 0;L: 100, 0.05
G  0, 25, 0;L: 100, 0.05
G  0, 30, 0;L: 100, 0.05
G  0, 40, 0;L: 100, 0.05
G  0, 45, 0;L: 100, 0.05
G  0, 55, 0;L: 100, 0.05
G  0, 60, 0;L: 100, 0.05

; headline
T  0, 5, 0, 3, 5, u;Linux Libertine und Linux Biolinum Schriften[J:c100]

; don't print, just create an internal preview bitmap
A [PREVIEW]
```

Listing 4.4: Using some custom fonts TrueType.lbl



Die Schrifthöhe ist nicht die Versalhöhe!

Text-Eigenschaften: Text1 editieren

Allgemein | Füllung & Kontur | Text | Datenquelle

Ihr Text

Die Schrifthöhe ist nicht die Versalhöhe!

Schrift | Schlüsselwörter | Sprache | Sonstiges

Schrift-Eigenschaften

Swiss 721

5,00 mm

100 %

Grundlinie 9,77 mm

Sperrsatz 0,0 mm

Text-Höhe vorgeben [ ]

Unterschneidung [✓]

Mehrfach-Linie

Linienabstand 0 %

Max Breite 0,0 mm

* Bewegen Sie die Maus darüber um Schrift Informationen zu erhalten *

Präfix

Suffix

Text- und Font-Einstellungen

Ändere Schriftart.

Infos über Schriften

Für das beste Ergebnis druckereigene Schriftarten verwenden.

Natürlich können auch PC-Schriftarten benutzt werden, aber dies kann zu Druck-Verzögerungen beim ersten Etikett führen.

Die Verzögerung variiert abhängig von:

- Anzahl der verschiedenen Fonts
- Größe der Schrift-Datei
- Speicherort im Drucker
- Dem Drucker selbst

Figure 4.6: In *cabLabel S3* a height of 5.0 mm was chosen for the font. This specification is later sent to the printer in JScript code. The cap height is only 3.91 mm and the descender is 1.09 mm. *cabLabel S3 Pro* shows you these values in a help bubble when you move the mouse pointer over the gray text below the values.

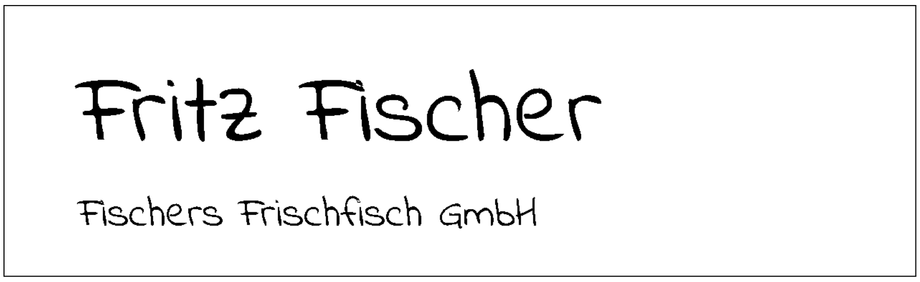


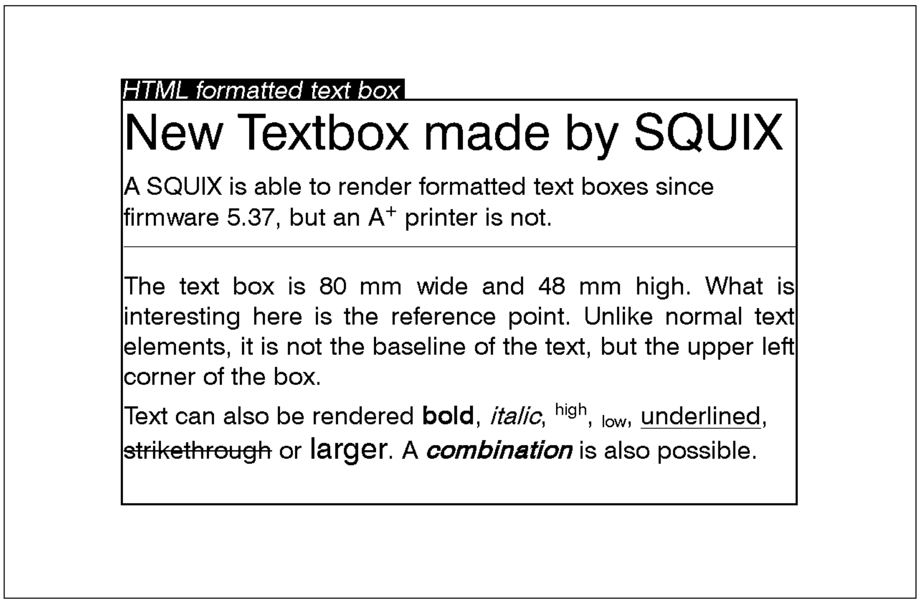
Figure 4.7: A handwriting gives an empathic impression on a name badge. A free font from the designer Kimberly Geswein was used here. IndieFlower-Regular.ttf

4.3.3 Textboxes

Text boxes are frameless rectangles that are filled with text, available since firmware 5.37 for all printers using an X4 mainboard. They are defined with the *W* command to specify their width and height, and the reference point for placement on the label is the upper left corner. Unlike a simple text element, the text in a text box is wrapped in multiple lines. Thus, the text always remains within the box. HTML markup can be used for further styling. Because this all sounds quite complicated here is an example (listing 4.5).

```
m m
J
S l1;0,0,68,71,100
T 10,10,0,3,3,s,n;HTML formatted text box
W:Textbox;10,10,0,80,48,3,3;<HTML>
<h1>New Textbox made by SQUIX</h1>
A SQUIX is able to render formatted text boxes since firmware 5.37, but an A<sup>+</sup> printer is not.
<hr>
<p align=justify>The text box is 80 mm wide and 48 mm high. What is interesting here is the reference point. Unlike normal text elements, it is not the baseline of the text, but the upper left corner of the box.</p>
Text can also be rendered <b>bold</b>, <i>italic</i>, <sup>high</sup>, <sub>low</sub>, <u>underlined</u>, <s>strikethrough</s> or <big>larger</big>. A <b><i>combination</i></b> is also possible.
</HTML>
G 10,10,0;R:80,48,0.25
A 1
```

Listing 4.5: Think of a text box as a blank page in a web browser that the printer fills to fit within its dimensions. Textbox.lbl



HTML formatted text box

New Textbox made by SQUIX

A SQUIX is able to render formatted text boxes since firmware 5.37, but an A$^+$ printer is not.

The text box is 80 mm wide and 48 mm high. What is interesting here is the reference point. Unlike normal text elements, it is not the baseline of the text, but the upper left corner of the box.

Text can also be rendered **bold**, *italic*, ^{high}, $_{low}$, underlined, ~~strikethrough~~ or larger. A ***combination*** is also possible.

Figure 4.8: The listing 4.5 leads to this result. As can be clearly seen, the line break is not taken over, but must be specified by an HTML command like *<br>* or *<p>. . . </p>*.

The most important tags available in the W command, as the (usually enclosing) HTML language elements are also called, are shown in the table 4.1.

Table 4.1: Format text boxes using the **W** command

Tag	Result
`<b>…</b>`	bold text
`<i>…</i>`	italic text (italic)
`<u>…</u>`	underlined text
`<s>…</s>`	striked out text
`<sup>…</sup>`	superscript text
`<sub>…</sub>`	subscript text
`<big>…</big>`	enlarged text
`<!-- … -->`	comments (will not be printed)
`<br>`	newline
`<p>…</p>`	paragraph
`<p align=left>…</p>`	left aligned paragraph (identical to <p>)
`<p align=right>…</p>`	right aligned paragraph
`<p align=center>…</p>`	centered paragraph
`<p align=justify>…</p>`	full justified paragraph
`<h1>…</h1>`	heading of 1st order
`<h2>…</h2>`	heading of 2nd order
`<h3>…</h3>`	heading of 3rd order
`<h4>…</h4>`	heading of 4th order
`<h5>…</h5>`	heading of 5th order
`<hr>`	draws a horizontal rule

To make the JScript code more readable there is an exception for the W command. The JScript interpreter does not recognize an end of line between the tags *<HTML>* and *</HTML>*. Thus, longer content in the W command can be written across multiple lines if you frame it appropriately in a <HTML> tag pair. In the listing 4.5 this can be seen (lines 5 to 19).

Please note that a textbox is always drawn without a border. In the example (listing 4.5), the frame is drawn separately by line 20 to show the positioning. A single text element is inserted in line 4. It is rendered with the additional parameters *s* and *n* to slant and invert the text. You can see in the figure 4.8 that the text sits exactly on the top edge of the text box. With the *T* command, the `10,10,0,...` refers to the baseline of the text. With a rotation of 0°, the text is set at the coordinate (10/10) on the baseline.

Inside the text box, which also has its anchor point at the coordinate (10/10) and a rotation of 0°, the text is placed in such a way that a minimum distance to the edge remains. By the way, the distance to the left and to the top is the same here. However, since the text used in the example does not reach completely to the k-line (see also Figure 4.3), the distance to the top appears larger.

It is not possible in the *W* command to set the baseline of the first letter as an anchor point. If you want to align text horizontally, you should not mix the *T* and *W* commands if possible.

4.3.4 Hyphenation in Text Boxes

A text box distributes the text without any knowledge of the language used. Thus, the printer cannot hyphenate, but only breaks the text where there are spaces. However, there is a special character in the Unicode character set which is not printed. It carries an information that the text can be hyphenated at exactly this position.

The cab printers use this special character in the W command to insert hyphenation. The special character is the Unicode character No. 173. Four ways allow its use in the W command. You can use the Unicode special command `[U:173]`, its hexadecimal notation `[U:$AD]` or the two other abbreviations, based on HTML, `&shy;` and `&#173;`.

```
J
S 0,0,68,71,100
H 100,0
W 10,65,90,60,5,3,2.3;Neil and Buzz left a flag at the moon in 1969 landing at
W 15,10,0,15,40,3,4;Mare Tran&shy;quil&#173;li[U:173]ta[U:$AD]tis
A 1
```

Listing 4.6: There are several ways to indicate where hyphenation may occur. All of them represent the Unicode character “SOFT HYPHEN”. Silbentrennung.lbl

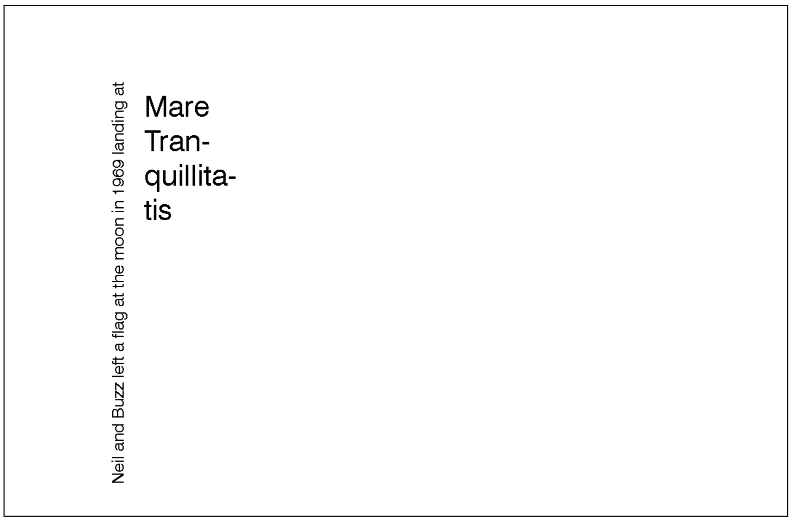


Figure 4.9: Listing 4.6 shows multiple ways to indicate a position for hyphenation.

4.3.5 Barcodes

Barcodes offer a wide range of possible parameters. The topic alone is so complex that more than 100 pages of the programming manual deal with barcodes.

Basically, JScript has the *B* command to create a barcode. It has the parameters:

1. X-position (from left edge to right).
2. Y-position (from top edge to bottom).
3. Rotation angle (only multiples of 90° allowed).
4. The type of barcode, e.g. Code128, EAN-13, Datamatrix or QR Code.
5. Options directly appended to the type with the plus sign (without comma in between).
6. The size whose parameters depend on the type.
7. After a semicolon the content of the barcode.

It is strongly recommended to use the programming guide to read all parameters for the respective barcode type. We will only look at a Datamatrix, a QR Code and a Code128 here to show the creation of three common barcodes as an example.

```
1 m m
2 J
3 S l1; 0, 0, 68, 71, 100
4 H 100, 1
5 B 10, 10, 0, DATAMATRIX+COLS18+ROWS18, 1;Hallo Welt!
6 B 10, 48, 0, CODE128, 8, 0.15;grosse Buchstaben - grosse Wirkung
7 B 10, 58, 0, code128, 8, 0.15;kleine Buchstaben - keine Wirkung
8 B 50,  5, 0, QRCode,  0.75;https://www.cab.de/de/kennzeichnung/etikettendrucker/squix
9 A 1
```

Listing 4.7: Examples of some barcodes Barcodes.lbl

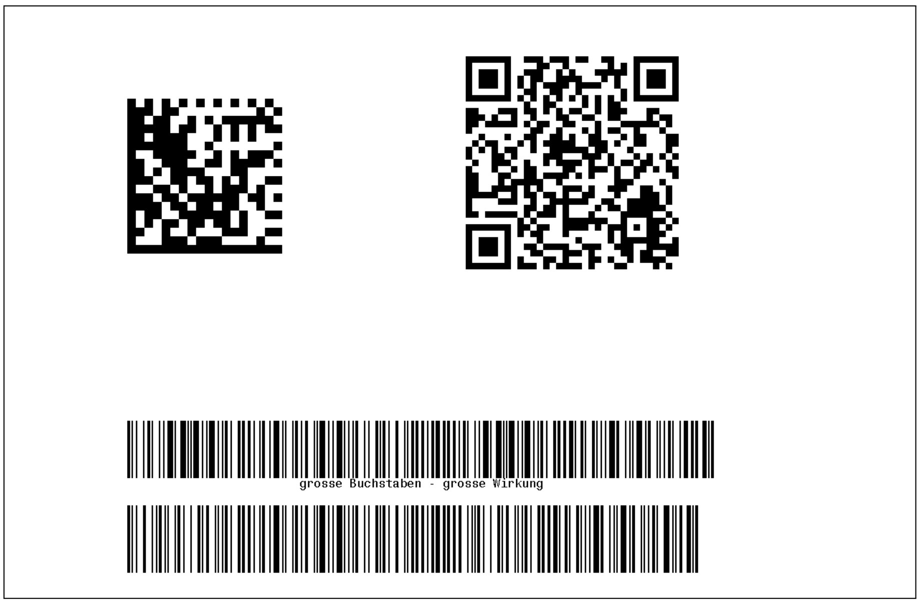


Figure 4.10: Label with several barcodes (created with listing 4.7)

Let’s look at the example in listing 4.7. The first barcode creates a data matrix with a fixed 18x18 matrix. The following two lines each generate a Code128, but with the difference that **capitalized type specifications generate a plain text line in addition to the barcode** (only for 1-D codes). The last line with a *B* command generates a QR Code. As with the Datamatrix, upper/lower case is not relevant here because the printer does not generate a text line for 2-D codes.

For Datamatrix and QR Code, the size specification is the module size, i.e., the edge length of the smallest square (called module) that makes up the codes.

For Code128, two size specifications are required. The first defines the height (in the example 8 mm), the second specification is the width of the smallest bar (here 0.15 mm).

If a code is specified in capital letters, i.e. with a plain text line, the font below the barcode is calculated with the specified height. Although both Code128 commands require a height of 8 mm, the height of the bars in the barcode generated in line 6 is therefore lower than in the barcode from line 7 (from listing 4.7).

Module width and line width are related to the resolution of the print bar, because with a thermal transfer printer no dots can be heated proportionally (there are no half dots).

The printer automatically determines the number of heating points that comes closest to the specified value. If the same label is printed on a 203 dpi and a 300 dpi printer, the widths of the barcodes and the sizes of the 2-D codes may differ. The smaller the barcode, the more clearly the effect is visible.

Exercise 4.9 Creating a WLAN access QR Code

Create a label where a QR code allows automated access to a (fictitious) WLAN. You are welcome to use the label for your own guest network. The important thing is that you create the content of a QR Code according to fixed specifications. These specifications for providing access information to a WLAN are described in this section of the scanner programming library wiki “zxing”:

https://github.com/zxing/zxing/wiki/Barcode-Contents#wi-fi-network-config-android-ios-11

Below is a small example as a suggestion:

```
m m
J
S l1; 0, 0, 68, 71, 100
H 100, 0
O R
T 10, 15, 0, 3, 5;Scan Code to enter the WiFi network!
B 30, 20, 0, qr-code, 1;WIFI:T:WPA;S:HoneyPotter;P:123456;;
A 1
```

Listing 4.8: QR codes allow automated access to WLAN networks. By the way, a cab printer also shows you such a QR code on the printer display when you set up a hotspot. You don’t have to type SSID and password laboriously. WLAN-QR-Code.lbl

Try it out for your own WLAN network. If you do not have one at hand, your printer can provide you a WLAN hotspot. Can you connect your smartphone automatically, just by using the camera app[a]?

[a] On iPhones Apple has built this feature into the camera app. If you are using an Android smartphone this will not work on all models. However, you can download several barcode scanner apps from the Google App Store that offer this feature.

4.3.6 Images

For the images to be integrated into a label, the limitations of thermal transfer printing technology apply. Individual dots are calculated, which are either printed or not printed (black and white), there are no fractions of dots (no gray representation).

Once this is accepted, it becomes clear that color or grayscale image formats such as JPEG are not particularly suitable for printing as an image on a label. Better suited are black and white formats such as those that can be stored as PNG (Portable Network Graphics). The integration is done with the *I* command (capital letter I like Image).

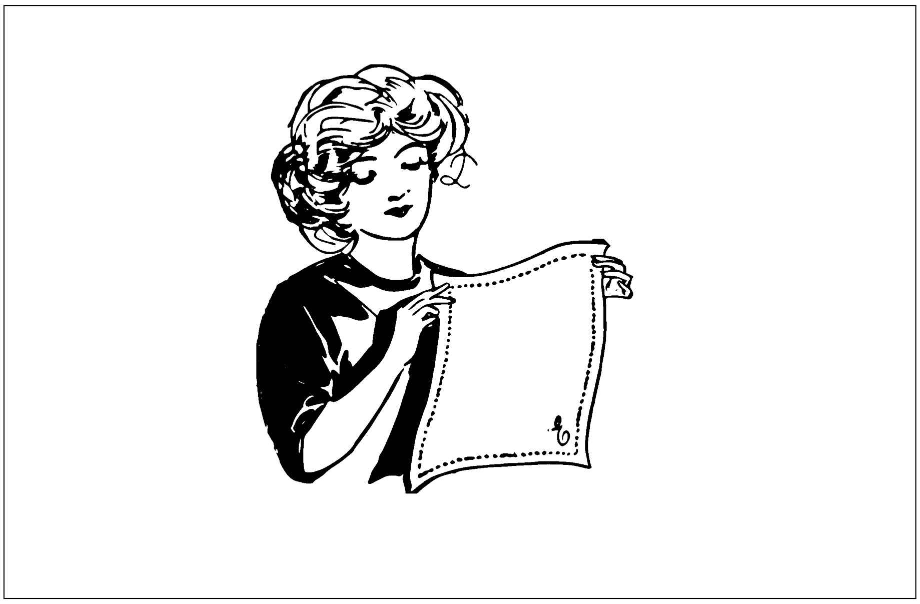


Figure 4.11: Drawing of a woman folding a napkin (line 5 from listing 4.9) Mechthild.png

```
m m
J
S l1; 0, 0, 68, 71, 100
H 100, 0
I 25, 5, 0, 1, 1,a;Mechthild
A 1
```

Listing 4.9: Positioning of an image Bitmapgrafik.lbl

In listing 4.9 the two magnification factors and the parameter *a* are used to use the autoload function. The parameters of *I* mean (line no. 5):

1. X-position (from left edge to right).
2. Y-position (from top edge to bottom).
3. Rotation angle (only multiples of 90° allowed).
4. Magnification factor in X dimension (integer from 1 to 10).
5. Magnification factor in Y dimension (integer from 1 to 10).
6. Option *a* to load the file automatically.
7. The file name separated by a semicolon (with or without file extension)

The image must be located in the subdirectory *images*. If no matching image is found, the output is omitted without an error message. The file extension is added by the printer itself if it is not specified. Allowed are the extensions *ASC*, *BMP*, *GIF*, *IMG*, *MAC*, *PCX*, *PNG* and *TIF*.

Images are always distributed 1:1 to the points on the print bar (integer magnifications of up to 10x are possible). Therefore images are always printed in different sizes on printers with different resolutions!

Exercise 4.10 Sticker with logo and QR code for mailings

Create a label as a return address label for mailings. In doing so, include a logo of your company. Save the logo in a suitable format in the subdirectory “images” and design your shipping label according to your own ideas.

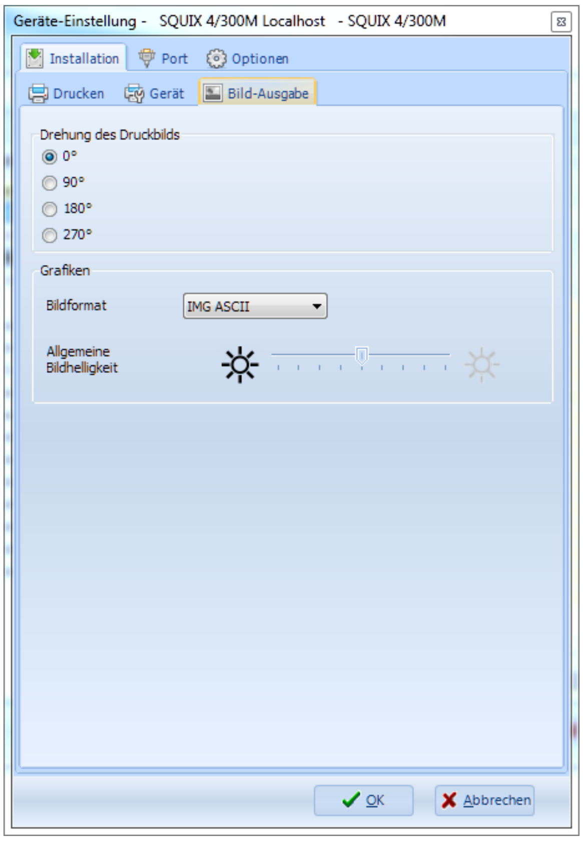
In addition to a graphic, you can also integrate a QR code that contains the address of your website as content (e.g. “https://www.cab.de”). If you would like to view a sample code for orientation, Figure 4.17 on page 49 shows you how to integrate a logo and a QR code into a product sticker. The source code for Figure 4.17 can be found in the listing 4.12 on page 45. Of interest are lines 8 and 21.

4.3.7 Images as embedded ASCII data

If you want to include the image data in the print job, JScript offers the format *ASC*. This is a special format for black and white graphics, which only works with the ASCII characters 0-9 and A-F.

You can read the exact structure of the format in the programming guide, but it is recommended to use cabLabel S3 for the calculation. In cabLabel’s printer configuration, set the image output to IMG ASCII and write the print data to a file (button “Memory card” in the ribbon menu). The figure 4.12 shows the setting in cabLabel S3. In listing 4.10 you see a sample code and its output in figure 4.13.

After the image is transferred to the printer’s RAM using the autoload option, the image is placed in the printer as if the image file were in fixed memory (e.g. on a USB stick). However, this is only possible until the printer is switched off or all image data is removed from RAM using `e IMG;*`.


Geräte-Einstellung - SQUIX 4/300M Localhost - SQUIX 4/300M

Drehung des Druckbilds
(● 0°) ○ 90° ○ 180° ○ 270°

Grafiken
Bildformat: IMG ASCII
Allgemeine Bildhelligkeit

Figure 4.12: The image format IMG ASCII means that the generated JScript file can be edited in ASCII editors.

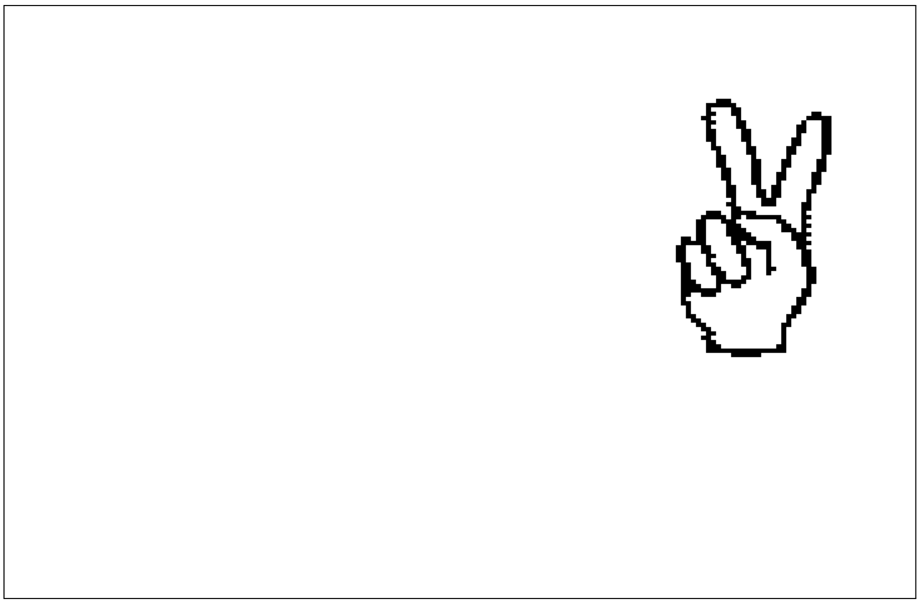


Figure 4.13: A graphic with enlargement (line 24 in listing 4.10)

```
; erase all images from RAM
e IMG;*
; get Victory.ASC into RAM
d ASC;Victory
003100490180017005800201F8058002010C058007018C000C000000080070304001F000000800701
86002300000008007010600630000000800701830063000000080070183004300000008007018300C300
00000180068100C3000000018006C181830000000018006418183000000001800660810200000000000
FF0201800660C30600000000001800630C20600000000000FF0201800630C60C000000001800610C40C00
000001800618 4C080000000018006186C18000000001800618 6C100000000018006083810000000000180
0618383000000000180060C003000000000800701CF802000000000800703EDFC30000000800706380620
0000008007061C03300000000008007061E03A0000000008007061F00F0000000008007360F80E0000000080
073E0DF070000000008007630670600000000800763061030000000008007618210300000000008007610310
300000008007318310300000000800730C11818000000000800730E110180000000080073063001800000000
8007387E001800000000080073C4C00100000000080073FC00030000000000800733800030000000080072000
00600000000008007300000600000000000FF028007100000C000000000800718000180000000080030C00
010480030600030480030300020480030180020480030300020480030180020480030 1C006048003
01FFFE040180020FC0040000FF0D07

m m
J
S l1; 0, 0, 68, 71, 100
H 150, 0
; place the image with autoload and 7x6 magnification
I 75, 10, 0, 7, 6,a;Victory
A 1
```

Listing 4.10: Modified JScript output of cabLabel S3 Victory.lbl

4.3.8 Graphical elements (circles, lines and rectangles)

They consist of one line starting with *G* and follow as parameter:

1. X-position (from left edge to right).
2. Y-position (from top edge to bottom).
3. Rotation angle (integer).
4. A letter separated by a semicolon for the element type (*C* = circle, *L* = line, *R* = rectangle), followed by a colon and further parameters depending on the selected type.

With the circle are the parameters:

5. Radius in the direction of the rotation angle.
6. Radius perpendicular to the angle of rotation.
7. Line width (starting from the radius inwards).[12]
8. If required, further options as special function (*fill*, *shade* or *outline*)

The parameters of a line are:

5. Length.
6. Width.
7. Shape of the beginning of the line (*a* = arrowhead, *r* = rounded, *s* = rectangular).
8. Shape of the line end (as at the beginning).

And follow with a rectangle:

[12] For a filled point with the center point (*X*|*Y*) and the radius *r* you can use the command "`G X,Y,0;C:r,r,r`."

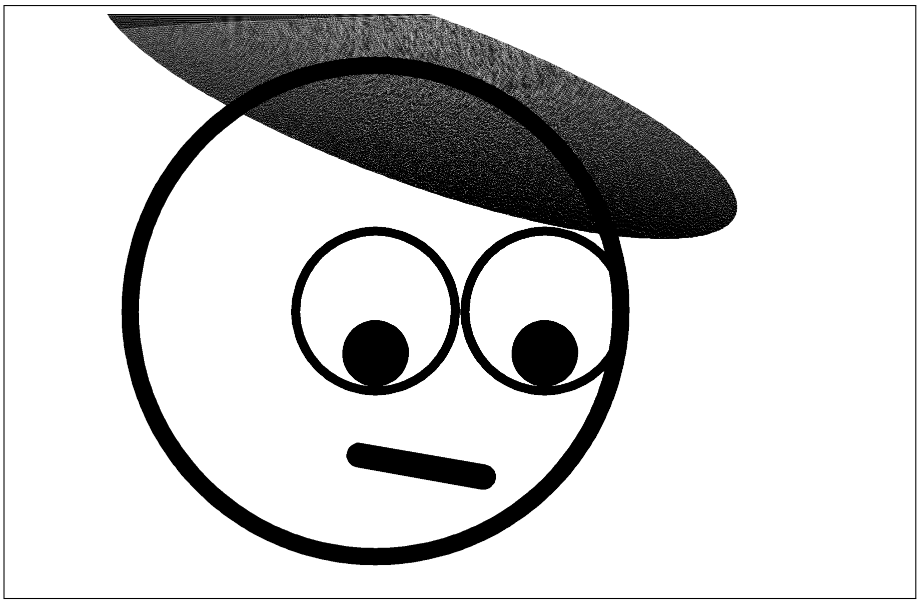


Figure 4.14: Face built from graphic elements (listing 4.11)

5. Width (edge length in rotation angle direction).
6. Height (edge length in clockwise direction perpendicular to the rotation angle).
7. Thickness of the horizontal lines.
8. Thickness of the vertical lines.
9. If required, further options as special function (*fill*, *shade* or *outline*)

The options *fill*, *shade* and *outline* are inserted as special functions (without a comma to separate them). The function name is *F*, *S* or *O*. In listing 4.11 a shading has been used (drawing the cap in line 5). The exact structure is explained in detail in the programming guide.

```
 1 m m
 2 J
 3 S l1; 0, 0, 68, 71, 100
 4 H 100, 2
 5 G 45, 10, 340;C: 40, 10, 44[S:100,50,80]
 6 G 40, 35,   0;C: 30, 30,  2
 7 G 40, 35,   0;C: 10, 10,  1
 8 G 60, 35,   0;C: 10, 10,  1
 9 G 40, 40,   0;C:  4,  4,  4
10 G 60, 40,   0;C:  4,  4,  4
11 G 38, 52, 350;L: 15,  3, r, r
12 A 1
```

Listing 4.11: Graphic elements form a face Gesicht.lbl

Exercise 4.11 The house of Santa Claus

S. 141

Do you remember the drawing exercise “The House of Santa Claus” from Kindergarten days? Create a label in which you draw Santa’s house using the “G” commands in JScript.
The original St. Nicholas Child’s play draws eight lines without stopping. Can you draw the house with less than 8 “G” commands?

4.4 Design the label dynamically

Up to now we have only built a label statically. In other words, we wrote the elements (text, barcodes, etc.) with fixed content into a label as a JScript file.

We will now look at an alternative approach, the separation of content and layout. To do this, we first create a layout whose content corresponds to what will later be printed. But the content is only a placeholder and will be replaced by the actual content before printing.

4.4.1 Referencing content

In order for a content to be dynamically changed (replaced), we need a *reference* to the content so that the printer knows later where to make a content change.

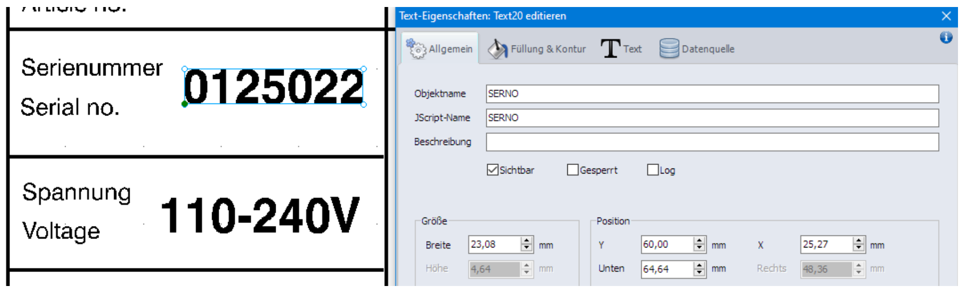
To include such a *reference*, a *field name*, which must start with a letter and may consist of letters and numbers, separated by a colon and a semicolon, is placed after the JScript command. Sounds complicated, looks like this (lines 18 to 21 in listing 4.12)

```
 1 ; usual header
 2 m m
 3 J
 4 S l1;0, 0, 68, 70, 100
 5 H 150, 0
 6
 7 ; static elements
 8 I 72,    2, 0, 1, 1,a;cab-logo
 9 T  4,   52, 0, 3,   3;Part number:
10 T  4,   30, 0, 3,   3;Product name:
11 T  4,   41, 0, 3,   3;Resolution:
12 T  4,    9, 0, 5, 5.4;cab Produkttechnik
13 T  4, 12.5, 0, 3, 2.5;Wilhelm-Schickard-Str. 14
14 T  4,   15, 0, 3, 2.5;76131 Karlsruhe
15 G  0,   18, 0;L: 100, 0.2, s, s
16
17 ; dynamic elements including their field names
18 T:PARTNO; 35, 52, 0, 3, 4;5954501
19 T:PROD;   35, 30, 0, 5, 8, b;A4+
20 T:RESOL;  35, 41, 0, 3, 4;300 dpi
21 B:SERNR;  74, 42, 0, qrcode+ELL+MODEL1, 0.92;0000000000001
22
23 ; no printout, just generate a preview bitmap
24 A [PREVIEW]
```

Listing 4.12: Example of a layout template. Variable elements can be referenced by field names (lines 18 to 21). LayoutVorlage.lbl

The example names four elements with the field names *PARTNO*, *PROD*, *RESOL* and *SERNR*. The first three are text elements, the last one is a QR code.

The last line with the content is important:
`A [PREVIEW]`



Text-Eigenschaften: Text20 editieren

Objektname: SERNO
JScript-Name: SERNO
Beschreibung:
[✓ Sichtbar] Gesperrt Log

Größe
Breite: 23,08 mm
Höhe: 4,64 mm

Position
Y: 60,00 mm
X: 25,27 mm
Unten: 64,64 mm
Rechts: 48,36 mm

Figure 4.15: A text field was created in cabLabel S3. It is first called "Text20". You can change the field name on the *General* tab (this is where "SERNO" is set). If you want to reuse content as described in section 4.4.2, use the *Data Source* tab in cabLabel S3 to create the link.

This does not trigger printing, but creates an internal bitmap. This can be viewed in the browser at the following address:

```
https://192.168.10.1/cgi-bin/bitmap
```

Where *192.168.10.1* is the IP address of the printer (here you must enter the address of *your* printer). If the connection to the printer is set to unencrypted (you can specify this in the printer settings), you must use *http* instead of *https*. Since the SQUIX an encrypted connection is possible, the A$^+$ and Hermes$^+$ printers communicate only unencrypted, and they are not capable of preview graphics using the `A [PREVIEW]` command.

Note: Technically correct, this type of "field name" is called an *identifier*. In colloquial language it is often also referred to as a *variable*. However, variables or data fields usually have a data type. JScript does not know any data types, the field names serve only as reference (pointer) to the content of a line. Strictly speaking, the content is always a character string, which the JScript interpreter converts into a number if required.

4.4.2 Reusing content

Assigning a field name alone does not bring progress. However, the names offer two main possibilities for dynamic design. On the one hand, the contents can be changed later, on the other hand, a content can be used several times in a label.

For multiple use, we simply use the field name like a special function and write the field name in square brackets. This makes it possible, for example, to add a self-selected font and size to a barcode, as the example in listing 4.13 illustrates.

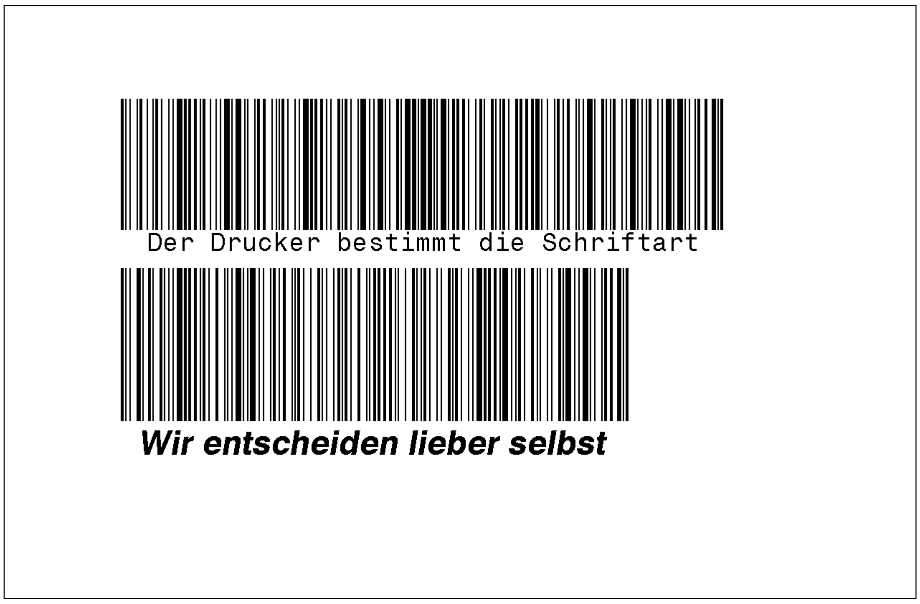

Der Drucker bestimmt die Schriftart
Wir entscheiden lieber selbst

Figure 4.16: The upper barcode has a plain text line generated by the printer, the lower barcode has its own formatting. The field name *SELFMADE* ensures that the correct content is always printed under the barcode if you change it later with only one *R* command. (Created by listing 4.13.)

```
1  m m
2  J
3  S l1; 0, 0, 68, 71, 100
4  H 150, 0
5  O R
6  ; if the barcodetyp is written in capital letters, the printer will add a plain text
7  B           10, 10, 0, CODE128, 18, 0.2;Der Drucker bestimmt die Schriftart
8  ; no capital letters - no additional plain text underneath
9  B:SELFMADE; 10, 30, 0, code128, 18, 0.2;Wir entscheiden lieber selbst
10 ; a selfmade plain text allows custom font and format
11 T           12, 52, 0, 3, 4, b, s;[SELFMADE]
12 A 1
```

Listing 4.13: Multiple use of a content by reference to the field name BarcodeMitTextzeile.lbl

In line 7 a Code128 is printed, which the printer adds a plain text line. But this makes the code smaller in height and we have no influence on the font size or the font style. Line 9 therefore prints a Code128 without a plain text line, but gives the code a field name "SELFMADE". This is used in the following text line to insert the contents of the barcode and thus to reuse the text "Wir entscheiden lieber selbst". The big advantage of this procedure is that both the barcode and the corresponding plain text line can later be assigned new content with a single command (see section 4.4.4).

4.4.3 Calling templates from storage

After we are satisfied with the label layout, we upload the layout template to the subdirectory *labels* on the printer (e.g. via FTP upload). To do this, you can specify the following address in the Windows file browser:

`ftp://ftpcard:card@192.168.10.1/`

An FTP user *ftpcard* always accesses the printer's default memory. This can point to the internal IFFS, or an external USB or SD-CARD root directory.

Here it is assumed that the password still has the default setting *card*. This default should be changed for security reasons! Therefore you have to use the IP of your printer instead of *192.168.10.1* and your password instead of *card* which you have set in the printer settings.

If an external storage medium is used, it should contain the subdirectories *fonts*, *images*, *labels* and *misc* in order to be accepted by the printer. If these directories do not exist, the printer will create them automatically. However, files are not automatically moved to the appropriate directories. They are simply not found.

The storage medium should be formatted with FAT16 or FAT32. However, FAT16 has a limitation of 8 capital letters for the file name, so you should format the storage media with FAT32 (explanations about the formats can be found in the Wikipedia).

In internal memory (IFFS) and when using FAT32 formatted external media, the maximum length of file names is limited to 50, even if the formats would allow more. There is not enough space in the printer's display to display more than 50 Unicode characters, the file names would be truncated.

In our example we save the layout as:

```
labels/LayoutVorlage.lbl
```

The label stored in the printer memory can be loaded with a JScript command:

```
M l LBL;Dateiname
```

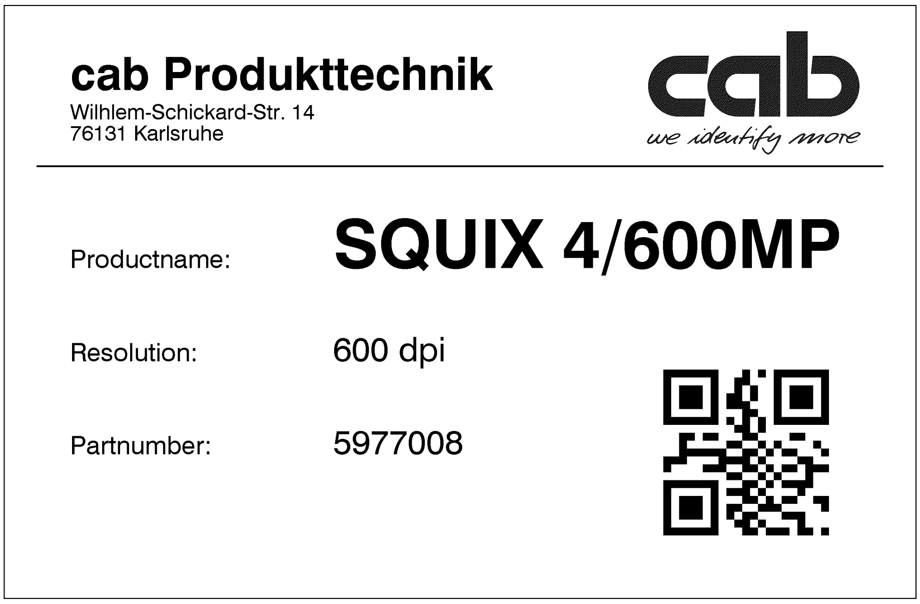
The *M* initiates all accesses to the memory. As parameter follows "*l LBL;*" (lower case *l* followed by the capital letters *LBL* and a *semicolon*). The small *l* stands for load and *LBL* for a JScript file. The file name must not have an extension, otherwise the file will not be found.[13]

4.4.4 Fill in new content

Loading a layout alone makes little sense for dynamic printing projects. Now the values/contents are to be changed. To do this, use the *Replace* Command *R*. If all values are newly set, the printout can be made.

The field name is added to the *R*, separated by a semicolon, and the new content is inserted at the position to which the field name points. The old content is always replaced. The result is shown in figure 4.17.

[13]With *M l LBL;filename.LBL* the printer would search for *filename.LBL.LBL*.

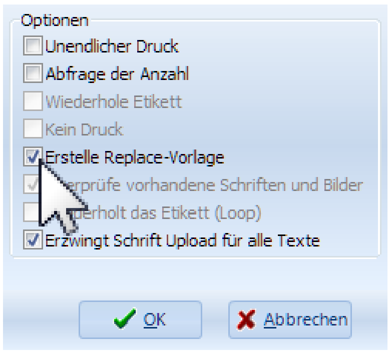


cab Produkttechnik
Wilhelm-Schickard-Str. 14
76131 Karlsruhe

cab
we identify more

Productname:	**SQUIX 4/600MP**
Resolution:	600 dpi
Partnumber:	5977008

Figure 4.17: The layout filled with new content cab-logo.png



Optionen
- [ ] Unendlicher Druck
- [ ] Abfrage der Anzahl
- [ ] Wiederhole Etikett
- [ ] Kein Druck
- [✓] Erstelle Replace-Vorlage
- [✓] erprüfe vorhandene Schriften und Bilder
- [ ] derholt das Etikett (Loop)
- [✓] Erzwingt Schrift Upload für alle Texte

Figure 4.18: *cabLabel S3* makes creating a layout template easy. By checking *Generate Replace template* you will not only get the JScript code for the layout, but also an example of a control file for e.g. a PLC.

```
M l LBL;LayoutVorlage
R PARTNO;5977008
R PROD;SQUIX 4/600MP
R RESOL;600 dpi
R SERNR;164162038304
A 1
```

Listing 4.14: The layout template is loaded. Subsequently, contents are updated. Replace.lbl

4.5 Summary or J SHOW BIG A

We learned how to create a layout and how to change the elements contained in it later. Of course, you could also print a label written in JScript without the detour of uploading, re-reading and modifying. In any case, the printing is only triggered by the final capital letter *A*. So JScript only starts printing when we enter an *A* command.

Using this mnemonic "*J SHOT BIG A*" you can easily remember the correct order of the typical commands of a label.

1. With **J** the job is started.
2. With **S** you set the size so that the printer knows how to arrange the following elements.
3. With **H** you define the print speed and the heat level.
4. Use **O** to select options such as 180° rotation.
5. Then follow with **W** (or **T**) the text boxes (or elements),
6. with **B** the barcodes,
7. with **I** the images and
8. with **G** the graphic elements (circles, lines, rectangles).
   The commands *T*, *B*, *I* and *G* can be mixed with each other, their sequence is arbitrary as long as *S*, *H* and *O* precede them.
9. With **A** the output is then started.

Labels can be completed without printing with the A command.

```
A [NOPRINT]
```

Afterwards the content can be adapted dynamically with the Replace command, e.g. by a PLC.

After an *A* command are until the next *J* command no longer allows for object-generating uppercase commands (after print preparation, contents can only be changed, but no new elements can be added).

5 Special functions

A dynamic label is not only the possibility to change the contents of a label with the Replace command. With its special functions, JScript offers an extremely wide range of possibilities, for example to have the printer calculate contents.

5.1 Syntax of the special functions

5.1.1 Include in square brackets

Basically, special functions can only be used within a content (e.g. in a text, a barcode or a file name for a graphic). The special functions then have an effect on the content similar to a function in other programming languages. The special function is indicated by enclosing square brackets. The square brackets are removed from the contents with the special function contained therein and replaced by a return value if the called special function returns a return value.

This can look like this, for example:

```
T 20, 10, 0, 3, 8;Today is [DATE].
```

Here a text is printed, at position (20,10) with the font Swiss 721 (font number 3) and a font size of 8 units. In the content *[DATE]* is removed and replaced by the current date, the return value of the special function *DATE*. The text around it is retained, so that the label contains a complete sentence (including the dot at the end), which always contains the date of printing.[14]

5.1.2 Pass function parameters

The function can also evaluate one or more parameters (called arguments in programming languages), depending on the special function. The *DATE* function can handle up to three arguments.

```
T 20, 20, 0, 3, 8;Tomorrow is [DATE:1].
```

For the *DATE* function, the first argument gives a date offset in days. Positive values indicate the future. The second argument gives the offset in months, the third in years. The values can be mixed.

```
T 20, 30, 0, 3, 8;Next month we have [DATE:0,1].
T 20, 40, 0, 3, 8;Next year we have [DATE:0,0,1].
T 20, 50, 0, 3, 8;Yesterday was [DATE:-1].
```

The arguments are separated by commas.

[14] If multiple labels are printed without using the single-buffer mode, and if a print started the day before is continued, the date of the previous day will be printed.

5.1.3 Use field names as special function

If a field name is written in square brackets which does not correspond to one of the existing special functions, then the system searches for a JScript line whose content was marked with exactly this name. The return value is the content to which the field name points.

```
T:Name; 20, 10, 0, 3, 4;Fischers Fritze[I]
T       20, 15, 0, 3, 4;[Name] fischt frische Fische.
```

In this example the function *name* is required, but it does not exist. The JScript interpreter then searches for a field name *Name* and finds it one line above. Thus the sentence "*Fischers Fritze fischt frische Fische.*" is printed (a common german tongue twister).

The special function *I* makes the entire line invisible. Therefore only the sentence from the lower line appears on the label (see section 5.2).

Such a referencing via the field name can also be used with one or two parameters. If parameters are used, they cut out a section of the return value. The first parameter is the position from which characters are taken. The second parameter specifies the number of characters.

If start position and length are specified for a field name referencing, these parameters must be separated from the field name by commas, not by a colon as is done for function names.

For clarification another example.

```
T:Name; 20, 10, 0, 3, 4;Fischers Fritze[I]
T       20, 15, 0, 3, 4;[Name,10,5] fischt frische Fische.
```

Here "Fritz fischt frische Fische." will be the result.

5.1.4 No nestings

Special functions cannot be nested. This means, that no further special function may be used as argument of a special function. It is therefore not allowed to use the following code:

```
; totally wrong!
T:Anzahl; 20, 10, 0, 3, 4;4[I]
T         20, 15, 0, 3, 4;Twice of [Anzahl] is [*:2,[Anzahl]].
```

But some functions allow field names as arguments, but you must not use brackets. The following code would be allowed:

```
; so much better...
T:Anzahl; 20, 10, 0, 3, 4;4[I]
T         20, 15, 0, 3, 4;Twice of [Anzahl] is [*:2,Anzahl].
```

5.2 Hiding elements

We have already seen that you can make the line invisible with *[I]*. This way you can hide lines, which are needed only for passing values. Here is an example to illustrate this.

```
T:GEWICHT;  0,  0, 0, 3, 5;10[I]
T          30, 20, 0, 3, 8;Gewicht: [GEWICHT] kg
```

In this case we reference the first line by a field name *GEWICHT*, to which a PLC can be connected. E.g. by

```
R GEWICHT;20
```

you can set the weight to 20 kg. The invisibility is preserved in such a case, only the actual value is changed from 10 to 20. The line with the text in front and behind is printed, but a PLC only has to pass the pure numerical value.

5.2.1 Conditional visibility

The *I* function allows one more optional argument. If it is 0, the line will not be invisible. Thus a conditional visibility is realizable, as shown in listing 5.1.

```
m m
J
S l1; 0, 0, 68, 71, 100
H 150, 0
T:drink; 5, 20, 0, 3, 10;Apple juice
T:sugar;  0,  0, 0, 3,  3;98[I]

; Do not hide if sugar > 0
T 5, 30, 0, 3, 5;Sugar content:   [sugar] g/l[I:!sugar]

; Hide if sugar > 0
T 5, 30, 0, 3, 3;Warning: This drink is not suspected of promoting[I:sugar]
T 5, 35, 0, 3, 3;obesity in children![I:sugar]

A 1
R drink;Mineral water
R sugar;0
A 1
```

Listing 5.1: Conditional Visibility Conditional-Invisible.lbl

A field name must be used as argument. If the content, to which the field name refers, differs from 0 (numeric zero), the condition becomes true and the [I:Fieldname] special function unfolds its effect. By placing an exclamation mark in front of the field name, the condition is reversed (invisible if content is numerically zero).

Always use numeric values (0 and 1) or a logical operation, but never a string. The strings “true”, “false”, “yes”, “no”, “to” and “off” are all identical for the JScript interpreter (they numerically correspond to zero).

In this way, images (e.g. danger symbols) can also be shown or hidden selectively.

Here is another example. Labels with a width of 80 mm and a height of 20 mm are processed to create name tags for sticking on. The data comes from a software which sends the following file to the printer (listing 5.2).

```
M l LBL;LayoutNamensschilder

R Company;Musterfirma GmbH
R Sex;male
R Name;Anton Müller
A 1
R Name;Bernhardt Schmidt
A 1
R Sex;female
R Name;Lydia Weber
A 1

R Company;Firma hinterm Deich AG
R Sex;female
R Name;Antonia Fischer
A 1
```

Listing 5.2: JScript code as it could be sent by a database system Namensschilder.lbl

What could the corresponding layout look like? A possible solution would be listing 5.3.

```
 1 m m
 2 J
 3 S 0, 0, 20, 23, 80
 4 H 100, 5
 5
 6 ; variables (all invisible)
 7 T:Company; 0, 0, 0, 3, 4;Beispielfirma GmbH[I]
 8 T:Sex;     0, 0, 0, 3, 4;female[I]
 9 T:Name;    0, 0, 0, 3, 4;Martina Musterfrau[I]
10
11 ; the logic part
12 T:WennMann; 0, 0, 0, 3, 4;[==:Sex,male][I]
13
14 ; name with salutation
15 T 5, 10, 0, 3, 6, b;Mr. [Name][I:!WennMann]
16 T 5, 10, 0, 3, 6, b;Ms. [Name][I:WennMann]
17
18 ; company name
19 T 5, 15, 0, 3, 4;[Company]
20
21 A [Preview]
```

Listing 5.3: The layout template for the name tags contains two texts, of which only one is printed at a time LayoutNamensschilder.lbl

In lines 7 to 9, three invisible texts are created, into which the Replace command is later used to write. In line 12 a string comparison takes place. If the character string “male” is in the field *Sex*, the return value of the compare function is 1, otherwise 0.

The lines 15 and 16 are interesting: Here “Mr. …” is printed, but this print becomes invisible if the content of the field *WennMann* does not differ (exclamation mark) from 0 (numeric zero). If we want to print a text, we must not hide it, so the condition must be reversed by the preceding exclamation mark. We use the same logic in line 16, where we hide the text “Ms. ...”, if the condition *WennMann* is true.

5.3 Ask the operator

The question mark leads to a dialog prompt on the printer display. The operator can then enter an answer on the display, accept the default value, or make an entry using a USB keyboard or USB barcode scanner.

```
T:AUFTRAG; 0, 0, 0, 3, 5;[I][?:scan production docket]
```

With this line of code, “scan production docket” is shown on the display and the operator must enter a value. He can use a USB barcode scanner for this purpose. The (scanned) value can be used later in the JScript code with [AUFTRAG].

After the text to be displayed, a default value can be passed, separated by commas, which the operator then only has to confirm by tapping the *ENTER* button.

```
T 0, 0, 0, 3, 5;[I][?:enter password,123456]
```

After the default value, you specify how often the specification is used before it must be re-entered. The default value is 1, so the input must be re-entered after each print. The last entered value is used as the new default value, so that the operator only has to tap the *ENTER* symbol when the same entry is used several times.

```
T 0, 0, 0, 3, 5;[I][?:enter PIN,0000,1,M!1111]
```

Here a four-digit input consisting of digits without spaces is required. A fourth parameter starting with “M” describes the permitted characters (input-*mask*). The exclamation mark prohibits the use of a space, the digit 1 stands for a number (more precisely a single digit). So the query here must consist of exactly four digits and must not contain a space.

The special function to query the display can be provided with many more arguments, for example to allow only a fixed number of certain characters. In the appendix you will find a more detailed description in the table on page 137.

Exercise 5.1 Handwritten name badges in stand-alone operation *S. 142*

Design labels as shown in Figure 4.7 on page 34 and modify the print job so that the label file can be stored on the printer for stand-alone operation. Copy the file to the printer memory (USB memory stick, SD card or IFFS) and recall the label from the printer. The names and the companies should be queried. After that one label at a time should be printed and immediately asked for the next name. The old name should appear as the new default value, in case someone wants to print a second label. The same applies to the company name.

5.4 Dynamic date and time functions

We have already learned about the date function *DATE*. It inserts the current date, always in the format that corresponds to the country setting on the printer.

If you want to define the format yourself, e.g. the month as two digits followed by a dash and a four-digit year value, you have to combine several other date functions.

```
T 20, 30, 0, 3, 8;Filled on: [MONTH02]/[YYYY]
```

5.5 Adding a time offset

All date functions allow up to three arguments, which give the offset in days, months and years.

```
T 20, 40, 0, 3, 8;Best before: [MONTH02:0,6]/[YYYY:0,6]
```

A date is output here as before, but with a 6-month offset. The offset must be carried out in the same way for each function if several functions are combined into one display.

Time functions allow an offset in up to three arguments, hours, minutes and seconds.

```
T 20, 50, 0, 3, 8;Three hours ahead it will be [H024:3]:[MIN:3].
```

5.6 Using counter

Counters are created with the *SER* function. The arguments are:

1. First value (will be printed onto the first label).
2. Increment value (steps between each label).
3. Stepping frequency (after this amoung of labels a stepping is done).

The start value must be specified, the two further arguments are assumed by the printer to be 1 if they are not specified.

The counter works like a mechanical numbering machine. When using the function *SER*, the first argument also determines the maximum number of digits by preceding zeros. Thus `[SER:001]` counts starting from 001 up to 999 and then falls back to the value 000. As a mechanical work would behave. Thus, the counter returns 001, 002, 003, 004, ..., 997, 998, 999, 000, 001, 002, ...

If you don't want leading zeros in print, you have to use the *D* Use function for the minimum number of digits to be printed. But this can only work with a calculation with which *SER* function the *C* and *D* are compatible Functions not. In the code example in listing 5.5 we see such a solution in line 7, where additionally an addition with a start value from a query was realized.

Exercise 5.2 Complex numbering of packages *S. 142*

20 components should be distributed evenly over 4 boxes. You want to print 100 mm wide and 10 mm high labels for identification and glue them on two opposite sides of a carton. Five components are packed in each of the four cartons. Each label is to be marked "01 to 05 of 20". Two copies of each label (for two opposite sides of a box). The next carton would then have two labels with the inscription "06 to 10 of 20". Complete the entire print in a single JScript file ending in "A 8".

5.7 Remember the counter value

This operation is equivalent to a value added from outside and used as a starting value. Unfortunately the code in listing 5.4 does not work like this.

```
m m
J
O R
S l1; 0, 0, 68, 71, 100
T:start; 0, 0, 0, 3, 5;[I][?:Enter start value,98]
; That will be a complete flop:
T 10, 50, 0, 5, 40;[SER:start]
A 4
```

Listing 5.4: The serial number function must not be used in this way Serienfehler.lbl

The idea would be that a starting value could be entered by the user. In principle a good idea, but the *SER* Function does not support referencing.

But there is hope (listing 5.5).

```
m m
J
O R
S l1; 0, 0, 68, 71, 100
T:start;   0,  0, 0, 5,  5;[?:Enter start value,98][I]
T:offset;  0,  0, 0, 5,  5;[SER:000][I]
T         10, 50, 0, 5, 40;[+:start,offset][D:1,0]
A 4
```

Listing 5.5: Setting a counter value the right way Counter-start-value.lbl

If the operator accepts the default value of 98, the printer delivers four labels with the imprint 98, 99, 100, 101. The *D* function makes sure that no leading zeros are printed.

5.8 Calculations and comparisons

As shown in the previous example, you can also do calculations in JScript. There are the functions +, -, *, /, % and many others. The special functions = and == compare the numeric value or string for equality. See the Programming Guide for a complete list.

Here is a small example (listing 5.6 and figure 5.1).

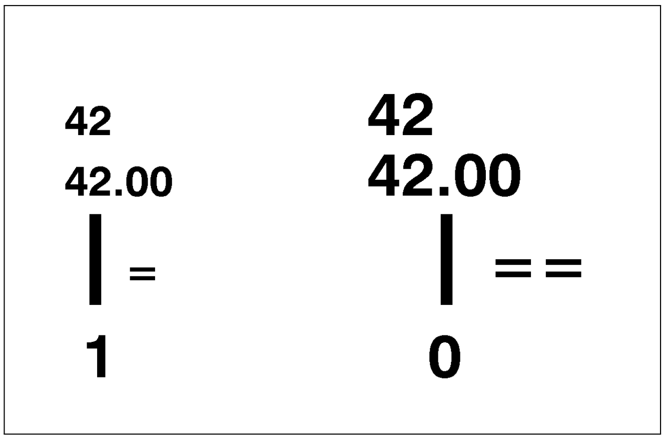


Figure 5.1: The result is 1 if they match, otherwise 0

```
m m
J
S l1; 0, 0, 68, 70, 100
H 150, 0

T:VAR1;  5, 20,   0, 5, pt20;42
T:VAR2;  5, 30,   0, 5, pt20;[*:6,7]
T       15, 45,   0, 5, pt20;=
G       10, 33, 270;L: 15, 2, s, a
T        8, 60,   0, 5, 10;[=:VAR1,VAR2]

T       55, 20,   0, 5, 10;[VAR1]
T       55, 30,   0, 5, 10;[VAR2]
T       75, 45,   0, 5, 10;==
G       68, 33, 270;L: 15, 2, s, a
T       65, 60,   0, 5, 10;[==:VAR1,VAR2]

A 1
```

Listing 5.6: Comparison of numerical values Numvergleich.lbl

It shows the multiplication (line 7 as `[*:6,7]`), the value comparison (left as `[=:VAR1,VAR2]`) and the string comparison (right as `[==:VAR1,VAR2]`). If you compare the values, 42.00 is identical with 42, if you compare the strings “42.00” with “42”, there is no match. Here you have to be a bit careful what you compare and how.

For clarification once again with strings as contents of the fields (listing 5.7 and illustration 5.2).

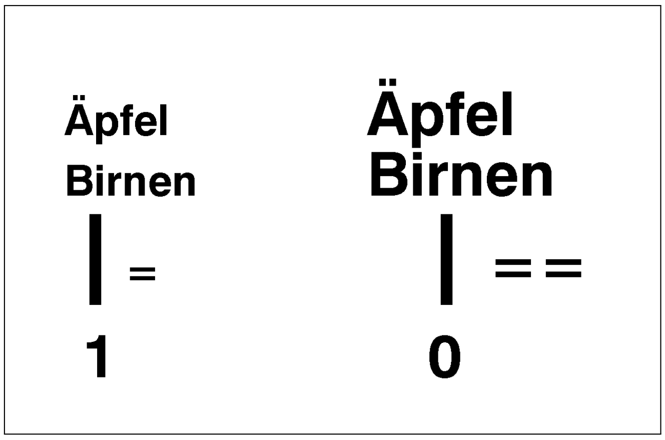


Figure 5.2: Apples and pears both yield 0 when converted to numerical values, which is numerically identical

```
m m
J
S l1; 0, 0, 68, 70, 100
H 150, 0

T:VAR1;  5, 20,   0, 5, pt20;Äpfel
T:VAR2;  5, 30,   0, 5, pt20;Birnen
T       15, 45,   0, 5, pt20;=
G       10, 33, 270;L: 15, 2, s, a
T        8, 60,   0, 5, 10;[=:VAR1,VAR2]

T       55, 20,   0, 5, 10;[VAR1]
T       55, 30,   0, 5, 10;[VAR2]
T       75, 45,   0, 5, 10;==
G       68, 33, 270;L: 15, 2, s, a
T       65, 60,   0, 5, 10;[==:VAR1,VAR2]

A 1
```

Listing 5.7: String comparison Stringvergleich.lbl

Exercise 5.3 Variable number of packages S. 143

Design two numbering labels on opposite sides of a box, as required in Exercise 5.2. However, modify the JScript file so that a dialog on the printer asks for the total number of pieces to be packaged and the number of pieces per box. The labels are then printed.

5.9 Format strings or numbers

If you want to format a number, it must be the result of a calculation. If there is no calculation at hand, you can multiply by 1 or add 0. Formatting is then done with the *D* Function, which expects the number of integer and decimal places as arguments. With the *C* function you can still define how to fill up to the required number of digits in the front.

Assuming `[value]` points to 123.4567, then `[*:1,value][C:0][D:5,0]` returns the result “00123”. Note that the dot is considered a decimal separator, even if the country setting is Germany instead of the USA. `[*:1,value][C: ][D:5,2]` returns “ 123.45”, with two leading spaces and without commercial rounding.

If you want a defined rounding instead of truncation, you can use the *R* function.
`[*:1,value][C: ][D:5,2][R:m]` would yield “ 123.46”.

If there are more places before the decimal point than given, all places are printed.
`[*:1,value][C: ][D:1,3][R:m]` returns “123.457”.

Without the *C* and/or *D* Function rounds the function *R* always to two places.
`[*:1,value][R:m]` would result in “123.46”.

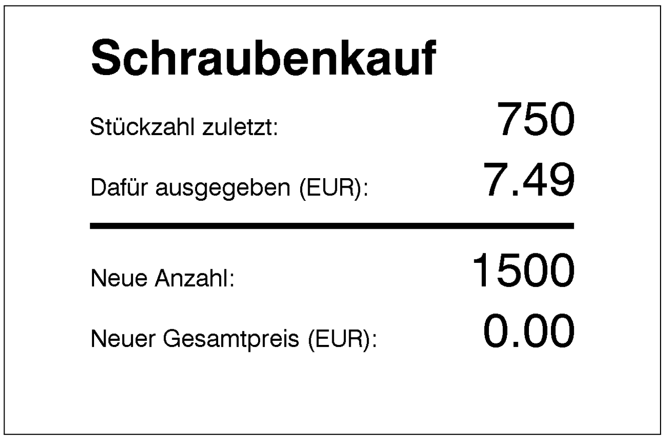
A character string can be freed from leading or trailing spaces with the *TRIM* function. With *J* you can arrange texts left, center or right. As an argument, after the lowercase letter *l*, *c* or *r*, you have to specify the area (in the selected unit of measurement) over which the alignment should take place. `[J:c50]` centers the text over 50 mm (or inches, depending on the country setting). You can find an example of the use of the alignment in the list 4.4 on page 33. There a text in the code lines 22 to 25 is aligned right.

5.10 Avoid rounding errors

If a calculation has several steps, you need several lines in JScript, because you cannot nest special functions. Usually you use invisible text lines and assign field names ending with a consecutive number, e.g. “ZS1” for the first intermediate step. cabLabel S3 uses letters.

The problem with these intermediate steps in JScript is that (unless otherwise defined) each result is output as a floating point number with two decimal places. The contents behind the field names are always strings in JScript, since they are derived from text.

But even with a simple rule of three, this can lead to problems. Let us assume we want to buy screws. Last time we bought 750 screws and paid 7,49 € for them. But today we need the double amount of 1500 screws. Now one could assume that we, with a rule of three calculation, also have to pay the double price, so exactly 14.98 €. If we look at the result of the calculation within the JScript label, a very special offer seems to work (listing 5.8).



Schraubenkauf

Stückzahl zuletzt:	750
Dafür ausgegeben (EUR):	7.49
Neue Anzahl:	1500
Neuer Gesamtpreis (EUR):	0.00

Figure 5.3: Today everything for free – JScript must be kidding?

```
J
S l1; 0, 0, 68, 71, 100
H 100, 0
O R
T              10, 10, 0, 5, 8;Schraubenkauf
T              10, 20, 0, 3, 4;Stückzahl zuletzt:
T:AnzahlZG;    10, 20, 0, 3, 8;[J:r80][?:Zuletzt gekauft (Stück),750]
T              10, 30, 0, 3, 4;Dafür ausgegeben (EUR):
T:GPZG;        10, 30, 0, 3, 8;[J:r80][?:Dafür bezahlt (EUR),7.49]
G              10, 35, 0;L: 80, 1
T              10, 45, 0, 3, 4;Neue Anzahl:
T:AnzahlNEU;   10, 45, 0, 3, 8;[J:r80][?:Anzahl jetzt (Stück),1500]
T              10, 55, 0, 3, 4;Neuer Gesamtpreis (EUR):

; now we calculate a rule of three in JScript
T:Einzelpreis; 0,  0, 0, 3, 3;[I][/:GPZG,AnzahlZG]
T:GPNEU;        0,  0, 0, 3, 3;[I][*:Einzelpreis,AnzahlNEU]

; print the result
T              10, 55, 0, 3, 8;[J:r80][GPNEU]
A 1
```

Listing 5.8: Caution with multi-step calculation and disregarding the intermediate result formatting Rundungsfehler.lbl

Exercise 5.4 Avoid errors in multi-step calculations S. 144

Change the listing 5.8 so that a correct result is calculated.

5.11 Inserting UNICODE characters

The special function *U* needs a number as argument and returns the corresponding Unicode character. Thus `[U:8364]` returns a Euro currency symbol. You can also specify the hexadecimal value, then a dollar sign has to be put in front. So `[U:$20AC]` also results in the Euro currency symbol.

A list of all official Unicode characters can be found here:
`http://www.unicode.org/charts/`

But you should check if very exotic Unicode characters are defined in the used font. Not all fonts return a unicorn face for the character `[U:$1F984]`.

Fonts with many UNICODE characters can also slow down the processing of a label if they are sent to the printer together with the print job. For example, the “Arial Standard” font file is only 756 KB in size, while the “Arial Unicode MS Standard” font requires 22.1 MB of disk space, almost thirty times as much.

Large TrueType font files should be kept on the printer and should only be included in the job using the *M l fnt;...* and *F* command.

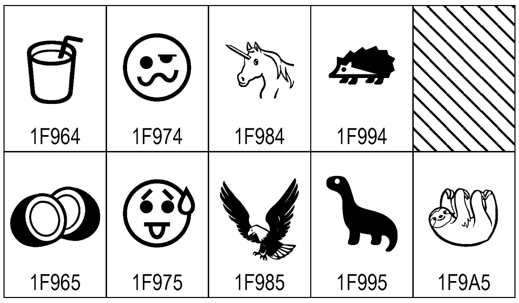


Figure 5.4: Some rare UNICODE characters

5.12 Single buffer mode

With the parameter *S* of the *O* command, the *single label buffer* option can be enabled. Normally, the printer pre-calculates as many labels as possible in advance for faster printing (no delay between printing single labels).

However, if a special function is part of a dynamic label, the calculation is only performed at the beginning of the process, and is then applied to all subsequent labels (if sufficient buffer memory is available).

You can see the effect in the following JScript code (listing 5.9), which uses BASIC to insert a delay of two seconds between the simulated pressing of the dispense button. Everything that is printed in the BASIC code with the command `PRINT` is sent to the JScript interpreter as a JScript line, so it is treated as if we had sent the code to the printer from outside. The command `POKE("io.xin")` simulates the keystroke on the demand key in dispense mode.

```
<ABC>
PRINT "m m"
PRINT "J"
PRINT "S l1; 0, 0, 68, 71, 100"
PRINT "H 150, 0"
REM option J = request labels by keystroke (force "print on demand" mode)
PRINT "O J"
PRINT "T 10, 40, 0, 5, 10;[TIME]"
PRINT "A"
FOR i = 1 TO 3
    PAUSE 2
    POKE("io.xin"), "START"
NEXT
END
</ABC>
```

Listing 5.9: Keystrokes made with BASIC SingleBufferBASICDemo.lbl

If you are confused about the BASIC lines, here is the pure JScript code (listing 5.10). Start it and press the trigger yourself several times with a pause of about two seconds.

```
m m
J
S l1; 0, 0, 68, 71, 100
H 150, 0
O J
T 10, 40, 0, 5, 10;[TIME]
A
```

Listing 5.10: One label with demand button on the display SingleBufferDemo.lbl

Exercise 5.5 Effect of the single buffer mode

Send the code from listing 5.9 to the printer and watch the printout. Now switch the printer to single buffer mode by changing line 7 to :

```
7 PRINT "O J, S"
```

Send the code to the printer again. What changes? If you see no difference, the option “single buffer mode” may set on in the printer’s menu. Turn it off to and try again if both printouts didn’t differ.

6 Using the different memories

If you want to remember values between the individual print jobs, e.g. the counter value of the last label printed, you have the choice between different storage locations. You can create a file and write the values into it. Or you can use one of the predefined memories.

6.1 The useful user memory

Creating a file and constantly writing to this file has a major technical disadvantage. Flash memories, such as an SD card, a USB memory stick or the internal IFFS have the technical characteristic of abrasion during write operations. While it is possible to read from a flash medium almost as often as desired, the writing processes are limited.

For this reason the cab printers have a user memory in the RAM of the battery backed up real time clock. It is not affected by abrasion, it can be written as often as you like. However, the memory is only 32 bytes large and global, i.e. shared by all JScript files (labels).

If global availability is no problem, [WUSER] and [RUSER] can be used to write to and read from the memory (see listing 6.1).

```
m m
J
S l1; 0, 0, 68, 71, 100
T:XVAL;  10, 10, 0, 3, 3;[RUSER][I]
T:SERNO; 10, 10, 0, 3, 3;[+:XVAL,1][D:0,0][I][WUSER]
T        10, 20, 0, 3, 8;Serial Number: [SERNO]
A 4
```

Listing 6.1: Write values into the user memory and read them out again WUSER-RUSER.lbl

6.2 The “I have finished!” info memory

Instead of using *[WUSER]* in the 32 bytes user memory, you can also write to the information memory. This is done with *[WINF]*. The big difference to *[WUSER]* is that there is no *[RINF]* function. The Info memory can only be read out using the escape function Esc *i*.

The special feature is that the actual writing process in the 128 bytes of volatile memory (RAM) only takes place after the label has been completely printed and (if working in the dispense mode) dispensed. If, for example, each label is provided with an unique serial number, this number can be written to the info memory. A PLC can then read the serial number with the appropriate escape command and ensure that the label was printed and (if provided) also dispensed.

6.3 Using a file to store a value

If you want to use a file as storage location, you can name the file yourself. This has the advantage that different JScript label jobs do not have to share a common memory. For example, different layouts for different product labels could each remember their own counter without affecting each other.

If a file is to be used, by the "*E TMP;...*" command (capital letter E), the file name (without file extension) must be defined first. After that, you can write and read with [WTMP] and [RTMP].

Writing to a file cannot be done in IFFS, a USB memory stick, SD card or WebDAV drive must be used. The command always refers to the default memory. If the IFFS is selected as default memory, an error message is displayed.

6.3.1 Using a TMP file

With a TMP file, the contents are completely overwritten with each write operation. So the file always contains only the last written content, which is limited to a length of 128 bytes.

The example shown for user memory can also be implemented with a file as storage location (listing 6.2).

```
1 m m
2 J
3 S l1; 0, 0, 68, 71, 100
4 E TMP;KillMyFlash
5 T:XVAL;  10, 10, 0, 3, 3;[RTMP][I]
6 T:SERNO; 10, 10, 0, 3, 3;[+:XVAL,1][D:0,0][I][WTMP]
7 T        10, 20, 0, 3, 8;Serial Number: [SERNO]
8 A 4
```

Listing 6.2: Saving a value to a TMP file WTMP-RTMP.lbl

New here is line number 4, where the file name to be used is defined (*KillMyFlash.TMP* in the subdirectory */misc*). As already mentioned in the name you should consider writing to a flash memory not too often.

6.3.2 Filling a LOG file

A LOG file can be handled in the same way as a TMP file. But there is only one *[WLOG]* special function and no *[RLOG]*. So you can only write to the file. But for this the write operations limited to 128 bytes are appended, the file content is kept and the new content is added as a new line.

```
1 m m
2 J
3 S l1; 0, 0, 68, 71, 100
4 E LOG;INFO
5 T:VAL;   5,  6, 0, 3, 3;[SER:0001][I]
6 T:PRINT; 5, 15, 0, 3, 3;Label [VAL] printed on [DATE] at [TIME].[WLOG]
7 A 3
```

Listing 6.3: Logging into a file Logging.lbl

Exercise 6.1 A CSV file as protocol of all printouts

S. 145

You have a donation run with many runners to look after. The runners who want to take part in the run can pay a starting fee/donation on site and will be assigned a starter number automatically. They leave their address in order to receive a certificate later on.

Create a JScript file that assigns an unique number to each runner starting with number 1. The runners enter their names in a dialog on the printer (a SQUIX with USB keyboard). You will receive a 100 mm wide and 68 mm high label with your name and starter number.

For the event manager the printer should create a file on a USB stick with the following content in CSV format: “start number”, “first name”, “name”, “street”, “house number”, “postcode”, “residence”, “date of entry” and “time of entry”. The CSV file already exists at the start of the event and consists of the first line with the field names. The JScript file should therefore only append the respective data record.

The runners may decide during the input whether the first and last name should be printed completely on the label next to the start number or whether the last name should be abbreviated with the first letter. After input, printout and data storage, the input process starts again.

Exercise 6.2 The cherry on top for the JScript expert

S. 146

Complete the previous exercise so that the font size for the name on the name tag is automatically set so that the name appears completely on the label.

Hint: Look in the programming guide for the special command `[LEN:field name]` and work with conditional visibility.

Save the print job under the name “DEFAULT.LBL” so that it starts automatically when the printer is switched on. Design the start number assignment so that accidentally switching off and restarting the printer does not confuse the start numbers.

7 The most common Escape commands

In this manual only the most important escape commands will be described. For a complete list please refer to the Programming Guide.

Escape commands provide an alternative approach to communicating with the printer. The main difference to the "normal" JScript is that it is not line-oriented, and escape commands are processed directly.

They are usable on all interfaces where JScript is received. However, they never reach the JScript interpreter, the software in the printer that processes JScript lines. Instead, they are filtered out by the respective interfaces and processed directly. If an escape command returns an answer, it comes directly from the same interface.

It is possible to merge escape commands into the JScript code. But this is not absolutely necessary. Escape commands can also be used alone on an interface without sending additional JScript code. For example, a PLC can query the status via an interface such as RS232 with the printer using escape commands, while the print data is sent as JScript code over the network from a PC.

7.1 Interrupt the printing process (Esc p1 and Esc p0)

Pressing the pause button on the printer can also be done via one of the data interfaces. To do this, send the character string Esc *p1* to a JScript interface to set the printer to pause mode and Esc *p0* to end the pause.

This means the character strings consisting of the special character Esc , i.e. the ASCII character no. 27, the lower case letter *p* and the number *1* or *0* (hexadecimal *1B 70 30*$_{16}$ or *1B 70 31*$_{16}$).

If a printer is set to the Pause state, it still finishes the current label to be printed and then interrupts the current job. This ensures that no label is printed incompletely. The Esc *t* command described below, on the other hand, immediately interrupts all printing activity, even if this means that a label is not completely printed.

7.2 When nothing works at all (Esc t and Esc ! Esc !)

Via one of the interfaces you can send the sequence Esc *t* (ASCII values 27 and the lower case letter "t", or hexadecimal *1B 74*$_{16}$). The printer will then reset as if the Cancel key had been pressed for at least three seconds[15].

If this does not help yet, you can switch off the printer and restart (reboot). For this purpose send Esc *!* Esc *!* or hexadecimal *1B 21 1B 21*$_{16}$.

[15] Pressing *Cancel* for three seconds will result in the A^{+} printer to a cancel with erasing all pending jobs. You can cause the same reaction on the EOS, SQUIX or Mach 4S by pressing the red symbol with the rejecting hand.

7.3 Query status (Esc s and Esc z)

With Esc *s* the status of the printer can be queried. Historically, Esc *s* is the older command and the response is exactly 9 characters. These have the following meaning:

1. The first letter indicates whether the printer is ready to receive (response is *Y*) or not (response is *N*).
2. The second letter is an error code. A "-" (minus character) indicates that there is no error. The detailed error list can be found in the programming manual and in the appendix on page 138.
3. The numbers in position three to eight in the answer indicate the number of labels still to be printed.
4. The last character is again a letter which indicates whether the JScript interpreter is currently active (*Y* = yes, *N* = no).

Later in the evolution of the script language JScript, the Esc *z* command was added. It returns twelve letters as response, which can be either *Y* (for yes) or *N* (for no). They indicate:

1. Printer is in pause mode
2. A job is loaded
3. Printer not able to receive more data
4. Label is moving
5. Pre-warning of foils (hardware-dependent)
6. Pre-warning of labels (hardware dependent)
7. Label in dispensing position
8. Label on vacuum plate (hardware dependent)
9. Applicator ready (hardware dependent)
10. External pause signal active (hardware dependent)
11. External Start signal active (hardware dependent)
12. Print head cleaning recommended (cleaning interval reached)

To be able to provide further information in the future, the answer closes with a carriage return character (hexadecimal $0D_{16}$).

For a printer with applicator, the answer after switching on is normally:

```
NNNNNNNNYNNN<CR>
```

7.4 Reading from the information memory (Esc i)

To exchange data between a PLC and the printer, and at the same time check whether the label has been dispensed, there is the information memory. It is 128 characters in size and volatile (part of the RAM).

This memory is filled with the special function `[WINF]`, which writes the contents of the line in which the function is used to the INF memory.

However, this is only done after the label has been completely finished. Completely output means that for a label in dispensing mode, the value is only written to the INF memory when it has actually been picked up under the photocell of the dispensing sensor.

The selected value for the INF memory may also be marked as a non-printable element by additionally adding the special function `[I]` to the same line of code.

The option "S" in line 5 is important: If the printer is not working in Single Buffer mode, all labels would have the same timestamp. If the printing process was started by the night shift, but only four labels were

```
m m
J
S l1; 0, 0, 68, 71, 100
H 150, -2
O R, S
T:ETINUM; 5,  6, 0, 3, 3;[SER:00001][I][WINF]
T         5, 15, 0, 3, 3;Label [ETINUM] printed on [DATE] at [TIME].
A 12
```

Listing 7.1: JScript writes the invisible text from line 6 to the INF memory WINF-ESCi.lbl

printed, the morning shift would find the date and time from the previous day on the 8 labels still to be printed.

`[DATE]` and `[TIME]` from the example are reserved function names and print the current date and time in the format typical for the country setting. However, because JScript is case-sensitive, `[Date]` and `[Time]` are references to the field names *Date* and *Time* (if defined as such).

The command Esc *i* thus returns the character strings in the example as an answer one after the other:

```
00001<CR>
00002<CR>
00003<CR>
00004<CR>
...
00012<CR>
```

But only after the corresponding label has been printed completely.

7.5 Start signal (Esc g)

If a printout is not to be made immediately for all desired copies, but for one copy at a time by activating a start contact, the command Esc *g* can also be used instead of the contact at the I/O interface (to be sent in hexadecimal form as two bytes $1B\,67_{16}$, without terminating line end character).

The command Esc *g* corresponds to exactly one single triggering of the *START* contact at the I/O interface. Thus, a printer can be controlled exclusively via a TCP/IP connection without having to use the I/O interface too.

But also a remote control of an EOS2 or EOS5 printer, which both do not have an I/O interface (in contrast to the SQUIX), will react to the Esc *g* command.

7.6 Trigger the I/O interface (Esc xin)

The possibilities to control a cab printer via TCP/IP connection (and without direct wiring of the I/O interface) are even more versatile. To control the I/O interface on the TCP/IP network there is the command Esc *xin*. The command must be followed by the name of the signal/connection, finished with a semicolon. An end of line is not necessary.

The commands are interpreted as triggers. It is therefore not possible to set the printer to pause state. For this purpose, the separate commands Esc *p1* and Esc *p0* must be used.

Since these are escape commands, they can be sent, for example, by a PLC to port 9100 of the printer, while the print data itself is sent to the printer via a PC or data server.

The commands are as follows:

Table 7.1: Trigger I/O signals Esc *xin*

command	result
Esc *xin FSTLBL;*	Corresponds to the signal input *Print first label* of the I/O interface. Only required if *Apply/Print* mode is selected on a printer with an applicator.
Esc *xin START;*	Starts printing a (further) copy.
Esc *xin STOP;*	Stops printing[16]
Esc *xin REPRINT;*	Repeats printing the last label.
Esc *xin RSTERR;*	Confirms an error message if the printer is blocked by an error.
Esc *xin LBLREM;*	If the printer is in dispensing mode but without a cab applicator, the signal *Label removed* must be used to give the printer permission to print the next copy of the print job.
Esc *xin JOBDEL;*	Deletes the currently pending print job. The printer returns to the *ready* state.

7.7 Reading the I/O interface (Esc xout)

The I/O interface can not only be triggered by an escape command, it is also possible to read the digital outputs of the printer. To do this, send the command Esc *xout* to the printer. Because it is an escape command, no line end is required.

The printer responds with a sequence of the letters *N* for No or *Y* for Yes. The letters are terminated by a carriage return character (decimal 13, hexadecimal $0D_{16}$). They stand in order for:

1. READY (Pin 10)
2. JOBRDY (Pin 9)
3. FEEDON (Pin 4)
4. ERROR (Pin 22)
5. RIBWARN (Pin 15)
6. PEELPOS (Pin 21)
7. HOMEPOS (Pin 5)
8. ENDPOS (Pin 3)
9. LBLWARN (Pin 2)
10. RIBERR (Pin 7)
11. MEDERR (Pin 8)

7.8 Outlook: Printer control in the industry 4.0 age with OPC UA

Today's machine communication is network-based, devices communicate with each other via TCP/IP based services and protocols. JScript and the escape commands are still designed for direct communication without complex protocols. In the past two decades, in which JScript exists, a lot has changed, especially in the last few years. The current cab printers (except Mach1 and Mach2) are using the OPC UA protocol. It is possible to replace all escape queries with the advantages of a modern protocol. Therefore OPC UA provides a mature security concept which allows to send commands encrypted only from authorized devices to the printers.

[16] The STOP signal interrupts each operation immediately and puts the printer into an error state (an EOS or SQUIX display shows *External Error*). A print command sent in the error state will not be executed and the printer will remain quiet. In contrast to pressing an emergency stop button, the error can be acknowledged and the printer can be used again by pressing Esc *xin RSTERR;*.

If you want to know more about OPC UA, please have a look at the cab web page. There you will also find special instructions how to integrate a cab printer into your production process control or IT infrastructure using the OPC UA protocol.

8 Access to databases

The access to databases can be done from a cab printer via JScript and SQL interface. The printer connects to a Windows service, the cab Database Connector. The connection is made via the database language SQL, in a language range reduced to this application.

For JScript itself the actual database remains invisible. The printer always speaks only with the Database Connector. Whether this interacts with a MySQL database server or a simple Excel file is not visible to the printer.

Besides accessing a database via the cab Database Connector the printer can also access a local file (stored on the printer) and act as database server itself. This is possible for the A, A^+ and $Hermes^+$ devices with a dBASE file and for all newer printers based on the SQUIX with a SQLite file besides dBASE.

This requires either a suitable dBASE or SQLite file in the subdirectory *misc/*, or a Windows PC that can be reached from the printer via an IP address and has access to the database. If the cab Database Connector is used, one service for any number of printers is sufficient.

The database is accessed from the cab Database Connector via the ODBC or OLEDB[17] interface. Therefore any database can be used which has a suitable ODBC or OLEDB interface. The cab Database Connector in turn uses a port to accept TCP connections to talk to the printers (usually port 1001).

8.1 Connecting to the cab Windows service

We need to know the IP address of the machine running cab Database Connector. Then we need the port on which the service listens and receives the SQL requests. In JScript we then use the following command line to connect to the database (routed through the service).

```
E SQL;192.168.10.34:1001
```

From now on, the printer sends all database queries to the IP address 192.168.10.34 on port 1001 (of course you will have to enter your address values here if you want to try the examples). There the cab Database Connector should be installed and running. How to install the cab Database Connector can be read in the programming manual (Chapter 7: cab Database Connector).

The software cab Database Connector and also the programming manual can be downloaded from the cab website.

`https://www.cab.de/en/programming`

8.2 Stand alone sulution: using a local SQLite file

In addition to a connection to the Database Connector, a standalone solution can also be implemented. For this purpose a database file compatible with *SQLite 3* has to be created and copied into the directory *misc/*. The printer then works with this SQLite file (possible since firmware 5.25).

We only have to tell the printer the file using the *E* command:

```
E SQLITE;filename
```

[17] OLEDB is no longer maintained by Microsoft and should not be used anymore. Windows, ODBC and OLEDB are registered trademarks of Microsoft Corporation.

If you omit the file extension, *.sqlite3* is assumed. However, *.db* or *.db3* are also possible, in which case the extension must also be specified.

The example *12.2 Print vouchers* on page 93 shows the use of a SQLite file stored locally on the printer.

Engaging the database connection (to a server or a local file) with the *E* command must always be done after the *J* command, otherwise an error message will appear on the printer.

8.3 Get a value from a database

For the query to a database there is the special function **[SQL:...]**. If it is used in a text or barcode content, the response of the database is used as return value.

```
1 m m
2 J
3 S l1; 0, 0, 68, 71, 100
4 H 200
5 E SQL;192.168.10.34:1001
6 T:ARTNR; 10,  5, 0, 3, 5;[?:Article number?,5560432,1,R,D]
7 T        10, 15, 0, 3, 5;[SQL:SELECT PRODNAME FROM TA WHERE ARTICLE='{ARTNR}']
8 A 1
```

Listing 8.1: Get a string from a database DBSearch.lbl

Within the special function, curly brackets can be used to insert the content of another JScript element (text or barcode), so to speak a reference within an SQL call. In this example, the user is first asked for an article number (line 6). Then this input is used in the following line. The cab Database Connector then receives this request:

```
SELECT PRODNAME FROM TA WHERE ARTICLE='5560432'
```

Assuming that the user has accepted the default by simply confirming.

8.4 Splitting the database response

The response is a single string, regardless of whether the database server returns one or more responses. If the response consists of multiple elements (strings), they are arranged one after the other and separated with a *GS* ASCII character (Group Separator, decimal *29*, hexadecimal $1D_{16}$).

The special function **[SPLIT:*name*,*pos*]** is available for processing in JScript. It can be used to refer to a content of a single JScript line and *pos* is used to specify the position within the content defined by the *GS* character separated string response from the database.

The position is counted from 1. If the referenced content contains no *GS* character, so if it consists of only one element, then [SPLIT:*name*,1] returns the same result as [*name*].

A more extensive example shows listing 8.2.

```
m m
J
S l1; 0, 0, 68, 71, 100
H 100, 0
O R

; definition of the IP and port address where the cab service is installed
E SQL;192.168.10.34:1001

; user entry via touchscreen
T:INPUT; 0, 0, 0, 3, 3;[?:Artikelnr.:, , , L7, R, D][I]

; Database Connector query (SQL Statement)
; (query the database about all fields of the table 'article'
;  where 'artnr' was equal to the JScript variable 'INPUT')
T:RESULT; 0, 0, 0, 3, 3;[SQL:SELECT * FROM article WHERE artnr='{INPUT}'][I]

; split the result from the database into
; multiple text elements onto the label
T:RES1; 30,  5, 0, 5, pt11;[SPLIT:RESULT, 1][I]
T:RES2; 30, 10, 0, 5, pt11;[SPLIT:RESULT, 2]
T:RES3; 30, 15, 0, 5, pt11;[SPLIT:RESULT, 3]
T:RES4; 30, 20, 0, 5, pt11;[SPLIT:RESULT, 4]
T:RES5; 30, 25, 0, 5, pt11;[SPLIT:RESULT, 5]
B       12, 30, 0, 2OF5INTERLEAVED, 25, 1, 15;[RES2]

; fixed text elements on the label:
T        0, 10, 0, 5, pt11;[J:r26]Artikelnr.:
T        0, 15, 0, 5, pt11;[J:r26]Beschreibung:
T        0, 20, 0, 5, pt11;[J:r26]Beschreibung:
T        0, 25, 0, 5, pt11;[J:r26]Einheit:

; Add an entry into the log table containing
; date, time and article number
T:DAT;   0,  0, 0, 3, 3;[DATE][I]
T:TIM;   0,  0, 0, 3, 3;[TIME][I]
T        0,  0, 0, 3, 3;[SQL:INSERT INTO log VALUES ('{DAT}','{TIM}','{RES2}')][I]

; query how many labels the user wants to print
A [?]
```

Listing 8.2: Splitting a database response DB-SPLIT.lbl

8.5 Write back to a database

Since the database language SQL does not only know queries, the special function **[SQL:...]** can also be used to write back to a database. The previous example shows this in line 37, where with the *INSERT INTO* command three values can be inserted into a table. The line itself is implemented in JScript as invisible text.

Besides the *SQL* special function, there is also the *SQLLOG* special function. It is only executed when the respective label has been completely processed (printed and if necessary dispensed). The procedure works in analogy to the *WINF* write command into the info memory. In the previous example (listing 8.2), line 37 could also look like this to create the database entry only at the time of printing.

```
37 T        0,  0, 0, 3, 3;[SQLLOG:INSERT INTO log VALUES ('{DAT}','{TIM}','{RES2}')][I]
```

Listing 8.3: If SQLLOG is used to write to a database, the write operation is only performed after complete printing and dispensing (similar to WINF).   DB-SPLIT-LOG.lbl

9 More than just printing

The cab printers allow via options (additional hardware mounted in front of the printer) different operation modes like cutting, dispensing or printing and rewinding. For these options there are capital letters in JScript to configure the respective option.

The commands to use (hardware) options must follow the command *S* (label size), otherwise an error message will appear on the printer.

9.1 Dispensing labels

To dispense a label you need two things. There must be a dispensing edge, combined either with a dispensing photoelectric sensor, an I/O interface or a label applicator. In addition, a label must be pushed forward after printing so that it can be removed by the user (or label applicator).

The dispensing mode is set with the command *P* in JScript. The command can be assigned an optional parameter, the dispensing offset, i.e. the distance in millimeters or inches by which the label is moved beyond the dispensing position. Positive and negative values are possible. If the values are positive, the label strip is transported further out of the printer. The default value 0.0 mm positions the rear edge of the label on the dispensing edge. Typical for the dispensing offset would be half the label distance (half the length of the gap).

```
P 1.5
```

In this example the dispensing distance is set to 1.5 mm and the dispensing mode is activated. After reaching the dispensing position, the labels are pushed forward by +1.5 mm. Afterwards the printing of the next label is waited until the removal of the label is confirmed.

9.2 Tear-off mode

When dispensing, the label is only moved forward in combination with a dispensing light barrier. Without such a light barrier there is a similar option, the tear-off mode. This mode is activated by the *T* parameter of the option command (*Tear-Off Mode*).

```
m m
J
S l1; 0, 0, 68, 71, 100
H 150, 0
O R, T2.5
T 12, 25, 0, 3, 6;Hallo Welt
A 1
```

Listing 9.1: With the T option, the printer switches to tear-off mode TearOff.lbl

The fifth line (O R,T2.5) activates the tear-off mode in addition to the 180° rotation.

The specification of how far beyond the tear-off position must be shifted can be defined directly after the letter “T” as an option argument. It is added to the parameter defined directly in the printer. Therefore, the value 0 should always be aimed for in the printer and only a slightly different value should be set in the printer menu to compensate for differences between several printers. If both values are set to 0.0 mm, the position is approached in such a way that the original accessory (toothed tear-off edge) matching the respective printer is positioned at the rear edge of the label.

The transport to the tear-off edge always takes place after the last label. When further labels are printed again, a retraction is first triggered and the front edge of the next label is positioned under the printing bar again. Then all new labels to be printed are printed without interruption and moved forward to the tear-off position.

9.3 Automatic retraction

If a label has been dispensed, the beginning of the next label is no longer under the print head, but has already passed it. To print the next label completely, the printer must retract the label. To do this, the counterpressure roller is turned backwards until the start of the next label is under the print head.

However, this does not necessarily have to be done in this way. The printer can also start printing the next label and print exactly until the dispensing position is reached. If the previous label is removed, the interrupted label is printed to the end. This usually results in a hair-thin white (unprinted) cross line on the label due to slippage.

If this is to be prevented, set the Back Transport parameter in the printer menu from *optimized* to *always*. In JScript, the option command parameter **D** can be used.

```
O R, D
```

This line would also force a permanent retraction (including a 180° rotation). If you want to force an optimized retraction instead, use the **P** parameter in the Options command.

Note that firmware 5.35 (published October, 27th 2020) comes with a new method of optimised printing to eliminate the hair-thin white cross line. So instead of turning off the optimisation, updating the firmware may be a better solution for you.

9.4 Ribbonsaver

If you have a printer with automatic saving, you can enable or disable the use of this option. To do this, the additional argument **R1** or **R0** is appended to the H command. The argument **R1** ("ribbon saver on") enables the use of automatic saving. The following line, however, would disable the automatic saver (if installed):

```
H 150, 0, R0
```

Where the first value sets the print speed to 150 mm/s, the second value adds a heat of 0 to the printer's internal value and the `R0` disables the saving function, if the function is not forced in the printer menu. In the settings in the printer you have the possibility to choose between "JScript" and "on". If you have chosen "on", a parameter **R0** cannot prevent the automatic saving activity. The printer setting has (exceptionally) priority over the JScript code here.

9.5 Cutting Labels

You can also use a cutter to cut off a label. The command is **C** followed by a letter that specifies the cut type or a number. The following commands are possible:

Table 9.1: The cut command *C* with its options

Command	Option
C *Integer*	Cut after defined among of labels
C e	Cut after the job's last label
C s	Cut at the job's start
C s,*Number*	Cut at job start after *number* units
C p	Perforation cut instead of cutting through, after each label
C p,*Zahl*	Perforation cut after *number* units of measurement
C sp	Start the job with a perforation cut

The position of the cut at the end of the label and an additional cut position can be selected as optional parameters. Only positive values are allowed for the further cutting position.

The additional value parameters to get an offset of the cut cannot be mixed. Therefore, you cannot mix an offset perforation with a cut through.

Probably the most common application is perforating on a shrink tube with final cutting at the end of the job.

```
m m
J

; here are the elements like label size, options, texts, barcodes, etc.
...

C p
C e
A 5
```

Listing 9.2: Using the cutting command in a label made for shrink tubes. You will get a perforation cut between the single labels and a fully cut through after the last label.

The example in listing 9.2 would create five markings (*labels*) on the heat shrink tube and then cut the tube. The markings are separated from each other by a perforation.

The use of a perforation knife allows both perforating and cutting, but is limited in the maximum cut width depending on the material to be cut. Talk to us in advance if you want to perforate very large materials.

9.6 Using an Applicator

An applicator can be attached to an A, A$^+$, SQUIX, Hermes A, Hermes$^+$ or HERMES Q. The printer offers additional parameters in the settings menu when an applicator is detected.[18]

[18]Printer and applicator communicate via the Sub-D connector on the front of the printer. The printer is able to recognize that an applicator is attached, but it cannot recognize which one it is.

9.6.1 Set parameters for an applicator

Parameters can be passed to the applicator with the JScript print data. This is done with the option command *O Ax=y*.

With the *O* command, several options can be spread over several command lines, or all options can be defined on a single command line and separated by a comma.

Table 9.2: Set parameters for an applicator

Command	Parameter [ms]
O A0=*n*	Start support air *n* milliseconds delayed
O A1=*n*	Stop support air *n* milliseconds delayed
O A2=*n*	Set print delay to *n* milliseconds
O A3=*n*	Set blocking time (suppress new start signals) to *n* milliseconds
O A4=*n*	Set blowing time (support air duration) to *n* milliseconds

9.6.2 Print and apply or apply and print?

In the printer settings, the mode can be selected between *Print and Apply* (default value) and *Apply and Print*.

Normally a label is printed and applied directly if the *A* command was given. If an applicator is connected to e.g. the Hermes A or Hermes$^+$, the digital input *START* must be triggered to start printing (apply +24 V between pin 13 and pin 25 at the I/O interface).

In *Apply and Print* mode, however, the applicator is “hold” in the waiting position and applies the held label while triggering the *START* signal. After the label is applied, a new label is automatically printed and brought into the waiting position. This procedure allows a faster response to the signal from a production process.

However, after loading a job, no label is yet in the waiting position, so the *START* signal does not trigger any action. It is mandatory to send the *FSTLBL* signal to the printer first (connect +24 V between pin 1 and pin 25 at the I/O interface).

The I/O interface can not only be wired directly with 24 V voltage, it is also possible to control it via TCP/IP networks. Send as START signal the escape command Esc *g* (hexadecimal *1B 67*$_{16}$). In *Apply and Print* mode, you must send the command Esc *xin FSTLBL;* once (hexadecimal *1B 78 69 6E 20 46 53 54 4C 42 4C 3B*$_{16}$).

10 Diagnostic options

The SQUIX offers several ways to log the data stream sent to the printer. On the one hand there is the possibility to print the received data directly in plain text. This possibility exists since a very long time, old devices from the A-series already master this.

This method is called *Monitor Mode*. It reaches its limits when interaction with the printer is required. In *Monitor Mode*, the printer does not process the incoming data stream, but only produces a printout. The width can be set in 5 mm steps from 50 mm to the maximum print width. The printer prints line by line and ignores the label detection. As with all diagnostic functions that do not take the label dimensions into account during printing, you can also turn the material around in monitor mode and print on the back of the carrier strip if no suitable continuous material is available.

A new feature of the SQUIX series is the possibility to write the data to a file, which can later be loaded from the printer via FTP or FTPS (FTP with TLS) connection or you can have the printer write directly to an SD card or USB memory.

10.1 Monitor Mode

Monitor mode can be turned on at the printer in the *Diagnostics* menu. If it is active, this is indicated in the printer display (Figure 10.1). Press the *Cancel* key to exit the monitor mode.

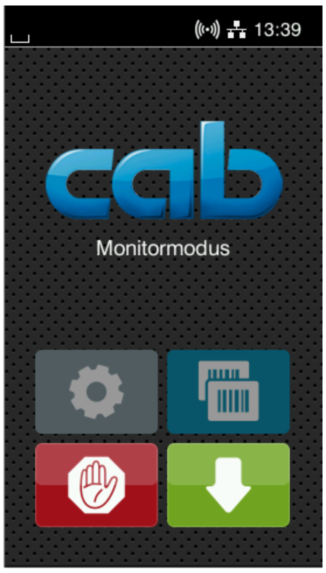


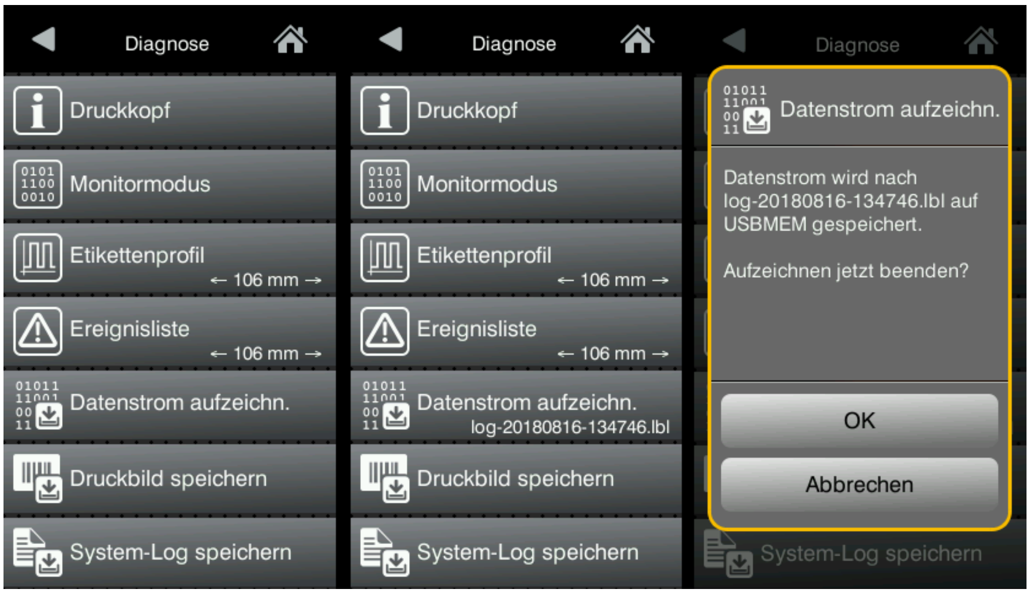
Figure 10.1: Monitor Mode was activated

10.2 Log incoming data into a file

But it is also possible to write a file directly.

To do this, press the *Record Data Stream* item in the Diagnostic Menu.

The figure 10.2 shows three screenshots of the SQUIX. In the Diagnostics menu you can start the recording by pressing *Record Data Stream*. A dialog follows in which you can specify name and location. By default, a file name is generated from the current date and time.



Datenstrom wird nach log-20180816-134746.lbl auf USBMEM gespeichert.

Aufzeichnen jetzt beenden?

Figure 10.2: A SQUIX printer allows to record all incoming data into a file direcly on an USB storage device

The file name is displayed to indicate that recording is in progress. Pressing the button again will stop the recording (with a query if you really want to stop).

While recording, the status bar on the main screen shows a circle with a dot flashing in the middle (similar to an audio or video recorder).

The recorded file contains the pure input data stream without any additions (like timestamps).

A missing line end after an *A* command can be quickly exposed both with the monitor mode and via data stream recording.

10.3 Preview a label without printing

With the following command you can force the printer to create an internal bitmap of the label, but inhibit to print it.

```
A [PREVIEW]
```

To show the bitmap of the last printed label (or forced preview), use this link in your browser:

https://192.168.10.1/cgi-bin/bitmap

If a user name and password are requested, use *admin* as the user name and the website password specified in the *Security* menu. The initial factory default for the password is *admin*. Use the IP address of your printer as the IP address. Starting with the SQUIX model series, you have the choice to force incoming connections via HTTP (unencrypted) or HTTPS (encrypted). Older printers can only communicate unencrypted.

11 Barcodes in global trade

In global trade, a number of barcode types have quickly established themselves which clearly classify goods. For example, the EAN-13 code, the *European Article Number*. It consists of 13 digits, 12 of which identify the article and the last digit acts as a check digit.

In addition to this code, there are similar codes like the ISBN for books and magazines. There is also a check digit, the value that can be uniquely calculated from the previous digits.

11.1 Data integraty for barcodes

Barcodes in general are a symbolic representation of content (numbers, character strings, etc.), which can be captured particularly well by machine. E.g. with bar code readers based on a laser beam for one-dimensional codes or sufficiently high-resolution cameras for two-dimensional codes.

When reading by machine, errors are to be ruled out as far as possible. Two methods are used for this purpose. On the one hand, the correctness of data can be checked using check digits. Depending on the method used to calculate these check digits, errors can be detected almost 100 %. However, the error detection can only confirm a read data set. If the calculated and read check digits do not match, the actual content cannot be restored. One can only discard the reading that was recognized as incorrect.

Therefore, there is not only the **error detection** but also the **error correction**. Because the correction data is already integrated during the compilation without knowing whether the reader has to restore the data later or not, this procedure is also called *FEC*, which means *forward error correction*. This means that more data is displayed in the barcode than actually needs to be transported. Later you can read the data, perform a check, and if the check is successful, you can process the data directly as checked values.

If the data cannot be verified as correct, an error correction procedure offers the possibility to restore the original content of the data. Depending on the method, this is possible to a considerable extent, but requires increased computing power.

11.1.1 Error detection by a check digit shown on the GTIN

There are some data that already have a built-in authenticity check. For example, a check digit is built into the GTIN, the *Global Trade Item Number*. The complete GTIN number contains an additional digit that does not contribute to the content and could actually be omitted. However, it provides confirmation that this sequence of digits follows a certain rule and can therefore be considered correct.

The GTIN-13 has 12 payload digits followed by a check digit. The first 12 digits clearly identify a product. For example, at a supermarket checkout, the price and product name can be determined with the help of a database after the checkout system has read the GTIN from a printed barcode by a laser scanner.

You could simply add up the sum of the single digits and calculate the difference to the next multiple of 10 (as a result, we only want to get a single digit, so the code standardized by GS1 uses this difference). The result would then be the check digit.

So if you had to form the GTIN for the sequence of digits 123456789012, you would do the calculation:

$$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 0 + 1 + 2 = 48$$

$$50 - 48 = 2$$

The check digit would thus be 2 and the total GTIN would be 123456789012**2**.

Why so complicated? Why not just use the last digit of the sum of all useful digits as check digit? The answer lies hidden in practical application. If we calculate the check digit as the difference to the next multiple of 10, then the sum of *all* digits including the check digit is always a multiple of 10. This makes it much easier for the scanner at the supermarket checkout to check whether the reading is valid. How much work we put into creating the label is not important, the process at the checkout must be fast.

In our example, the scanner calculates like this:

$$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 0 + 1 + 2 + 2 = 50$$
$$50 = 5 \cdot 10 + 0 \quad \mapsto \quad \text{good read}$$

If you now take any digit and replace it by another digit, e.g. the fifth digit by a 7, the reading result 1234**7**67890122 would not be conform to the rules any more. Because the sum of all digits now results in 52.

But it is a little more complicated. If the barcode cannot be read by the scanner, a human would have to read the GTIN from a plain text imprint and enter it on a keyboard. This can lead to an interchanging of two neighboured digits, which happens more often with a manual input than typing a wrong digit. However, the GTIN 123**54**6789012**2** would be valid according to the rule we have set up.

Therefore, starting from the back (with the check digit), each digit is multiplied alternately by 1 and by 3 before the sum is formed. This sum must be a multiple of 10.

The GTIN check digit for the useful digits 123456789012 is actually calculated in this way:

$$1 + 3 + 5 + 7 + 9 + 1 = 26$$
$$2 + 4 + 6 + 8 + 0 + 2 = 22$$
$$26 + 3 \cdot 22 = 26 + 66 = 92$$
$$100 - 92 = 8$$

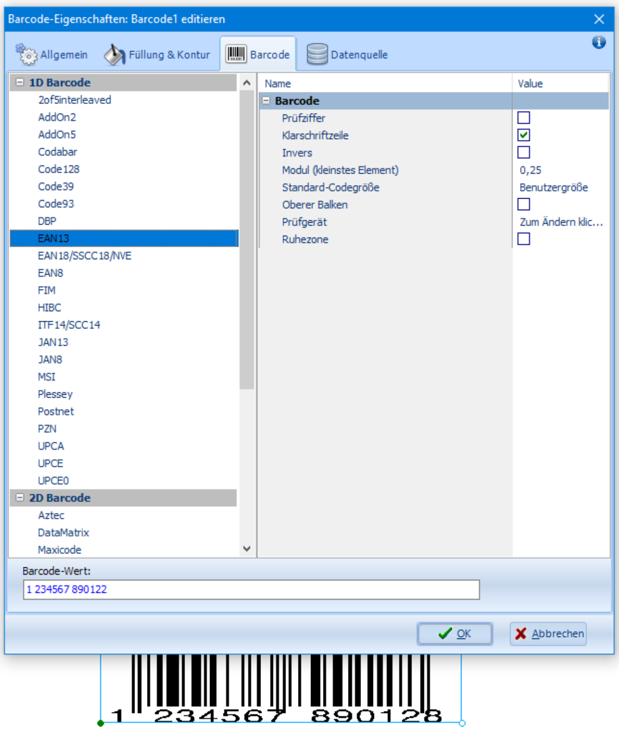
A final check will result in this (we multiply each digit alternatively with 1 and 3, starting from the last digit):

$$8 \cdot 1 + 2 \cdot 3 + 1 \cdot 1 + 0 \cdot 3 + 9 \cdot 1 + 8 \cdot 3 + 7 \cdot 1 + 6 \cdot 3 + 5 \cdot 1 + 4 \cdot 3 + 3 \cdot 1 + 2 \cdot 3 + 1 \cdot 1 = 100 \quad \mapsto \quad \text{good read}$$

The actual GTIN is 1234567890128. 123**54**6789012**8** would no longer conform to the rules.

$$8 \cdot 1 + 2 \cdot 3 + 1 \cdot 1 + 0 \cdot 3 + 9 \cdot 1 + 8 \cdot 3 + 7 \cdot 1 + 6 \cdot 3 + 4 \cdot 1 + 5 \cdot 3 + 3 \cdot 1 + 2 \cdot 3 + 1 \cdot 1 = 102 \quad \mapsto \quad \text{bad read}$$

There are other check digits within the data defined by the GS1 organization. For example, certain GTINs contain a weight specification in addition to the identification of the goods. This allows you to offer goods in bulk and to charge them by weight at the supermarket checkout. The procedures for these additional check digits are much more complex and are defined in the GS1 organization's manual GS1 General Specifications.


Barcode-Eigenschaften: Barcode1 editieren

Allgemein | Füllung & Kontur | Barcode | Datenquelle

1D Barcode
- 2of5interleaved
- AddOn2
- AddOn5
- Codabar
- Code128
- Code39
- Code93
- DBP
- EAN13
- EAN18/SSCC18/NVE
- EAN8
- FIM
- HIBC
- ITF14/SSCC14
- JAN13
- JAN8
- MSI
- Plessey
- Postnet
- PZN
- UPCA
- UPCE
- UPCE0

2D Barcode
- Aztec
- DataMatrix
- Maxicode

Name	Value
Barcode	
Prüfziffer	☐
Klarschriftzeile	✓
Invers	☐
Modul (kleinstes Element)	0,25
Standard-Codegröße	Benutzergröße
Oberer Balken	☐
Prüfgerät	Zum Ändern klic...
Ruhezone	☐

Barcode-Wert:
1 234567 890122

OK | Abbrechen

1 234567 890128

Figure 11.1: Both, the printer and the cabLabel S3 software are calculating the check digit. If the digit the user entered is wrong, the software will replace it automatically by the correct digit.

11.1.2 Error correction in QR and Datamatrix codes with the Reed-Solomon method

The two mathematicians Irving Stoy Reed and Gustave Solomon from the MIT Lincoln Laboratory, a research facility of the United States Department of Defense, developed a data structure in the 1960s that allows data to be reliably identified as faulty and even clearly reproduced again in the event of the loss of individual parts of the data or corruption (e.g. the mixing up of digits).

The Reed-Solomon code is used by the Datamatrix in the form *ECC200* and allows to correct up to 25 % of destroyed or unreadable parts. *ECC200* means *error correction code* and describes the procedure and the amount of redundant information in the code. There exist a couple of different levels of error correction in Datamatrix. But today's Datamatrix always uses the *ECC200* correction code. This does not have to be specified in the JScript, the older procedures *ECC00* to *ECC140* are not supported.

The situation is different with the QR Code. The company Denso Wave, which invented the code in 1994 and released it for public use, defines different levels of redundancy. In JScript, for example, you have the possibility to append the options *+ELx* to the type identifier *QR-Code*, where the *x* indicates the level of redundancy (Error Correction Level).

In the 4.7 listing on page 38 the error correction level was not specified. The printer has chosen level 1 by itself. You can recognize this by the fact that both bits are set in the format definition. This way the code takes up the least space. If a higher level is to be printed, the corresponding barcode option must be used.

Table 11.1: Error Correction Levels of a QR Code

Code type including option	Alternative spelling	Result
QR-CODE+ELL	QR-CODE+EL1	Level 1, able to reproduce 7 % of the original code.
QR-CODE+ELM	QR-CODE+EL2	Level 2, able to reproduce 15 % of the original code.
QR-CODE+ELQ	QR-CODE+EL3	Level 3, able to reproduce 25 % of the original code.
QR-CODE+ELH	QR-CODE+EL4	Level 4, able to reproduce 30 % of the original code.

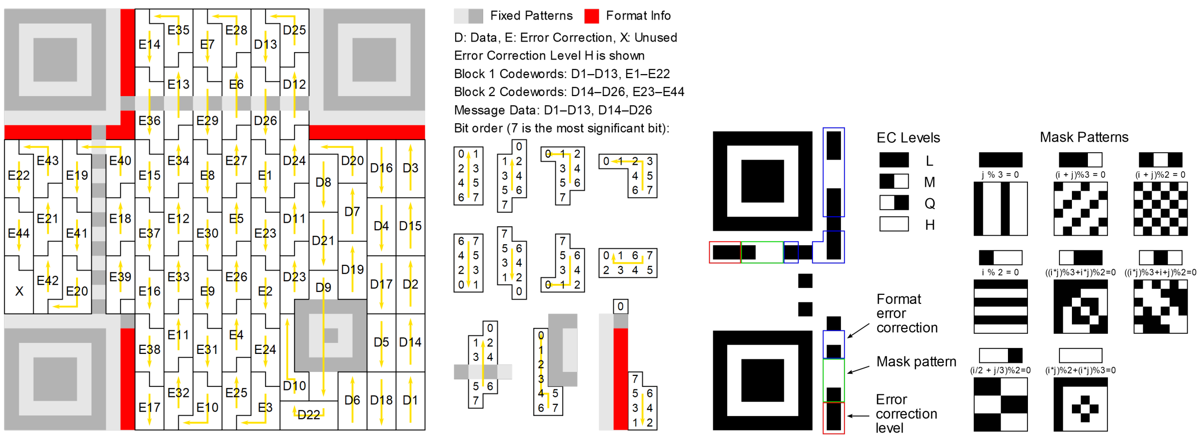


Figure 11.2: Basic structure of a QR code with error level and mask pattern. Besides the three large search patterns, which are connected to the two lines for timing, and the other smaller one, the code also contains the format information in addition to the user data and the correction parameters (source: Wikipedia).

11.2 GS1 Data Structure

The GS1 is a global organization that defines a special data structure and associated barcodes. The data itself is divided into so-called *Application Identifier*, also called *AI*. These AI are basically numbers that have the same meaning worldwide. The following table 11.2 lists some of these AI.

Table 11.2: AI starting with 0, 1 or 2 of the GS1 data structure

AI	Description	Format
00	Serial Shipping Container Code (SSCC)	N2+N18
01	Global Trade Item Number (GTIN)	N2+N14
02	GTIN of contained trade items	N2+N14
10	Batch or lot number	N2+X..20
11	Production date (YYMMDD)	N2+N6
12	Due date (YYMMDD)	N2+N6
13	Packaging date (YYMMDD)	N2+N6
15	Best before date (YYMMDD)	N2+N6
16	Sell by date (YYMMDD)	N2+N6
17	Expiration date (YYMMDD)	N2+N6
20	Internal product variant	N2+N2
21	Serial number	N2+X..20
22	Consumer product variant	N2+X..20
235	Third Party Controlled, Serialised Extension of GTIN (TPX)	N3+X..28
240	Additional product identification assigned by the manufacturer	N3+X..30
241	Customer part number	N3+X..30
242	Made-to-Order variation number	N3+N..6
243	Packaging component number	N3+X..20
250	Secondary serial number	N3+X..30
251	Reference to source entity	N3+X..30
253	Global Document Type Identifier (GDTI)	N3+N13+X..17
254	GLN extension component	N3+X..20
255	Global Coupon Number (GCN)	N3+N13+N..12

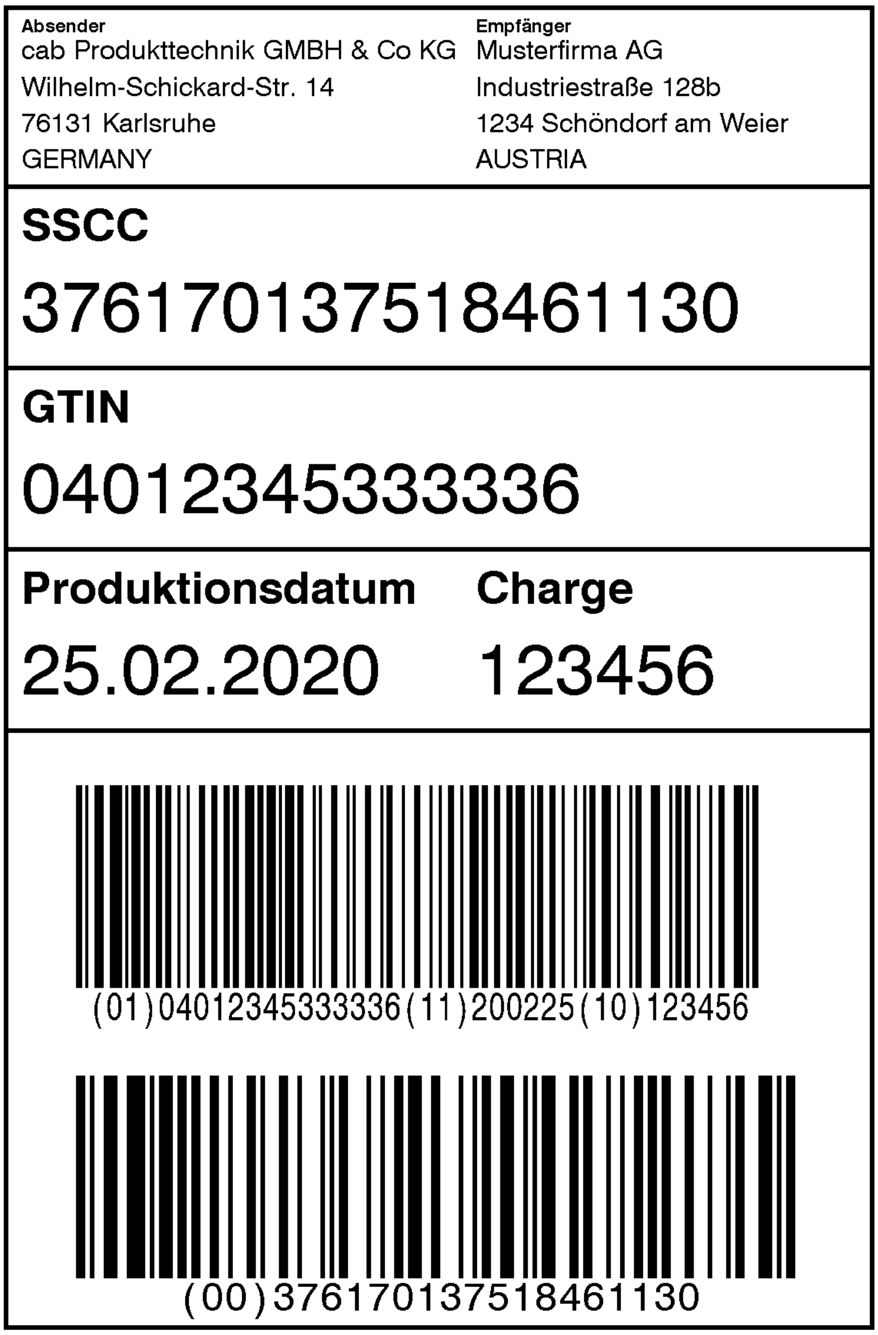
All AI can be found on the Website of the GS1. The format describes the required digits or characters. N2 means that you need exactly two digits. X..20 refers to a string of letters and/or numbers, which may have a maximum length of 20 characters. With N2+X..20 as defined for the serial number, for example, you can include up to 20 letters or digits in this information. The data set then consists of up to 23 characters. If a data record is flexible in length (and is not the last data record in a listing), it must be followed by a special character (usually an FNC1). A FNC1 special character is always followed by an AI.

Barcodes like the GS1-128 consist of several AI, which are listed one after the other in the barcode. If a human-readable plain text line is added to a code, the AI is placed in round brackets to increase readability. However, the code itself does not contain brackets. It consists only of the completely formed sequence of AI and their respective contents, in case of a flexible length it ends with a special character before a following AI.

If a GS1 barcode is defined in JScript, the data is also specified with the brackets, even if it is not contained in the actual code. See 11.1 for an example of a GS1-style shipping label.

11.2.1 Floating numbers as a content of an AI

There are AI, which have a decimal number as content, e.g. the AI 310x. The meaning of this AI is to indicate the net weight in the unit kilogram. The AI is formed in the form N4+N6, i.e. 4 digits are used for the AI itself and then always 6 digits for the value. **The last digit of the AI always indicates the number of decimal digits.**


Absender
cab Produkttechnik GMBH & Co KG
Wilhelm-Schickard-Str. 14
76131 Karlsruhe
GERMANY

Empfänger
Musterfirma AG
Industriestraße 128b
1234 Schöndorf am Weier
AUSTRIA

SSCC
376170137518461130

GTIN
04012345333336

Produktionsdatum
25.02.2020

Charge
123456

(01)04012345333336(11)200225(10)123456

(00)376170137518461130

Figure 11.3: A GS1 conform shipping label

If you want to display five kilograms net weight in a GS1 barcode, you can use the following line in JScript.

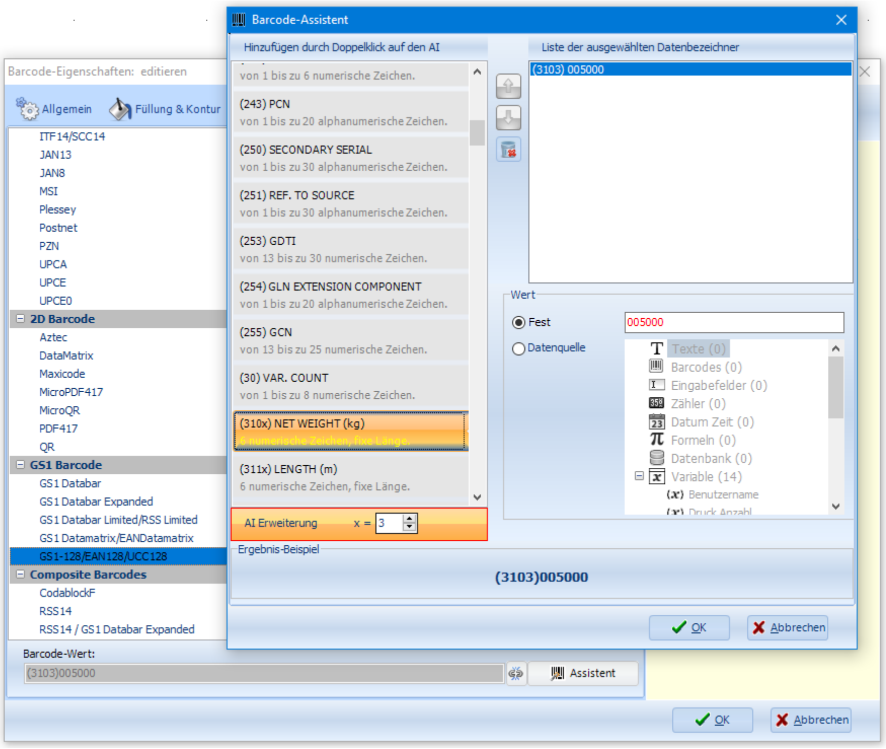
```
B 10, 30, 0, GS1-128, 20, 0.33;(3100)000005
```

Here the last digit of the AI is 0, so the coded value has no decimal digits. If you want to specify the value in grams, you could use the following line. Then 5 kg = 5000 g.

```
B 10, 30, 0, GS1-128, 20, 0.33;(3103)005000
```

Grams and kilograms are frequently used, but all specifications of the number of decimal digits up to the value of 5 are still acceptable. In our example, it would therefore also be possible to form the AI including data as follows:

(3100)000005 = (3101)000050 = (3102)0000500 = (3103)005000 = (3104)050000 = (3105)500000



von 1 bis zu 6 numerische Zeichen.
(243) PCN
von 1 bis zu 20 alphanumerische Zeichen.
(250) SECONDARY SERIAL
von 1 bis zu 30 alphanumerische Zeichen.
(251) REF. TO SOURCE
von 1 bis zu 30 alphanumerische Zeichen.
(253) GDTI
von 13 bis zu 30 numerische Zeichen.
(254) GLN EXTENSION COMPONENT
von 1 bis zu 20 alphanumerische Zeichen.
(255) GCN
von 13 bis zu 25 numerische Zeichen.
(30) VAR. COUNT
von 1 bis zu 8 numerische Zeichen.
(310x) NET WEIGHT (kg)
6 numerische Zeichen, fixe Länge.
(311x) LENGTH (m)
6 numerische Zeichen, fixe Länge.
AI Erweiterung x = 3
Ergebnis-Beispiel: (3103)005000
Liste der ausgewählten Datenbezeichner: (3103) 005000
Wert: Fest 005000

ITF14/SCC14, JAN13, JAN8, MSI, Plessey, Postnet, PZN, UPCA, UPCE, UPCE0
2D Barcode: Aztec, DataMatrix, Maxicode, MicroPDF417, MicroQR, PDF417, QR
GS1 Barcode: GS1 Databar, GS1 Databar Expanded, GS1 Databar Limited/RSS Limited, GS1 Datamatrix/EANDatamatrix, GS1-128/EAN128/UCC128
Composite Barcodes: CodablockF, RSS14, RSS14 / GS1 Databar Expanded
Barcode-Wert: (3103)005000

Figure 11.4: In cabLabel S3 Pro you get assistance by a software tool. In this dialog window you will have to enter the number of digits of some AIs in a seperate field.

```
m m
J
S l1; 0, 0, 150, 153, 100
H 150, 0

; create the variable fields
T:GTIN;   0, 0, 0, 3, 3;04012345333336[I]
T:SSCC;   0, 0, 0, 3, 3;376170137518461130[I]
T:PDATE;  0, 0, 0, 3, 3;[YY][MONTH02][DAY02][I]
T:Charge; 0, 0, 0, 3, 3;123456[I]

; draw a rectangle and some lines onto the label
G 2,  2, 0;R:96, 146, 0.5, 0.5
G 2, 22, 0;L:96, 0.5
G 2, 42, 0;L:96, 0.5
G 2, 62, 0;L:96, 0.5
G 2, 82, 0;L:96, 0.5

; sender address
T 4,  5, 0, 5, 2;Absender
T 4,  8, 0, 3, 3;cab Produkttechnik GMBH & Co KG
T 4, 12, 0, 3, 3;Wilhelm-Schickard-Str. 14
T 4, 16, 0, 3, 3;76131 Karlsruhe
T 4, 20, 0, 3, 3;GERMANY

; receiver address
T           54,  5, 0, 5, 2;Empfänger
T:ToLine1; 54,  8, 0, 3, 3;Musterfirma AG
T:ToLine2; 54, 12, 0, 3, 3;Industriestraße 128b
T:ToLine3; 54, 16, 0, 3, 3;1234 Schöndorf am Weier
T:ToLine4; 54, 20, 0, 3, 3;AUSTRIA

; NVE plain text
T 4, 28, 0, 5, 5;SSCC
T 4, 38, 0, 3, 8;[SSCC]

; GTIN plain text
T 4, 48, 0, 5, 5;GTIN
T 4, 58, 0, 3, 8;[GTIN]

; production date and batch no. as plain text
T 4,  68, 0, 5, 5;Produktionsdatum
T 4,  78, 0, 3, 8;[PDATE,5,2].[PDATE,3,2].20[PDATE,1,2]
T 54, 68, 0, 5, 5;Charge
T 54, 78, 0, 3, 8;[Charge]

; barcode for the AI 01, 10 and 11
; To avoid the need for an additional FNC1, the AI10 is placed at the end.
B 10,  88, 0, GS1-128, 26, 0.3;(01)[GTIN](11)[PDATE](10)[Charge]

; barcode for the SSCC
B 10, 120, 0, GS1-128, 26, 0.5;(00)[SSCC]

A [PREVIEW]
```

Listing 11.1: A GS1 conform shipping label GS1Versandetikett.lbl

11.3 Barcode types for GS1 data

The GS1 usually does not use a new development, but uses already existing barcode types. These barcode types are filled with data according to precisely defined rules in order to distinguish a GS1 barcode from its original form. In most of the used codes you simply start with the special character *FNC1* and let the sequence of AI numbers and AI contents follow. If an AI has a variable length, a special character *FNC1* is inserted at the end of the user data to indicate that the AI's data ends here and a new AI begins after the special character. This method can be used to convert Datamatrix, QR Codes and Code-128 barcodes to GS1 compliant data carriers.

However, the method is not a legal standard, but a non-proprietary regulation of the non-profit organization GS1. It would not be punishable by law, but it would be unwise to use a *FNC1* special character in the same way when using a Datamatrix, QR Code or Code-128 barcode, because it can be assumed that all scanners in the world will master the GS1 rules and identify the code as a GS1 code.

However, there are also barcode types that differ from this basic structure. So you can form the barcode types EAN8 and EAN13. They always represent the GTIN or the shorter GTIN8. This is a separate barcode that does not contain the special character *FNC1* or the AI identifier (01) for the GTIN.

12 Best practice examples

In this chapter some more complex examples will be explained, as they are more common in practice. Even if you are not familiar with the concrete task of the respective examples, it is still worth to take a closer look into the code, because it shows the systematic procedure for individual solutions with the cab printer language JScript.

The examples are simplified and focus on the detail to be demonstrated. The label dimensions of 68 x 100 mm correspond to the "training label" as it is inserted on the training printers at the cab training center in Karlsruhe/Germany. In practice there would be more elements like eg. logos on the labels.

12.1 Single digit month, shift identification and daily counter reset

In this example we will create a single barcode containing a searial number. The serial number is constructed as:

1. Year of production (two digits)
2. a one digit month code (where October to December are uppercase letters O, N and D)
3. day of month (two digits)
4. a single uppercase letter as a shift code (F, S or N)
5. a five digit number starting with 00001 each new day

Listing 12.1 on page 92 will show the example.

It gets interesting from line 9 on. A special function *[MONTH1]* is simulated, which generates a **one-digit month code**. First the string "123456789OND" is printed invisibly as *AlleMonate*. Then the month is determined with the special function [MONTH] and saved as the content of *AktuellerMonat*.[19] The value of the current month is now used as index to the previously defined string to generate the one-character code.

The **shift code** looks a bit more complex. Let's assume that the early shift starts at 6:00, the late shift takes over at 14:00 and the night shift works from 22:00 until the next early shift starts. The shift abbreviations are defined in line 15 (F, S and N).

Then the current time is determined in line 16. Since we use the options *S* and *J* (single buffer and demand button, see line 7), the calculation only takes place when the trigger is pressed (symbol on the printer display or external button). Three conditions now check if the current time is greater than the start time of the shifts. There is no *[>=:...]* special function, so we check if the time is greater than one minute before start.

Each check returns 1 (numeric one) or 0 (numeric zero), depending on whether the condition is true or false. Line 20 adds all checks and adds 1 more. We have to do this because we cannot use 0 as index, but all three conditions could be false (e.g. at 3:47 in the morning).

[19] Saving to a variable is equivalent to invisible printing with a field name as reference. A "variable command" does not exist in JScript, instead, **T:*variable name*;0,0,0,3,3;*variable content*[I]** can be used to simulate assigning a value to a variable.

```
 1 m m
 2 J
 3 S l1; 0, 0, 68, 71, 100
 4 H 150, 0
 5
 6 ; options: print on demand, single buffer, 180° rotation
 7 O J, S, R
 8
 9 ; single digit month
10 T:AlleMonate;     0, 0, 0, 3, 3;123456789OND[I]
11 T:AktuellerMonat; 0, 0, 0, 3, 3;[MONTH][I]
12 T:MONTH1;         0, 0, 0, 3, 3;[AlleMonate,AktuellerMonat,1][I]
13
14 ; shift code (F from 6:00, S from 14:00, N from 22:00)
15 T:SchichtKuerzel; 0, 0, 0, 3, 3;NFSN[I]
16 T:AktuelleZeit;   0, 0, 0, 3, 3;[H24][MIN][I]
17 T:SC1;            0, 0, 0, 3, 3;[>:AktuelleZeit,559][I]
18 T:SC2;            0, 0, 0, 3, 3;[>:AktuelleZeit,1359][I]
19 T:SC3;            0, 0, 0, 3, 3;[>:AktuelleZeit,2159][I]
20 T:SC;             0, 0, 0, 3, 3;[+:SC1,SC2,SC3,1][I]
21 T:Schichtcode;    0, 0, 0, 3, 3;[SchichtKuerzel,SC,1][I]
22
23 ; serial number with storage in user memory and reset at the beginning of the day
24 T:Nutzerspeicher;      0, 0, 0, 3, 3;[RUSER][I]
25 T:AlteSN;              0, 0, 0, 3, 3;[Nutzerspeicher,6][I]
26 T:AlterTag;            0, 0, 0, 3, 3;[Nutzerspeicher,1,5][I]
27 T:NeuerTag;            0, 0, 0, 3, 3;[YY][DOFY][I]
28 T:Ruecksetzung;        0, 0, 0, 3, 3;[=:NeuerTag,AlterTag][I]
29 T:NeueSN;              0, 0, 0, 3, 3;[*:AlteSN,Ruecksetzung][D:5,0][C:0][I]
30 T:Seriennummer;        0, 0, 0, 3, 3;[+:NeueSN,1][D:5,0][C:0][I]
31 T:NeuerNutzerspeicher; 0, 0, 0, 3, 3;[YY][DOFY][Seriennummer][WUSER][I]
32
33 ; barcode centred on the label
34 B 0, 15, 0, CODE128, 20, 0.5;[YY][MONTH1][DAY02][Schichtcode] [Seriennummer][J:c100]
35
36 ; endless printing (on demand, see line no. 7)
37 A
```

Listing 12.1: With some extra work, a complex serial number can be generated, which is reset daily Seriennummer.lbl

If the time is eg. exactly 14:00:11 (2 p.m. plus 11 seconds), then the calculation in line 16 to 21 will result in:

JScript Code	Calculation	Result
T:**AktuelleZeit**; 0, 0, 0, 3, 3;**[H24][MIN]**[I]		1400
T:**SC1**; 0, 0, 0, 3, 3;**[>:AktuelleZeit,559]**[I]	1400 > 559	1
T:**SC2**; 0, 0, 0, 3, 3;**[>:AktuelleZeit,1359]**[I]	1400 > 1359	1
T:**SC3**; 0, 0, 0, 3, 3;**[>:AktuelleZeit,2159]**[I]	1400 > 2159	0
T:**SC**; 0, 0, 0, 3, 3;**[+:SC1,SC2,SC3,1]**[I]	1 + 1 + 0 + 1	3
T:**Schichtcode**; 0, 0, 0, 3, 3;**[SchichtKuerzel,SC,1]**[I]	N F **S** N	S

The barcode should represent a unique serial number, but the counter is only five digits long. However, when counting up ten thousand, we would have the problem that the code would have one more digit in total. This is because the formatting [D:5.0][C:0] fills up to five digits, but does not cut off any digits if the calculated value has six digits or more. Therefore we want to **reset the counter once a day**.

The counter itself is stored in user memory, but not alone. The first five digits of the user memory are made up of the current year and the day of the year, which JScript always fills with zeros in three digits. Let's assume that on New Year's Eve 2019, work was only done until 2 p.m. and 2612 products were produced. On Thursday, January 2nd, 2020, the early shift starts production at 6.00 a.m. The user memory then contains the string "**19365**02612" (20**19** is not a leap year and therefore has **365** days). The comparison of "19365" with "20002" now returns 0 (false = numerical zero). Multiply 0 by 02612 to get 0, add one and you get the new serial number 00001.

So we have to save the date in a unique form and compare it with the current day's date. The result (1 if identical, otherwise 0) is multiplied by the old (also stored) counter value and only then incremented by 1. However, if no leading zeros and two decimal places isn't wanted (standard format within JScript for calculations), you have to format them appropriately, e.g. like here with *[D:5,0][C:0]*.

The second label in the new year would read "**20002**00001" from the user memory. The comparison of "20002" (content user memory) with "20002" (current value) now returns 1. multiplied by the old counter value and increased by one we get formatted to five digits now "00002".

A summary of the most important date functions can be found in the appendix on page 135.

12.2 Print vouchers

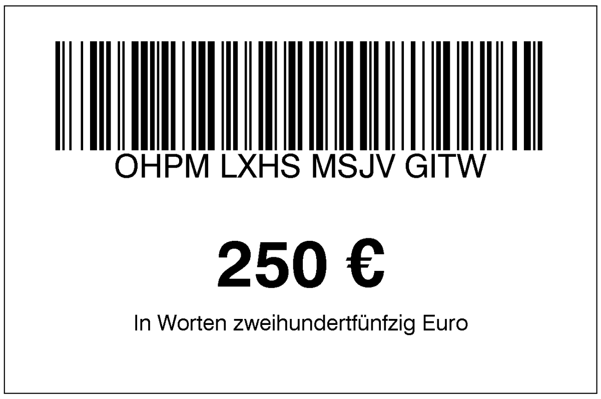


Figure 12.1: A voucher of 250 € (Listing 12.3)

Vouchers from 10 € to 500 € should be printed. The operator can enter the desired amount at the printer. Only multiples of 10 € are allowed. A voucher is printed with a barcode consisting of a sixteen-digit random chain of letters. This information is written into a database. In order to be able to use the code later in the online store, the content is also given in plain text, divided into four-digit segments.

Here the Advanced BASIC Compiler is used to generate the random sequence of capital letters. In JScript a check is performed. If the desired amount is not a multiple of 10 € or if the amount is smaller than 10 € or higher than 500 €, only a “INVALID!” is printed on the label instead of the voucher.

In the lines 1 to 14 a BASIC program is included. If the label is loaded from memory or sent to the printer, the JScript interpreter passes lines 2 to 13 to the Advanced BASIC compiler and continues from line 16.

The BASIC compiler translates the program and executes it directly. It runs in a continuous loop (*REPEAT – UNTIL* lines 4 to 6). Line 3 specifies that characters can be sent directly to JScript. The actual magic of the program is done with the commands *JGET$* and *JPUT*. In line 5, the data stream is permanently queried from JScript. You get an empty string if nothing is (yet) available. If the *a$* assigned string differs from the empty string, it can escape the *UNTIL* loop in line 6. Another loop is used to generate a random string and send it to JScript. Afterwards the BASIC program is terminated in line 13.

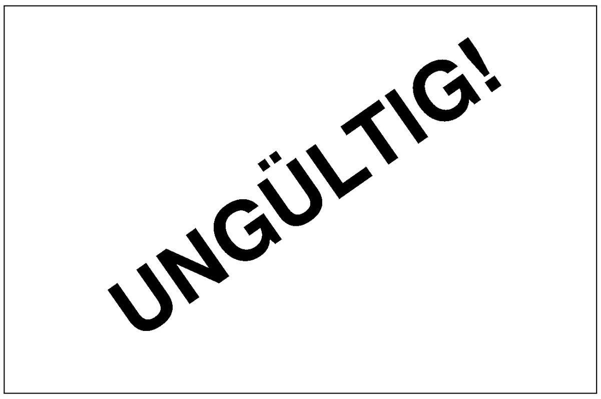


Figure 12.2: 125€ are not allowed, because it’s undividable by 10 (see line 47 and 48 in listing 12.3).

```
1  ; random generator with BASIC
2  <ABC>
3  POKE "bypass",1
4  REPEAT
5      a$ = JGET$
6  UNTIL (a$ <> "")
7
8  Zufall$=""
9  FOR i = 1 TO 16
10     Zufall$ = Zufall$ + CHR$(INT(RAN(24))+ASC("A"))
11 NEXT
12 JPUT Zufall$
13 END
14 </ABC>
```

Listing 12.2: A BASIC software generates a random set of characters Coupon.lbl

From line 16 on, the actual JScript code follows, which uses the BASIC program that is now running in the background.

In line 27, the user is asked for a desired amount. The user can enter a maximum of three digits.

In line 28 the desired amount is sent to *abc* (our BASIC program). The answer is used via the identifier *Zufall* later in line 54 and line 55.

If you read the BASIC code carefully you will notice that the sent value is not used at all. Only line 6 is used to check if something was sent at all, otherwise the program will continue to wait in a continuous loop. However, the special command *[ABC:. . .]* requires a field name as argument, and *Wunschbetrag* in line 28 is the only field name defined in JScript so far (see also section 13.13.2 on page 126).

Line 55 uses the **cutting a section of a string special command** via the two parameters *start position* and *length*. This procedure was already used in the previous example. New is the **indexing using SPLIT** in lines 37 and 42. *SPLIT* expects an identifier and an index as parameters. The record to which the identifier points should be separated with the special character *Group Separator*, accessible via *[U:GS]*. On the newer printers based on the SQUIX board, other separators can also be used, but the *Group Separator* is compatible with all printers and is also used by the Database Connector.

Exciting are the checks, which are completely implemented in JScript here. They are single checks, which must result in a 1 if everything is ok. If we multiply the results of the checks, a wrong result (value 0) is enough to make the whole multiplication become 0.

Line 45 only accepts values starting from 10 (only three-digit integers are to be expected from the query). Line 46 limits up to and including 500 and lines 47 and 48 only allow multiples of 10.

In the layout part the elements become conditionally (un)visible, depending on whether the line ends with *[I:CHECK]* or *[I:!CHECK]*. With *[I:CHECK]* the line becomes invisible if all conditions are fulfilled together. *[I:!CHECK]* reverses the logic, the element is hidden if one of the conditions is not fulfilled.

In any case the input is written to the database together with the random number and a date/time stamp. So you can see the wrong input in the database. The database entry is only made when the voucher has also been printed.

```
J
S l1; 0, 0, 68, 71, 100
H 150, 0

; options: single buffer, 180° rotation, prohibit a reprint
O S, R, U

; database connection to the file misc/Gutscheine.sqlite3
E SQLITE;Gutscheine

; user query and random calculation of the character string
T:Wunschbetrag; 0, 0, 0, 3, 3;[?:Wunschbetrag eingeben,,,D,M111][I]
T:Zufall;       0, 0, 0, 3, 3;[ABC:Wunschbetrag][I]

; create text string for plain text display of price
T:Hunderter;    0, 0, 0, 3, 3;[U:GS]einhundert[U:GS]zweihundert[U:GS]dreihundert[U:GS]…
    vierhundert[U:GS]fünfhundert[I]
T:Zehner;       0, 0, 0, 3, 3;[U:GS]zehn[U:GS]zwanzig[U:GS]dreißig[U:GS]vierzig[U:GS]fü…
    nfzig[U:GS]sechzig[U:GS]siebzig[U:GS]achtzig[U:GS]neunzig[I]

; calculations
T:HUND1; 0, 0, 0, 3, 3;[/:Wunschbetrag,100][I]
T:HUND2; 0, 0, 0, 3, 3;[+:HUND1,1][I]
T:HTEXT; 0, 0, 0, 3, 3;[SPLIT:Hunderter,HUND2][I]

T:ZEHN1; 0, 0, 0, 3, 3;[%:Wunschbetrag,100][I]
T:ZEHN2; 0, 0, 0, 3, 3;[/:ZEHN1,10][I]
T:ZEHN3; 0, 0, 0, 3, 3;[+:ZEHN2,1][I]
T:ZTEXT; 0, 0, 0, 3, 3;[SPLIT:Zehner,ZEHN3][I]

; checks
T:CHK1;  0, 0, 0, 3, 3;[>:Wunschbetrag,9][I]
T:CHK2;  0, 0, 0, 3, 3;[<:Wunschbetrag,501][I]
T:CHK3;  0, 0, 0, 3, 3;[%:Wunschbetrag,10][I]
T:CHK4;  0, 0, 0, 3, 3;[=:CHK3,0][I]
T:CHECK; 0, 0, 0, 3, 3;[*:CHK1,CHK2,CHK4][I]

T 20, 60, 35, 3, 15, b;UNGÜLTIG![I:CHECK]

; layout
B  0,  5,  0, code128, 20, 0.4;[Zufall][J:c100][I:!CHECK]
T  0, 30,  0, 3,  6;[Zufall,1,4] [Zufall,5,4] [Zufall,9,4] [Zufall,13,4][J:c100][I:!…
    CHECK]
T  0, 50,  0, 3, 12, b;[Wunschbetrag] [U:$20AC][J:c100][I:!CHECK]
T  0, 58,  0, 3,  4;In Worten [HTEXT][ZTEXT] Euro[J:c100][I:!CHECK]

; filling the database
T:DAT; 0, 0, 0, 3, 3;[DATE][I]
T:TIM; 0, 0, 0, 3, 3;[TIME][I]
T      0, 0, 0, 5, pt10;[SQLLOG:INSERT INTO Gutscheine VALUES ('{DAT}','{TIM}','{Zufall…
    }','{Wunschbetrag}')][I]

A 1
```

Listing 12.3: A voucher with plain text and barcode on it logged into a database Coupon.lbl

If you want to write the incorrect entries into a separate table, you can change the code. The example would then end as in listing 12.4 (replaces the part from line 59 on). Gutscheine.sqlite3

```
59 ; filling the database
60 T:DAT;        0, 0, 0, 3, 3;[DATE][I]
61 T:TIM;        0, 0, 0, 3, 3;[TIME][I]
62
63 T:CHECKIDX;   0, 0, 0, 3, 3;[+:CHECK,1][I]
64 T:Tabellen;   0, 0, 0, 3, 3;Fehleingaben[U:GS]Gutscheine[I]
65 T:wohindamit; 0, 0, 0, 3, 3;[SPLIT:Tabellen,CHECKIDX][I]
66
67 T 0, 0, 0, 5, pt10;[SQLLOG:INSERT INTO {wohindamit} VALUES ('{DAT}','{TIM}','{Zufall…
       }','{Wunschbetrag}')][I]
68
69 A 1
```

Listing 12.4: Log incorrect entries in a separate table FehlerGesondertLoggen.lbl

12.3 Printing data from a CSV file

As a more complex example of programming with ABC, the import of a CSV file shall demonstrate it. The following program (listing 12.5), stored as *CSVPrinter.lbl* in the *labels* directory of the printer's default memory, will read a CSV file, ignore the first line and then pass the twenty-first to twenty-fifth record as *CSV* using the JScript Replace command and print the desired number of labels (here one label at a time).

```
<ABC>
REM Set parameters as a block of variables
CSV_Datei$ = "regional-averages-tm-year.csv"
Titelzeile = TRUE
Trenner$ = "|;,"
Layout$ = "CSV-Layout"
Anzahl = 1
von = 21
bis = 25

REM ------8<------------------------------------------

REM load the layout
PRINT "M l LBL;" + Layout$

REM open the comma separated value file for read
OPEN #1,CSV_Datei$,"r"

REM skip the first line (it usually contains the field names)
IF (Titelzeile) THEN
    LINE INPUT #1 Titelzeile$
ENDIF

REM where are we? (remember the line number)
Position = 0

WHILE(NOT EOF(1))
    REM read in a complete line (without <CR>)
    LINE INPUT #1 Zeile$

    REM ignore empty lines
    IF (Zeile$ = "") THEN
        CONTINUE
    ELSE
        Position = Position + 1
    ENDIF

    REM go to the selected area of data
    IF (Position < von) THEN
        CONTINUE
    ENDIF
    IF (Position > bis) THEN
        BREAK
    ENDIF

    REM change separator character to [GS]
    FOR i = 1 TO LEN(Zeile$)
        IF (INSTR(Trenner$, MID$(Zeile$,i,1))) THEN
            MID$(Zeile$,i,1) = CHR$(29)
        ENDIF
    NEXT

    REM print the right among of labels
    PRINT "R CSV;" + Zeile$
    PRINT "A " + STR$(Anzahl)
WEND

REM and don't forget to close the opened files
CLOSE #1
</ABC>
```

Listing 12.5: Template to read in a CSV file with abc as a data source for JScript CSVPrinter.lbl

The program can be used universally. Only the first part with the parameters must be adapted. If you want to print all records, you have to select 1 and 999999 (a number higher than the number of expected records) for *von* and *bis*. All characters in the character string *Trenner$* are exchanged by the character *[GS]* (Group Separator, for the JScript special function *[SPLIT:CSV,Index]*).

The data of this example is provided by the German Weather Service and can be downloaded here:

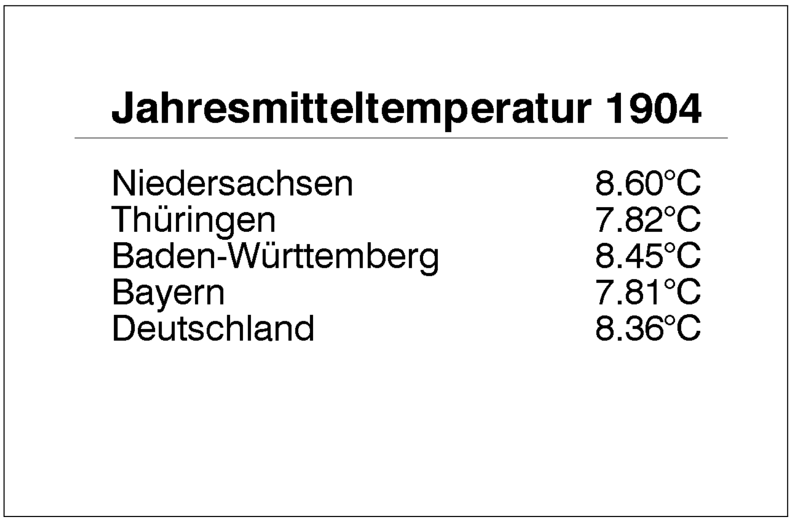
https://opendata.dwd.de/climate_environment/CDC/regional_averages_DE/annual/air_temperature_mean/

From there, load the CSV file “regional_averages_tm_year.txt” and change the file extension to “.csv” if you want to reproduce the example.

Note that here all three characters from the *Trenner$* parameter are replaced by a Group Separator. A CSV file should therefore be read appropriately, in this case better that way:

```
Trenner$ = ";"
```

The layout was stored in the file *CSV-Layout.lbl*. It could look like this (listing 12.6).



Jahresmitteltemperatur 1904

Niedersachsen	8.60°C
Thüringen	7.82°C
Baden-Württemberg	8.45°C
Bayern	7.81°C
Deutschland	8.36°C

Figure 12.3: Annual average temperatures are read from a CSV file with climate data (source: Deutscher Wetterdienst). regional-averages-tm-year.csv

```
m m
J
S l1; 0, 0, 68, 71, 100
H 100, 0

; get Data from csv file
T:CSV; 0,  0, 0, 3, 3;nix[I]

T 10, 15, 0, 5, 6;Jahresmitteltemperatur [SPLIT:CSV,1]
G  5, 17, 0;L: 90, 0.1

T 10, 25, 0, 3, 5;Niedersachsen
T 10, 30, 0, 3, 5;Thüringen
T 10, 35, 0, 3, 5;Baden-Württemberg
T 10, 40, 0, 3, 5;Bayern
T 10, 45, 0, 3, 5;Deutschland

T 70, 25, 0, 3, 5;[SPLIT:CSV,9]°C
T 70, 30, 0, 3, 5;[SPLIT:CSV,18]°C
T 70, 35, 0, 3, 5;[SPLIT:CSV,5]°C
T 70, 40, 0, 3, 5;[SPLIT:CSV,6]°C
T 70, 45, 0, 3, 5;[SPLIT:CSV,19]°C
```

Listing 12.6: In the layout the identifier CSV can be used to process the file from the source line by line. The SPLIT command leads to the individual values. CSV-Layout.lbl

13 Advanced BASIC Compiler

In addition to the JScript code, BASIC code can also be embedded. This code must be embedded between the two lines *<ABC>* and *</ABC>*. The JScript interpreter then passes the abc code block as a whole to the abc compiler. This will translate the code and run the generated program. The generated abc program starts independently of the JScript interpreter as its own task. Execution start and speed are out of sync with JScript.

You can only pass a single such code block. If a new code block is passed to the abc compiler while the old BASIC program is still running, the newly passed code is initially only buffered. Only when an abc program is terminated can new code be compiled from the buffer and brought to execution.

13.1 This is not a BASIC manual

The BASIC language is not explained here. Ask your favorite search engine for BASIC tutorials and books, there are plenty of them.

The abc BASIC used in cab printers is derived from *Yabasic*. A (admittedly not very beginner-friendly) guide to *Yabasic* can be found on the net:

`http://www.yabasic.de/yabasic.htm`

13.2 Either JScript or abc

If BASIC is used, you should not integrate JScript into the same label file, but embed the JScript code in the BASIC program via the PRINT command. This is because it causes synchronization problems when pure JScript code with an abc block is placed in a single file. The abc code usually takes a little longer to be compiled and then executed. However, the timing cannot be predicted with certainty. If the JScript portion is sent from BASIC via the PRINT command, it is clearly determined in the time sequence of the abc BASIC program when it is executed.

13.3 Comments in abc

Not only JScript allows to insert comment lines, also in abc BASIC you can make the program code readable by comments for later maintenance. abc goes even further, allowing a line of other code to be commented on by two slashes until the end of the line. Mixing command and comment in one line is not possible in JScript.

If you want to insert a pure comment line use the REM command. The listing 13.1 shows the use of both comment forms.

13.4 The PRINT command

The way from abc to JScript is quite simple. Just use the command *PRINT* and “print” into JScript.

From within the BASIC program, you can send a string to the JScript interpreter as a command line using the PRINT command. The string is given without line end characters, the PRINT command automatically appends the line end.

It should always be remembered that the timing of the processing of an embedded BASIC code is unpredictable because the abc compiler and the JScript interpreter run independently of each other as separate processes in the printer. The X2 motherboards still contained cab’s own monolithic architecture without multitasking. The only process running was a continuous loop.

However, this disadvantage of the older systems (Linux was first introduced with the X3 motherboard) offered predictability in the sequence of processing JScript and Advanced BASIC. This can lead to the fact that solutions programmed for older systems can lead to problems on X4 mainboards today if JScript and BASIC are not properly separated.

If a semicolon is appended to the PRINT command, the automatically appended <*CR*> Character omitted. Do not forget to end the line with a PRINT command without a semicolon, otherwise you will get a non-executable JScript code. An example shows the listing 13.5.

13.5 Conditional tasks and jumps

13.5.1 IF (condition) THEN . . . ELSIF (condition) THEN . . . ELSE . . . ENDIF

The IF statement can be used in abc to attach a command block to a condition. The condition must be followed by the keyword THEN. The ELSE part is optional and is executed if the condition results in *FALSE*. The keyword ENDIF, with which the construct must be terminated, is mandatory. ELSIF blocks can be inserted in between. Any number of them is allowed. The listing 13.2 shows examples of conditional statements.

A condition is a logical expression enclosed in round brackets. The following symbols are available (table 13.1). They have different values for determining the order. Comparison operators are evaluated *before* the logical operators (similar to the “dot-before-dash rule” in calculations). Mathematical calculation operators are processed before the comparison operators.

Table 13.1: Logical operators in abc

Operator	Content
=	equal
<>	not equal
>	greater than
<	less than
>=	greater or equal
<=	less or equal
AND	logical AND (*TRUE* if both are *TRUE* , else *FALSE*)
OR	logical OR (*TRUE* at least one argument is TRUE, else *FALSE*)
NOT	logical NOT (*TRUE* if *FALSE*, else *FALSE*)

There is also a short form of the IF instruction. If you write the condition and the conditional command together with the IF in only one line of code, you can do without THEN and ENDIF.

Several conditional statements are used in the listing 13.2. It should also be noted that abc BASIC makes a short circuit for logical calculations. That is, if an AND statement results *FALSE* in the first argument, or an OR statement results *TRUE* in the first argument, then all the following arguments are no longer evaluated. This behavior can be seen in the listing 13.1

```
<ABC>
PRINT "J"
PRINT "S 0, 0, 68, 71, 100"
PRINT "H 100, 0"
PRINT "O R"

REM If the first element of an AND condition is false,
REM the second element won't be evaluated (logical shortcut).
PRINT "T 5, 30, 0, 3, 4;[J:c90]FALSE AND FALSE: ";
IF (falsch("with") AND falsch("out")) THEN
    PRINT "** Oh no! **";    // This should never become visible ...
ENDIF
PRINT " short circuit"

REM The same happens to OR conditions,
REM if the first part is true.
PRINT "T 5, 40, 0, 3, 4;[J:c90]TRUE  OR  TRUE: ";
IF (richtig("with") OR richtig("out")) THEN
    PRINT " short circuit"
ENDIF

PRINT "A 1"

REM Subroutines has to be encapsulated in SUB name(arguments) ... END SUB
SUB falsch(text$)
    PRINT text$;   // print the given text towards JScript
    RETURN FALSE   // and return FALSE for the logic evaluation
END SUB

SUB richtig(text$)
    PRINT text$;   // print the given text towards JScript
    RETURN TRUE    // and return TRUE for the logic evaluation
END SUB
</ABC>
```

Listing 13.1: abc is using logical shortcuts LogicShortcut.lbl

13.5.2 GOTO and GOSUB

You can also jump within a program. The GOTO command jumps to the position with a matching LABEL command, while the GOSUB command remembers the current position in the program and jumps back there when the program hits the RETURN command. Both can be seen in the abc program in listing 13.2.

```
<ABC>
OPEN #1, "/DEV/RAWIP", "r"
OPEN #2, "/DEV/RAWIP", "w"

DO
    LINE INPUT #1 Eingabe$
    IF (Eingabe$ = "exit") BREAK
    GOSUB Antworte
LOOP

GOTO goodbye

LABEL Antworte
IF (Eingabe$ = "time") THEN
    PRINT #2 TIME$
    RETURN
ELSIF (Eingabe$ = "date") THEN
    PRINT #2 DATE$
    RETURN
ELSIF (Eingabe$ = "random") THEN
    PRINT #2 STR$(RAN(100))
    RETURN
ELSE
    PRINT #2 "syntax error: ", Eingabe$
    RETURN
ENDIF

LABEL goodbye
PRINT #2 "goodbye"
CLOSE #1
CLOSE #2
END
</ABC>
```

Listing 13.2: Conditional instructions and jumps. If you want to test the program, you need to start a Telnet session on port 9100 of the printer. IF-THEN.lbl

13.6 Loops

In BASIC there are various forms to repeat a code specifically. Implemented in the printer is a *FOR/NEXT*, *DO/LOOP*, *WHILE/WEND* und *REPEAT/UNTIL* loop construction.

13.6.1 FOR Variable = start TO end STEP stepping . . . NEXT

This example will print four labels with a serial number (listing 13.3)

You can exchange the example by this abc program (listing 13.4).

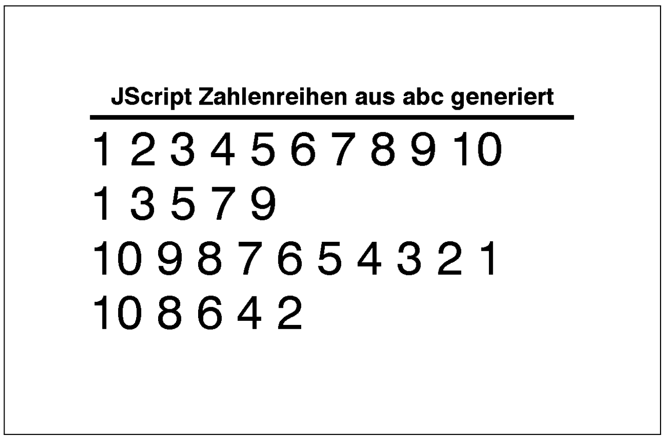
```
m m
J
S l1;0,0,68,71,100
H 150,0
O R
T 40,40,0,5,20;[SER:1]
A 4
```

Listing 13.3: A label containing a serial number JScript-SER.lbl

```
<ABC>
PRINT "m m"
FOR i = 1 TO 4 STEP 1
    PRINT "J"
    PRINT "S l1; 0, 0, 68, 71, 100"
    PRINT "H 150, 0"
    PRINT "O R"
    PRINT "T 40, 40, 0, 5, 20;" + STR$(i)
    PRINT "A 1"
NEXT
</ABC>
```

Listing 13.4: abc creates a serial number with a loop abc-SER.lbl

A FOR loop must be completed with the NEXT command. The loop variable is first assigned to the start value. The code between FOR and NEXT is repeated until the variable reaches the final value. The STEP command is optional and specifies the increment. If you omit STEP, abc counts up with 1. Negative values count down. If the end value is exceeded, the loop content will no longer be executed, see also the listing 13.5.



JScript Zahlenreihen aus abc generiert

```
1 2 3 4 5 6 7 8 9 10
1 3 5 7 9
10 9 8 7 6 5 4 3 2 1
10 8 6 4 2
```

Figure 13.1: Result of the FOR/NEXT loops in listing 13.5

```
<ABC>
PRINT "m m"
PRINT "J"
PRINT "S l1; 0, 0, 68, 71, 100"
PRINT "H 150, 0"
PRINT "O R"

PRINT "T 10, 15, 0, 5, 4;[J:c80]JScript Zahlenreihen aus abc generiert"
PRINT "G 10, 17, 0;L: 80, 0.75"

REM some examples using different steppings
PRINT "T 10, 25, 0, 3, 8;";
FOR i = 1 TO 10
    PRINT i;
NEXT
PRINT

PRINT "T 10, 34, 0, 3, 8;";
FOR i = 1 TO 10 STEP 2
    PRINT i;
NEXT
PRINT

PRINT "T 10, 43, 0, 3, 8;";
FOR i = 10 TO 1 STEP -1
    PRINT i;
NEXT
PRINT

PRINT "T 10, 52, 0, 3, 8;";
FOR i = 10 TO 1 STEP -2
    PRINT i;
NEXT
PRINT

PRINT "A 1"
END
</ABC>
```

Listing 13.5: Some FOR/NEXT loops, keep an eye on the semicolon at the end of a PRINT command. FOR-NEXT.lbl

13.6.2 DO . . . LOOP

In a DO/LOOP loop the code is repeated endlessly within the loop. You can, however, break out of the loop with the BREAK command. abc will continue with the code after the LOOP command in a BREAK command.

A continuous loop without end shows listing 13.6 (by pressing the red Cancel key on the display or pressing Cancel on the Navigator Pad for at least three seconds on older printers, an abc program can be terminated at any time).

```
<ABC>
OPEN #1,"/DEV/RS232","w"
OPEN #2,"/DEV/RAWIP","r"

DO
    x = PEEK(#2)
    IF (x <> -1) POKE #1,x
LOOP
</ABC>
```

Listing 13.6: Redirect the IP interface to the RS232 interface IP-Umleiten.lbl

13.6.3 WHILE (condition) . . . WEND

After the WHILE command a condition must be specified. This is similar to the IF command, except that the condition is directly followed by the loop block, without another command word like THEN or DO. When the loop block is finished, the WEND command is used to inform the compiler of the end of the loop.

With a WHILE loop, it is also possible that the loop block is not executed at all if the condition already results in *FALSE* during the first check. As long as the condition results in *TRUE* the loop is executed repeatedly. An example is reading files line by line. If a file has no content, then nothing is processed, as seen in listing 13.7.

13.6.4 REPEAT . . . UNTIL (condition)

If a loop is to be run through at least once, the REPEAT UNTIL loop can be used to check the condition only after a loop has been run through. However, the loop is executed again if the condition results in *FALSE*. Such a loop can also be found in the listing 13.7. The two commands MOUSEX and MOUSEY return the numeric value -1 if the display is not touched or the respective position if the display is touched.

Both the WHILE and the REPEAT loop can be replaced with a suitably designed DO loop if the IF (condition) BREAK command is used in it to exit from the repetition.

```
<ABC>
REM This program has to be started from
REM the printer's default storage memory!
OPEN #1,"/card/labels/WHILE.lbl","r"
OPEN WINDOW 272, 480
y = 0
POKE "lcd", 1

WHILE (NOT EOF(#1))
    LINE INPUT #1 Zeile$
    TEXT 0,y,Zeile$
    y = y + 12
WEND

TEXT 65, 460, "*** PLEASE TOUCH ME! ***"

REPEAT
    PAUSE 0.1
UNTIL (MOUSEX > -1)

POKE "lcd", 0
CLOSE WINDOW
</ABC>
```

Listing 13.7: A file reads itself and shows up on the display of a SQUIX or EOS printer. The command EOF(#1) indicates whether the end of the file has been reached. WHILE.lbl

13.7 Subroutines by SUB name(arguments) . . . END SUB

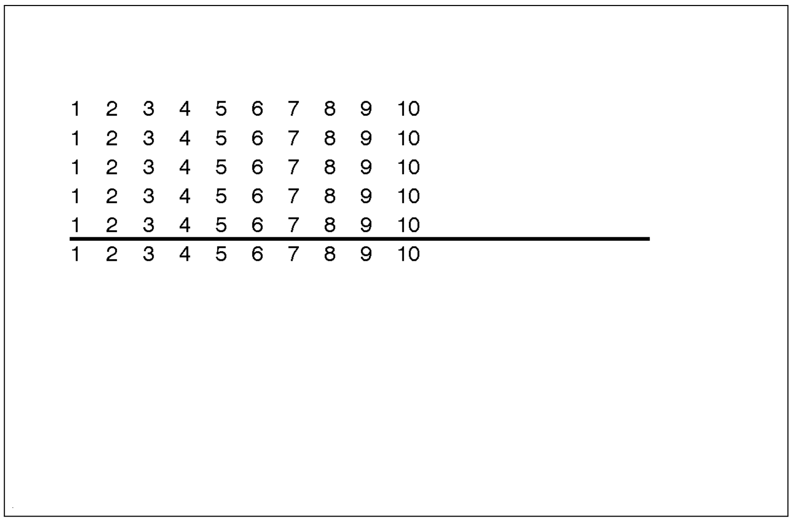
When BASIC was invented, Structured Programming with encapsulated routines had not yet been thought of. At that time BASIC was invented as a pure imperative programming language. Later, various dialects of the language introduced structured programming through concepts such as encapsulated subroutines. This also applies to abc. Also when programming cab printers you can define functions and call them in the program using the function name followed by arguments in round brackets. A function can, but does not have to return a value. The return from a function is done by the RETURN command, as with the GOSUB command. You can let RETURN be followed by another value. This is then used as the function return value.

Note, however, that each variable is also defined globally within a subroutine. If you want to encapsulate a variable in a function, you have to declare it once with the prefixed keyword LOCAL before using it. A variable declared this way within a subroutine is not available outside the subroutine. If the subroutine is terminated, all local variables are removed from the RAM again. If you want to keep a variable value over several calls, you must use the STATIC keyword. Variables declared with STATIC are automatically also local variables, i.e. they are not accessible outside the subroutine.

How locally declared variables leave the global variables untouched can be seen in the listing 13.8. If you use subroutines, always make sure to declare all variables with LOCAL or STATIC. BASIC (and therefore abc) does not pay attention to this detail. Otherwise you will get an error like in listing 13.9.

```
<ABC>
REM global variables
a = 1
b = 2
c = 3

Test(a, b)

REM all variables inside the subroutine are local
SUB Test(b, c)
    LOCAL a
    a = 30
    b = 40
    c = 50
END SUB

PRINT "J"
PRINT "S l1; 0, 0, 68, 71, 100"
PRINT "H 100, 0"
PRINT "O R"
PRINT "T 10, 65, 0, 3, 20;", a, b, c
PRINT "A 1"
</ABC>
```

Listing 13.8: Variables that have been declared with LOCAL or passed as arguments are local in a subroutine, the output on the label is "1 2 3". LOCAL.lbl



```
1   2   3   4   5   6   7   8   9   10
1   2   3   4   5   6   7   8   9   10
1   2   3   4   5   6   7   8   9   10
1   2   3   4   5   6   7   8   9   10
1   2   3   4   5   6   7   8   9   10
1   2   3   4   5   6   7   8   9   10
```

Figure 13.2: The listing 13.9 should originally produce ten rows with the numbers from 1 to 10. But four rows are missing, because the global variable "i" is changed in the subroutine.

```
<ABC>
PRINT "J"
PRINT "S l1; 0, 0, 68, 71, 100"
PRINT "H 100, 0"
PRINT "O R"

REM The first five rows are counted upwards by the global variable "y"
FOR y = 1 TO 5
    Zahlenreihe(y, 10)
NEXT

REM division line on the label
PRINT "G 5, 31, 0;L:80,0.5"

REM "i" is used globally for the last five rows (underneath the division line)
FOR i = 1 TO 5
    Zahlenreihe(i, 30)
NEXT

PRINT "A 1"

SUB Zahlenreihe(a, b)
    FOR i = 1 TO 10   // ATTENTION: The variable "i" is still used globally!
        PRINT "T " + STR$(5 * i) + "," + STR$(4 * a + b) + ",0,3,3;" + STR$(i)
    NEXT
END SUB
</ABC>
```

Listing 13.9: Also a counter variable in a FOR/NEXT loop is global within a subroutine. For the exercise change the code so that ten rows are printed. Kapselung.lbl

The listing 13.10 is based on a lecture about bad programming style and the resulting program errors caused by it. Here the subroutines access global variables and the names of the passed variables do not match the names in the SUB(. . .) definition. Who has to deal with such a code has after few minutes only spaghetti in the brain. The listing 13.11 solves the same task in a more understandable way, even if the subroutine also accesses the global array.

```
<ABC>
REM global variables
DATA 45, 23, 34, 12
READ vinegar, oil, salt, pepper

SUB Christoph(pepper, salt)
IF (salt > pepper) THEN
    oil = salt
    vinegar = pepper
ENDIF
END SUB

SUB Anton(vinegar, salt)
IF (vinegar < salt) THEN
    oil = vinegar
    pepper = salt
ENDIF
END SUB

SUB Daniela(vinegar, oil)
IF (oil > vinegar) THEN
    pepper = oil
    salt = vinegar
ENDIF
END SUB

SUB Britta(oil, pepper)
IF (pepper > oil) THEN
    salt = pepper
    vinegar = oil
ENDIF
END SUB

REM four people stir a salad
Anton(pepper, oil)
Britta(salt, vinegar)
Christoph(oil, vinegar)
Daniela(pepper, salt)

PRINT "J"
PRINT "S l1; 0, 0, 68, 71, 100"
PRINT "H 100, 0"
PRINT "O R"
PRINT "T 10, 35, 0, 3, 10;";
IF (oil > salt) THEN
    PRINT vinegar, salt, oil, pepper
ELSE
    PRINT vinegar, oil, salt, pepper
ENDIF
PRINT "A 1"
</ABC>
```

Listing 13.10: Change the four numbers in the DATA line and observe the result. Can you explain the program? Salad.lbl

```
<ABC>
REM global variables
DATA 45, 23, 34, 12
DIM Values(4)       // four values stored into one global array
FOR i = 1 TO 4
    READ Values(i)
NEXT

REM use a pointer to an array to manipulate a global array
SUB sortiere(a(),x,y)
LOCAL puffer       // this variable exists only during a function call
IF (a(x) > a(y)) THEN  // swap the two numbers in the related array
    puffer = a(x)
    a(x)   = a(y)
    a(y)   = puffer
ENDIF
END SUB

REM Network Sort over four values
sortiere(Values(),1,3)
sortiere(Values(),2,4)
sortiere(Values(),1,2)
sortiere(Values(),3,4)
sortiere(Values(),2,3)

REM The JScript code is equal to the label "Salad.lbl"
PRINT "J"
PRINT "S l1; 0, 0, 68, 71, 100"
PRINT "H 100, 0"
PRINT "O R"
PRINT "T 10, 35, 0, 3, 10;";
FOR i = 1 TO 4
    PRINT Values(i);
NEXT
PRINT   // Don't forget the line ending character, or JScript will fail!
PRINT "A 1"
</ABC>
```

Listing 13.11: Does the same as listing 13.10 NetworkSort.lbl

13.7.1 Using a pointer to an array as an argument of a subroutine

If one of the arguments passed to a function is an array where the round bracket remains empty, a pointer to that array is passed to the function. The function then edits the global array under the name used locally in the function definition. You can use this oddity to create more than one return value. For an example of an abc array pointer, see listing 13.11.

13.8 String operations

In abc BASIC, a variable can represent a number or a string. Numbers have a simple variable name that must begin with a letter and can only consist of letters and numbers. A distinction is made between upper case and lower case. Here in this guide, variable names are always lower case and commands are capitalized. But you can also use abc commands in lower case. If a variable contains a string, a dollar sign must be appended to the variable name.

You can also place multiple contents numbered in one variable. Such an entity is called array. The array must be announced in size before first being used with the *DIM* command (declaration).

Here is an example (listing 13.12). The TOKEN command breaks down a string into multiple elements and writes it to an array.

```
<ABC>
zahl = 42
antwort$ = "We have the solution for every challenge."
DIM a$(10)
anzahl = TOKEN(antwort$, a$(), " ")
IF (a$(4) = "solution") END
ERROR "Oh no, that went wrong!"
</ABC>
```

Listing 13.12: A meaningless BASIC program. If the abc symbol disappears from the display of the printer, the program has been successfully executed. Change line 6 to get the error message from line 7 presented. BASIC-Variablen.lbl

13.8.1 num = SPLIT(string$, return_array$(), dividing_character$)

The SPLIT command requires a string to be split as the first argument. Then a pointer to an array must follow (array name with dollar signs and empty brackets) that allows the SPLIT command to edit the array. The last argument is expected to be a string with the separator characters.

The SPLIT command copies character by character from the string to the first element of the array. If it reads a delimiter, it will not be transferred, but it will be switched to the next array element. At the end of the procedure, you have one array element more than the string contains separator characters. The return value of the SPLIT command is the number of elements generated in the array. If the array was previously too small with the DIM command, the SPLIT command corrects the array size to fit it.

13.8.2 num = TOKEN(string$, array$(), separator$)

The TOKEN command works like the SPLIT command, but with the difference that no empty strings are created as elements of the array. So you can't say that TOKEN creates exactly one more element in the array than delimiters exist. For clarity, see listing 13.13.

13.8.3 Handling sub strings with LEFT$, MID$ and RIGHT$

You can cut out a part of a string from it. For this purpose there are the commands LEFT$(*string$*, *number*), RIGHT$(*string$*, *number*) and MID$(*string$*, *position*, *number*). If you omit the last parameter in the MID$ command, the string is returned or replaced from the position (second parameter) to the end.

The special thing about this is that you can also reassign parts of a string by using the function on the left side of an assignment operation. The listing 13.14 shows how to use the MID$ function for a reassignment of a part of a string. But be careful, the reassignment only works for replacements of equal length!

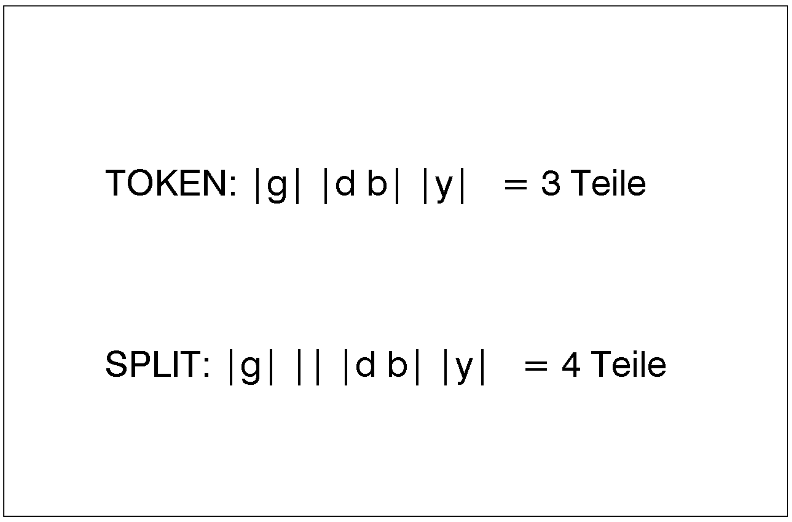
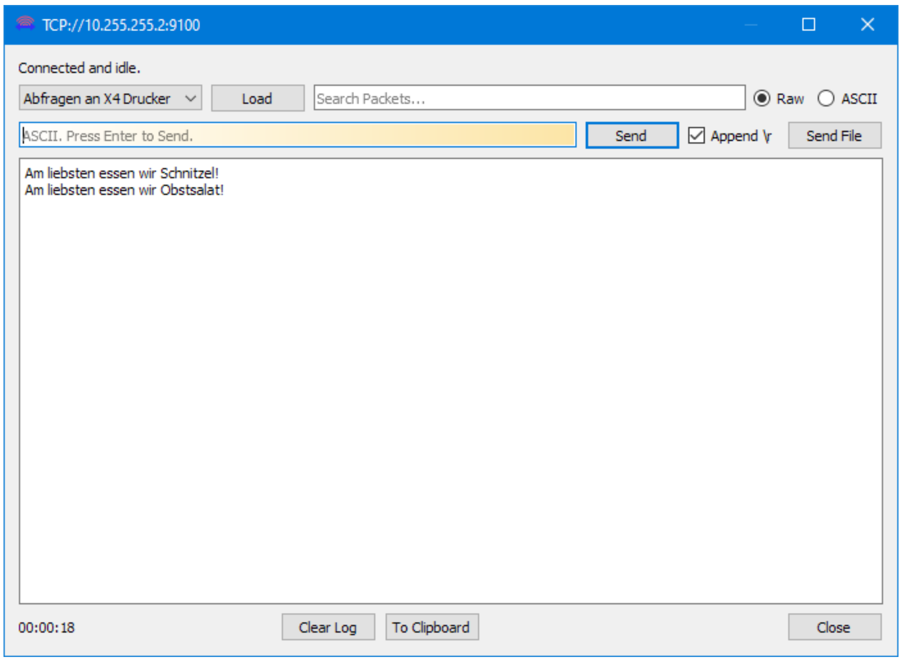

TOKEN: |g| |d b| |y| = 3 Teile
SPLIT: |g| || |d b| |y| = 4 Teile

Figure 13.3: With SPLIT you get one element more than separators are present in the string. TOKEN, on the other hand, doesn't create empty elements (listing 13.13).



```
Am liebsten essen wir Schnitzel!
Am liebsten essen wir Obstsalat!
```

Figure 13.4: With the software Packet Sender we send a line to the RAW-IP interface and receive an accordingly manipulated line as response (listing 13.14).

```
<ABC>
PRINT "J"
PRINT "S l1; 0, 0, 68, 71, 100"

DIM t$(1)
t = TOKEN("good boy", t$(), "o")
PRINT "T 10, 25, 0, 3, 5;TOKEN: ";
FOR i = 1 TO t
    PRINT "|" + t$(i) + "| ";
NEXT
PRINT "  = ", t, " Teile"

DIM s$(1)
s = SPLIT("good boy", s$(), "o")
PRINT "T 10, 50, 0, 3, 5;SPLIT: ";
FOR i = 1 TO s
    PRINT "|" + s$(i) + "| ";
NEXT
PRINT "  = ", s, " Teile"

PRINT "A 1"
</ABC>
```

Listing 13.13: For processing a CSV file, it is better to use SPLIT() instead of TOKEN() so that you do not encounter problems with empty fields. SPLIT-TOKEN.lbl

13.8.4 pos = INSTR(string$, search_pattern$)

Searches for the occurrence of a search pattern in a string and returns the position of the first occurrence. If you do not want to search from left to right, you can use the RINSTR command. If the pattern is not found, the result is 0 (numeric zero). If the string starts with the search pattern, the answer is 1.

13.8.5 Sweeping blank spaces at the beginning and end of a string

The TRIM$(*a$*) command removes all spaces at the beginning and end of the *a$* string. If you only want to remove spaces at the beginning, you have to use the LTRIM$() command, for removing spaces at the end you can use the RTRIM$() command. In the listing 13.16 this is used to make the constructed “LOAD” command more robust. So you can use any number of spaces between “LOAD” and the file name.

13.8.6 More commands to manipulate strings

There are more commands to process strings. For example, you can use LEN(*a$*) to get the length of the string *a$*. With VAL(*number$*) you can convert the string *number$* into a numeric value. The reverse is possible with STR$(*number*). With UPPER(*a$*) you get a string of uppercase letters, with LOWER(*a$*) lowercase letters are generated.

```
<ABC>
OPEN #1, "/DEV/RAWIP", "r"
OPEN #2, "/DEV/RAWIP", "w"

REPEAT
    LINE INPUT #1 Text$

    REM we replace animal products with healthier alternatives (fruit salad)
    pos = INSTR(Text$, "Schnitzel")
    WHILE (pos > 0)
        MID$(Text$, pos, 9) = "Obstsalat"
        pos = INSTR(Text$, "Schnitzel")
    WEND

    PRINT #2 Text$
UNTIL (Text$ = "ende")
</ABC>
```

Listing 13.14: Send a line to the RAW-IP interface and look at the answer BesserVegan.lbl

13.9 Read and write on the interfaces

In abc you can use the command *OPEN* to read or write from the following interfaces:

- /dev/rs232:baud,handshake
- /dev/usb
- /dev/rawip
- /dev/lpr
- /dev/panel
- /dev/keyboard
- /dev/jscript
- /card/filename.ext
- /iffs/name.ext
- mailto:address

We are looking at an example which is often sent to a cab printer for diagnostic purposes (you already know it as the 13.6 listing on page 106). It reads all incoming characters from */dev/rawip* and sends them to */dev/rs232*. So you can listen to the IP interface on the RS232 interface. If you swap the two interfaces, you can also use a cab printer to listen to an RS232 connection in the same way. If you mix the two, you can use the printer to access an RS232 interface via port 9100 (listing 13.15).

```
<ABC>
REM from IP interface to RS232
OPEN #1,"/DEV/RAWIP","r"
OPEN #2,"/DEV/RS232","w"

REM and vice versa
OPEN #3,"/DEV/RS232","r"
OPEN #4,"/DEV/RAWIP","w"

DO
    // IP --> RS232
    x = PEEK(#1)
    IF (x <> -1) POKE #2,x
    // RS232 --> IP
    x = PEEK(#3)
    IF (x <> -1) POKE #4,x
LOOP
</ABC>
```

Listing 13.15: Using a cab printer as a bridge between RS232 and ethernet. Probably clever, if you want to access a RS232 connected machine for diagnostics at a factory hall, where your modern laptop no longer have an interface to RS232. IP-RS232-Bridge.lbl

13.10 Creating a parser

Sometimes a printer has to be integrated into an existing system that uses a completely different syntax than JScript and whose rebuilding is not possible or only possible with great effort. Here we want to think of an alternative dialect as an example. There are only four rules:

1. Each command has it's own line of code.
2. If the line starts with *LOAD*, a filename has to follow.
3. Filling fields (variables) by using the equation sign.
4. If a line starts with *PRINT*, the number of labels has to follow.

We will open the RAW IP interface and listen to the incoming data. If we can convert the incoming code, we send it as JScript code with abc's *PRINT* command.

```
<ABC>
OPEN #1, "/dev/rawip", "r"

DO
    LINE INPUT #1 Zeile$

    REM load layout
    IF (UPPER$(LEFT$(Zeile$, 5)) = "LOAD ") THEN
        PRINT "M l LBL;" + TRIM$(MID$(Zeile$, 6))
    ENDIF

    REM replace field values
    pos = INSTR(Zeile$, "=")
    IF (pos > 0) THEN
        PRINT "R " + LEFT$(Zeile$, pos-1) + ";" + MID$(Zeile$, pos+1)
    ENDIF

    REM print the right among of labels
    IF (UPPER$(LEFT$(Zeile$, 6)) = "PRINT ") THEN
        PRINT "A " + MID$(Zeile$, 7)
    ENDIF
LOOP
</ABC>
```

Listing 13.16: Parsing an unknown stream of data using BASIC logic. Parser.lbl

```
LOAD LayoutVorlage
PARTNO=5977008
PROD=SQUIX 4/600MP
RESOL=600 dpi
SERNR=164162038304
PRINT 1
```

Listing 13.17: Thanks to the parser we can send this code directly to the printer. Alternativsyntax.lbl

13.11 Writing onto the graphical display

The printer display can also be used by abc. For this purpose, a suitable part of the RAM must first be reserved, which is done with the command OPEN WINDOW. The arguments of the command are the number of pixels in X and Y direction. With a SQUIX we control a 272 x 480 pixel display.

After that the bitmap can be edited in memory until we delete it from RAM with CLOSE WINDOW. The command POKE "lcd" switches between the display of the standard controls and the abc bitmap, 1 is for the bitmap, with 0 the controls are displayed again.

You can only write into the bitmap. After creation it is completely filled in white. With the command POKE "bcolor" a background color can be defined. But only as an index from 0 (always black) to 255 (always white). Which color is hidden behind the index can be defined with the POKE "color#*num*", where for *num* a number from 1 to 254 is allowed.

The listing 13.18 shows the commands for graphical output with abc.

The command TEXT can be used to output a text at the coordinates X, Y. In contrast to JScript, the upper left corner is used here, not the baseline. With the FONT command you can select a font, e.g. "Swiss" or "Swiss Bold", followed by the height in display pixels. If a TrueType font is to be used, it must be located in the printer's RAM. Since there is no font load command, we leave this task to JScript, as shown in the example of the 13.18 listing.

```
<ABC>
REM Let JScript load the font Indie Flower into RAM
PRINT "M l FNT;IndieFlower-Regular"

REM Assign a part of the RAM for a bitmap of the printer screen
OPEN WINDOW 272, 480

REM background color set to black
POKE "bcolor", 1

REM Fill the bitmap completely with background color
CLEAR WINDOW

REM switch from printer's status display to abc's bitmap
POKE "lcd", 1

REM Set the font to be used (font name, height in pixel)
FONT "Indie Flower,25"

REM define a random color and write some text onto the screen
FOR x = 1 TO 5
    POKE "color#1", RAN(DEC("FFFFFF"))   // generate random color (RGB hex value),
    POKE "bcolor", 1                      // assign to color no. 1
    CLEAR WINDOW                           // and wipe the screen with the new color
    FOR y = 1 TO 25
        POKE "color#1", RAN(DEC("FFFFFF"))         // change color again randomly
        POKE "fcolor", 1                            // assign new color to index 1
        TEXT RAN(120), RAN(455), "colorful dreams"  // randomly position your text
    NEXT
    PAUSE 2  // a break of 2 seconds before we end
NEXT

REM clean up everything
POKE "lcd", 0   // first we switch back to the printer's status display,
CLOSE WINDOW    // then we purge the bitmap out of the RAM
</ABC>
```

Listing 13.18: abc can overtake the printer's display ColorfulDreams.lbl

13.11.1 Example abc program: global climate warming up

As another example, the listing 13.19 will show a parser that uses values from the already known CSV file from the practical example on page 99 (section 12.3).

These are annual mean temperatures from 1881 to today, published by the German Weather Service (the file "regional_averages_tm_year.txt" must be placed in the `/misc` sub directory of the printer and the file extension must be changed to `.csv`):

https://opendata.dwd.de/climate_environment/CDC/regional_averages_DE/annual/air_temperature_mean/

The abc program uses the possibility to show a bitmap not only as a display on the printer's screen, but also to transfer it to a JScript graphic in the RAM via the WINDOW TRANSFER command. JScript can then easily transfer the bitmap into a label using the *I* command (Image). However, you should note that within the abc bitmap, the coordinate system starts in the upper left corner and positive X values point to the right and positive Y values point to the bottom.

You could use *PEEK("resolution")* to determine the printer resolution and create the usable pixels or a conversion factor for millimeters yourself. Here, however, it is only about a temperature trend without any scale. The focus is on the possibility to create a graphic in abc and use it in JScript.

On the other hand the example shows how to create a selection list in abc and display it in JScript. You get a value back using the JGET$/JPUT combination. Since you use the special function *SELECT*, you have to finish the label with *A [PREVIEW]* without printing it, otherwise the selection will not be displayed.

After that you have to rebuild the label. In the first label, which is not to be printed, the size was intentionally chosen small to show that an S command is required, but the exact dimensions are irrelevant if you only need the result of a screen dialog.

```
<ABC>
OPEN #1, "regional-averages-tm-year.csv", "r"

REM The first line contains only remarks and is skipped
LINE INPUT #1 Zeile$

REM We create a fictitious label to fetch a value from JScript
DIM Bundesland$(1)
LINE INPUT #1 Zeile$
Anzahl = SPLIT(Zeile$, Bundesland$(), ";")

PRINT "e IMG;*"
PRINT "J"
PRINT "S 0, 0, 10, 11, 12"
PRINT "T:Laender; 0, 0, 0, 3, 3;[I]";

FOR i = 3 TO Anzahl
    PRINT Bundesland$(i) + "[U:GS]";
NEXT
PRINT

PRINT "T:LandNr; 0, 0, 0, 3, 3;[I][SELECT:Wähle ein Land aus,Laender,3,1]"
PRINT "T 0, 0, 0, 3, 3;[I][ABC:LandNr]"
PRINT "A [PREVIEW]"

REM Now follows to usual JGET$ --> JPUT query
REPEAT
    J$ = JGET$
UNTIL (J$ <> "")
JPUT(J$)

REM skip the first two rows
Auswahl = VAL(J$) + 2

REM Create a WINDOW object and edit it graphically from the data
OPEN WINDOW 800, 600
```

```
DIM Werte$(1)
NEW CURVE
DO
    LINE INPUT #1 Zeile$
    Anzahl = SPLIT(Zeile$, Werte$(), ";")
    IF (Anzahl < Auswahl) BREAK
    REM The coordinate origin is at the top left
    x = (VAL(Werte$(1)) - 1880) * 5
    y = 600 - VAL(TRIM$(Werte$(Auswahl))) * 50
    LINE TO x, y
LOOP

REM Store it directly into RAM as a JScript image
WINDOW TRANSFER TO "Temperaturen"
CLOSE WINDOW

REM Finally print the actual label
PRINT "J"
PRINT "S l1; 0, 0, 68, 71, 100"
PRINT "H 50, 0"
PRINT "O S"
PRINT "T 5, 10, 0, 3, 5;" + Bundesland$(Auswahl)

PRINT "I 5, 15, 0;Temperaturen"
PRINT "A 1"

CLOSE #1
</ABC>
```

Listing 13.19: This selection dialog can only be performed on the newer printers with an X4 motherboard. It has only become possible with the touch display and its user interface. Temperaturverlauf.lbl

13.11.2 Rectangles and lines on the display

It is relatively easy to draw a rectangle. To do this, use the RECT command, followed by the coordinates of two opposite corners of the rectangle.

```
RECT x1, y1, x2, y2
```

As a result, a rectangle is drawn. You can't influence the pen width, it's always one pixel wide. The rectangle can be filled out. Preceding the FILL command fills it with the foreground color, CLEAR fills it with the background color.

If a line is to be drawn instead of a rectangle, it must be initiated with the NEW CURVE command. The individual points of the line strokes are then defined with LINE TO *x*, *y* Unlike Yabasic, however, abc BASIC does not have a CLOSE CURVE command. You must return to the first point by yourself, and you cannot fill the resulting surface. The CIRCLE and TRIANGLE commands are also not available.

The use of the RECT and LINE TO commands shows listing 13.20.

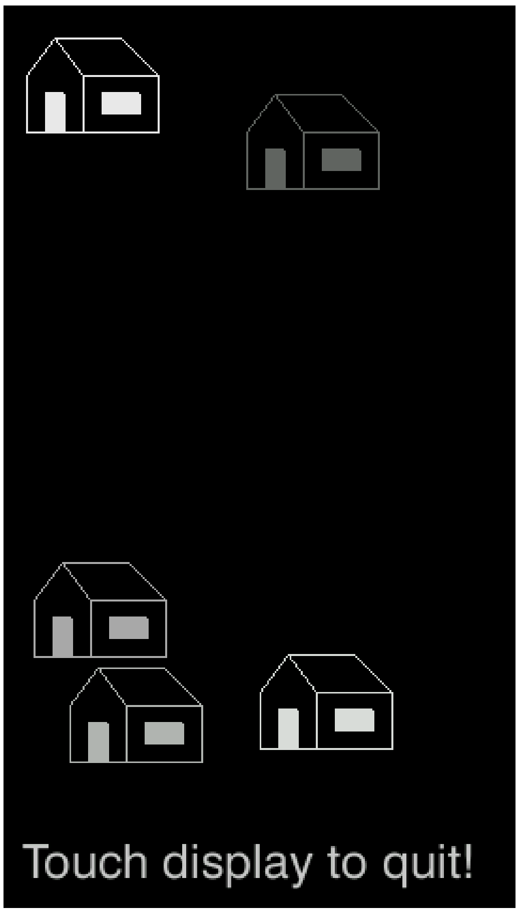


Figure 13.5: The listing 13.20 creates five buildings on the display and waits for a touch input.

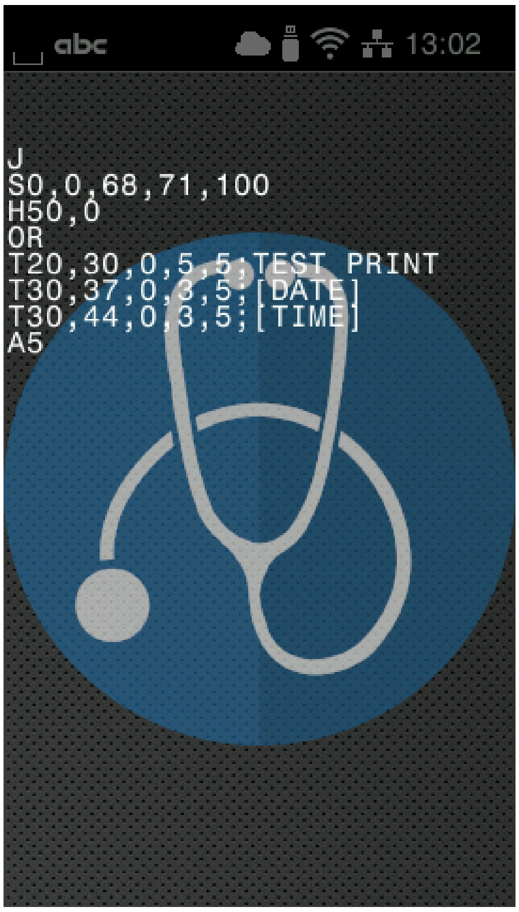
```
<ABC>
OPEN WINDOW 272, 480
POKE "bcolor", 0      // 0 = black, 255 = white
CLEAR WINDOW
POKE "lcd", 1

FOR wdh = 1 TO 5
    REM By default all colors are grayscaled
    POKE "fcolor", 100 + RAN(155)
    Hausbau(RAN(200), RAN(380))
NEXT

POKE "fcolor", 200
FONT "Swiss721,25"
TEXT 10, 440, "Touch display to quit!"
REPEAT PAUSE 0.02 UNTIL (MOUSEX>-1)
POKE "lcd", 0
CLOSE WINDOW
END

REM House outline
DATA 15, 0, 30, 20, 70, 20, 50, 0, 15, 0, 0, 20, 0, 50, 70, 50, 70, 20, 30, 20, 30, 50

SUB Hausbau(xrel, yrel)
    REM clean up space for drawing one house
    CLEAR FILL RECT xrel, yrel, xrel + 70, yrel + 50
    REM draw the outline of one house
    NEW CURVE
    RESTORE
    FOR i = 1 TO 11
        READ x, y
        LINE TO x + xrel, y + yrel
    NEXT
    REM draw door and window
    FILL RECT xrel + 10, yrel + 50, xrel + 20, yrel + 30
    FILL RECT xrel + 40, yrel + 40, xrel + 60, yrel + 30
END SUB
</ABC>
```

Listing 13.20: Drawing five buildings on the screen Village.lbl

13.11.3 Example: a transparent onscreen logo

If you have a graphic in PNG format that has transparency information (an alpha channel), an X4 printer (SQUIX or newer) can process this transparency. We will look at this in an example. The listing 13.21 loads two graphics into the abc window (lines 4 and 5). Actually row 5 would overwrite the graphic from row 4, but the file has a transparency, so both will mix in the display.

The rest of the program is relatively simple, it opens the RAW-IP interface for reading, and everything sent to this interface is output to the display instead of being processed by the JScript interpreter. So it's a kind of monitor mode without labels to print on. If the text reaches the bottom of the display, the display is erased again and reassembled from the two graphics. The incoming text is displayed line by line starting from the top again.



```
J
S0,0,68,71,100
H50,0
OR
T20,30,0,5,5;TEST PRINT
T30,37,0,3,5;[DATE]
T30,44,0,3,5;[TIME]
A5
```

Figure 13.6: The stethoscope merges with the background using transparency (listing 13.21)
SQUIX-background.png Stethoskop.png

Interesting about listing 13.21 is also line 8. The POKE command causes the abc window to appear on the display, 36 pixels down. This keeps the upper toolbar visible (see figure 13.6). Besides the vertical offset *lcdy* there is also *lcdx* available for a horizontal offset.

```
1  <ABC>
2  REM abc window with transparent background logo
3  OPEN WINDOW 272,444
4  WINDOW READ FROM "SQUIX-background"
5  WINDOW READ FROM "Stethoskop"
6  FONT "Monospace,16"
7  POKE "fcolor", 255
8  POKE "lcdy", 36
9  POKE "lcd", 1
10
11 REM listen to input stream
12 OPEN #1, "/dev/rawip", "r"
13
14 position = 20
15 DO
16     LINE INPUT #1 Zeile$
17     REPEAT
18         position = position + 14
19         IF (position > 430) THEN
20             position = 34
21             CLEAR WINDOW
22             WINDOW READ FROM "SQUIX-background"
23             WINDOW READ FROM "Stethoskop"
24         ENDIF
25         TEXT 2, position, LEFT$(Zeile$, 27)
26         Zeile$ = MID$(Zeile$, 28)
27     UNTIL (Zeile$ = "")
28     IF (MOUSEX > -1) BREAK
29 LOOP
30 </ABC>
```

Listing 13.21: The inputs to the RAW-IP interface are written line by line to the display. The logo from line 5 is transparent so that it merges with the graphic below. RAWIP2Display.lbl

13.11.4 Interact with the user by catching touches

A cab printer with a X4 motherboard (SQUIX series or newer) has not only a color display for a graphical user interface, but also a resistive touch sensor. This makes it possible to detect keystrokes on the display. In abc BASIC the touch sensor is represented in the two commands MOUSEX and MOUSEY. If the screen is not touched, both have the value -1. If pressure is applied to the display, the two commands return the respective coordinates of the pressure point.

Try it, the listing 13.22 creates a polyline from the pressure points by using the two commands MOUSEX and MOUSEY. Do you manage to draw a house of Santa Claus? Compared to a capacitive touch screen, a resistive sensor can also be controlled with a glove. Multiple finger gestures like on a smartphone are not necessary on the display of a printer, so it is not a big deal that a resistive sensor can't handle it.

The request of the mouse position is sent internally via bus to the display driver. In order not to flood the driver unnecessarily, the request should only be sent again after a pause of 20 ms or longer.

```
<ABC>
REM Open graphical display and show message
OPEN WINDOW 272, 480
POKE "fcolor", 0                  // black
FILL RECT 0, 440, 272, 480
TEXT 5, 5, "Draw the Nikolaus puzzle by touching the screen!"
POKE "fcolor", 255                // white
FONT "Monospace,20"
TEXT 110, 450, "QUIT"

REM set up drawing color and switch to abc screen
POKE "color#1", DEC("0000FF")  // blue
POKE "fcolor", 1
POKE "lcd", 1

REM continue drawing lines until message box is pressed
NEW CURVE
REPEAT
    x = MOUSEX
    y = MOUSEY
    IF (x>-1 AND y<440) THEN
        LINE TO x, y
        REM wait until finger is released from screen
        REPEAT PAUSE 0.02 UNTIL (MOUSEX = -1)
    ENDIF
UNTIL (y > 440)

REM always make a clean exit
POKE "lcd", 0
CLOSE WINDOW
</ABC>
```

Listing 13.22: Bring a nice house of Santa Claus to the display of a cab printer Linienzug.lbl

13.12 A permanent abc loop

Important with examples like a parser is that abc in a *DO … LOOP* continuous loop has to work. An *abc* symbol is shown in the printer display during this time. The execution can be terminated by pressing the *Cancel* icon (or button on older printers).

13.13 The interaction between JScript and abc

So far, we have only ever learned about the interaction between JScript and abc in one direction. From abc, which should work as far as possible without inparallel processing JScript code, JScript lines are sent to the interpreter with the command *PRINT*.

But there is another way to interact. Values can be passed to an abc program working in the background and the return value is transferred to the JScript interpreter with the special function *[ABC:]*

13.13.1 Just a dream

Now one could assume that the first parameter is a function name, followed by the arguments to be passed to the function. The value returned by the BASIC command *RETURN* is then included in the contents of the JScript line as the result of the special function.

This could look like listing 13.23.

```
<ABC>
SUB Umklammern$(Text$)
    RETURN "<" + Text$ + ">"
END SUB
</ABC>
J
S e;0,0,30,32,100
H 100,0,T
O S
T:text;10,10,0,3,5;[SER:1]
; Nice, but this dream will never become true!
T 10,20,0,3,5;[ABC:Umklammern$,text]
A 5
```

Listing 13.23: Calling a BASIC sub routine directly from JScript isn't allowed

Unfortunately, it doesn't work that way because we have a thinking error in this example. abc is a BASIC compiler, not an interpreter. It is not possible for the JScript interpreter to jump into the abc code and execute a subroutine.

13.13.2 The correct way to interact (JGET$ and JPUT)

So we have to create an abc program ourselves that is constantly looking in the background that data is submitted by the JScript interpreter. In concrete terms, this is done as in a listing 13.24.

```
<ABC>
POKE "bypass", 1
DO
    REPEAT
        a$ = JGET$
    UNTIL (a$ <> "")
    JPUT "<" + a$ + ">"
LOOP
</ABC>

J
S e;0,0,30,32,1000
H 100,0,T
O S
T:text;10,10,0,3,5;[SER:1]
T 10,20,0,3,5;[ABC:text]
A 5
```

Listing 13.24: Interact with JGET$ and JPUT JGET-JPUT-Zusammenspiel.lbl

There is a function *JGET$*, which reads the input buffer, which is filled by the JScript special function *[ABC:]*. After passing the parameter to the buffer, the JScript interpreter stops until a return value is received via the abc function *JPUT*. Then the JScript interpreter continues its work and uses this return value.

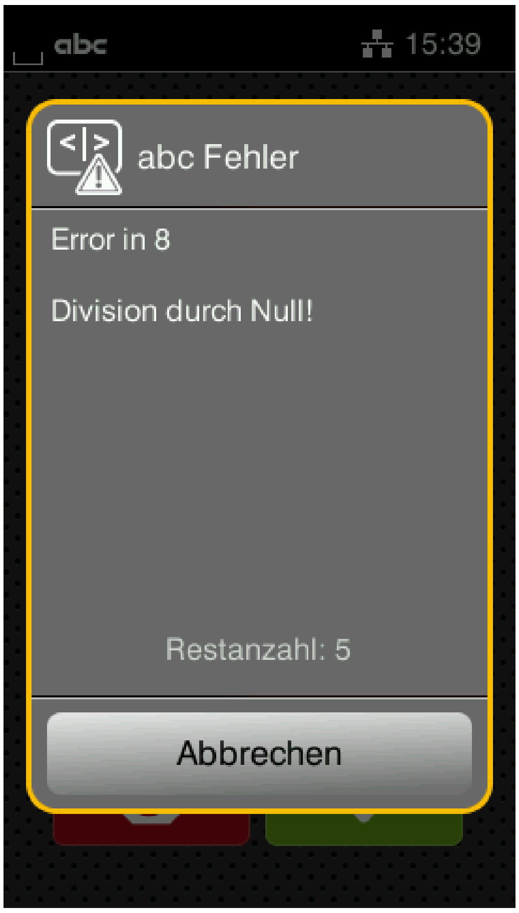
The only parameter allowed for the special function *[ABC:]* is a field identifier. In this example *text*.

The use of this special function is also only allowed once. It is not possible to calculate several values or even to use different calculation functions.

13.14 Error handling

Error handling in abc BASIC is only possible in a very simplified way. There is the command *ERROR* to abort the program with an error message. The abort corresponds to a complete reset, which also clears the JScript buffer. After the *ERROR* command a character string must follow, that is shown in the printer display. The printer automatically adds the line number from which the command was called.

Please note that the abc BASIC compiler only receives and numbers the lines after a <ABC> and before the closing </ABC> from the JScript interpreter. In the example from listing 13.25, line 8 is mentioned at the printer. It is the eighth line received by the abc BASIC compiler, but line 9 in the JScript code. The resulting message on the printer touchscreen is shown in figure 13.7.


abc Fehler

Error in 8

Division durch Null!

Restanzahl: 5

Figure 13.7: The abc command *ERROR* forces an error message onto the display

```
<ABC>
POKE "bypass",1
DO
    REPEAT
        a$ = JGET$
    UNTIL (a$ <> "")
    a = VAL(a$)
    IF (a = 0) THEN
        ERROR "Division durch Null!"
    ENDIF
    b$ = STR$(1/a)
    JPUT b$
LOOP
</ABC>

J
S l1; 0, 0, 68, 71, 100
H 100, 0, T
O R, S
T:Zahl; 10, 10, 0, 3, 5;[SER:5,-1][I]
T       10, 20, 0, 3, 5;1 / [Zahl] = [ABC:Zahl]
A 10
```

Listing 13.25: Forcing the printer to stop and display an error message ERROR.lbl

14 Appendix

The appendix lists some useful tables, sorted by topic. Use the appendix to this guide to look up and support your thoughts. This is not a complete collection. For a complete listing, please refer to the programming manual.

You can find the manual on the cab website:

`https://cab.de/en/programming`

14.1 A typical JScript label in detail

The following tables show the JScript commands of a typical label. Optional parameters are printed in italics and can be omitted.

The two commands **J** and **A** have been omitted. A can also have *[PREVIEW]* instead of a concrete number as a parameter, in which case the label is not printed but internally processed as a bitmap. In addition, since firmware 5.20 you can also give the A command a field name in square brackets as a parameter, then the number is taken from this “variable”.

The tables are taken from the programming manual, some of them are shortened. In the title line of the tables, the syntax of the programming manual is used. Optional parameters, which can also be omitted, are enclosed in curly brackets. You must not use the curly brackets in JScript code, they only indicate what you may omit. If the brackets include several parameters, they can only be used together or not at all.

For better readability, spaces were used in the syntax representation, which should not be used in the JScript code. A closing semicolon must not be followed by a space, because immediately after the last semicolon in a JScript command line follows the content of the text, barcode or the name of a file. If there is a space between the semicolon and the file name, the file will not be found. Spaces before text move the text by the spaces and barcodes are corrupted in their content.

Table 14.1: S = Label Size

S	*{ptype;}* xo, yo, ho, dy, wd *{,dx,col} {;name}*
ptype;	Gap sensor type (l0 = reflex below, l1 = transmitted light, l2 = reflex above, e = continuous material)
xo,	Offset in X direction (from the left)
yo,	Offset in Y direction (from above)
ho,	Label hight
dy,	Label start distance (height + gap)
wd	Label width
,dx	Distance from the edge of the first label to the edge of the next label in the horizontal direction
,col	Number of labels horizontally (columns) (default value = 1)
;name	Text that is shown on the printer display

Table 14.2: H = Setting heat level and printing speed

H	speed *{,h} {,t} {,s} {,Br}*
speed	The speed as an integer in mm/s (permitted values vary depending on the printer)
,h	Heat value in points (from -20 to +20), is added to the printer setting
,t	Type: T = transfer, D = direct thermal (default: setup value in the printer's menu)
,s	Ribbon saver on/off (R0=off, R1=on)
,Br	Retraction speed of the material in millimeters or in inches e.g. “B100” pulls the media back at 100 mm/s after printing, provided the printer is set to mm.

Table 14.3: O = setting options

O	*{Ax=y} {,B} {,Cx} {,D} {,E} {,F} {,Hx} {,J} {,Lx} {,M} {,N} {,P} {,R} {,S} {,Tx} {,U} {,Wy}*
Ax=y	see table on page 79 (applicator parameters)
,B	The printout on the underside of the material is identical to the printout on the top. (Only for double-sided printers)
,Cx	Setting the cutting depth on the perforation knife. Values for x = 0.0 – 10.0 (value that influences the cutting depth.)
,D	Printing or dispensing of labels always with return transport
,E	Ignore end of paper (not permitted when operating with continuous material)
,F	Discard label position (material is re-synchronized, only useful for double-sided printers)
,Hx	Additional offset between the upper and lower printhead in the transport direction. (Only available for double-sided printers) x = value in millimeters.
,J	Printing on demand (manual start via the display or interface)
,Lx	Length parameter that is used to compress or stretch the printout on the label. Specification in %. Values for x are from -5 to 5.
,M	Complete label content is mirrored
,N	Negative (inverted) printout of the complete label
,P	There is no return transport[20], faster printing, white stripe in the label possible
,R	Rotate the label by 180°
,S	Single label buffer. The next label will only be processed when the current label has been completely printed.
,Tx	Tear off mode – advances a printed label to make it easier to remove. x = optional offset, positive or negative value in mm or inches.
,U	Unique label - suppresses the pause/reprint option so that no label can be accidentally printed twice.
,Wy	Define waiting position after job. y = n = next label start, y = i = at the end of the job. *Wi* can be defined with an offset. With the dispensing module, the offset is relative to the dispensing position. This command is only effective in connection with *P* (Peel Off), is then also effective for the following jobs and must be reset with *O Wi0*.

[20] No return transport is not completely correct, within the print order “is printed” if the next label is already known.

Table 14.4: T = Text

T	*{:name;}* x, y, r, font, size *{,effects}*;Text
:name;	Creates a reference (field name) to the content of the text in order to be able to use it later in special functions or the *R* command
x,	Position in X-direction (from left)
y,	Position in Y-direction (from above)
r,	Rotation angle in degrees (integer counterclockwise around the starting point)
font,	font number (3 = Swiss 721, 5 = Swiss 721 Bold, 596 = Monospace 821)
size	Height of the font (spacing of the p–k lines, see section *4.3.1 Texts*)
,b	bold
,s	slanted (works with every TrueType font)
,i	italic (an italic font file must be loaded into the printer's memory before)
,z	Left shear
,l	light (a proper font file must be loaded first)
,u	underline
,k	Kerning
,v	Text with vertical alignment (individual letters rotated by 90°)
,qn	Compress or expand text (given in percent). The default value is 100. Possible values: 10-1000
,hn	Width of a large "H", width n in millimeters or inches (float value).
,mn	Horizontal text spacing, width n in millimeters or inches (float value).
;text	the content (text) to be printed in the given codepage

Table 14.5: W = Textbox

W	*{:name;}* x, y, r, font, size *{,effects}*;Text
:name;	Creates a reference (field name) to the content of the text in order to be able to use it later in special functions or the *R* command
x,	Position in X-direction (from left)
y,	Position in Y-direction (from above)
r,	Rotation angle in degrees (integer counterclockwise around the starting point)
width,	width of the box (framing invisible rectangle)
height,	height of the box (framing invisible rectangle)
font,	font number (3 = Swiss 721, 5 = Swiss 721 Bold, 596 = Monospace 821)
size	Height of the font (spacing of the p–k lines, see section *4.3.1 Texts*)
;text	the content (text) to be positioned as HTML (find the possible HTML-Tags in table 4.1 on page 36)

Table 14.6: B = Barcode

B	*{:name;}* x, y, r, type*{+options}*, size;content
:name;	Creates a reference to the content of the barcode in order to be able to use it later in special functions or the *R* command
x,	Position in X-direction (from left)
y,	Position in Y-direction (from above)
r,	Rotationswinkel in Grad (nur Vielfache von 90° gegen den Uhrzeigersinn)
type,	The barcode type is specified here. Barcodes with uppercase letters are printed with their plain text line (readable characters usually below the actual barcode), while the plain text line is suppressed for barcodes in lowercase letters.
size;	The size can consist of one or more values, depending on the selected type (height and width of a line for 1-D codes or size of a module for 2-D codes).
content	content of the barcode (look for special rules when creating GS1 barcodes)

You can find an example in the section *4.3.5 Barcodes* on page 37. Please refer to the programming manual for the exact syntax. There you will find the information on all codes on over 100 pages, which would go far beyond the scope of these instructions.

Table 14.7: I = (auto loaded) Images

I	*{:name;}* x, y, r, mx, my, a ;image name
:name;	Creates a reference to the content of the barcode to be able to use it later in special functions or the *R* command.
x,	Position in X-direction (from left)
y,	Position in Y-direction (from above)
r,	Rotation angle in degrees (only multiples of 90° counterclockwise)
mx,	Horizontal magnification factor. Integer value 1-10. Multiplies the image size by this factor in the x direction.
my,	Vertical magnification factor. Integer value 1-10. Multiplies the image size by this factor in the y direction.
a;	Autoload (small *a* as parameter) allows you to call up an image from the memory card. The field remains empty if no graphic was found. It is necessary that the values for mx and my are set when using Autoload.
image name	The file name without the ending. The file must be located in the subfolder *images*.

The autoload option simplifies the use of a graphic. Normally, the graphic would first have to be loaded into the RAM so that it can then be positioned using the *I* command. With this original syntax, the parameters *mx* and *my* are optional.

```
M l IMG;Logo
I 10,10,0;Logo
```

The following line does the same as before:

```
I 10,10,0,1,1,a;Logo
```

Table 14.8: **G**. . . **;L:** = Lines

G	x, y, r; L:length, width *{,Start{,End}}*
x,	Position in X-direction (from left)
y,	Position in Y-direction (from above)
r,	Rotation angle in degrees (integer counterclockwise around the lower, left corner)
L:	Defines that we want a line
length,	Value in millimeters or inches
width	Value in millimeters or inches
,start	Line start design (s = rectangular, r = rounded, a = arrow)
,end	Line end design (s = rectangular, r = rounded, a = arrow)

Table 14.9: **G**. . . **;R:** = Rectangles

G	x,y,r;R:width,height{,hD {,vD}}
x,	Position in X-direction (from left)
y,	Position in Y-direction (from above)
r,	Rotation angle in degrees (integer counterclockwise around the lower, left corner)
R:	Defines a rectangle as a graphic element
width,	Width of the entire rectangle (horizontal edge length)
hight	Height of the entire rectangle (vertical edge length)
,hD	Horizontal edge thickness (thickness of the line)
,vD	Vertical edge thickness (thickness of the line)

Table 14.10: **G**. . . **;C:** = Circles

G	x,y,r;C:Radius1{,Radius2{,width}}
x,	Position in X-direction (from left)
y,	Position in Y-direction (from above)
r,	Rotation angle in degrees (integer counterclockwise around the center)
C:	Defines a circle as the graphic element
radius1	Horizontal radius
,radius2	Vertical radius
,width.	Width of the circle line

Circles (as well as lines or rectangles) can contain other options. For a complete overview, it is worth taking a look at the programming manual. In the section *4.3.8 Graphical elements (circles, lines and rectangles)* on page 43 you will find an example for the creative use of circles and ellipses including one Shading as an additional option.

14.2 Short view on special functions

Table 14.11: Date functions

Function	Result
[DATE...]	Print current date in national format
[DAY...]	Print the day of the month numerically (1–31)
[DAY02...]	Same as before, but always two digits
[DOFY...]	Print the numeric day of the year as three digits (001–366)
[ISODATE...]	Print the ISO date
[ISOORDINAL...]	Print the date in ISO ordinal format
[ODATE...]	Date with offset
[WDAY...]	Print the numeric weekday (0–6)
[wday...]	Print weekday name (0 = Sunday)
[wday2...]	Print weekday name, 2-digit short (e.g. So)
[wday3...]	Print weekday, 3-digit (e.g. Sun)
[ISOWDAY...]	Print numerical weekday (1–7)
[WEEK...]	Print week number (1–53)
[WEEK02...]	Same as before, but always two digits (01–53)
[OWEEK...]	Numerical week of year with offset
[mon...]	Name of the month, always 3 characters (e.g. Jan)
[month...]	Complete name of month (e.g. April)
[MONTH...]	The numerical month (1–12)
[MONTH02...]	Same as before, but always 2 digits
[YY...]	Two digit year (70–38)
[YYYY...]	Four digit year (1970-2038)
options	*DD,MM,YY* for an offset in days, months and years, separated from the function name by a colon

Example:

```
T 20,50,0,3,4;Mindestens haltbar bis: [MONTH02:0,6]/[YYYY:0,6]
```

Generates a text with a date in the format “02/2021”, calculated with an offset of 6 months in the font *Swiss 721 Regular* with 4 mm font size. The value *02/2021* would have been e.g. calculated on 08/20/2020.

A positive date or time offset always points to the future. Negative values are possible to calculate in the past.

Table 14.12: Time functions

Function	Result
[H12…]	Hour in 12-hour format (1–12)
[H24…]	Hour in 24-hour format (1–23)
[H012…]	Two digit version of the hour in 12-hour format (01–12)
[H024…]	Two digit version of the hour in 24-hour format (01–23)
[ISOTIME…]	Time in ISO format
[MIN…]	Minutes (00–59)
[SEC…]	Seconds (00–59)
[TIME…]	Time using the matching format to the printer's setup language
[XM…]	am / pm indicator
options	*HH,MM,SS* offset in hours, minutes and seconds

Table 14.13: Math functions

Function	Result
[+:op1,op2, …]	Addition
[-:op1,op2]	Subtraction
[*:op1,op2, …]	Multiplication
[/:op1,op2]	Division
[%:op1,op2]	Modulo
[|:op1,op2]	Logical OR (1 if at least one argument is 1, else 0)
[&:op1,op2]	Logical AND (0 if at least one argument is 0, else 1)
[<: op1,op2]	numerical comparison – less than (1=True, 0=FALSE)
[=:op1,op2]	numerical comparison – equal (1=True, 0=FALSE)
[>:op1,op2]	numerical comparison – greater than (1=True, 0=FALSE)
[MOD10:x]	Calculates and prints the modulo 10 check digit
[MOD36:x]	Calculates and prints the modulo 36 check digit
[MOD43:x]	Calculates and prints the modulo 43 check digit
[P:name,.,–]	Prints the value of the field and uses the characters after the comma for thousands, decimal sign and as a placeholder if the value has no fraction part
[R:*x*]	Rounding method, values for *x* are n = no rounding (default), u = rounding up, d = rounding down and m = commercial rounding (DIN 1333)
[==:text1,text2]	string comparison (1=TRUE, 0=FALSE)

Table 14.14: User query in standalone operation [?:...]

?:	x,y,z{,D}{,Lx}{,Mx}{,R}{,J}
x,	Text message to be displayed (max.16 characters)
y,	Optional default value that is shown in the display for the first query, otherwise the previous display is shown.
z	Defines how often the input must be entered.
,D	Erases last user input
,Lx	Max allowed length of input (x=1–200)
Mx...x	Masks the input according to the following scheme (see examples below) *0* = Numeric, decimal separators and characters *1* = Numbers only *2* = Lower case letters only *3* = Lower case letters and numbers *4* = Capital letter *5* = Alphanumeric with capital letters *6* = All letter, lower case and capital *7* = Alphanumeric with lower case and capital letters *8* = All characters allowed at this position *!* = Spaces are not allowed if the exclamation mark is after the M option.
,R	Repeats the query if a value was not found in the database.
,J	Repeats the prompt when the printer asks for the number of labels to print. (A[?,R]) defines a simple query loop for the number of labels to be printed.

The following are some examples of queries in standalone mode. The queries are to be written as special functions in the content of a text or barcode line.

Prompts for the **Artikelnr:** and sets the value 7733214 for the next 3 labels to be printed. After that, the input is cleared, which only appears the first time it is called:

```
[?:Artikelnr:,7733214,3,D]
```

Prompts for the **Artikelnr:** with the preset value of **Schraube**. The maximum length of the input here is 8 characters:

```
[?:Artikelnr:,Schraube,,L8]
```

Asks for the **number** with the preset value 7733214 and masks the input for purely numeric values:

```
[?:Nummer,7733214,,M1111111]
```

Prompts for **ArtNr?** and allows 3 digits and 4 capital letters during input:

```
[?:ArtNr?,,1,M1114444]
```

Prompts for **Article?** with no preset value, input limited to 7 characters and repeat query if database entry is not found:

```
[?:Artikel?,,1,M1111111,R,D]
```

Prompts for **Article?** with preset value 22003 and masks the input for 5 digits without spaces:

```
[?:Artikel?,22003,,,L5,M!11111]
```

Example of a simple loop:

```
m m
J
S l1; 0, 0, 68, 71, 100
T 10, 15, 0, 3, 10;[SER:1]
; Diese Abfrage wird nur einmal angezeigt
T 10, 30, 0, 3, 10;[?:Eingabe 1:]
; Diese Abfrage wird wiederholt
T 10, 45, 0, 3, 10;[?:Eingabe 2:,,,J]
A [?,R]
```

When the label is finished, displays the prompt again until the print job is terminated by *Cancel*.

14.3 Error codes for the Esc s command

If you send Esc *s* to the printer, you get a 9 character status back. The second character is the error code. The error code should always be a "-" (minus character), that indicates we have no error at all.

Table 14.15: Error codes for the Esc *s* command

Error code	Cause
-	No error
a	Applicator error: Applicator did not reach the upper position
b	Applicator error: Applicator did not reach the lower position
c	Applicator error: Vacuum plate is empty
d	Applicator error: Label not deposit
e	Applicator error: Host stop/error
f	Applicator error: Reflective sensor blocked
g	Applicator error: Tamp pad 90° error
h	Applicator error: Tamp pad 0° error
i	Applicator error: Table not in front position
j	Applicator error: Table not in rear position
k	Applicator error: Head liftet
l	Applicator error: Head down
m	Scan result negative
n	Network error
o	Pressure error
p	Wrong media
r	RFID - error
s	System error (during boot up check)
u	USB error
x	Stacker full - printer goes on Pause (only with a specified cutter)
B	abc error, caused by the ERROR command (example on page 128)
U	User error, e.g. caused by the abc code `POKE "usererror","Attention: User defined error message!"`

15 Solutions to some exercises

The exercises in this manual are used in the JScript training putting what you have learned into practice. We have not printed all exercises from the training in this manual and have not published a solution for all exercises. If you have any questions about individual exercises or sections of this manual, please contact our training or support team by e-mail or phone. You can find the contact details on our website.

`https://www.cab.de/en/information/contact/`

Solution 4.1 Three at one stroke

S. 25

Since your label dimensions were not known when creating this guide, we assume they would be 50 mm wide, 30 mm high and the gap would be 3 mm.

```
J
S 0, 0, 30, 33, 50
A 3
```

Solution 4.2 Error in label size

S. 25

Assuming your label would be 50 mm wide, 30 mm high and the gap would be 3 mm. Then two parameters have to be changed to enlarge the label hight by 20 mm.

```
    J
    S 0, 0, 50, 53, 50
    A 3
```

Now apparently 6 labels are printed. But this is not quite correct. If you specify a label height to the printer with the label size command, no attention is paid to gaps during this distance. So the printer passes the gap by the label height that is 20 mm too long and only then starts to search for the next label. You can easily try this. Add the following line before the line “A 3” (even if the contents may not mean anything to you):

```
T 5, 5, 0, 3, 3;Hello World!
```

You can now see that in fact every second label is skipped without being printed. There are three labels, but the height was given wrong.

Solution 4.3 Set a print area within the label

S. 26

The page definition in the JScript code must be as follows to limit the print area according to Figure 4.2 at page 26.

```
S l1; 9, 8, 35, 53, 63
```

Solution 4.6 Headstand of the labels

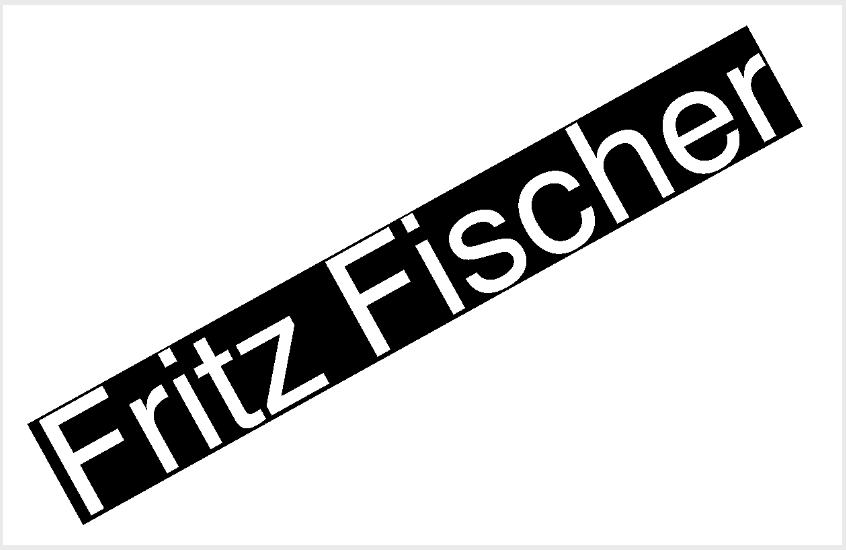
S. 28

If the line “O R” is omitted, the label is not printed rotated by 180°. On table printers the text is then upside down. So why is the normal state not like with the option? In the late 1990’s printers printed the label upside down. On these printers, the text was still readable without the option to rotate, even without standing upside down. Today, however, printers output the label forward, so the option “R” should usually always be selected. The cab printers do not have a setting “Rotate all labels 180°”, you have to specify this option with each label. If you use cabLabel for label design, this option can be set once for the printer and is automatically applied to each label.

Solution 4.7 Pretty weird

S. 30

Since when this PDF was created, it was not known how big your label is and what your name is, here is an example of the training labels.



```
m m
J
; The training labels are 100 mm wide and 68 mm high
S l1; 0, 0, 68, 71, 100
; Because of the rotation, the base point must be 7 mm from the edge
T 7, 65, 29, 3, 19, n;Fritz Fischer
A 1
```

Listing 15.1: Texts can also be printed crossways Quername.lbl

Solution 4.8 Handwritten name badges

S. 32

```
m m

; load the font from the default storage memory into RAM
M l FNT;IndieFlower-Regular

J
S l1; 0, 0, 30, 33, 100

; assign font no. 13
F 13;Indie Flower

; ... and use this font in the text elements
T 5, 15, 0, 13, 12;Fritz Fischer
T 5, 25, 0, 13,  5;Fischers Frischfisch GmbH

A 1
```

Listing 15.2: The file name was used for the load command and the name of the font family for the font number assignment. If you do not know the name of the font family, you can also specify the file name of the Regular font. NamensschildHandschrift.lbl

Solution 4.11 The house of Santa Claus

S. 44

You can draw Santa's house with only four "G" commands:

```
m m
J
S l1; 0, 0, 68, 71, 100
H 100, 0
O R

; large square (base)
G 40, 30,   0;R:    20,    20, 0.14, 0.14

; small square (roof)
G 40, 30,  45;R: 14.14, 14.14, 0.14, 0.14

; two small sides of the lowest triangle
G 50, 40, -45;L: 14.14,  0.14
G 50, 40, 225;L: 14.14,  0.14

A 1
```

Listing 15.3: The house of St. Nicholas can also be drawn as a structure of two squares and a triangle. However, since there is no triangle command, a total of four graphic commands are required. Nikolaushaus.lbl

Solution 5.1 Handwritten name badges in stand-alone operation

S. 55

The JScript file from the exercise “Handwritten name badges” must be supplemented by the special function for dialog input. At the end of the file, the command *A* without parameters is used to print an unlimited number of labels. If the third parameter in the dialog function is a 1, the system prompts for a new parameter after each print.

```
m m
M l FNT;IndieFlower-Regular
J
S l1; 0, 0, 30, 33, 100
F 13;Indie Flower
; Third parameter for re-query after each label
T 5, 15, 0, 13, 12;[?:Vorname und Name,Fritz Fischer,1]
T 5, 25, 0, 13,  5;[?:Firmenname,Fischers Frischfisch GmbH,1]
; Unlimited printing leads to re-query the values each time
A
```

Listing 15.4: The additional parameter repeats the query for each print ErneuteAbfrage.lbl

Solution 5.2 Complex numbering of packages

S. 56

And this is how a solution could look like to apply two labels to all four cartons.

```
m m
J
S l1; 0, 0, 10, 13, 100
H 150, 0
O R
T 15, 9, 0, 5, 9;[SER: 01, 5, 2] bis [SER: 05, 5, 2] von 20
A 8
```

Listing 15.5: Two labels for the outer boxes of each shipment of goods Versandkartons.lbl

Solution 5.3 Variable number of packages

S. 59

In the exercise 5.2, two labels with the dimensions 100 mm width and 10 mm height per box are required.

```
 1 m m
 2 J
 3 S l1;0,0,10,13,100
 4 H 150,0
 5 O R
 6
 7 ; query dialog
 8 T:Gesamt;0,0,0,3,3;[?:Anzahl Packstücke?,20][I]
 9 T:JeKarton;0,0,0,3,3;[?:Stücke je Karton?,5][I]
10
11 ; define counter
12 T:KartonNr;0,0,0,3,3;[SER:00000,1,2][I]
13
14 ; do the calculation
15 T:Hilfsvariable;0,0,0,3,3;[*:KartonNr,JeKarton][D:1,0][I]
16 T:VON;0,0,0,3,3;[+:Hilfsvariable,1][D:1,0][I]
17 T:BIS;0,0,0,3,3;[+:Hilfsvariable,JeKarton][D:1,0][I]
18
19 ; print onto the label
20 T 15,9,0,5,9;[VON] bis [BIS] von [Gesamt]
21
22 ; calculate and print the number of labels required
23 T:AnzahlKartons;0,0,0,3,3;[/:Gesamt,JeKarton][D:1,0][I]
24 T:AnzahlEtiketten;0,0,0,3,3;[*:AnzahlKartons,2][D:1,0][I]
25 A [AnzahlEtiketten]
```

Listing 15.6: Labeling with variable number of packages Packstuecke.lbl

To calculate with integers, the special function `[D:1,0]` was added after each calculation, which formats the result with at least one digit before but no decimal place. The solution thus found always works if the number of units per box is an integer divisor of the total number, i.e., the last box can be filled completely. With the code, test packing 17 pieces at 4 pieces per box. Better would be (the first 13 lines are identical):

```
14 ; do a more advanced calculation
15 T:Hilfsvariable;0,0,0,3,3;[*:KartonNr,JeKarton][D:1,0][I]
16 T:VON;0,0,0,3,3;[+:Hilfsvariable,1][D:1,0][I]
17 T:BIS;0,0,0,3,3;[+:Hilfsvariable,JeKarton][D:1,0][I]
18 T:ÜberBIS;0,0,0,3,3;[>:BIS,Gesamt][I]
19
20 ; print onto the label
21 T 15,9,0,5,9;[VON] bis [BIS] von [Gesamt][I:ÜberBIS]
22 T 15,9,0,5,9;[VON] bis [Gesamt] von [Gesamt][I:!ÜberBIS]
23
24 ; calculate and print the number of labels required more clever
25 T:AnzahlVolleKartons;0,0,0,3,3;[/:Gesamt,JeKarton][D:1,0][I]
26 T:Krümelchen;0,0,0,3,3;[%:Gesamt,JeKarton][D:1,0][I]
27 T:Zusatzkarton;0,0,0,3,3;[>:Krümelchen,0][I]
28 T:AnzahlAlleKartons;0,0,0,3,3;[+:AnzahlVolleKartons,Zusatzkarton][D:1,0][I]
29 T:AnzahlEtiketten;0,0,0,3,3;[*:AnzahlAlleKartons,2][D:1,0][I]
30 A [AnzahlEtiketten]
```

Listing 15.7: Extended version for a variable number of packages PackstueckePRO.lbl

Solution 5.4 Avoid errors in multi-step calculations

S. 61

The problem lies in the formatting of the intermediate results. If this is not specified, JScript always chooses a display with two decimal places. This would correspond to the special function `[D:1,2]`.

We can start here and provide line 16 with more decimal places accordingly.

```
16 T:Einzelpreis; 0,  0, 0, 3, 3;[I][/:GPZG,AnzahlZG][D:1,8]
```

Solution 6.1 A CSV file as protocol of all printouts

S. 66

Here is a possible solution for a running event.

```
; usual header
m m
J
S l1;0,0,68,71,100
H 150,0

; define a log file
E LOG;Spendenlauf

; query data from a display dialog
T:Vorname;0,0,0,3,3;[?:Vorname][I]
T:Name;0,0,0,3,3;[?:Name][I]
T:Straße;0,0,0,3,3;[?:Straße][I]
T:HNR;0,0,0,3,3;[?:Hausnummer][I]
T:PLZ;0,0,0,3,3;[?:Postleitzahl][I]
T:Ort;0,0,0,3,3;[?:Wohnort][I]
T:Anonym;0,0,0,3,3;[?:Nachname kürzen? (ja/nein),nein][I]

; logic for shortening the name (string comparison)
T:Kürzen;0,0,0,3,3;[==:Anonym,ja][I]

; generate start number
T:AlteNummer;0,0,0,3,3;[RUSER][I]
T:Startnummer;0,0,0,3,3;[+:1,AlteNummer][D:1,0][WUSER][I]

; write data into log file
T 0,0,0,3,3;[Startnummer],[Vorname],[Name],[Straße],[HNR],[PLZ],[Ort],[DATE],[TIME][WLOG][I]

; print the label
T 0,20,0,3,10;[Vorname] [Name][J:c100][I:Kürzen]
T 0,20,0,3,10;[Vorname] [Name,1,1].[J:c100][I:!Kürzen]
T 0,55,0,5,40;[Startnummer][J:c100]

; repeat in an infinite loop, reloading this JScript file
A 1
M r
```

Listing 15.8: Label creates a CSV file of all runners Spendenlauf.lbl

Solution 6.2 The cherry on top for the JScript expert

S. 66

First of all, the solution already worked with the user memory. This memory is technically located in the RAM of the buffered clock. Even when the printer is switched off, voltage is applied there so that the clock chip can keep the time running. Therefore, the memorized value of the last start number is already protected against accidentally switching off the printer.

It becomes more complicated with the automatic font size adjustment. Here (within the scope of this manual) only a single distinction between two sizes is to be made in order not to make the process too complex. The first 18 lines are identical to the previous solution.

```
; logic for shortening the name (string comparison)
T:Kürzen;0,0,0,3,3;[==:Anonym,ja][I]
T:Namensliste;0,0,0,3,3;[Vorname] [Name][U:GS][Vorname] [Name,1,1].[I]
T:Namenswahl;0,0,0,3,3;[+:Kürzen,1][I]

; generate start number
T:AlteNummer;0,0,0,3,3;[RUSER][I]
T:Startnummer;0,0,0,3,3;[+:1,AlteNummer][D:1,0][WUSER][I]

; write data into log file
T 0,0,0,3,3;[Startnummer],[Vorname],[Name],[Straße],[HNR],[PLZ],[Ort],[DATE],[TIME…
    ][WLOG][I]

; is there enough space for font height 10 (max. 22 characters)?
T:VollerName;0,0,0,3,3;[SPLIT:Namensliste,Namenswahl][I]
T:Zeichenlänge;0,0,0,3,3;[LEN:VollerName][I]
T:ZuLang;0,0,0,3,3;[>:Zeichenlänge,22][I]

; print the label (compress names to 85% or 50% character width)
T 0,20,0,3,10,q85;[VollerName][J:c100][I:ZuLang]
T 0,20,0,3,8,q50;[VollerName][J:c100][I:!ZuLang]
T 0,55,0,5,40;[Startnummer][J:c100]

; repeat in an infinite loop, reloading this JScript file
A 1
M r
```

Listing 15.9: Extended version of the donation run label SpendenlaufPRO.lbl

The decisive factor is that only one conditional visibility can be used per text element. Therefore, both decisions (last name shortened and string length) must be combined to one decision about the font height. For this purpose, a list is generated which contains both the full name and the anonymized name. The evaluation of the user input (field name “Anonym”) is then used to access the first or second element of the list.

Deutschland
cab Produkttechnik GmbH & Co KG
Karlsruhe
Tel. +49 721 6626 0
www.cab.de

USA
cab Technology, Inc.
Chelmsford, MA
Tel. +1 978 250 8321
www.cab.de/us

Taiwan
cab Technology Co., Ltd.
Taipeh
Tel. +886 (02) 8227 3966
www.cab.de/tw

China
cab (Shanghai) Trading Co., Ltd.
Guangzhou
Tel. +86 (020) 2831 7358
www.cab.de/cn

Frankreich
cab Technologies S.à.r.l.
Niedermodern
Tel. +33 388 722501
www.cab.de/fr

Mexiko
cab Technology, Inc.
Juárez
Tel. +52 656 682 4301
www.cab.de/es

China
cab (Shanghai) Trading Co., Ltd.
Shanghai
Tel. +86 (021) 6236 3161
www.cab.de/cn

Südafrika
cab Technology (Pty) Ltd.
Randburg
Tel. +27 11 886 3580
www.cab.de/za

© cab