
Intermec
Fingerprint®
v8.70.0 / v10.0.0
## Contents

### 1 Introduction
- Fingerprint Programming Language ............................................................. 2
- Fingerprint Documentation ............................................................................. 2
- Differences between Fingerprint 8.70.0 and 10.0.0 ........................................... 3
- News in Intermec Fingerprint v8.70.0 .............................................................. 3
- General Improvements .................................................................................... 3
- New Instructions .............................................................................................. 3
- Modified Instructions ...................................................................................... 3
- File System with Directories .......................................................................... 4
- Files in Intermec Fingerprint v8.70.x and v10.0.X ............................................ 5

### 2 Program Instructions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>8</td>
</tr>
<tr>
<td>ACTLEN</td>
<td>9</td>
</tr>
<tr>
<td>ALIGN (AN)</td>
<td>10</td>
</tr>
<tr>
<td>ASC</td>
<td>11</td>
</tr>
<tr>
<td>BARADJUST</td>
<td>14</td>
</tr>
<tr>
<td>BARCODENAMES$</td>
<td>15</td>
</tr>
<tr>
<td>BARFONT (BF)</td>
<td>16</td>
</tr>
<tr>
<td>BARFONT ON/OFF (BF ON/OFF)</td>
<td>17</td>
</tr>
<tr>
<td>BARHEIGHT (BH)</td>
<td>20</td>
</tr>
<tr>
<td>BARMAG (BM)</td>
<td>21</td>
</tr>
<tr>
<td>BARRATIO (BR)</td>
<td>22</td>
</tr>
<tr>
<td>BARSER</td>
<td>23</td>
</tr>
<tr>
<td>BARTYPE (BT)</td>
<td>24</td>
</tr>
<tr>
<td>BEEP</td>
<td>25</td>
</tr>
<tr>
<td>BREAK</td>
<td>26</td>
</tr>
<tr>
<td>BREAK ON/OFF</td>
<td>27</td>
</tr>
<tr>
<td>BUSY</td>
<td>28</td>
</tr>
<tr>
<td>CHDIR</td>
<td>29</td>
</tr>
<tr>
<td>CHECKSUM</td>
<td>30</td>
</tr>
<tr>
<td>CHRS$</td>
<td>31</td>
</tr>
<tr>
<td>CLEANFEED</td>
<td>32</td>
</tr>
<tr>
<td>CLEAR</td>
<td>33</td>
</tr>
<tr>
<td>CLIP</td>
<td>34</td>
</tr>
<tr>
<td>CLL</td>
<td>35</td>
</tr>
<tr>
<td>CLOSE</td>
<td>36</td>
</tr>
<tr>
<td>COM ERROR ON/OFF</td>
<td>37</td>
</tr>
<tr>
<td>COMSET</td>
<td>38</td>
</tr>
<tr>
<td>COMSET OFF</td>
<td>39</td>
</tr>
<tr>
<td>COMSET ON</td>
<td>40</td>
</tr>
<tr>
<td>COMSTAT</td>
<td>41</td>
</tr>
<tr>
<td>CONT</td>
<td>42</td>
</tr>
<tr>
<td>COPY</td>
<td>43</td>
</tr>
<tr>
<td>COUNT&amp;</td>
<td>44</td>
</tr>
<tr>
<td>CURDIR$</td>
<td>45</td>
</tr>
<tr>
<td>CSUM</td>
<td>46</td>
</tr>
<tr>
<td>CUT</td>
<td>47</td>
</tr>
<tr>
<td>CUT ON/OFF</td>
<td>48</td>
</tr>
<tr>
<td>DATE$</td>
<td>49</td>
</tr>
<tr>
<td>DATEADD$</td>
<td>50</td>
</tr>
<tr>
<td>DATEADD$</td>
<td>51</td>
</tr>
</tbody>
</table>

Intermec Fingerprint v8.70.0/v10.0.0 Programmer’s Reference Manual
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATEDIFF</td>
<td>59</td>
</tr>
<tr>
<td>DBBREAK</td>
<td>60</td>
</tr>
<tr>
<td>DBBREAK OFF</td>
<td>61</td>
</tr>
<tr>
<td>DBEND</td>
<td>62</td>
</tr>
<tr>
<td>DBSTDIO</td>
<td>63</td>
</tr>
<tr>
<td>DBSTEP</td>
<td>64</td>
</tr>
<tr>
<td>DELETE</td>
<td>65</td>
</tr>
<tr>
<td>DELETEPFSVAR</td>
<td>66</td>
</tr>
<tr>
<td>DEVICES</td>
<td>67</td>
</tr>
<tr>
<td>DIM</td>
<td>70</td>
</tr>
<tr>
<td>DIR</td>
<td>71</td>
</tr>
<tr>
<td>DIRNAME$</td>
<td>74</td>
</tr>
<tr>
<td>END</td>
<td>75</td>
</tr>
<tr>
<td>EOF</td>
<td>76</td>
</tr>
<tr>
<td>ERL</td>
<td>77</td>
</tr>
<tr>
<td>ERR</td>
<td>78</td>
</tr>
<tr>
<td>ERR$</td>
<td>79</td>
</tr>
<tr>
<td>ERROR</td>
<td>80</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>82</td>
</tr>
<tr>
<td>FIELD</td>
<td>83</td>
</tr>
<tr>
<td>FIELDNO</td>
<td>84</td>
</tr>
<tr>
<td>FILE&amp; LOAD</td>
<td>85</td>
</tr>
<tr>
<td>FILENAMES$</td>
<td>86</td>
</tr>
<tr>
<td>FILES</td>
<td>87</td>
</tr>
<tr>
<td>FLOATCALC$</td>
<td>89</td>
</tr>
<tr>
<td>FONT (FT)</td>
<td>90</td>
</tr>
<tr>
<td>FONTD</td>
<td>92</td>
</tr>
<tr>
<td>FONTNAME$</td>
<td>93</td>
</tr>
<tr>
<td>FONTS</td>
<td>94</td>
</tr>
<tr>
<td>FOR...TO...NEXT</td>
<td>95</td>
</tr>
<tr>
<td>FORMAT</td>
<td>96</td>
</tr>
<tr>
<td>FORMAT DATES$</td>
<td>98</td>
</tr>
<tr>
<td>FORMAT INPUT</td>
<td>99</td>
</tr>
<tr>
<td>FORMAT TIMES$</td>
<td>100</td>
</tr>
<tr>
<td>FORMAT$</td>
<td>101</td>
</tr>
<tr>
<td>FORMFEED (FF)</td>
<td>104</td>
</tr>
<tr>
<td>FRE</td>
<td>105</td>
</tr>
<tr>
<td>FUNCTEST</td>
<td>106</td>
</tr>
<tr>
<td>FUNCTEST$</td>
<td>107</td>
</tr>
<tr>
<td>GET</td>
<td>108</td>
</tr>
<tr>
<td>GETASSOC$</td>
<td>109</td>
</tr>
<tr>
<td>GETASSOCNAMES$</td>
<td>110</td>
</tr>
<tr>
<td>GETPFSVAR</td>
<td>111</td>
</tr>
<tr>
<td>GOSUB</td>
<td>112</td>
</tr>
<tr>
<td>GOTO</td>
<td>114</td>
</tr>
<tr>
<td>HEAD</td>
<td>115</td>
</tr>
<tr>
<td>HEAD, cont.</td>
<td>116</td>
</tr>
<tr>
<td>IF...THEN...(ELSE)</td>
<td>117</td>
</tr>
<tr>
<td>IMAGE BUFFER MIRROR</td>
<td>119</td>
</tr>
<tr>
<td>IMAGE BUFFER SAVE</td>
<td>120</td>
</tr>
<tr>
<td>IMAGE LOAD</td>
<td>121</td>
</tr>
<tr>
<td>IMAGENAME$</td>
<td>122</td>
</tr>
<tr>
<td>IMAGES</td>
<td>123</td>
</tr>
<tr>
<td>IMMEDIATE</td>
<td>124</td>
</tr>
<tr>
<td>INKEY$</td>
<td>127</td>
</tr>
<tr>
<td>INPUT (IP)</td>
<td>128</td>
</tr>
<tr>
<td>INPUT ON/OFF</td>
<td>130</td>
</tr>
<tr>
<td>INPUT#</td>
<td>131</td>
</tr>
</tbody>
</table>
Contents

INPUT$ .......................................................... 132
INSTR ......................................................... 133
INVIMAGE (II) ................................................. 134
KEY BEEP ..................................................... 135
KEY ON/OFF ................................................ 136
KEYBMAP$ .................................................. 137
KILL ............................................................. 139
LAYOUT ......................................................... 141
LAYOUT END ............................................... 144
LAYOUT INPUT ............................................ 145
LAYOUT RUN ............................................... 146
LBLCOND .................................................... 147
LED ON/OFF ................................................ 149
LEFT$ .......................................................... 150
LEN ............................................................. 151
LET ............................................................. 152
LINE INPUT .................................................. 153
LINE INPUT# ................................................ 154
LIST ........................................................... 155
LISTPF$VAR ................................................ 157
LOAD .......................................................... 158
LOC ............................................................ 159
LOF ............................................................. 160
LSET ........................................................... 161
LTS& ON/OFF .............................................. 162
MAG ............................................................ 163
MAKEASSOC ............................................ 164
MAP ........................................................... 165
MERGE ......................................................... 167
MIBVAR& .................................................... 168
MID$ ........................................................... 169
MKDIR ........................................................ 170
NAME DATES ............................................... 171
NAME WEEKDAY$ ...................................... 172
NASC .......................................................... 173
NASCD ......................................................... 175
NEW ........................................................... 176
NORIMAGE (NI) .......................................... 177
ON BREAK GOSUB .................................... 178
ON COMSET GOSUB .................................. 179
ON ERROR GOTO ....................................... 181
ON GOSUB .................................................. 182
ON GOTO ...................................................... 183
ON HTTP GOTO ......................................... 184
ON KEY GOSUB ........................................ 185
ON/OFF LINE ............................................. 187
ON MIBVAR& GOSUB .................................. 188
OPEN ........................................................ 189
OPTIMIZE BATCH ON/OFF ......................... 192
PORTIN ....................................................... 193
PORTOUT ON/OFF ...................................... 194
PRBAR (PB) .................................................. 195
PRBOX (PX) .................................................. 196
PRBUF ........................................................ 201
PRIMAGE (PM) ............................................ 202
PRINT (?) ..................................................... 202
PRINT KEY ON/OFF ................................... 204
PRINT# ....................................................... 205
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRINTFEED (PF)</td>
<td>206</td>
</tr>
<tr>
<td>PRINTONE</td>
<td>208</td>
</tr>
<tr>
<td>PRINTONE#</td>
<td>209</td>
</tr>
<tr>
<td>PRLINE (PL)</td>
<td>210</td>
</tr>
<tr>
<td>PRPOS (PP)</td>
<td>211</td>
</tr>
<tr>
<td>PRSTAT</td>
<td>213</td>
</tr>
<tr>
<td>PRTXT (PT)</td>
<td>215</td>
</tr>
<tr>
<td>PUT</td>
<td>217</td>
</tr>
<tr>
<td>RANDOM</td>
<td>218</td>
</tr>
<tr>
<td>RANDOMIZE</td>
<td>219</td>
</tr>
<tr>
<td>READY</td>
<td>220</td>
</tr>
<tr>
<td>REBOOT</td>
<td>221</td>
</tr>
<tr>
<td>REDIRECT OUT</td>
<td>222</td>
</tr>
<tr>
<td>REM (' )</td>
<td>223</td>
</tr>
<tr>
<td>REMOVE IMAGE</td>
<td>224</td>
</tr>
<tr>
<td>RENDER ON/OFF</td>
<td>225</td>
</tr>
<tr>
<td>RENUM</td>
<td>226</td>
</tr>
<tr>
<td>REPRINT ON/OFF</td>
<td>227</td>
</tr>
<tr>
<td>RESUME</td>
<td>228</td>
</tr>
<tr>
<td>RETURN</td>
<td>229</td>
</tr>
<tr>
<td>REWINDCONTROL</td>
<td>230</td>
</tr>
<tr>
<td>REWINDVOID</td>
<td>231</td>
</tr>
<tr>
<td>RIGHT$</td>
<td>232</td>
</tr>
<tr>
<td>RSET</td>
<td>233</td>
</tr>
<tr>
<td>RUN</td>
<td>234</td>
</tr>
<tr>
<td>SAVE</td>
<td>235</td>
</tr>
<tr>
<td>SET FAULTY DOT</td>
<td>236</td>
</tr>
<tr>
<td>SETASSOC</td>
<td>237</td>
</tr>
<tr>
<td>SETPFSVAR</td>
<td>238</td>
</tr>
<tr>
<td>SETSTDIO</td>
<td>239</td>
</tr>
<tr>
<td>SETUP</td>
<td>241</td>
</tr>
<tr>
<td>SETUP GET</td>
<td>244</td>
</tr>
<tr>
<td>SETUP KEY</td>
<td>245</td>
</tr>
<tr>
<td>SETUP WRITE</td>
<td>246</td>
</tr>
<tr>
<td>SGN</td>
<td>248</td>
</tr>
<tr>
<td>SORT</td>
<td>249</td>
</tr>
<tr>
<td>SOUND</td>
<td>250</td>
</tr>
<tr>
<td>SPACE$</td>
<td>251</td>
</tr>
<tr>
<td>SPLIT</td>
<td>252</td>
</tr>
<tr>
<td>STOP</td>
<td>253</td>
</tr>
<tr>
<td>STORE IMAGE</td>
<td>254</td>
</tr>
<tr>
<td>STORE INPUT</td>
<td>255</td>
</tr>
<tr>
<td>STORE OFF</td>
<td>256</td>
</tr>
<tr>
<td>STRS$</td>
<td>257</td>
</tr>
<tr>
<td>STRING$</td>
<td>258</td>
</tr>
<tr>
<td>SYSHEALTH</td>
<td>259</td>
</tr>
<tr>
<td>SYSHEALTH$</td>
<td>261</td>
</tr>
<tr>
<td>SYSVAR</td>
<td>262</td>
</tr>
<tr>
<td>TAGFIELD</td>
<td>270</td>
</tr>
<tr>
<td>TAGFORMAT</td>
<td>272</td>
</tr>
<tr>
<td>TAGPROTECT</td>
<td>274</td>
</tr>
<tr>
<td>TAGREAD</td>
<td>275</td>
</tr>
<tr>
<td>TAGWRITE</td>
<td>276</td>
</tr>
<tr>
<td>TESTFEED</td>
<td>277</td>
</tr>
<tr>
<td>TICKS</td>
<td>278</td>
</tr>
<tr>
<td>TIMES$</td>
<td>279</td>
</tr>
<tr>
<td>TIMEADDS$</td>
<td>280</td>
</tr>
<tr>
<td>TIMEDIFF</td>
<td>281</td>
</tr>
</tbody>
</table>
Contents

3 Image Transfer

Protocols .................................................................................................................. 302
Image Format .......................................................................................................... 303
UBI10 .................................................................................................................... 306
PRBUF Protocol .................................................................................................... 308

4 Character Sets and Fonts

Character Sets .................................................................................................... 312
UTF-8 Character Set .......................................................................................... 336
Scaleable Fonts ................................................................................................... 339
Bitmap Fonts ....................................................................................................... 339
Font Aliases ......................................................................................................... 339
OCR-A BT Character Set ................................................................................... 340
OCR-B 10 Pitch BT Character Set ..................................................................... 341
DingDings SWA Character Set ........................................................................... 342

5 Bar Codes

Introduction ......................................................................................................... 344
Supported Bar Codes .......................................................................................... 345
One-and Two-Dimensional Bar Codes ................................................................ 347
   Code 39 .......................................................................................................... 348
   Code 128 ....................................................................................................... 349
   Data Matrix .................................................................................................. 351
   EAN-8 .......................................................................................................... 352
   EAN-13 ....................................................................................................... 353
   EAN 128 ..................................................................................................... 354
   Interleaved 2 of 5 ........................................................................................ 356
   MaxiCode .................................................................................................... 357
   MicroPDF417 .............................................................................................. 359
   PDF417 ....................................................................................................... 361
   QR Code ...................................................................................................... 363
   RSS-14 ....................................................................................................... 364
   RSS-14 Truncated ......................................................................................... 365
   RSS-14 Stacked ............................................................................................ 366
   RSS-14 Stacked Omnidirectional ................................................................. 367
   RSS-14 Limited ............................................................................................ 368
Contents

RFID Tag Formats
  RFID Tags ..................................................................................................................... 371
    EPCGlobal Tag Format Specifications ................................................................. 371
    Uniform Resource Identifier (URI) ........................................................................ 406
    Other EPC Tag Input Methods .............................................................................. 410
    EPC Tag Writing Example ..................................................................................... 410
    Tag Memory Allocation .......................................................................................... 411
    Non-standard Tag Formats ..................................................................................... 416

Error Messages
  Error table ................................................................................................................... 418

Keyboards
  EasyCoder PF2i/4i/4ci ................................................................................................. 422
    Actual Keyboard Appearance ............................................................................... 422
    ID Numbers ........................................................................................................... 422
    Position Numbers ................................................................................................. 423
    ASCII Values ......................................................................................................... 423
  EasyCoder PM4i ........................................................................................................ 424
    Actual Keyboard Appearance ............................................................................... 424
    ID Numbers ........................................................................................................... 424
    Position Numbers ................................................................................................. 425
    ASCII Values ......................................................................................................... 425
  EasyCoder PX4i/6i ..................................................................................................... 426
    Actual Keyboard Appearance ............................................................................... 426
    ID Numbers ........................................................................................................... 426
    Position Numbers ................................................................................................. 427
    ASCII Values ......................................................................................................... 427
  EasyCoder PX4i/6i Alphanumeric keyboard ............................................................ 428
Contents

Actual Keyboard Appearance ................................................................. 428
Position Numbers ................................................................................ 428
ID Numbers and ASCII Values ........................................................... 428
Mapping the Keyboard ........................................................................ 429
EasyCoder PD41 .................................................................................. 430

Index

Index ........................................................................................................ 432
This chapter gives a quick introduction to the Intermec Fingerprint v8.70.0/v10.0.0 programming languages, lists available documentation regarding Fingerprint, and describes the most important differences between this version and the previously documented version of Fingerprint (v8.60).

It also lists the files included in the Fingerprint v8.70.0/v10.0.0 firmware package.
Chapter 1—Introduction

Fingerprint Programming Language

Intermec Fingerprint is a BASIC-inspired, printer-resident programming language that has been developed for use with the Intermec EasyCoder bar code label printers. Version 8.70.0 is supported by EasyCoder PF2/4i, EasyCoder PM4i, and EasyCoder PX4/6i printers. Version 10.0.0 has been developed for the EasyCoder PD41 (release in January 2006). Other Intermec printers may require version 6.XX or 7.XX of Fingerprint, or different firmware altogether. See your printer documentation.

The Intermec Fingerprint firmware is an easy-to-use intelligent programming tool for label formatting and printer customizing, which allows you to design your own label formats and write your own printer application software. You may easily create a printer program by yourself that exactly fulfils your own unique requirements. Improvements or changes due to new demands can be implemented quickly and without vast expenses.

Intermec Fingerprint also contains an easy-to-use slave protocol, called Intermec Direct Protocol. It allows layouts and variable data to be downloaded from the host and combined into labels, tickets and tags with a minimum of programming. Intermec Direct Protocol also includes a versatile error handler and a flexible counter function.

Fingerprint Documentation

This *Programmer’s Reference Manual* contains detailed information on all programming instructions in the Intermec Fingerprint programming language in alphabetical order. It also contains other program-related information that is common for all Fingerprint-compatible printer models from Intermec.

Those who wish to get started with Fingerprint programming and printer control are recommended to, in addition to this manual, refer to the *Fingerprint v8.XX Tutorial*.

The Intermec Direct Protocol, while being a subset of Intermec Fingerprint, is described in more detail in the *Intermec Direct Protocol v8.XX Programmer’s Reference Manual*.

All information needed by the operator, like how to run the printer, how to load the media or ribbon supply and how to maintain the printer, can be found in the User’s Guide for the printer model in question. You will also find information on installation, setup, print resolution, media specifications, and other technical information, which is specific for each printer model.

All these manuals are available for download from the Intermec Global Sales and Service web site, see [www.intermec.com](http://www.intermec.com).
Chapter 1—Introduction

Differences between Fingerprint 8.70.0 and 10.0.0

Fingerprint 10.0.0 has been designed for use with the EasyCoder PD41. Although most commands work exactly the same as for Fingerprint 8.XX series, the differences in the printer platforms cause the firmware to have some important differences. Some general differences are listed below:

- The EasyCoder PD41 has no display. Commands to control the display can be used and the results be read over the network (SNMP).
- The EasyCoder PD41 has no beeper. Commands BEEP and SOUND will be silent, yet create equally long delays.
- The EasyCoder PD41 has only a Print button. Applications requiring user input on the printer keyboard will therefore not work. Changing SETUP parameters through the console is not possible.
- The EasyCoder PD41 does not support as many optional interface cards. This affects the devices available.
- The EasyCoder PD41 does not send a power fail signal. Thus variables cannot be saved with the SETPFSVAR unless the printer is rebooted using the REBOOT command. This naturally affects the functionality of GETPFSVAR, DELETEPFSVAR and LISTPFSVAR.

News in Intermec Fingerprint v8.70.0

A new naming convention for Fingerprint version numbers is introduced. The last digit is added and is stepped up for new versions containing only corrections and no new functionality. A new version with only bug fixes would thus be called 8.70.1. A new Fingerprint version containing new functionality causes the the middle digits to be stepped up, for example 8.71.0.

General Improvements

- EPC Gen 2 RFID support.
- Ability to disable Setup access from console
- Enhanced input from control panel

New Instructions

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETUP KEY</td>
<td>Enable/disable setup access via console.</td>
</tr>
</tbody>
</table>

Modified Instructions

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAGFIELD</td>
<td>New Gen 2 segment names.</td>
</tr>
<tr>
<td>TESTFEED</td>
<td>Identifies RFID media.</td>
</tr>
</tbody>
</table>
Chapter 1—Introduction

File System with Directories

Two parts of the printer's memory support the use of directories, namely the read-only memory (rom) and the read/write permanent storage memory (c). Directories cannot be used in any other parts of the memory or in CompactFlash memory cards (card1).

The slash letter (/) is used as a divisor between directories and files, that is, the path "/c/DIR1/DIR2/FILE" refers to a file or directory named FILE in the directory DIR2, which in its turn is located in the directory DIR1 in the root of the device /c (the printer's permanent memory). The maximum length of a path is 255 characters.

The “old” device names (c:, rom:, tmp:, and so on) are now aliases (“shortcuts”) to the new directories (/c/, /rom/, /tmp/, and so on). The file STDIO on c: (/c) can thus be accessed using either c:STDIO or /c/STDIO. Writing c: is equivalent to writing /c/.

The philosophy in the design of the different commands and output formats is to be as backwards-compatible as possible, whilst giving the user access to the new features—directories. Examples of this are:

- FILES give a size of 0 for directories to minimize impact on applications that parse the output.
- FILENAME$ only report files to minimize impact on applications that use FILENAME$ to get file listings.

To relieve the user from always having to use the entire path when referring to a directory above the current one, each directory (including the root directories) contains a “parent directory”. This parent directory is called “..”. It refers to the directory’s parent directory. It is listed by FILES,A.

Each directory also has a reference to itself (“.”), that is, "/c./DIR1././FILE" refers to "/c/FILE" (or, using the legacy format, to "c:FILE").

Example:

CHDIR "/c/DIR1/DIR2" Changes the directory
CHDIR "..." Go up to "/c/DIR1"
CHDIR ".../" Go up to /c. Note that a trailing slash (/) may be used.

Note: A file or directory name may contain all printable characters except “:” (colon) and “/” (slash). Only /c (c:) supports creating and removing directories.
Files in Intermec Fingerprint v8.70.X and v10.0.X

The Intermec Fingerprint v8.70.X firmware contains the following files. Files marked with footnote * are not found in Fingerprint 10.0.X. Some files in "/c/" only appear after default settings have been changed.

In device "/rom/"

- .FONTALIAS
  - coms
  - profile
  - setup.saved
  - uartx
  - ubiqufr1.bin
  - wi_firm
  - CHESS2X2.1
  - CHESS4X4.1
  - DIAMONDS.1
  - ERRHAND.PRG
  - FILELIST.PRG
  - GLOBE.1
  - LBLSHTXT.PRG
  - LINE AX.PRG
  - LSHPXP1.SUB
  - MKAUTO.PRG
  - PUP.BAT
  - SHELLXP.PRG*
  - WINXP.PRG*

- default.html
- home.htmf
- htmlhead.htmf
- images/
  - iclogo1.gif
- monitor
- nav.html
- passwd
- restrictions
- secure/
  - config.js
  - configtree.html
  - ddns.html
  - empty.htm
  - ftie4style.css
  - ftiens4.js
  - ftv2blank.gif
  - ftv2doc.gif
  - ftv2folderclosed.gif
  - ftv2folderopen.gif
  - ftv2lastnode.gif
  - ftv2link.gif
  - ftv2mlastnode.gif
  - ftv2mnode.gif
  - ftv2node.gif

Create reference fonts
System file
System file
Default setup values
System file
Standard fonts
Firmware for EasyLAN Wireless
Standard image for test labels
Standard image for test labels
Standard image for test labels
Error Handler
List the lines of a file
Standard image for test labels
Intermec Shell auxiliary file
Intermec Shell Line Analyzer
Intermec Shell auxiliary file
Create a startup (autoexec) file
Intermec Shell Startup file
Intermec Shell startup program
Intermec Shell auxiliary file

EasyLAN home page
EasyLAN home page
EasyLAN home page
EasyLAN home page
System file
EasyLAN home page
Default password file
Default restriction file
EasyLAN home page
EasyLAN home page
EasyLAN home page
EasyLAN home page
EasyLAN home page
EasyLAN home page
EasyLAN home page
EasyLAN home page
EasyLAN home page
EasyLAN home page
EasyLAN home page
EasyLAN home page
In device "/c/"

.boot/
  .setup.saved
  passwd
  rmmcz`
  resu`
  psa`

.STDIO

.APPLICATION

.ADMIN/

.restrictions

Current setup values (prt section)
Password storage
Kernel file
Kernel file
Kernel file
Intermec Shell auxiliary file
Intermec Shell auxiliary file
Restrictions storage

To read the contents of these files, run the FILELIST.PRG program or COPY the file in question to the serial port "uart1:".

* Not in Fingerprint 10.0.X.
This chapter explains all program instructions in alphabetic order, lists their syntaxes and input parameters, and gives some examples how to use the instructions in simple programs.
Syntax

In the syntax descriptions which follow, certain punctuation marks are used to indicate various types of data. They must not be included in the program.

[ ] indicate that the enclosed entry is optional.

| indicates alternatives on either side of the bar.

< > indicate grouping.

..... indicate repetition of the same type of data.

↔ indicates a compulsory space character between two keywords.

" is a quotation mark (ASCII 34 dec).

\n indicates a carriage return or linefeed on the host.

Uppercase letters indicate keywords, which must be entered exactly as listed, with the exception that lowercase letters also are allowed unless otherwise stated.

The following abbreviations are used:

<scon> string constant
<ncon> numeric constant
<sexp> string expression
<nexp> numeric expression
<svar> string variable
<nvar> numeric variable
<stmt> statement
<line label> line label
ABS

Purpose
Function returning the absolute value of a numeric expression.

Syntax
ABS(<nexp>)

<nexp> is a numeric expression, from which the absolute value will be returned.

Remarks
The absolute value of a number is always positive or zero. Note that the expression must be enclosed within parentheses.

Examples
PRINT ABS(20-25)
5
PRINT ABS(25-20)
5
PRINT ABS(5-5)
0
PRINT ABS(20*-5)
100
**ACTLEN**

**Purpose**
Function returning the length of the most recently executed PRINTFEED, FORMFEED, or TESTFEED statement.

**Syntax**
ACTLEN

**Remarks**
The length of the most recently executed paper feed operation, resulting from a PRINTFEED, FORMFEED, or TESTFEED statement, will be returned as a number of dots. Due to technical reasons concerning the stepper motor control and label gap detection, a small deviation from the expected result may occur.

**Example**
In this example, a 12 dots/mm printer is loaded with 90 mm (1080 dots) long labels separated by a 3 mm (36 dots) gap. Start- and stopadjust setup values are both set to 0:

```
10 FORMFEED
20 PRINT ACTLEN
RUN
```

yields:

```
1121
```

The deviation from the expected result (1116) is normal and should have no practical consequences (less than 1 mm).
ALIGN (AN)

Purpose
Statement specifying which part (anchor point) of a text field, bar code field, image field, line, or box will be positioned at the insertion point.

Syntax
ALIGN|AN\(<nexp>\)

\(<nexp>\)  is the anchor point of the object (1–9).
Default value: 1
Reset to default by: PRINTFEED execution

Remarks
Each text, bar code, or image field has nine possible anchor points, whereas lines and boxes have three. One of these points must be selected, or the default value (1) will be used. The selected anchor point decides the position of the object in relation to the insertion point, which is decided by the nearest preceding PRPOS statement. Furthermore, the field will be rotated around the anchor point according to the nearest preceding DIR statement.

The nine anchor points of a text, bar code, or image field are located in the same manner as, for example, the numeric keys on a computer keyboard:

```
  7  8  9
  4  5  6
  1  2  3
```

Lines and boxes have three anchor points only: left, center, and right.
The anchor points for the various types of field are illustrated below.

Text field:

```
   7  8  9
   4  5  6
   1  2  3
```

A text field makes up an imaginary box limited in regard of width by the length of the text, and in regard of height by the matrix size of the selected font. In text fields, the anchor points 4, 5, and 6 are situated on the baseline, as opposed to bar code fields and image fields.
ALIGN (AN), cont.

Bar Code Field:

A bar code field makes up an imaginary box sufficiently large to accommodate the bar code interpretation, regardless if it will be printed or not (provided that the selected type of bar code may include an interpretation at all).

However, for EAN and UPC codes, the box is restricted in regard of width by the size of the bar pattern, not by the interpretation. This implies that the first digit of the bar code interpretation will be outside the imaginary box:

For composite bar codes, the human readable bar code interpretation for the 2-dimensional element is outside the imaginary box.

Image field:

The size of an image field is decided when the field is created. Note that an image field consists of the entire area of the original image, even possible white or transparent background.
ALIGN (AN), cont.

Line:

The anchor points are situated at the lower side of the line or box in relation to how text is printed in the selected direction. Lines and boxes have only three anchor points, each of which can be specified by means of three different numbers.

A special case is multi-line text fields in a box. The fields can be aligned in nine positions in relation to the box, whereas the box itself only has three anchor points, as described above. Refer to the PRBOX statement for more information on alignment of multi-line text fields.

Example

Printing of a label with a single line of text being aligned left on the baseline:

10 PRPOS 30,250
20 DIR 1
30 ALIGN 4
40 FONT "Swiss 721 BT"
50 PRTXT "Hello!"
60 PRINTFEED
RUN

The text “Hello!” will be positioned with the baseline aligned left to the insertion point specified by the coordinates X=30; Y=250 in line 10.
**ASC**

**Purpose**
Function returning the decimal ASCII value of the first character in a string expression.

**Syntax**

```
ASC(<sexp>)
```

*<sexp>* is a string expression, from which the ASCII decimal value of the first character will be returned.

**Remarks**
ASC is the inverse function of CHR$. The decimal ASCII value will be given according to the selected character set (see NASC statement).

**Examples**

```
10  A$="GOOD MORNING"
20  PRINT ASC(A$)
RUN
    71

10  B$="123456"
20  C% = ASC(B$)
30  PRINT C%
RUN
    49
```
**BARADJUST**

**Purpose**

Statement for enabling/disabling automatic adjustment of bar code position in order to avoid faulty printhead dots.

**Syntax**

```plaintext
BARADJUST<nexp1>,<nexp2>
```

- `<nexp1>` is the maximum left offset in dots.
- `<nexp2>` is the maximum right offset in dots.

**Default:** 0,0 (BARADJUST disabled)

**Remarks**

Under unfortunate circumstances, a printer may have to be run for some time with a faulty printhead, before a replacement printhead can be installed. Single faulty dots will produce very thin “white” lines along the media. This may be tolerable for text, graphics, and vertical (ladder) bar codes, but for horizontal bar codes (picket fence), this condition is likely to render the bar code unreadable.

If the bar code is moved slightly to the left or right, the trace of a faulty dot may come between the bars of the bar code and the symptom is remedied for the time being. The statement will attempt to adjust the bar code even if several faulty dots have been specified, but will return error 1052 if it was unable to adjust the bar code succesfully.

The BARADJUST statement allows the Intermec Fingerprint firmware to automatically readjust the bar code position within certain limits, when a faulty dot is detected (see HEAD function) and marked as faulty (see SET FAULTY DOT statement). The maximum deviation from the original position, as specified by the PRPOS statement, can be set up separately for the directions left and right. Setting both parameters to 0 (zero) will disable BARADJUST.

The BARADJUST statement does not work with:

- Vertically printed bar codes (ladder style)
- Stacked bar codes (for example Code 16K)
- Bar codes with horizontal lines (for example DUN-14/16)
- EAN/UPC-codes (interpretation not repositioned)

**Examples**

Enabling BARADJUST within 10 dots to the left and 5 dots to the right of the original position for a specific bar code, then disabling it:

```plaintext
10 BARADJUST 10,5
20 PRPOS 30,100
30 BARSET "CODE39",2,1,3,120
40 BARFONT ON
50 PRBAR "ABC"
60 BARADJUST 0,0
70 PRINTFEED
```
Chapter 2—Program Instructions

BARCODENAME$

Purpose

Function returning the names of the bar code generators stored in the printer's temporary memory ("tmp:").

Syntax

BARCODENAME$(<nexp>)

<nexp> the result of the expression should be either false or true, where...
False (0) indicates first bar code.
True (≠0) indicates next bar code.

Remarks

BARCODENAME$(0) produces the first bar code name in alphabetical order. BARCODENAME$(≠0) produces next name. Can be repeated as long as there are any bar code names left.

Example

Use a program like this to list the names of all bar codes in "tmp:". Note that bar codes with dynamic downloading will not appear before they have been called by a BARSET or BARTYPE statement.

```
10  A$ = BARCODENAME$ (0)
20  IF A$ = "" THEN END
30  PRINT A$
40  A$ = BARCODENAME$ (-1)
50  GOTO 20
RUN
```

yields for example:

ADDON2
ADDON5
C2OF5
C2OF5IND
C2OF5INDC
C2OF5MAT
CODABAR
CODE11
CODE128
CODE128A
CODE128B
CODE128C
CODE16K
CODE39
CODE39A
CODE39C
CODE49
CODE93
DATAMATRIX
DUN
EAN128
EAN128A
EAN128B
etc, etc.
BARFONT (BF)

Purpose
Statement specifying fonts for the printing of bar code interpretation.

Syntax
BARFONT|BF[#<ncon>,]<sexp1>,[<nexp1>,]<sexp2>[,]<nexp2>[,]<nexp3>
[,<nexp4>[,<nexp5>[,<nexp6>[,<nexp7>[,<nexp8>[,<nexp9>]]]]]](ON)

#<ncon> is, optionally, the start parameter in the syntax above.
<sexp1> is the name of the font selected for bar code interpretations.
<nexp1> is the height in points of the font.
<nexp2> is the clockwise slant in degrees (0-90°).
<nexp3> is the distance in dots between bar code and bar font.
<nexp4> is the magnification in regard of height (1-4).
<nexp5> is the magnification in regard of width (1-4).
<nexp6> is the width enlargement in percent relative the height (1-1000).
Default: 100. Does not work with bitmap fonts.
<nexp7> is the insertion point for the bar code interpretation.
<nexp8> is the horizontal offset from the insertion point for the bar code interpretation.
<nexp9> is the vertical offset from the insertion point for the bar code interpretation.
ON optionally enables the printing of bar code interpretation.
Reset to default by: PRINTFEED execution.

Remarks
Start Parameter:
The start parameter specifies which parameter in the syntax above should be the first parameter in the statement. Thereby you may bypass some of the initial parameters.
Default value: #1

Font Name:
This parameter corresponds to the FONT statement, but will only affect bar code interpretation. Double-byte fonts cannot be used.
Default: Swiss 721 BT

Font Size:
This parameter corresponds to the FONT statement, but will only affect bar code interpretation. The size is specified in points. (1 point = 1/72 inch ≈ 0.352 mm.)
Default: 12

Font Slant:
This parameter corresponds to the FONT statement, but will only affect bar code interpretation. Slanting increases clockwise. Values greater than 65-70° will be unreadable.
Default: 0
BARFONT (BF), cont.

Vertical Offset:
The distance between the bottom of the bar code pattern and the top of the character cell is given as a number of dots.
Default value: 6

Magnification:
Two parameters allow you to specify the magnification separately in regard to height and width (corresponding to MAG statement). Note that if a MAG statement is executed after a BARFONT statement, the size of the barfont will be affected by the MAG statement.
Default value for both parameters: 1

Width:
A scalable font can be enlarged in regard to width relative to height. The value is given as percent (1-1000). This means that if the value is 100, there is no change in the appearance of the characters, whereas if the value is given as for example 50 or 200, the width will be half the height or double the height respectively. When using this parameter, all parameters in the syntax must be included in the statement, (name, height, slant, and width).

Insertion Point:
The position of the bar code interpretation can be selected by setting this parameter. A value between 1 and 9 can be set and corresponds to the positions in the figure. The values 4, 5, and 6 are interpreted as 7, 8, and 9 respectively.
This function overrides the ALIGN command.
Default: 0 (Not Enabled)

Insertion Point Offsets:
The parameters represent the offset in dots with regard to the insertion point. The offsets are set with respect to the bar code direction, not necessarily the paper feed direction. The insertion point must be set for these parameters to have meaning.
Default: 0
Chapter 2—Program Instructions

BARFONT (BF), cont.

Enabling Interpretation Printing:
The printing of bar code interpretation can enabled by a trailing ON, which corresponds to a BARFONT ON statement.

Exceptions:
Note that in all EAN and UPC bar codes, the interpretation is an integrated part of the code. Such an interpretation is not affected by a BARFONT statement, but will be printed in according to specification, provided that interpretation printing has been enabled by a BARFONT ON statement.

Certain bar codes, like Code 16K, cannot contain any interpretation at all. In such a case, the selected barfont will be ignored.

Example

Programming a Code 39 bar code, selecting the same barfont for all directions, and enabling the printing of the bar code interpretation, can be done this way:

```
10  PRPOS 30,400
20  DIR 1
30  ALIGN 7
40  BARSET "CODE39",2,1,3,120
50  BARFONT "Swiss 721 BT",10,8,5,1,1,100 ON
60  PRBAR "ABC"
70  PRINTFEED
80  END
```
Chapter 2—Program Instructions

BARFONT ON/OFF (BF ON/OFF)

Purpose
Statement enabling or disabling the printing of bar code interpretation.

Syntax
```
BARFONT ON|BARFONT OFF[,<nexp>]
BF ON|BF OFF[,<nexp>]
```

- `<nexp>` For EAN/UPC bar codes disables/enables guard bar printing.
- Default: BARFONT OFF
- Reset to default by: PRINTFEED execution

Remarks
Usually, you start your program by selecting a suitable bar code interpretation font, see BARFONT. Then use BARFONT ON and BARFONT OFF statements to control whether to print the interpretation or not, depending on application.

BARFONT ON can be replaced by a BARFONT statement appended by a trailing ON, see BARFONT statement.

When printing EAN/UPC bar codes, the trailing parameter 0 can be specified after the BARFONT OFF command to disable the printing of the guard bars, which are set to print by default. BARFONT OFF, 0 will disable the printing of the guard bars. BARFONT OFF,1 is the default (guard bars enabled) and is equivalent to BARFONT OFF.

Examples
Programming a Code 39 bar code, selecting a barfont for each direction and enabling the printing of the bar code interpretation. Compare with the example for BARFONT statement:
```
10  PRPOS 30,400
20  DIR 1
30  ALIGN 7
40  BARSET "CODE39",2,1,3,120
50  BARFONT "Swiss 721 BT",10,8,5,1,1
60  BARFONT ON
70  PRBAR "ABC"
80  PRINTFEED
90  END
```

Programming a EAN8 bar code, and choosing to disable guard bars:
```
10  BARTYPE "EAN8"
20  BARFONT OFF,0
30  PRBAR "1234567"
40  PRINTFEED
BARHEIGHT (BH)

Purpose
Statement specifying the height of a bar code.

Syntax
BARHEIGHT|BH<nexp>

<nexp> is the height of the bars in the bar code expressed in number of dots.
Default value: 100 dots.
Reset to default by: PRINTFEED execution.

Remarks
The barheight specifies the height of the bars, that make up the code. In bar codes consisting of several elements on top of each other, for example Code 16K, the barheight specifies the height of one element. The height is not affected by BARMAG statements.
BARHEIGHT can be replaced by a parameter in the BARSET statement.

Example
Programming a Code 39 bar code, selecting a barfont for all directions and enabling the printing of the bar code interpretation:

```plaintext
10  PRPOS 30,400
20  DIR 1
30  ALIGN 7
40  BARTYPE "CODE39"
50  BARRATIO 2,1
60  BARHEIGHT 120
70  BARMAG 3
80  BARFONT "Swiss 721 BT"ON
90  PRBAR "ABC"
100 PRINTFEED
```

A more compact method is illustrated by the example for BARSET statement.
Chapter 2—Program Instructions

BARMAG (BM)

Purpose

Statement specifying the magnification in regard to width of the bars in a bar code.

Syntax

```
BARMAG|BM<nexp>
```

<nexp> is the magnification in regard to width of the bars, which make up the bar code.

Allowed input: Depends on type of bar code.

Default value: 2

Reset to default by: PRINTFEED execution.

Remarks

The magnification only affects the bar code width, not the height of the bars (see BARHEIGHT). For example, by default the BARRATIO is 3:1 and the BARMAG is 2, which means that the wide bars will be 6 dots wide and the narrow bars will be 2 dots wide (2 × 3:1 = 6:2).

The magnification also affects the interpretation in EAN and UPC bar codes, since the interpretation is an integrated part of the EAN/UPC code.

BARMAG can be replaced by a parameter in the BARSET statement.

Example

Programming a Code 39 bar code, selecting a barfont for all directions and enabling the printing of the bar code interpretation:

```
10  PRPOS 30,400
20  DIR 1
30  ALIGN 7
40  BARTYPE "CODE39"
50  BARRATIO 2,1
60  BARHEIGHT 120
70  BARMAG 3
80  BARFONT "Swiss 721 BT" ON
90  PRBAR "ABC"
100 PRINTFEED
```

A more compact method is illustrated by the example for BARSET statement.
**BARRATIO (BR)**

**Purpose**
Statement specifying the ratio between the wide and the narrow bars in a bar code.

**Syntax**
BARRATIO|BR<\nexp_1>,<\nexp_2>

<\nexp_1> is the thickness of the wide bars relative to the narrow bars.
<\nexp_2> is the thickness of the narrow bars relative to the wide bars.
Default value: 3:1
Reset to default by: PRINTFEED execution.

**Remarks**
This statement specifies the ratio between the wide and the narrow bars in a bar code in relative terms. To decide the width of the bars in number of dots, the ratio must be multiplied by the BARMAG value.

Example:
The default BARRATIO is 3:1 and the default BARMAG is 2.
(3:1) × 2 = 6:2
that is, the wide bars are 6 dots wide and the narrow bars are 2 dots wide.

Note that certain bar codes have a fixed ratio, for example EAN and UPC codes. In those cases, any BARRATIO statement will be ignored. Refer to Chapter 5, “Bar Codes” later in this manual.

BARRATIO can be replaced by two parameters in the BARSET statement.

**Example**
Programming a Code 39 bar code, selecting a barfont for all directions and enabling the printing of the bar code interpretation:

```
10   PRPOS 30,400
20   DIR 1
30   ALIGN 7
40   BARTYPE "CODE39"
50   BARRATIO 2,1
60   BARHEIGHT 120
70   BARMAG 3
80   BARFONT "Swiss 721 BT"ON
90   PRBAR "ABC"
100  PRINTFEED
```

A more compact method is illustrated by the example for BARSET statement.
Chapter 2—Program Instructions

BARSET

Purpose
Statement specifying a bar code and setting additional parameters to complex bar codes.

Syntax
BARSET[#<ncon>,][<sexp>],[<nexp1>],<nexp2>],[<nexp3>],[<nexp4> ],<nexp5>],[<nexp6>],[<nexp7>],[<nexp8>],[<nexp9>],[<nexp10>]]

#<ncon> is the start parameter in the syntax above.
<sexp> is the bar code type. No default.
<nexp1> is the ratio of the large bars. Default: 3.
<nexp2> is the ratio of the small bars. Default: 1.
<nexp3> is the enlargement. Default: 2
<nexp4> is the height of the code in dots. Default: 100.
<nexp5> bar code specific parameter, see Chapter 5. Default: 2.
<nexp6> bar code specific parameter, see Chapter 5. Default: 3.
<nexp7> bar code specific parameter, see Chapter 5. Default: 1.
<nexp8> bar code specific parameter, see Chapter 5. Default: 0.
<nexp9> bar code specific parameter, see Chapter 5. Default: 0.
<nexp10> bar code specific parameter, see Chapter 5. Default: 0.

Reset to default by: PRINTFEED execution.

Remarks
This statement can replace the statements BARHEIGHT, BARRATIO, BARTYPE, and BARMAG. Although being primarily intended for 2-dimensional and composite bar codes, it can be used for any type of bar code if non-relevant parameters are left out (for example <nexp5> to <nexp10>).

Note: The parameter descriptions under the syntax do not apply to all bar codes that can be specified using BARSET. Some bar codes have other meanings or use certain parameters only as place holders. See Chapter 5 “Bar Codes” for more information.

Start Parameter:
Start parameter specifies which parameter in the syntax above should be the first optional parameter (#1-#11). Thereby you may bypass some of the initial parameters, for example ratio and enlargement.
Default value: #1

#1 Bar Code Type:
The bar code type parameter corresponds to the BARTYPE statement.

#2 and #3 Bar Code Ratio:
The two ratio parameters correspond to the BARRATIO statement.
Default value: 3:1

#4 Enlargement:
The enlargement parameter corresponds to the BARMAG statement.
Default value: 2
Chapter 2—Program Instructions

**BARSET, cont.**

#5 Bar Code Height:
The height parameter corresponds to the BARHEIGHT statement. In QR Code, this parameter is used to specify model (1 or 2).
Default value: 100 dots

#6–#11 Bar code specific parameters:
These parameters have different meanings for each bar code or are used only as placeholders. Refer to the descriptions of the various 2-dimensional and composite bar codes in Chapter 5 for complete syntax descriptions and programming examples.

**Example**

This example shows how a BARSET statement is used to specify a Code 39 bar code (compare for example with the example for BARTYPE stmt):

```
10   PRPOS 30,400
20   DIR 1
30   ALIGN 7
40   BARSET "CODE39",2,1,3,120
50   BARFONT "Swiss 721 BT",10,8,5,1,1 ON
60   PRBAR "ABC"
70   PRINTFEED
```
Chapter 2—Program Instructions

BARTYPE (BT)

Purpose
Statement specifying the type of bar code.

Syntax
BARTYPE|BT<sexp>

<sexp> specifies the type of bar code.
Allowed input: Valid bar type name.
Default value: None.

Remarks
The selected bar code type must exist in the printer's memory and be entered in the form of a string expression. Please refer to Chapter 5, “Bar Codes” later in this manual for a list of the bar codes that are included in the Intermec Fingerprint firmware and their respective designations.

BARTYPE can be replaced by a parameter in the BARSET statement.

Example
Programming a Code 39 bar code, selecting a barfont for all directions, and enabling the printing of the bar code interpretation:

10  PRPOS 30,400
20  DIR 1
30  ALIGN 7
40  BARTYPE "CODE39"
50  BARRATIO 2,1
60  BARHEIGHT 120
70  BARMAG 3
80  BARFONT "Swiss 721 BT" ON
90  PRBAR "ABC"
100 PRINTFEED

A more compact method is illustrated by the example for BARSET statement.
### BEEP

**Purpose**
Statement ordering the printer to emit a beep.

**Syntax**
```
BEEP
```

**Remarks**
This statement makes the printer’s built-in buzzer sound at ≈800 Hz for 1/4 of a second. If a different frequency and/or duration is desired, use a SOUND statement instead.
The EasyCoder PD41 does not have a beeper, and will thus not emit a sound. The BEEP command will however cause an equally long delay in the PD41.

**Example**
In this example, a beep is emitted when an error occurs:
```
10     ON ERROR GOTO 1000
       .......
       .......
       1000   BEEP
1010   RESUME NEXT
```
**BREAK**

**Purpose**
Statement specifying a break interrupt character separately for the keyboard and each serial communication channel.

**Syntax**

```
BREAK <nexp1>,<nexp2>
```

- `<nexp1>` is one of the following devices:
  - 0 = "console:"
  - 1 = "uart1:"
  - 2 = "uart2:"
  - 3 = "uart3:"
  - 7 = "uart4:"
  - 8 = "uart5"

- `<nexp2>` is the decimal ASCII value for the break interrupt character.

**Default:**
- Comm. channels: ASCII 03 decimal
- Console: ASCII 158 decimal (<Shift> + <Pause>)

**Remarks**

The execution of a program can be interrupted using a method specified by the BREAK statement. In addition, the printing of a batch of labels can also be interrupted and resumed by pressing the <Pause> or the <Print> key on the printer’s front panel.

To issue a break interrupt, by default, hold down the <Shift> key and press the <Pause> key. Together these keys will produce the ASCII character 158 decimal (128 + 30), see Appendix A.

It is not possible to perform a break from the console on the PD41 printer.

It is possible to remap the keyboard, which may affect the keys used for break interrupt. Please refer to the variable KEYBMAP$.

Another method is to transmit the character ASCII 03 decimal (default) to the printer on one of the serial communication channels. The execution will be interrupted regardless of any INPUT waiting (that is, INPUT [#], LINE INPUT [#], and INPUT$).

The BREAK statement allows you to specify other ways of interrupting the execution, for example by pressing another combination of keys on the printer’s keyboard or transmitting another ASCII character from the host.

A specified break interrupt character is saved in the temporary memory until the printer is restarted or REBOOTed, which may be confusing for example when switching between programs. To change a break interrupt character, specify a new one for the same device using a BREAK statement and to remove it from memory, use a BREAK OFF statement.

The use of break interrupt is enabled or disabled separately for each device by BREAK ON or BREAK OFF statements. By default, break interrupt on the "console:" is enabled, while break interrupt on any of the communication channels is disabled.
BREAK, cont.

It is strongly recommended to include some facility for issuing a break interrupt from the host computer in startup (autoexec) files. If not, you may find yourself with an erroneous program running in a loop without being able to break it!

Examples

In this example, the ASCII character 127 decimal is selected and enabled as BREAK character on the communication channel "uart1:"

```
10  BREAK 1,127
20  BREAK 1 ON
    . . . .
```

In next example, BREAK characters are specified for both the keyboard ("console:"), and the serial communication channel "uart1:". The loop can be interrupted either by pressing the key usually marked “F1” on the printer’s keyboard, or by typing an uppercase X on the keyboard of the host:

```
10  BREAK 0,1:BREAK 1,88
20  BREAK 0 ON:BREAK 1 ON
30  GOTO 30
RUN
```

Reset BREAK to default by turning the printer off and on.
Chapter 2—Program Instructions

BREAK ON/OFF

Purpose
Statement enabling or disabling break interrupt separately for the keyboard and each serial communication channel.

Syntax

```
BREAK<nexp>ON|OFF
```

<nexp> is one of the following devices:

```
0 = "console:"
1 = "uart1:"
2 = "uart2:"
3 = "uart3:"
7 = "uart4:"
8 = "uart5"
```

Default:
Comm. ports: Disabled
Console: Enabled

Remarks
The use of the break interrupt specified by a BREAK statement can be enabled or disabled separately for each serial communication channel or for the printer’s built-in keyboard by BREAK ON or BREAK OFF statements. By default, break interrupt is enabled from the printer’s keyboard and disabled from all communication channels.

It is not possible to perform a break from the console on the PD41 printer. BREAK OFF deletes any existing break interrupt character stored in the printer’s temporary memory for the specified device.

Example
In this example, the ASCII character 127 decimal is selected and enabled as BREAK character on the communication channel "uart1:". At the same time, BREAK from the printer’s keyboard is disabled.

```
10 BREAK 1,127
20 BREAK 1 ON:BREAK 0 OFF
```

........
........
........
**BUSY**

**Purpose**
Statement ordering a busy signal, for example XOFF, CTS/RTS, or PE, to be transmitted from the printer on the specified communication channel.

**Syntax**

```
BUSY[<nexp>]
```

<nexp> optionally specifies the channel as:

1 = "uart1:"
2 = "uart2:"
3 = "uart3:"
4 = "centronics:"
7 = "uart4:"
8 = "uart5"

**Remarks**

Communication protocol usually contain some “busy” signal, which tells the host computer that the printer, for some reason, is unable to receive data.

The BUSY statement allows you to order a busy signal to be transmitted on the specified communication channel. If no channel is specified, the signal will be transmitted on the standard OUT communication channel, see SETSTDIO statement.

To allow the printer to receive more data, use a READY statement.

For the optional "centronics:" communication channel, BUSY/READY control the PE (paper end) signal on pin 12 according to an error-trapping routine (BUSY = PE high).

**Example**

You may, for example, want to prevent the printer from receiving more data on "uart2:" during the process of printing a label (running this example requires an optional interface board to be fitted):

```plaintext
10 FONT "Swiss 721 BT"
20 PRTXT "HELLO!"
30 BUSY3
40 PRINTFEED
50 READY3
RUN
```
CHDIR

Purpose

Statement specifying the current directory.

Syntax

CHDIR<scon>

<scon> specifies the current directory (see DEVICES)

Default: "/c"

Remarks

By default, the printer’s permanent memory ("/c") is the current directory, which means the directory that is used if the Intermec Fingerprint instruction does not contain any reference to a directory, for example FILES. This implies that to access the temporary memory ("tmp:"), the storage part of the RTC/Dallas key circuit ("storage:"), or an optional memory card ("/rom" or "card1:"), you must include such a reference in your instructions, for example FILES "/rom".

The CHDIR statement allows you to appoint another directory than "/c" as the current directory. Obviously, this implies that you must specify the permanent memory ("/c") whenever you want to access it.

Example

In this example, the current directory is changed to "card1:", all files in "card1:" are listed, and finally the current directory is changed back to "/c". (This example is only included to illustrate the principles of changing the current directory. It is more efficient to use FILES "card1:" to read its contents.)

10  CHDIR"card1:"  
20  FILES  
30  CHDIR="/c"  
RUN
CHECKSUM

Purpose
Statement calculating the checksum of a range of program lines in connection with the transfer of programs.

Syntax
CHECKSUM(<nexp1>,<nexp2>)

<nexp1> is the number of the first line in a range of program lines.
<nexp2> is the number of the last line in a range of program lines.

Remarks
The checksum is calculated from parts of the internal code using an advanced algorithm. Therefore, it is recommended to let the printer calculate the checksum before the transfer of a program. After the transfer is completed, let the receiving printer do the same calculation and compare the checksums.

Example
In this example, the checksum is calculated of all program lines between line 10 and line 2000 in the program "DEMO.PRG".

NEW
LOAD "DEMO.PRG"
PRINT CHECKSUM(10,2000)

yields:
60095
**CHR$**

**Purpose**  
Function returning the readable character from a decimal ASCII code.

**Syntax**  
`CHR$(<nexp>)`

- `<nexp>` is the decimal ASCII code to be converted to a readable character.

**Remarks**  
This function is useful for entering characters that cannot be produced from the keyboard of the host, for example non-printable characters ASCII 0-31 dec. Only integers between 0 and 255 are allowed. Input less than 0 or larger than 255 will result in an error condition (Error 41, “Parameter out of range.”)

**Example**  
The decimal ASCII code for “A” is 65 and for “B” is 66.

```plaintext
10  A$ = CHR$(65)
20  B$ = CHR$(40+26)
30  PRINT A$
40  PRINT B$
RUN
```

yields:

```
A
B
```
CLEANFEED

Purpose
Statement running the printer’s feed mechanism.

Syntax
```
CLEANFEED<nexp>
```

<nexp> is the feed length expressed as a positive or negative number of dots.

Remarks
The CLEANFEED statement activates the stepper motor that drives the printer’s platen roller (the rubber roller beneath the printhead). In case of thermal transfer printers, it also often drives the ribbon mechanism. The motor will run regardless of possible error conditions, for example if the printhead is lifted or not, or if there is no ribbon or media supply left.

Thus, the CLEANFEED statement is suitable for cleaning and for the loading of transfer ribbon.

A positive CLEANFEED value makes the stepper motor rotate the rollers forward, that is as when feeding out a label.

A negative CLEANFEED value makes the stepper motor rotate the rollers backwards, that is as when pulling back a label.

The execution of a CLEANFEED statement, as opposed to TESTFEED, does not affect the adjustment of the label stop sensor or black mark sensor, regardless what type of media or other objects that passes the sensor.

Note that the CLEANFEED statement, as opposed to FORMFEED, always must be specified in regard of feed length.

Example
In order to pull a cleaning card back and forth under the printhead three times, three 1200 dots long positive CLEANFEEDs and then the same amount of negative CLEANFEEDs are performed:
```
10 FOR A%=1 TO 3
20 CLEANFEED 1200
30 CLEANFEED -1200
40 NEXT
RUN
```
CLEAR

Purpose
Statement clearing strings, variables, and arrays in order to free memory space.

Syntax
CLEAR

Remarks
The CLEAR statement empties all strings, sets all variables to zero, and resets all arrays to their default values. As a result, more free memory space becomes available.

Example
In this example, more free memory space is obtained after the strings have been emptied by means of a CLEAR statement:
10   A$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20   B$ = "abcdefghijklmnopqrstuvwxyz"
30   FOR I%=0 TO 3:FOR J%=0 TO 3:FOR K%=0 TO 20
40   C$(I%,J%)=C$(I%,J%)+A$
50   NEXT K%:NEXT J%:NEXT I%
60   PRINT "String A before: ";A$
70   PRINT "String B before: ";B$
80   PRINT "Free memory before: ";FRE(1)
90   CLEAR
100  PRINT "String A after: ";A$
110  PRINT "String B after: ";B$
120  PRINT "Free memory after: ";FRE(1)
RUN

yields:
String A before: ABCDEFGHIJKLMNOPQRSTUVWXYZ
String B before: abcdefghijklmnopqrstuvwxyz
Free memory before: 1867368
String A after:
String B after:
Free memory after: 1876200

Ok
**CLIP**

**Purpose**
Statement for enabling/disabling the printing of partial fields.

**Syntax**

```plaintext
CLIP [BARCODE [HEIGHT|INFORMATION|X|Y]][ON|OFF]
```

- **BARCODE** toggles between partial bar code fields enable/disable.
- **BARCODE HEIGHT** clips the height of the bar so the bar code will fit inside the print window. A one-dimensional bar code may still be readable.
- **BARCODE INFORMATION** clips the bar code lengthwise, so some bars will be missing, making the bar code unreadable.
- **BARCODE X** clips the part of the bar code that is outside the X-dimension of the print window.
- **BARCODE Y** clips the part of the bar code that is outside the Y-dimension of the print window.
- **ON** enables use of partial text, image, line, and box fields.
- **OFF** disables use of partial text, image, line, and box fields.

**Remarks**
Partial fields means that the firmware will accept and print text, bar code, image, lines, and box fields even if they extend outside the print window as specified by the printer’s setup in regard of X-Start, Width, and Length. Even negative PRPOS values are allowed. However, all parts the fields outside the print window will be excluded from the printout, that is they will be clipped at the borders of the print window.

There are two main cases:

- **CLIP BARCODE [HEIGHT|INFORMATION|X|Y]** is used for bar code fields only. (Note that some bar codes, like Maxicode, consist of images and should in this context be regarded as image fields.)
- **CLIP ON|OFF** is only used for text, image, line, and box fields.

When the use of partial fields is disabled, the Error 1003, “Field out of label” will result if any field extends outside the print window.

Note the difference between the physical size of the label and the size of the print window specified by the printer’s setup. It is the latter that decides were the fields will be clipped.

**Example**
In this example, only the last part of the text will be printed:

```plaintext
10  CLIP ON
20  PRPOS -100,30
30  PRTXT "INTERMEC PRINTER"
40  PRINTFEED
RUN
```


**CLL**

**Purpose**

Statement for partial or complete clearing of the print image buffer.

**Syntax**

```
CLL [<nexp>]
```

- `<nexp>` optionally specifies the field from which the print image buffer should be cleared.

**Remarks**

The print image buffer is used to store the printable image after processing awaiting the printing to be executed. The buffer can be cleared, partially or completely, by the use of a CLL statement:

- **CLL**<nexp> partially clears the buffer from a specified field to the end of the program. The field is specified by a FIELDNO function.

  Partial clearing is useful in connection with print repetition. To avoid superfluous reprocessing, one or several fields can be erased from the buffer and be replaced by other information, while the remaining parts of the label are retained in the buffer.

  Note that there must be no changes in the layout between the PRINTFEED and the CLL statements, or else the layout will be lost. Also note that partial clearing always starts from the end, which means that the fields which are executed last are cleared first.

- **CLL** (without any field number) clears the buffer completely.

  When certain error conditions have occurred, it is useful to be able to clear the print image buffer without having to print a faulty label. Should the error be attended to, without the image buffer being cleared, there is a risk that the correct image will be printed on top of the erroneous one on the same label. It is therefore advisable to include a CLL statement in your error-handling subroutines, when you are working with more complicated programs, in which all implications may be difficult to grasp.
**CLL, cont.**

**Examples**

Partial clearing:

Two labels are printed, each with two lines of text. After the first label is printed, the last line is cleared from the print image buffer and a new line is added in its place on the second label:

```
10  PRPOS 100,300
20  FONT "Swiss 721 BT"
30  PRTXT "HAPPY"
40  A%=FIELDNO
50  PRPOS 100,250
60  PRTXT "NEW YEAR!"
70  PRINTFEED
80  CLL A%
90  PRPOS 100,250
100 PRTXT "BIRTHDAY!"
110 PRINTFEED
RUN
```

Complete clearing:

In this example, the print image buffer will be cleared completely if Error 1030, “Character missing in chosen font” occurs.

```
10  ON ERROR GOTO 1000
   .
   .
   .
1000 IF ERR=1030 GOSUB 1100
1010 RESUME NEXT
   .
   .
1100 CLL
1110 PRINT "CHARACTER MISSING"
1120 RETURN
```
CLOSE

Purpose
Statement closing one or several files and/or devices for input/output.

Syntax
CLOSE[#] <nexp> [, [#] <nexp>...]]

Remarks
This statement revokes OPEN. Only files or devices, which already have been OPENed, can be CLOSEd.

A CLOSE statement for a file or device OPENed for sequential output entails that the data in the buffer will be written to the file/device in question automatically before the channel is closed.

When a file OPENed for random access is CLOSEd, all its FIELD definitions will be lost.

END, NEW, and RUN will also close all open files and devices.

Examples
This statement closes all open files and devices:
200 CLOSE

A number of files or devices (No. 1-4) can be closed simultaneously using any of the following types of statement:

200 CLOSE 1,2,3,4
or
200 CLOSE #1,#2,#3,#4
or
200 CLOSE 1,2,#3,4
**COM ERROR ON/OFF**

**Purpose**
Statement enabling/disabling error handling on the specified communication channel.

**Syntax**
```
COM <nexp> ERROR[ON|OFF]
```

<nexp> is one of the following communication channels:
1 = "uart1:"
2 = "uart2:"
3 = "uart3:"
4 = "centronics:"
7 = "uart4:"
8 = "uart5"

Default: COM ERROR OFF on all channels.

**Remarks**
This function is closely related to COMSET, ON COMSET GOSUB, COMSET ON, COMSET OFF, COMSTAT, and COMBUF$.

Each character received is checked for the following errors:
- Received break
- Framing error
- Parity Error
- Overrun error

If any such communication error occurs and COM ERROR is ON for the channel in question, the reception will be interrupted. This condition can be read by means of a COMSTAT function, but you cannot read exactly what type of error has occurred. COM ERROR OFF disables this type of error-handling for the specified channel.

COM ERROR ON cannot be used with USB (comm. channel #6).

**Example**
In this example, a message will appear on the screen when the reception is interrupted by any of four COMSET conditions being fulfilled:

```
10  COM ERROR 1 ON
20  A$="Max. number of char. received"
30  B$="End char. received"
40  C$="Attn. string received"
50  D$="Communication error"
60  ON COMSET 1, "A", CHR$(90), ",", "BREAK", 20
70  ON COMSET 1 GOSUB 1000
80  COMSET 1 ON
90  IF QDATA$="" THEN GOTO 90
100 END
1000 QDATA$=COMBUF$(1)
1010 IF COMSTAT(1) AND 2 THEN PRINT A$
1020 IF COMSTAT(1) AND 4 THEN PRINT B$
1030 IF COMSTAT(1) AND 8 THEN PRINT C$
1040 IF COMSTAT(1) AND 32 THEN PRINT D$
1050 PRINT QDATA$:RETURN
```
COMBUF$  

**Purpose**  
Function reading the data in the buffer of the communication channel specified by a COMSET statement.

**Syntax**  
COMBUF$ (<nexp>)

<nexp> is one of the following communication channels:
1 = "uart1:"
2 = "uart2:"
3 = "uart3:"
4 = "centronics:"
5 = "net1:"
6 = "usb1:"
7 = "uart4:"
8 = "uart5"

**Remarks**  
This function is closely related to COMSET, ON COMSET GOSUB, COMSET ON, COMSET OFF, COM ERROR ON/OFF, and COM-STAT. Using COMBUF$, the buffer can be read and the content be used in your program.

When the communication has been interrupted by any of the three conditions "end character", "attention string", or "max. no. of char." (see COMSET), you may use an ON COMSET GOSUB subroutine and assign the data from the buffer to a variable as illustrated in the example below.

Note that COMBUF$ filters out any incoming ASCII 00 dec. characters (NUL) by default. Filtering can be enabled/disabled using SYSVAR(44).

**Example**  
In this example, the data from the buffer is assigned to the string variable A$ and printed on the screen:

```plaintext
1    REM Exit program with #STOP&
10   COMSET1, "#", "&", "XYZ", ",=", 50
20   ON COMSET 1 GOSUB 2000
30   COMSET 1 ON
40   IF A$ <> "STOP" THEN GOTO 40
50   COMSET 1 OFF
.....
1000  END
2000  A$ = COMBUF$(1)
2010  PRINT A$
2020  COMSET 1 ON
2030  RETURN
```
**COMSET**

**Purpose**
Statement setting the parameters for background reception of data to the buffer of a specified communication channel (see COMBUF$).

**Syntax**

```
COMSET<nexp1>,<sexp1>,<sexp2>,<sexp3>,<sexp4>,<nexp2>
```

- `<nexp1>` is one of the following communication channels:
  1 = "uart1:"
  2 = "uart2:"
  3 = "uart3:"
  4 = "centronics:"
  5 = "net1:"
  6 = "usb1:"
  7 = "uart4:"
  8 = "uart5"

- `<sexp1>` specifies the start of the message string (max. 12 char).
- `<sexp2>` specifies the end of the message string (max. 12 char).
- `<sexp3>` specifies characters to be ignored (max. 42 char).
- `<sexp4>` specifies the attention string (max. 12 char).
- `<nexp2>` specifies the max. number of characters to be received. Enter a value ≥1. (If `<nexp2>` = 0, the first character will be lost.)

**Remarks**

Data can be received by a buffer on each of the communication channels without interfering with the running of the current program. At an appropriate moment, the program can fetch the data in the buffer and use them according to your instructions. Such background reception has priority over any ON KEY GOSUB statement.

Related instructions are COMSTAT, ON COMSET GOSUB, COMSET ON, COMSET OFF, COM ERROR ON/OFF, and COMBUF$.

The communication channels are explained in connection with the DEVICES statement.

The start and end strings are character sequences which tells the printer when to start or stop receiving data. Max. 12 characters, may be "".

It is possible to make the printer ignore certain characters. Such characters are specified in a string, where the order of the individual characters does not matter. Max. 42 characters, may be "".

The attention string interrupts reception. Max. 12 characters, may be "".

The length of the afore-mentioned COMSET strings are checked before they are copied into the internal structure. If any of these strings are too long, Error 26, “Parameter too large” will occur.

When the printer has received the specified maximum number of characters, without previously having encountered any end string or attention string, the transmission will be interrupted. The maximum number of characters also decides how much of the memory will be allocated to the buffer.
The reception of data to the buffer can be interrupted by four conditions:

- If an end string being encountered.
- If an attention string being encountered.
- If the maximum number of characters being received.
- If error-handling is enabled for the communication channel in question (see COM ERROR ON/OFF) and an communication error occurs. This condition can be checked by a COMSTAT function.

Any interruption will have a similar effect as a COMSET OFF statement, that is close the reception, but the buffer will not be emptied and can still be read by a COMBUF$ function. After the reception has been interrupted, an ON COMSET GOSUB statement can be issued to control what will happen next.

COMSET does not support auto-hunting (see SETSTDIO).

**Example**

This example shows how "uart1:" is opened for background communication. Any record starting with the character # and ending with the character & will be received. The characters Z, Y and X will be ignored. The character = will stop the reception. Max. 50 characters are allowed.

```
1    REM Exit program with #STOP&
10   COMSET1,"#","&","ZYX","=",50
20   ON COMSET 1 GOPSUB 2000
30   COMSET 1 ON
40   IF A$ <> "STOP" THEN GOTO 40
50   COMSET 1 OFF
   ....
1000  END
2000  A$ = COMBUF$(1)
2010  PRINT A$
2020  COMSET 1 ON
2030  RETURN
```
COMSET OFF

Purpose
Statement turning off background data reception and emptying the buffer of the specified communication channel.

Syntax

\[
\text{COMSET}<\text{nexp}>\text{OFF}
\]

\(<\text{nexp}>\) is one of the following communication channels:
1 = "uart1:"
2 = "uart2:"
3 = "uart3:"
4 = "centronics:"
5 = "net1:"
6 = "usb1:"
7 = "uart4:"
8 = "uart5"

Remarks
This statement is closely related to COMSET, ON COMSET GOSUB, COMSTAT, COMSET ON, COM ERROR ON/OFF, and COMBUF$.

The COMSET OFF statement closes the reception and empties the buffer of the specified communication channel.

Example
In this example, the COMSET OFF statement is used to close "uart1:” for background reception and empty the buffer:

\[
\begin{align*}
1 & \quad \text{REM Exit program with #STOP&} \\
10 & \quad \text{COMSET1,"#","&","ZYX","="},50 \\
20 & \quad \text{ON COMSET 1 GOSUB 2000} \\
30 & \quad \text{COMSET 1 ON} \\
40 & \quad \text{IF A$ <> "STOP" THEN GOTO 40} \\
50 & \quad \text{COMSET 1 OFF} \\
\end{align*}
\]

\[
\begin{align*}
1000 & \quad \text{END} \\
2000 & \quad \text{A$ = COMBUF$(1)} \\
2010 & \quad \text{PRINT A$} \\
2020 & \quad \text{COMSET 1 ON} \\
2030 & \quad \text{RETURN}
\end{align*}
\]
COMSET ON

**Purpose**
Statement emptying the buffer and turning on background data reception on the specified communication channel.

**Syntax**

```
COMSET<nexp>ON
```

<nexp> is one of the following communication channels:

1 = "uart1:"
2 = "uart2:"
3 = "uart3:"
4 = "centronics:"
5 = "net1:"
6 = "usb1:"
7 = "uart4:"
8 = "uart5"

**Remarks**
This statement is closely related to COMSET, ON COMSET GOSUB, COMSTAT, COMSET OFF, COM ERROR ON/OFF, and COMBUFS. It allows you to open any of the communication channels for background data reception with an empty buffer, provided the communication parameter for the channel has already been set up by a COMSET statement.

When the reception has been interrupted by the reception of an end character, an attention string or the max. number of characters, the buffer can be emptied and the reception reopened by issuing a new COMSET ON statement.

**Example**
In this example, the COMSET ON statement on line 30 is used to open "uart1:" for background reception. After the buffer has been read, it is emptied and the reception is reopened by a new COMSET ON statement in the subroutine on line 2020:

```
1  REM Exit program with #STOP&
10  COMSET1, "#", "&", "ZYX", ",", 50
20  ON COMSET 1 GOSUB 2000
30  COMSET 1 ON
40  IF A$ <> "STOP" THEN GOTO 40
50  COMSET 1 OFF
.....
1000  END
2000  A$= COMBUFS$(1)
2010  PRINT A$
2020  COMSET 1 ON
2030  RETURN
```
COMSTAT

Purpose

Function reading the status of the buffer of a communication channel.

Syntax

\texttt{COMSTAT(<nexp>)}

<\texttt{nexp}> is one of the following communication channels:

1 = "uart1:"
2 = "uart2:"
3 = "uart3:"
4 = "centronics:"
5 = "net1:"
6 = "usb1:"
7 = "uart4:"
8 = "uart5"

Remarks

This function is closely related to COMSET, ON COMSET GOSUB, COMSET ON, COMSET OFF, COM ERROR ON/OFF, and COMBUF$. It allows you to find out if the buffer is able to receive background data, or—if not—what condition has caused the interruption.

The buffer’s status is indicated by a numeric expression, which is the sum of the values given by the following conditions:

- Copy of hardware handshake bit (not on "net1:" or "usb1:"), 0 or 1
- Interruption: Max. number of characters received, 2
- Interruption: End character received, 4
- Interruption: Attention string received, 8
- Interruption: Communication error (not on "net1:" or "usb1:"), 32

Example

A message will appear on the screen when the reception is interrupted by any of four COMSET conditions being fulfilled:

10 \texttt{COM ERROR 1 ON}
20 \texttt{A$="Max. number of char. received"}
30 \texttt{B$="End char. received"}
40 \texttt{C$="Attn. string received"}
50 \texttt{D$="Communication error"}
60 \texttt{COMSET 1, "A",CHR$(90),"#","BREAK",20}
70 \texttt{ON COMSET 1 GOSUB 1000}
80 \texttt{COMSET 1 ON}
90 \texttt{IF QDATA$="" THEN GOTO 90}
100 \texttt{END}
1000 \texttt{QDATA$=COMBUF$(1)}
1010 \texttt{IF COMSTAT(1) AND 2 THEN PRINT A$}
1020 \texttt{IF COMSTAT(1) AND 4 THEN PRINT B$}
1030 \texttt{IF COMSTAT(1) AND 8 THEN PRINT C$}
1040 \texttt{IF COMSTAT(1) AND 32 THEN PRINT D$}
1050 \texttt{PRINT QDATA$}
1060 \texttt{RETURN}
CONT

Purpose
Statement for resuming execution of a program that has been interrupted by means of a STOP, BREAK, or DDBREAK statement.

Syntax
CONT

Remarks
The CONT statement may be used to resume program execution after a STOP, BREAK, or DDBREAK statement has been executed. Execution continues at the point where the break happened with the STDIO settings restored.

CONT is usually used in conjunction with DDBREAK or STOP for debugging. When execution is stopped, you can examine or change the values of variables using direct mode statements. You may then use CONT to resume execution. CONT is invalid if the program has been edited during the break.

It is also possible to resume execution at a specified program line using a GOTO statement in the immediate mode.

Example
10 A%=100
20 B%=50
30 IF A%=B% THEN GOTO QQQ ELSE STOP
40 GOTO 30
50 QQQ:PRINT "Equal"

Ok
RUN
Break in line 30

Ok
PRINT A%
100

Ok
PRINT B%
50

Ok
B%=100

Ok
CONT
Equal

Ok
COPY

Purpose
Statement for copying files.

Syntax
COPY<sexp₁>[,<sexp₂>]

<sexp₁> is the name and optionally directory of the original file.
<sexp₂> is, optionally, a new name and/or directory for the copy.

Remarks
This statement allows you to copy a file to another name and/or directory as an alternative to LOADING the file in question and then SAVEing it.

If no directory is specified for the original and/or copy, the current directory will be used by default (see CHDIR statement). By default, the current directory is "/c", which is the printer’s permanent memory. If the file is to be copied from or to another directory than the current one, the file name must contain a directory reference.

A file cannot be copied to the same name in the same directory.

In addition to copying files to the printer’s permanent or temporary memory or a DOS-formatted memory card, a file can also be copied to an output device such as the printer’s display or a serial communication channel. Copying a program to the standard OUT channel has the same effect as LOADING and LISTing it.

Note that bitmap fonts and images are not files and therefore cannot be copied.

Examples
In the following examples, "/c" is the current directory.
Copying a file from "card1:" to the current directory without changing the file name:
COPY "card1:LABEL1.PRG"

Copying a file from "card1:" to the current directory and changing the file name:
COPY "card1:FILELIST.PRG", "COPYTEST.PRG"

Copying a file from "/c:" to a directory other than the current one without changing the file name:
COPY "/c/FILELIST.PRG", "card1:FILELIST.PRG"

Copying a file in the current directory to a new name within the same directory:
COPY "LABEL1.PRG", "LABEL2.PRG"

Copying a file in the current directory to serial channel "uart1:":
COPY "LABEL1.PRG", "uart1:"
**COUNT&**

**Purpose**
Statement for creating a counter (Intermec Direct Protocol only).

**Syntax**

\[ \text{COUNT&}<\text{sexp}_1>,<\text{nexp}_1>,<\text{sexp}_2> \]

- \(<\text{sexp}_1>\)
  - is the type of counter parameter to be set:
    - START (start value)
    - WIDTH (minimum number of digits)
    - COPY (number of copies before update)
    - INC (increment/decrement at update)
    - STOP (stop value)
    - RESTART (restart counting at this value)

- \(<\text{nexp}_1>\)
  - is the counter reference number (integers only)

- \(<\text{sexp}_2>\)
  - is the parameter value

**Remarks**
This instruction can only be used in the Intermec Direct Protocol.

The counters can be used in text and bar code fields and are global, which means that they are not connected to any special label or layout, but will be updated at every execution of PRINTFEED statements where the counter in question is used.

Counters are designated using positive integers, for example 1, 2, or 3. When used for printing, they are referred to by “CNT<ncon>$” variables, where <ncon> is the number of the counter as specified by COUNT&, for example CNT5$. A counter variable without a matching counter will be regarded as a common string variable.

The value of the start, stop, and restart parameters decide the type of counter (alpha or numeric). If different types of counter are specified in these parameters, the last entered parameter decides the type. Alpha counters count A-Z whereas numeric counters use numbers without any practical limit.

Counters are not saved in the printer’s memory, but will have to be recreated after each power up. Therefore, it may be wise to save the COUNT& statements as a file in the host.

**START:**
Decides the first value to be printed. If a single letter is entered (A-Z), the counter will become an alpha counter, and if one or several digits are entered the counter will be numeric. Numeric values can be positive or negative. Negative values are indicated by a leading minus sign.
Default: 1 (numeric) or A (alpha)
COUNT&, cont.

WIDTH:
This parameter can only be used in numeric counters and decides the minimum number of digits to be printed. If the counter value contains a lesser number of digits, leading zero (0) characters will be added until the specified number of digits is obtained. If the number of digits in the counter value is equal to or larger than specified in the width parameter, the value will be printed in its entity.
Default: 1 (no leading zeros)

COPY:
Decides how many copies (labels etc.) will be printed before the counter is updated according to the INC parameter.
Default: 1

INC:
Decides by which value the counter should be incremented or decremented when it is updated. In case of decrementation, the value should contain a leading minus sign.
Default: 1

STOP:
Decides the value after which the counter should start all over again at the value specified by the RESTART parameter. If a single letter is entered (A-Z), the counter will become an alpha counter, and if one or several digits are entered the counter will be numeric. When a counter is decremented, a stop value less than the start value must be given, since the default stop value will never be reached.
Default: 2,147,483,647 (numeric) or Z (alpha)

RESTART:
Decides at which value the counter should start all over again after having exceeded the STOP parameter value. If a single letter is entered (A-Z), the counter will become an alpha counter, and if one or several digits are entered the counter will be numeric.
Default: 1 (numeric) or A (alpha)

Examples
In this example, a counter is created. It will start at number 100 and be updated by a value of 50 after every second label until the value 1000 is reached. Then the counter will start again at the value 200. All values will be expressed as 4-digit numbers with leading zeros.

COUNT& "START",1,"100"  
COUNT& "WIDTH",1,"4"  
COUNT& "COPY",1,"2"  
COUNT& "INC",1,"50"  
COUNT& "STOP",1,"1000"  
COUNT& "RESTART",1,"200"
### CURDIR$

**Purpose**
Function returning the current directory as the printer stores it.

**Syntax**

```plaintext
CURDIR$
```

**Example**

```plaintext
CHDIR "/c/DIR1/DIR2"
PRINT CURDIR$
```

yields:

```
/c/DIR1/DIR2
```
**CSUM**

**Purpose**
Statement calculating the checksum of an array of strings.

**Syntax**

\[
\text{CSUM}<\text{ncon}>,<\text{svar}>,<\text{nvar}>
\]

- \(<\text{ncon}>\) is the type of checksum calculation:
  1: Longitudinal Redundancy Check (LRC)
  2: Diagonal Redundancy Check (DRC)
  3: Longitudinal Redundancy Check (LRC)

- \(<\text{svar}>\) is the array of strings of which the checksum is to be calculated.
  - if \(<\text{ncon}> = 1\) or \(<\text{ncon}> = 2\): The array of strings of which the checksum is to be calculated.
  - if \(<\text{ncon}> = 3\): Checksum string.

- \(<\text{nvar}>\) is the variable in which the result will be presented.

**Remarks**
These types of checksum calculation can only be used for string arrays, not for numeric arrays. In case of \(\text{CSUM 3,}<\text{svar}>,<\text{nvar}>\), the resulting variable will be the indata for next \(\text{CSUM}\) calculation, unless the variable is reset.

**Example**
In this example, the DRC checksum of an array of strings is calculated:

```plaintext
10  ARRAY$(0) = "ALPHA"
20  ARRAY$(1) = "BETA"
30  ARRAY$(2) = "GAMMA"
40  ARRAY$(3) = "DELTA"
50  CSUM 2,ARRAY$,B%
60  PRINT B% :REM DRC CHECKSUM
RUN
```

Yields:

252
CUT

Purpose
Statement activating an optional cutter.

Syntax
CUT

Remarks
Obviously, this statement only works with printers fitted with a cutter. A cutter is normally used to cut non-adhesive paper strip or to cut through the liner between self-adhesive labels.

When a PRINTFEED statement is executed, the printer feeds out a certain amount of the media according to the printer's setup in regard of startadjust and stopadjust, as explained in its User's Guide. Then the cutter can be activated by a CUT statement.

Example
This program orders the printer to print a text and then cut off the media:
10 PRPOS 250,250
20 DIR 1
30 ALIGN 4
40 FONT "Swiss 721 BT"
50 PRTXT "Hello everybody!"
60 PRINTFEED
70 CUT
RUN
CUT ON/OFF

Purpose
Statement enabling or disabling automatic cutting after PRINTFEED execution and optionally adjusting the media feed before and after the cutting.

Syntax
CUT [<nexp>] ON|CUT OFF

<nexp> is optionally the length of media to be fed out before cutting and pulled back after cutting.

Default: CUT OFF

Remarks
This statement makes it possible to enable or disable automatic execution of a CUT operation directly after the execution of each PRINTFEED statement. If any extra media feed in connection with the cutting operation is required, use startadjust and stopadjust setup or specify the desired length of media to be fed out before the cutting is performed and pulled back afterwards in the CUT ON statement.

The amount of media feed (<nexp>) will not automatically be reset to 0 (zero) by an CUT OFF statement, but must be manually be specified as 0 (CUT 0 ON:CUT OFF). However, a reboot resets it to 0.

Example
This program enables automatic cutting and orders the printer to print a text and feed out an extra amount of strip before cutting the media. The media is then pulled back the same distance:

```
10 CUT 280 ON
20 PRPOS 250,250
30 DIR 1
40 ALIGN 4
50 FONT "Swiss 721 BT"
60 PRTXT "Hello everybody!"
70 PRINTFEED
RUN
```
**DATE$**

**Purpose**
Variable for setting or returning the current date.

**Syntax**

<table>
<thead>
<tr>
<th>Setting the date:</th>
<th>DATE$=&lt;sexp&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;sexp&gt;</td>
<td>sets the current date by a 6-digit number specifying Year, Month and Day (YYMMDD).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returning the date:</th>
<th>&lt;svar&gt;=DATE$[(&lt;sexp&gt;)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;svar&gt;</td>
<td>returns the current date according to the printer’s calendar.</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>is an optional flag “F”, indicating that the date will be returned according to the format specified by FORMAT DATE$.</td>
</tr>
</tbody>
</table>

**Remarks**
This variable works best if a real-time clock circuit (RTC) is fitted on the printer’s CPU board. The RTC is battery backed-up and will keep record of the time even if the power is turned off or lost.

If no RTC is installed, the internal clock will be used. After startup, an error will occur when trying to read the date or time before the internal clock has been manually set by means of either a DATE$ or a TIMES$ variable. If only the date is set, the internal clock starts at 00:00:00 and if only the time is set, the internal clock starts at Jan 01, 1980. After setting the internal clock, you can use the DATE$ and TIMES$ variables the same way as when an RTC is fitted, until a power off or REBOOT causes the date and time values to be lost.

Date is always entered and, by default, returned in the order YYMMDD, where:
YY = Year (Last two digits (for example 2003 = 03))
MM = Month (Two digits (01-12))
DD = Day (Two digits (01-28|29|30|31))

Example: December 1, 2003 is entered as “031201”.

The built-in calendar corrects illegal values for the years 1980-2048, for example the illegal date 031232 will be corrected to 040101.

The format for how the printer will return dates can be changed by means of a FORMAT DATE$ statement and returned by DATE$“(F)”.

**Example**
Setting the date and then returning the date in two different formats:

10 DATE$ = ”031201” (sets date)
20 FORMAT DATE$ ”DD/MM/YYYY” (sets date format)
30 PRINT DATE$ (returns unformatted date)
40 PRINT DATE$“(F)” (returns formatted date)

**RUN**

yields:

031201
01/12/2003
**DATEADD$**

**Purpose**
Function returning a new date after a number of days have been added to, or subtracted from, the current date or optionally a specified date.

**Syntax**

```
DATEADD$([<sexp1>],[<nexp>][,<sexp2>])
```

- `<sexp1>` is any date given according to the DATE$ format, which a certain number of days should be added to or subtracted from.
- `<nexp>` is the number of days to be added to (or subtracted from) the current date or optionally the date specified by `<sexp1>`.
- `<sexp2>` is one or two optional flags "F" or "M":
  - "F" specifies that the date will be returned according to the format specified by FORMAT DATE$.
  - "M" specifies the value given in `<nexp>` will mean months instead of days.

**Remarks**
The original date (`<sexp1>`) should be entered according to the syntax for the DATE$ variable, that is in the order YYMMDD, where:

- **YY** = Year  Last two digits  (for example 2003 = 03)
- **MM** = Month  Two digits  (01-12)
- **DD** = Day  Two digits  (01-28|29|30|31)

Example: December 1, 2003 is entered as "031201".

The built-in calendar corrects illegal values for the years 1980-2048, for example the illegal date 031232 will be corrected to 040101. When for example, adding one month to the date 2004-01-31, the result will be rounded off to 2004-02-29 (2004 is a leap year).

The number of days to be added or subtracted should be specified as a positive or negative numeric expression respectively.

If no "F" flag is included in the DATEADD$ function, the result will be returned according to the DATE$ format, see above.

If the DATEADD$ function includes an "F" flag, the result will be returned in the format specified by FORMAT DATE$.

"F" and "M" flags can be combined in the same instruction without any separating character, that is, "FM" or "MF". No other characters are accepted.
DATEADD$, cont.

Examples

```
10    DATE$ = "031201"
20    A%=15
30    B%=-10
40    FORMAT DATE$ "DD/MM/YY"
50    PRINT DATEADD$("031201",A%)
60    PRINT DATEADD$("031201",A%,"F")
70    PRINT DATEADD$(B%,"F")
RUN

```
yields:

```
031216
16/12/03
21/11/03

DATE$="040131"
FORMAT DATE$ "YYYY/MM/DD"
? DATEADD$(1,"F"),DATEADD$(1,"M"),DATEADD$(1,"FM")
2004/02/01  040229  2004/02/29
Ok
? DATEADD$("040131",1,"F"),→
→DATEADD$("040131",2,"M"),→
→DATEADD$(040131",3,"FM")
2004/02/01  040331  2004/04/30
Ok

```
**DATEDIFF**

**Purpose**
Function returning the difference between two dates as a number of days.

**Syntax**

```
DATEDIFF(<sexp₁>,<sexp₂>)
```

- `<sexp₁>` is one of two dates (date 1).
- `<sexp₂>` is the other of two dates (date 2).

**Remarks**

To get the result as a positive numeric value, the two dates, for which the difference is to be calculated, should be entered with the earlier of the dates (date 1) first and the later of the dates (date 2) last, see the first example below.

If the later date (date 2) is entered first, the resulting value will be negative, see the second example below.

Both dates should be entered according to the syntax for the DATE$ variable, that is in the order YYMMDD, where:

- **YY** = Year, last two digits (for example 2003 = 03)
- **MM** = Month, two digits (01-12)
- **DD** = Day, two digits (01-28/29/30/31)

Example: December 1, 2003 is entered as "031201".

The built-in calendar corrects illegal values for the years 1980-2048, for example the illegal date 031232 will be corrected to 040101.

**Examples**

Calculation of the difference in days between the dates October 1, 2003 and November 30, 2003:

```
10   A$=DATEDIFF("031001","031130")
20   PRINT A$
RUN
```
yields:

60

If the later date is entered first, the result will be negative:

```
10   A$=DATEDIFF("031130","031001")
20   PRINT A$
RUN
```
yields:

-60
**DBBREAK**

**Purpose**
Statement for adding or deleting a breakpoint for the Fingerprint Debugger.

**Syntax**

```
DBBREAK<nexp>|<sexp>[ON|OFF]
```

- `<nexp>` is the line number where the debugger will break and also the name of the breakpoint.
- `<sexp>` is the line label where the debugger will break and also the name of the breakpoint.
- `ON` adds the specified breakpoint (default).
- `OFF` deletes the specified breakpoint.

**Remarks**
The execution of a program will break at each program line, that has been specified as a breakpoint, and the message “break in line nnn” will be transmitted on the Debug STDOUT port. If a CONT statement is issued, the execution will continue at next line, whereas if RUN is issued, the execution will start again from the first program line.

The line number or line label does not have to exist when a breakpoint is added, but if a non-existing breakpoint is deleted an error will occur (Error 39 or 70).

There is no error given if a breakpoint is added more than once. When a breakpoint is deleted, all breakpoints with the same name are deleted at the same time. There will only be one break for each line even if there are more than one breakpoint on that line.

When a NEW statement is issued, all breakpoints will be deleted.

If a breakpoint is set on a line with a call to a FOR or WHILE loop, there will only be one break on that line (the first time it is executed).

Related instructions are DBBREAK OFF, DBEND, DBSTDIO, DBSTEP, and STOP.

**Example**

```
10 PRINT "A"
20 PRINT "B"
30 PRINT "C"
DBBREAK 20 ON
RUN
```

```yields:
A
Break in line 20```
DBBREAK OFF

**Purpose**
Statement for deleting all breakpoints for the Fingerprint Debugger.

**Syntax**
```
DBBREAK OFF
```

**Remarks**
This statement is similar to `DBBREAK<nexp>|<sexp>OFF` but deletes all breakpoints instead of just one breakpoint at the time.

Related instructions are `DBBREAK`, `DBEND`, `DBSTDIO`, and `DBSTEP`.
DBEND

Purpose
Statement for terminating the Fingerprint Debugger.

Syntax
DBEND

Remarks
This statement is used for terminating the Fingerprint Debugger prematurely and restore the STDIO settings as they were before the Debugger was started.

Related instructions are DBBREAK, DBBREAK OFF, DBSTDIO, and DBSTEP.
DBSTDIO

Purpose
Statement for selecting the standard IN/OUT channel for the Fingerprint Debugger.

Syntax

```
DBSTDIO <nexp_1>,<nexp_2>[,<sexp_1>,<sexp_2>]
```

```
DBSTDIO [<nexp_1>,<nexp_2>,]<sexp_1>,<sexp_2>
```

- `<nexp_1>` is the desired Debug STDIN channel:
  - 0 = "console:"
  - 1 = "uart1:" (default)
  - 2 = "uart2:"
  - 3 = "uart3:"
  - 4 = "centronics:"
  - 5 = "net1:"
  - 6 = "usb1:"
  - 7 = "uart4:"
  - 8 = "uart5:"

- `<nexp_2>` is the desired Debug STDOUT channel:
  - 0 = "console:"
  - 1 = "uart1:" (default)
  - 2 = "uart2:"
  - 3 = "uart3:"
  - 5 = "net1:"
  - 6 = "usb1:"
  - 7 = "uart4:"
  - 8 = "uart5:"

- `<sexp_1>` Preamble (default: empty string)
- `<sexp_2>` Postamble (default: empty string)

Remarks
The maximum size of the preamble and postamble strings is 12 characters.

Related instructions are CONT, DDBREAK, DDBREAK OFF, DBEND, DBSTEP, and STOP.

Example
This statement selects "uart2:" as Debug STDIO channel. Preamble is specified as “in” and postamble as “out”:
```
DBSTDIO 2,2,"in","out"
```
Chapter 2—Program Instructions

DBSTEP

Purpose
Statement for specifying the interval between breaks for the Fingerprint Debugger and execute the program accordingly.

Syntax
```
DBSTEP<ncon>
```

<ncon> is the number of lines to be executed before break. Default: 1 line.

Remarks
If <ncon> is omitted, one line will be executed, but if <ncon> is specified as 0, nothing at all will happen.

DBSTEP cannot be used in execution mode (yields Error 78).

When DBSTEP is used on the last line in a program, the line will be executed but there will be no break.

If DBSTEP is used in a program with a FOR or WHILE loop, there will only be one break on the line which is calling for the FOR or WHILE loop (the first time it is executed).

Related instructions are CONT, DBBREAK, DBBREAK OFF, DBEND, and DBSTDIO.

Example
```
10 PRINT "11"
20 PRINT "22"
30 PRINT "33"
40 PRINT "44"
50 PRINT "55"
60 PRINT "66"
70 PRINT "77"
80 PRINT "88"
90 PRINT "99"
DBSTEP 4
11
22
33
44
Break in line 50
Ok
DBSTEP
55
Break in line 60
Ok
DBSTEP 2
66
77
Break in line 80
CONT
88
99
Ok
```
DELETE

Purpose
Statement deleting one or several consecutive program lines from the printer’s working memory.

Syntax
DELETE<ncon1>[<ncon2>]

<ncon1> is the line, or the first line in a range of lines, to be deleted.
<ncon2> is (optionally) the last line in a range of program lines to be deleted.

Remarks
This statement can only be used for editing the current program in the Immediate Mode and cannot be included as a part of the program execution.

Examples
DELETE 50  deletes line 50 from the program.
DELETE 50-100 deletes line 50 thru 100 from the program.
DELETE 50- deletes all lines from line 50 to the end of the program.
DELETE -50 deletes all lines from the start of the program to line 50.
Chapter 2—Program Instructions

**DELETEPFSVAR**

**Purpose**
Statement for deleting variables saved at power failure.

**Syntax**
```
DELETEPFSVAR <sexp>
```

<sexp> is the name of the variable to be deleted.

**Remarks**
Related instructions are SETPFSVAR, GETPFSVAR, and LISTPFSVAR.

**Examples**
```
DELETEPFSVAR "QCPS%"
DELETEPFSVAR "QS$"
```
DEVICES

Purpose

Statement for returning the names of all devices on the standard OUT channel.

Syntax

DEVICES

Remarks

All devices available to the user in the Intermec Fingerprint firmware will be listed, regardless if they are installed or not. The only exception is that centronics will be listed for the EasyCoder PD41 only if installed. There are also a number of devices for internal use only. The list below indicates if and how the device can be OPENed (see OPEN statement). If you try to OPEN a device, which is not fitted or is disconnected, the message "Error in file name" will be printed to the standard OUT channel (see SETST-DIO). Note that all names of devices are lowercase and most are appended by a colon (:)..

<table>
<thead>
<tr>
<th>Device</th>
<th>Explanation</th>
<th>Can be OPENed for...</th>
</tr>
</thead>
<tbody>
<tr>
<td>c: (= &quot;/c/&quot; )</td>
<td>Printer's permanent memory</td>
<td>Input/Output/Random</td>
</tr>
<tr>
<td>card1:</td>
<td>CompactFlash memory card</td>
<td>Input/Output/Random</td>
</tr>
<tr>
<td>centronics:</td>
<td>Parallel communication port</td>
<td>Input</td>
</tr>
<tr>
<td>console:</td>
<td>Printer's display and/or keyboard</td>
<td>Input/Output</td>
</tr>
<tr>
<td>dll:</td>
<td>Special applications only</td>
<td>–</td>
</tr>
<tr>
<td>finisher:</td>
<td>The finisher interface</td>
<td>Input/Output</td>
</tr>
<tr>
<td>lock:</td>
<td>Electronic keys</td>
<td>Input</td>
</tr>
<tr>
<td>net1:</td>
<td>EasyLAN</td>
<td>Input/Output</td>
</tr>
<tr>
<td>par:</td>
<td>Special applications only</td>
<td>–</td>
</tr>
<tr>
<td>rom: (= &quot;/rom/&quot; )</td>
<td>Kernel and Read-only memory card</td>
<td>Input</td>
</tr>
<tr>
<td>rs485:</td>
<td>RS-485 communication</td>
<td>Input/Output</td>
</tr>
<tr>
<td>storage:</td>
<td>Electronic keys</td>
<td>Input/Output/Random</td>
</tr>
<tr>
<td>tmp:</td>
<td>Printer's temporary memory</td>
<td>Input/Output/Random</td>
</tr>
<tr>
<td>uart1:</td>
<td>Serial communication port</td>
<td>Input/Output</td>
</tr>
<tr>
<td>uart2:</td>
<td>Serial communication port</td>
<td>Input/Output</td>
</tr>
<tr>
<td>uart3:</td>
<td>Serial communication port</td>
<td>Input/Output</td>
</tr>
<tr>
<td>uart4:</td>
<td>Serial communication port</td>
<td>Input/Output</td>
</tr>
<tr>
<td>uart5:</td>
<td>Serial communication port</td>
<td>Input/Output</td>
</tr>
<tr>
<td>usb1:</td>
<td>Serial communication port</td>
<td>Input/Output</td>
</tr>
<tr>
<td>wand:</td>
<td>Data from Code 128 bar code</td>
<td>Input</td>
</tr>
</tbody>
</table>

c: or /c/ is the printer's permanent read/write memory (Flash SIMMs). It supports file system with directories and will retain its content when the power is switched off. For compatibility with programs created in previous versions of Intermec Fingerprint, the designation "ram:" will also be accepted.

card1: is a read/write DOS-formatted CompactFlash memory card inserted in the printer's memory card adapter.

centronics: is the Centronics parallel port. Three different types can be selected by means of SYSVAR(25).
DEVICES, cont.

console: is the printer’s display and keyboard. The keyboard can be used for input only and the display for output only.

dll: is used for special applications only.

finisher: is the device controlling the finisher interface, where for instance a cutter can be connected.

lock: is an electronic key item that has been specified as locks by means of special software. An electronic key may contain several key items with different properties (counter, lock, or storage). The device name calls all key items with the corresponding properties. Each key item has a 4-character name, usually appended by a delimiter (?) and a 4-character password. Also see OPEN statement.

net1: is the communication channel for an EasyLAN interface board.

par: is used for special applications only.

rom: or /rom/ is both the read-only kernel sectors in the Boot-Bank flash SIMM, and any resource files on a CompactFlash memory card inserted in the printer’s memory card adapter. It supports file system with directories.

rs485: is used in connection with RS-485 point-to-point or multidrop communication to specify that the RS-485 protocol is used and to specify the protocol address of the unit, for example "rs485:23".

storage: is all electronic key items in the printer that has been specified as storages by means of special software. Note that this memory is comparatively slow.

tmp: is the printer’s temporary read/write memory (SDRAM SIMMs). It will lose its content when the power is turned off or at a power failure. Thus, do not use SDRAM for valuable data that cannot be recreated, but copy it to "/c/". One advantage of using "tmp:" instead of "/c/" is that data can be written to SDRAM faster than to the flash memory. To speed up operation, the Intermec Fingerprint firmware (except program modules with dynamic downloading) is copied from "/rom/" to "tmp:" at startup and used from "tmp:".

uart1: is the standard RS-232 port.

uart2: is an additional serial port on an optional interface board.

uart3: is an additional serial port on an optional interface board.

uart4: is an additional serial port on an optional interface board.

uart5: is an additional serial port on an optional interface board.

usb1: is the standard USB (Universal Serial Bus) port.

wand: is any input from an Code 128 bar code not containing any FNC3 character via a bar code wand or reader connected to the wand interface.
Example

DEVICES

c:
card1:
centronics: (only if an optional parallel interface board is fitted)
console:
dll:
finisher:
lock:
net1: (only if an EasyLAN interface board is fitted)
par:
rom:
rs485: (only if an optional serial interface board is fitted)
storage:
tmp:
uart1: (only if an optional serial interface board is fitted)
uart2: (only if an optional serial interface board is fitted)
uart3: (only if two optional serial interface boards are fitted)
uart4: (only if two optional serial interface boards are fitted)
uart5:
usb1:
wand:
DIM

Purpose
Statement specifying the dimensions of an array.

Syntax
```
DIM<nvar>|<svar>>(<nexp1>[,<nexp2>...])....[,<nvar>|<svar>]
```

- `<nvar>|<svar>` is the name of the array.
- `<nexp1>` is the max. subscript value for the first dimension.
- `<nexp2-10>` are, optionally, the max. subscript value for the following dimensions (No. 2-10).

Remarks
An array is created by entering a variable followed by a number of subscripts (max 10) separated by commas. All the subscripts are enclosed by parentheses. Each subscript represents a dimension. The number of subscripts in an array variable, the first time (regardless of line number) it is referred to, decides its number of dimensions. The number of elements in each dimension is by default restricted to four (No. 0-3).

If more than 4 elements in any dimension is desired, a DIM statement must be issued. Note that 0 = 1:st element, 1 = 2:nd element, etc.

For example ARRAY$(1,2,3) creates a three-dimensional array, where the dimensions each contain 4 elements (0-3) respectively. This corresponds to the statement DIM ARRAY$(3,3,3).

It is not possible to change the number of dimensions of an array that already has been created during runtime. (Error 57, “Subscript out of range” will occur.)

Considering the printer’s limited memory and other practical reasons, be careful not to make the arrays larger than necessary. A DIM statement can be used to limit the amount of memory set aside for the array.

Examples
This example creates an array containing three dimensions with 13 elements each:
```
100 DIM NAME$(12,12,12)
```

Here, two one-dimensional arrays are created on the same program line:
```
10 DIM PRODUCT$(15), PRICE%(12)
20 PRODUCT$(2)="PRINTER"
30 PRICE%(2)=1995
40 PRINT PRODUCT$(2); "$"; PRICE%(2)
RUN
```
yields:

```
PRINTER $1995
```
**DIR**

**Purpose**
Statement specifying the print direction.

**Syntax**

```
DIR<nexp>
```

- `<nexp>` is the print direction (1, 2, 3, or 4).
- Default value: 1
- Reset to default by: PRINTFEED execution

**Remarks**
A change of print direction affects all printing statements, that is PRTXT, PRBAR, PRIMAGE, PRBOX, and PRLINE statements that are executed later in the program until a new DIR statement or a PRINTFEED statement is executed.

The print direction is specified in relation to the media feed direction as illustrated below. The print direction affects the various types of objects as follows:

Text:
DIR, cont.

Bar Codes:

Horizontal "picket fence" printing vs. vertical "ladder" printing.

Images:

The relation of the image and the print direction depends how the image was drawn. An image can only be “rotated” 180°. Thus, it may be useful to have two copies of the image available with different extensions for either horizontal or vertical printing:

DIR 1 & 3, use extension .1
DIR 2 & 4, use extension .2
**Examples**

Printing a label with one line of text and drawing a line beneath the text:

```
10 PRPOS 30,300
20 DIR 1
30 ALIGN 4
40 FONT "Swiss 721 BT",18
50 PRTXT "TEXT PRINTING"
60 PRPOS 30,280
70 PRLINE 555,10
80 PRINTFEED
RUN
```

Printing the same information vertically necessitates new positioning to avoid Error 1003, "Field out of label."

```
10 PRPOS 300,30 (new position)
20 DIR 4 (new direction)
30 ALIGN 4
40 FONT "Swiss 721 BT",18
50 PRTXT "TEXT PRINTING"
60 PRPOS 320,30 (new position)
70 PRLINE 555,10
80 PRINTFEED
RUN
```
DIRNAME$

Purpose
Function returning the names of the directories stored in the specified part of the printer's memory.

Syntax

**DIRNAME$[(<sexp>)]**

<sexp> is the name of the memory device from which the first directory name will be listed.

Remarks
In <sexp>, parts of directory names and wildcards (*) are allowed. If <sexp> is omitted, the next directory name in the same memory device is listed. Can be repeated. When there are no directories left to list, the output string will be empty. Also see FILENAME$.

Example

FILES,A  
Files on /c  
./          0  ../          0  
.setupsaved  239  DIR1/  
STDIO         3  

4124672 bytes free  242 bytes used  

PRINT DIRNAME$("/c/")  
.

Ok  
PRINT DIRNAME$  
..  

Ok  
PRINT DIRNAME$  
DIR1  

Ok
### END

**Purpose**

Statement ending the execution of the current program or subroutine and closing all OPENed files and devices.

**Syntax**

```
END
```

**Remarks**

END can be placed anywhere in a program, but is usually placed at the end. It is also useful for separating the “main” program from possible subroutines with higher line numbers. It is possible to issue several END statements in the same program.

**Example**

A part of a program, which produces fixed line-spacing, may look this way:

```
10  FONT"Swiss 721 BT"
20  X%=300:Y%=350
30  INPUT A$
40  PRPOS X%,Y%
50  PRTXT A$
60  Y%=Y% -50
70  IF Y%>=50 GOTO 30
80  PRINTFEED
90  END
```

The Y-coordinate will be decremented by 50 dots for each new line until it reaches the value 50. The END statement terminates the program.
EOF

Purpose
Function for checking for an end-of-file condition.

Syntax
EOF(<nexp>)

<nexp> is the number assigned to the file when it was OPENed.

Remarks
The EOF function can be used with files OPENed for sequential input in connection with the statements INPUT#, LINE INPUT#, and INPUT$ to avoid the error condition “Input past end” which has no error message. When the EOF function encounters the end of a file, it returns the value -1 (true). If not, it returns the value 0 (false).

Example
10    DIM A%(10)
20    OPEN "DATA" FOR OUTPUT AS #1
30    FOR I%=1 TO 10
40    PRINT #1, I%*1123
50    NEXT I%
60    CLOSE #1
70    OPEN "DATA" FOR INPUT AS #2
80    I%=0
90    WHILE NOT EOF(2)
100   INPUT #2, A%(I%):PRINT A%(I%)
110   I%=I%+1:WEND
120   IF EOF(2) THEN PRINT "End of File"
RUN

yields:

1123
2246
3369
4492
5615
6738
7861
8984
10107
11230
End of File
**ERL**

**Purpose**
Function returning the number of the line on which an error condition has occurred.

**Syntax**
```
ERL
```

**Remarks**
Also useful in connection with an ON ERROR GOTO statement.

**Examples**
You can check at which line the last error since power up occurred like this:
```
PRINT ERL
```
yields for example
```
40
```
In this example, the line number of the line, where an error has occurred, decides the action to be taken (in this case the font size is too large for the label width):
```
10    ON ERROR GOTO 1000
20    FONT "Swiss 721 BT",100
30    PRTXT "HELLO EVERYBODY"
40    PRINTFEED
50    END
1000  IF ERL=40 THEN PRINT "PRINT ERROR"
1010  RESUME NEXT
RUN
```
yields:
```
PRINT ERROR
```
You can use the ERL function in programs without line numbers too, because such programs have automatically generated hidden line numbers that are revealed when the program is LISTed. This is the same program as above but without line numbers:
```
NEW
IMMEDIATE OFF
ON ERROR GOTO QAAA
FONT "Swiss 721 BT",100
PRTXT "HELLO EVERYBODY"
PRINTFEED
END
QAAA:  IF ERL=40 THEN PRINT "PRINT ERROR"
       RESUME NEXT
IMMEDIATE ON
RUN
```
yields:
```
PRINT ERROR
```
ERR

Purpose
Function returning the code number of an error that has occurred.

Syntax

Remarks
The firmware is able to detect a number of error conditions. The errors are represented by code numbers according to Chapter 7, “Error Messages.” The ERR function enables the program to read the coded error number. Thereby you may design your program to take proper action depending on which type of error that may have occurred.

Example
In this example, the code number of the error decides the action to be taken:

```
10    ON ERROR GOTO 1000
   . . . . .
   . . . . .
100   PRTXT "HELLO"
110   PRINTFEED
120   END
   . . . . .
   . . . . .
1000  IF ERR=1005 THEN PRINT "OUT OF PAPER"
1010  RESUME NEXT
```

You can also check the number of the last error since power up:

```
PRINT ERR
```

yields for example:

```
1022
```
**ERR$**

**Purpose**
Function for returning the explanation of an error code in plain text.

**Syntax**

```
ERR$(<nexp>)
```

- `<nexp>` is the error code number

**Remarks**
The explanation of the error is returned in English according to Chapter 7 “Error Messages.”

**Example**

```plaintext
PRINT ERR$(1003)  yields:
Field out of label
```
Chapter 2—Program Instructions

ERROR

Purpose
Statement for defining error messages and enabling error handling for specified error conditions (Intermec Direct Protocol only).

Syntax
```
ERROR <nexp>[,<sexp>]
```

- `<nexp>` is the number of the error condition.
- `<sexp>` is the desired error message.

Remarks
The ERROR statement can only be used in the Intermec Direct Protocol for the purpose of enabling error-handling and creating customized error messages, as described below.

The built-in error-handler of the Intermec Direct Protocol will always handle the following standard errors (display message inside brackets):
- 15 Font not found [Font not found]
- 18 Disk full [Disk full]
- 37 Cutter device not found [Cutter device not found]
- 43 Memory overflow [Memory overflow]
- 1003 Field out of label [Field out of label]
- 1005 Out of paper [Out of paper]
- 1006 No field to print [No field(s)]
- 1022 Head lifted [Head lifted]
- 1027 Out of transfer ribbon [Out of ribbon]
- 1031 Next label not found [Label not found]
- 1058 Transfer ribbon is installed [Ribbon installed]
- 1606 Testfeed not done [Testfeed not done]

Other errors will not be handled unless they have been specified by an ERROR statement. The number of the error should be entered according to the list of error messages at the end of this manual.

The ERROR statement also allows you to edit a suitable message in any language. This message will appear in the printer's display window if the error occurs. The error message will be truncated to 33 characters. Character No. 1-16 will appear on the upper line and character 18-33 will appear on the lower line, whereas character No. 17 always is ignored.

ANSI control characters can be used in the error message string, see “Printer Function Control; Display” in the Intermec Fingerprint, Tutorial. An empty string removes any previously entered message for the error in question. Likewise, an existing message can be replaced by a new one.

When a standard error or an error defined by an ERROR statement is detected, the printer sets its standard IN port to BUSY, sets the “Status” LED to red, and displays the error messages. The error message will be cleared, the LED is set to green, and the standard IN port will be set to READY when the printer’s <Print> key is pressed. In some cases, the error must also be cleared, for example by loading a fresh stock of labels.
**Note:** In printers, that are not fitted with the Intermec Ready-to-Work Indicator console, the green Ready LED will be switched off and the red Error LED will be switched on when an error is detected. The opposite will happen when the error is cleared.

Error messages are not saved in the printer’s memory, but new ERROR statements will have to be downloaded after each power up. Therefore, it is recommended to save a set of ERROR statements as a file in the host computer.

The ERROR statements affect both the error messages in the printer’s display window and the error messages returned to the host via the standard OUT channel (see SETSTDIO statement).

By default, no error messages are returned to the host in the Intermec Direct Protocol, since the statement INPUT ON sets the verbosity level to off, that is SYSVAR (18)= 0. However, the verbosity level can be changed by means of VERBON/VERBOFF statements or the SYSVAR (18) system variable.

Different types of error messages to be returned on the standard OUT channel can be selected by means of the SYSVAR (19) system variable. If SYSVAR (19) is set to 2 or 3, the error message specified by ERROR is transmitted. If no such error message is available, a standard error message in English will be transmitted (see list of Error Messages in Chapter 7).

**Examples**

In these examples, a few errors are specified. Note the blank spaces for character position 17 in each message (space characters are indicated by doubleheaded arrows):

ERROR 1010, "HARDWARE ↔↔↔↔↔↔↔↔↔ ERROR"
ERROR 1029, "PRINTHEAD↔VOLT-↔↔AGE↔TOO↔HIGH"
EXECUTE

Purpose
Statement for executing a Fingerprint program line or a file with Finger-
print program lines from within another Fingerprint program.

Syntax
EXECUTE<sexp>

<sexp> is one line of Fingerprint instructions or the name of a file con-
taining at least one line of a Fingerprint program.

Remarks
This statement allows you to create a library of layouts, subroutines, texts, etc, which can be executed as a part of a program without having to merge the programs.

The program called by EXECUTE must not contain any line numbers or line labels.

If the EXECUTE statement is followed by a string of Fingerprint instructions, they should be separated by colons.

When an error occurs in an EXECUTE file, the line number in the error message is that of the EXECUTE file, not of the program where the EXECUTE statement is issued.

EXECUTE is only allowed in the execute mode, not in the immediate mode (yields Error 69).

Recursive call of EXECUTE is not allowed (yields Error 78).

Example
This example shows how a preprogrammed file containing a bar code is executed as a part of a Fingerprint program, where the input data and printfeed are added:
IMMEDIATE OFF
DIR 1
ALIGN 7
BARSET "CODE39",2,1,3,120
BARFONT "Swiss 721 BT",10,8,5,1,1
BARFONT ON
IMMEDIATE ON
SAVE "tmp:BARCODE.PRG",L

NEW
10 PRPOS 30,400
20 EXECUTE "tmp:BARCODE.PRG"
30 PRBAR "ABC"
40 PRINTFEED
RUN
FIELD

Purpose

Statement for creating a single-record buffer for a random file and dividing the buffer into fields to which string variables are assigned.

Syntax

FIELD[#]<nexp1>,<nexp2>AS<svar1>[,<nexp3>AS<svar2>...]

# indicates that whatever follows is a number. Optional.
<nexp> is the number assigned to the file when it was OPENed.
<nexp> is the number of bytes to be reserved for the string variable that follows. (Null not allowed.)
<svar> is the designation of the string variable, for which space has been reserved.

Remarks

The buffer is divided into fields, each of which is given an individual length in bytes. A string variable is assigned to each field. This statement does not put any data in the buffer, it only creates and formats the buffer, allowing you to place the data using LSET and RSET statements.

Before using this statement, consider the maximum number of characters (incl. space characters) needed for each variable and check that the total does not exceed the record size given when the file was OPENed (by default 128 bytes).

When a file is CLOSED, all its FIELD definitions will be lost.

Example

This example opens and formats a file buffer for a single record. The buffer is divided into three fields, with the size of 25, 30, and 20 bytes respectively.

10 OPEN "ADDRESSES" AS #8 LEN=75
20 FIELD#8,25 AS F1$, 30 AS F2$, 20 AS F3$

(Imagine a spreadsheet matrix where the file is the complete spreadsheet, the records are the lines and the fields are the columns. The buffer can only contain one such line at the time.)
**FIELDNO**

**Purpose**
Function getting the current field number for partial clearing of the print buffer by a CLL statement.

**Syntax**

```
FIELDNO
```

**Remarks**
By assigning the FIELDNO function to one or several numeric variables, you can divide the print buffer into portions, which can be cleared using a CLL statement.

**Example**

```
10  PRPOS 100,300
20  FONT "Swiss 721 BT"
30  PRTXT "HAPPY"
40  A%=FIELDNO
50  PRPOS 100,250
60  PRTXT "NEW YEAR"
70  B%=FIELDNO
80  PRPOS 100, 200
90  PRTXT "EVERYBODY!"
100 PRINTFEED
110 CLL B%
120 PRPOS 100,200
130 PRTXT "TO YOU!"
140 PRINTFEED
150 CLL A%
160 PRPOS 100,250
170 PRTXT "BIRTHDAY"
180 PRPOS 100,200
190 PRTXT "DEAR TOM!"
200 PRINTFEED
RUN
```

yields three labels:

<table>
<thead>
<tr>
<th>#1</th>
<th>#2</th>
<th>#3</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAPPY</td>
<td>HAPPY</td>
<td>HAPPY</td>
</tr>
<tr>
<td>NEW YEAR</td>
<td>NEW YEAR</td>
<td>BIRTHDAY</td>
</tr>
<tr>
<td>EVERYBODY!</td>
<td>TO YOU!</td>
<td>DEAR TOM!</td>
</tr>
</tbody>
</table>
**FILE& LOAD**

**Purpose**
Statement for receiving and storing binary files in the printer’s memory.

**Syntax**
```
FILE& LOAD[<nexp1>,]<sexp>,<nexp2>,[<nexp3>]
```
- `<nexp1>` is optionally the number of bytes to skip before starting to read the file data.
- `<sexp>` is the desired name of the file when stored in the printer’s memory.
- `<nexp2>` is the size of the file in number of bytes.
- `<nexp3>` optionally specifies a communication channel OPENed for INPUT by the number assigned to the device. (Default: Std IN channel.)

**Remarks**
This statement prepares the printer to receive a binary file on the standard IN channel (see SETSTDIO statement) or on another communication channel OPENed for INPUT.

Another, but more cumbersome, way of obtaining the same result is to use the TRANSFER KERMIT statement.

Image files and font files can also be downloaded using the IMAGE LOAD statement.

As opposed to IMAGE LOAD and TRANSFER KERMIT statements, FILE& LOAD will not immediately install the fonts, but the font files will remain as files in the printer’s memory until next power-up.

The optional first parameter makes it possible to use this statement in MS-DOS (CR/LF problem).

The name of the file, when stored in the printer’s memory, may consist of max. 30 characters including possible extension.

The size of the original file should be given in bytes according to its size in the host.

Before the FILE& LOAD statement can be used on a serial channel, the setup must be changed to 8 characters, RTS/CTS handshake. When a FILE& LOAD statement is executed, the execution stops and waits for the number of bytes specified in the statement to be received. During the transfer of file data to the printer, there is a 25 sec. timeout between characters. If a new character has not been received within the timeout limit, an error occurs (Error 80, “Download timeout”). When the specified number of characters have been received, the execution is resumed.

**Example**
```
10    OPEN "uart2:" FOR INPUT AS 5
20    FILE& LOAD "FILE1.PRG",65692,5
30    CLOSE 5
```
FILENAME$

Purpose
Function returning the names of the files stored in the specified part of the printer’s memory.

Syntax
FILENAME$[([<sexp>])]

<sexp>

is the name of the memory device from which the first file name (in alphabetical order) will be listed. Parts of file names and wildcards (*) are allowed. Maximum size is 30 characters.

If <sexp> is omitted, the next file name in the same memory device is listed. Can be repeated. When there are no files left to list, the output string will be empty.

Remarks
The specified memory device must be mounted. The file name must correspond to the name of the file stored in the memory device in regard of upper- and lowercase characters. Wildcards (*) = ASCII 42 dec.) can be used. The list may include all type of files. Even system files, that are preceded by a period character (for example .FONTALIAS), may be listed. No directories will be listed and the order of listing is not specified. Also see DIRNAME$.

Example
This example shows how all files in the printer’s permanent memory (/c) are listed:
FILES,A
Files on /c

./ 0 .. /
.setup.saved 239 DIR1/ 0
STDO 3

4124672 bytes free 242 bytes used

PRINT FILENAME$("/c/")
.setup.saved
Ok
PRINT FILENAME$
STDO

PRINT FILENAME$
Ok
FILES

Purpose

Statement for listing the files stored in one of the printer's directories to the standard OUT channel.

Syntax

`FILES[<sexp>][,R][,A]`

- `<sexp>` optionally specifies the directory (see DEVICES).
- `R` lists directories recursively
- `A` lists all files including system files (that is, files with a name starting with a period (.) character).

Remarks

If no directory is specified, files in the printer's current directory will be listed. As default, the current directory is the printer's permanent memory ("/c"), see CHDIR statement.

By including a reference to a memory device ("/c", "tmp:", "/rom", "card1:", "lock:", or "storage:", see DEVICES statement), the files of the specified directory will be returned without having to change the current directory.

If the “A” flag is omitted, all files, except system files, will be listed. The flags A and R can be entered in any order, but R is always processed first.

The number of bytes for each file and the total number of free and used bytes in the specified directory will also be included in the list.

Examples

The presentation may look like this on the host screen:

FILES "/c", R
Files on /c

```
STDIO          2 FILE2          4
DIR1/          0
```

Files on /c/DIR1/

```
FILE1          4 DIR2/          0
STDIO          2
```

No files on /c/DIR1/DIR2

4121600 bytes free  12 bytes used
## FILES, cont.

FILES, R, A
Files on /c

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>./</td>
<td>0</td>
<td>./</td>
</tr>
<tr>
<td>DIR1/</td>
<td>0</td>
<td>FILE2</td>
</tr>
<tr>
<td>STDIO</td>
<td>2</td>
<td>.setup.saved</td>
</tr>
</tbody>
</table>

Files on /c/DIR1/

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>./</td>
<td>0</td>
<td>./</td>
</tr>
<tr>
<td>DIR2/</td>
<td>0</td>
<td>STDIO</td>
</tr>
<tr>
<td>FILE1</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Files on /c/DIR1/DIR2/

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>./</td>
<td>0</td>
<td>./</td>
</tr>
</tbody>
</table>

4121600 bytes free   251 bytes used

FILES "/c/DIR1"
Files on /c/DIR1

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE1</td>
<td>4</td>
<td>DIR2/</td>
</tr>
<tr>
<td>STDIO</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

4121600 bytes free   6 bytes used
**FLOATCALC$**

**Purpose**
Function for calculation with float numbers.

**Syntax**

```plaintext
FLOATCALC$ (<sexp1>, <sexp2>, <sexp3>[,<nexp1>])
```

- `<sexp1>` is the first operand.
- `<sexp2>` is the operator (+, -, *, or /).
- `<sexp3>` is the second operand.
- `<nexp1>` is, optionally, the precision in decimals (default 10).

**Remarks**
Operands are float numbers, that is, a string of digits with a decimal point to separate decimals from integers. Operands can also contain leading plus (+), minus (-), and space characters. Space characters are ignored, whereas the usual mathematical rules apply to plus and minus signs. All other characters (or plus, minus, and space characters in other positions than leading) generate errors.

Note the mathematical rules:
- - yields +
- + yields -
+ - yields -
+ + yields +

The following arithmetic operators are allowed:

- `+` addition ASCII 043 dec
- `-` subtraction ASCII 045 dec
- `*` multiplication ASCII 042 dec
- `/` division ASCII 047 dec

Any other type of operators or other characters will generate an error.

The precision parameter optionally specifies the number of decimals in the result of the calculation. The result will be truncated accordingly. For example, if the number of decimals is specified as 5, the result 5.76123999 will be presented as 5.76123. The result of a FLOATCALC$ function can be formatted using a FORMAT$ function.

**Examples**

**Addition:**
```
A$ = "234.9"
B$ = "1001"
PRINT FLOATCALC$ (A$, "+", B$, 5)
```

yields:

```
1235.90000
```

**Subtraction:**
```
A$ = "234.9"
C% = 2
PRINT FLOATCALC$ (A$, "-", 100.013, C%)
```

yields:

```
134.88
```
Chapter 2—Program Instructions

**FONT (FT)**

**Purpose**
Statement for selecting a scaleable TrueType or TrueDoc single-byte font or a single-byte bitmap font for the printing of the subsequent PRTXT statements.

**Syntax**
```
FONT|FT<sexp1>[[,<nexp1>][,<nexp2>][,<nexp3>]]]
```

- `<sexp1>` is the name of the font. Default: "Swiss 721 BT".
- `<nexp1>` is optionally the height in points of the font. Default: 12 points. Use MAG to enlarge with bitmap fonts.
- `<nexp2>` is the clockwise slant in degrees (0–90°). Default: 0. Does not work with bitmap fonts.
- `<nexp3>` is the width enlargement in percent relative the height (1-1000). Default: 100. Does not work with bitmap fonts.

Reset to default by: PRINTFEED execution.

**Remarks**
Intermec Fingerprint supports scaleable fonts in TrueType and TrueDoc format that comply with the Unicode standard. A large number of scaleable fonts are available on special request, so it is quite possible that your printer is fitted with a unique selection of fonts. Use a FONTS statement to list the names of all fonts installed in your own printer to the standard OUT channel.

To maintain compatibility with programs created in earlier versions of Intermec Fingerprint, you can also specify bitmap font names, for example "SW030RSN" or "MS060BMN.2". In case of standard bitmap font name, the firmware will select the corresponding scaleable font in the printer’s memory and set its parameters so its direction, appearance, and size come as close to the specified bitmap font as possible. A prerequisite is that the printer’s memory contains the standard complement of outline fonts. Non-standard bitmap fonts can also be used. They will not produce any outline fonts, but will retain their bitmap format. Any extension to the bitmap font name is of no consequence. See Chapter 6, “Fonts” in this manual.

The height of the font is given in points (same as in your PC), which means that a text will be printed in the same size regardless of the print-head density of the printer. The unit of measure is points (1 point = 1/72 inch = 0.352 mm) and specifies the height of the font including ascenders and descenders. Sizes less than 4 points will be unreadable. In case of bitmap fonts, it is recommended to use MAG to enlarge the font instead of specifying a font height (works only in multiples of 12 points).

Any font may be magnified up to 4 times separately in regard of height and width using a MAG statement. Bitmap fonts will get somewhat jagged edges when magnified, whereas outline fonts will remain smooth.
Slanting means that you can create the same effect as in ITALIC characters. The higher value, the more askew the upright parts of the characters will come. Slanting increases clockwise. Values greater than 65-70° will be unreadable. Slanting cannot be used with bitmap fonts.

Slanting value: 10

\begin{center}
\textbf{ABCDEFGH}
\end{center}

Slanting value: 20

\begin{center}
\textbf{ABCDEFGH}
\end{center}

A scaleable font can enlarged in regard of width relative the height. The value is given as percent (1-1000). This means that if the value is 100, there is no change in the appearance of the characters, whereas if the value is given as, for example, 50 or 200, the width will the half the height or double the height respectively. When using this parameter, all parameters in the syntax must be included in the statement, that is, name, height, slant, and width.

The standard complement of fonts listed in Chapter 6 can be supplemented with more fonts using three methods:

**Downloading fonts from a Font Install Card**
The card must be inserted before the printer is started. At startup the fonts are automatically downloaded, installed, and permanently stored in the printer’s memory. The fonts can be used without the card being present.

**Using fonts from a Font Card**
The card must be inserted before the printer is started. At startup the fonts are automatically installed, but not copied to the printer’s memory. Thus, the card must always be present before such a font can be used.

**Downloading font files**
Font files can be downloaded and installed by means of either of the two statements IMAGE LOAD and TRANSFER KERMIT. There is no need to restart the printer before using the font in question.

It is possible to create aliases for one or several font to get shorter or more adequate names. Refer to Chapter 6 for further explanation.

**Examples**

Printing one line of 12p text with default direction and alignment:

```
10  FONT "Swiss 721 BT"
20  PRTXT "HELLO"
30  PRINTFEED
RUN
```

Printing the same text but with 24p size, 20° slant, and 75% width:

```
10  FONT "Swiss 721 BT",24,20,75
20  PRTXT "HELLO"
30  PRINTFEED
RUN
```
CHAPTER 2—Program Instructions

FONTD

Purpose
Statement for selecting a scaleable TrueType or TrueDoc double-byte font for the printing of the subsequent PRTXT statements.

Syntax

\[ \text{FONTD} <\text{sexp}_1>[,<\text{nexp}_1>[,<\text{nexp}_2>[,<\text{nexp}_3>]]] \]

- \(<\text{sexp}_1>\) is the name of the font. Default: none.
- \(<\text{nexp}_1>\) is optionally the height in points of the font. Default: 12 points.
- \(<\text{nexp}_2>\) is the clockwise slant in degrees (0-90°). Default: 0.
- \(<\text{nexp}_3>\) is the width enlargement in percent relative the height (1-1000). Default: 100.

Reset to default by: PRINTFEED execution or CLL.

Remarks
This statement is identical to the FONT statement, but is used for fonts specified by a double byte (16 bits) instead of a single byte (7 or 8 bits). To use a double-byte font, a double-byte character set must be selected using a NASCD statement. Usually, if the first byte has an ASCII value between 161 dec. (A1 hex) and 254 dec (FE hex), the character will be treated as a double-byte character and the firmware waits for next byte to make the 16 bit address complete. The character will be printed using the font specified by FONTD and according to the character set specified by NASCD and the Unicode standard.

On the other hand, if the first byte has an ASCII value below 161 dec. (A1 hex), the character is treated as a single byte character and next byte received will be regarded as the start of a new character. This implies that the character set specified by NASC and the font specified by FONT will be used. However, the selected Unicode double-byte character set may specify some other ASCII value as the breaking point between single and double byte character sets.

Note that 8 bit communication must be selected.

Only writing from left to right in the selected print direction is supported.

Example
The following text contains both single- and double-byte fonts. The double-byte font and its character set are stored in a Font Install Card:

```
10   NASC 46
20   FONT "Swiss 721 BT", 24, 10
30   FONTD "DLC Ming Medium"
40   NASCD "rom:BIG5.NCD"
50   PRTXT CHR$(65);CHR$(161);CHR$(162)
60   PRINTFEED
RUN
```

This program yields a printed text line that starts with the Latin character A (ASCII 65 dec.) followed by the character in the DLC Ming Medium font that corresponds to the address 161+162 dec. in the character set “BIG5.NCD.”
Chapter 2—Program Instructions

**FONTNAME$**

**Purpose**
Function returning the names of the fonts stored in the printer’s memory.

**Syntax**

```
FONTNAME$(<nexp>)
```

- `<nexp>` the result of the expression should be either false or true, where...
  - False (0) indicates first font.
  - True (≠0) indicates next font.

**Remarks**

- FONTNAME$(0)$ produces the first name in the memory.
- FONTNAME$(≠0)$ produces next name. Can be repeated as long as there are any fontnames left.

**Example**

Use a program like this to list all fontnames:

```
10  A$ = FONTNAME$ (0)
20  IF A$ = "" THEN END
30  PRINT A$
40  A$ = FONTNAME$ (-1)
50  GOTO 20
```

RUN

yields for example:

- UPC11.1
- UPC11.2
- UPC21.1
- UPC21.2
- UPC31.1
- UPC31.2
- UPC51.1
- UPC51.2
- Century Schoolbook BT
- DingDings SWA
- Dutch 801 Bold BT
- Dutch 801 Roman BT
- Futura Light BT
- Letter Gothic 12 Pitch BT
- MS030RMN
- MS030RMN.1
- MS030RMN.2
- MS050RMN
- MS050RMN.1
- MS050RMN.2
- MS060BMN
- MS060BMN.1
- MS060BMN.2
- Monospace 821 BT
- Monospace 821 Bold BT
- OB035RM1
- etc, etc.
Chapter 2—Program Instructions

FONTS

Purpose

Statement returning the names of all fonts stored in the printer’s memory to the standard OUT channel.

Syntax

FONTS

Example

A list of the fonts stored in the printer may look like this:

FONTS

 yields for example:

Century Schoolbook BT
Dutch 801 Bold BT
Futura Light BT
MS030RMN
MS030RMN.2
MS050RMN.1
MS050RMN.2
MS060BMN
MS060BMN.1
MS060BMN.2
Monospace 821 Bold BT
OB035RM1
OB035RM1.1
OB035RM1.2
OCR-A BT
Prestige 12 Pitch Bold BT
SW020BSN
SW020BSN.1
SW030RSN
SW030RSN.1
SW050RSN
SW050RSN.1
SW050RSN.2
SW060BSN
SW060BSN.1
SW060BSN.2
SW080BSN
SW080BSN.1
SW120BSN
SW120BSN.1
SW120BSN.2
Swiss 721 Bold BT
Swiss 721 Bold Condensed BT
Zurich Extra Condensed BT

3569264 bytes free 1717240 bytes used
Ok
**FOR...TO...NEXT**

**Purpose**
Statement for creating a loop in the program execution, where a counter is incremented or decremented until a specified value is reached.

**Syntax**
```
FOR<nvar>=<nexp1>TO<nexp2>[STEP<nexp3>]NEXT[<nvar>]
```
- `<nvar>` is the variable to be used as a counter.
- `<nexp1>` is the initial value of the counter.
- `<nexp2>` is the final value of the counter.
- `<nexp3>` is the value of the increment (decrement).

**Remarks**
This statement is always used in connection with a NEXT statement.

The counter `<nvar>`) is given an initial value by the numeric expression `<nexp1>`) If no increment value is given (STEP `<nexp3>), the value 1 is assumed. A negative increment value will produce a decremental loop. Each time the statement NEXT is encountered, the loop will be executed again until the final value, specified by `<nexp2>`, is reached. Then the execution will proceed from the first line after the NEXT statement.

If the optional variable is omitted in the NEXT statement, the program execution will loop back to the most recently encountered FOR statement. If the NEXT statement does include a variable, the execution will loop back to the FOR statement specified by the same variable.

FOR...NEXT loops can be nested, which means that a loop can contain another loop, etc. However, each loop must have a unique counter designation and the inside loop must be concluded by a NEXT statement before the outside loop can be executed.

**Examples**
The counter A% is incremented from 10 to 50 in steps of 20:
```
10  FOR A%=10 TO 50 STEP 20
20  PRINT A%
30  NEXT
RUN
```
yields:
```
10
30
50
```

The counter B% is decremented from 50 to 10 in steps of 20:
```
10  FOR A%=50 TO 10 STEP -20
20  PRINT A%
30  NEXT
RUN
```
yields:
```
50
30
10
```
FORMAT

Purpose
Statement for formatting the printer's permanent memory, or formatting a CompactFlash memory card.

Syntax
FORMAT<sexp>[<nexp1>,<nexp2>][,A]

<sexp> specifies the device to be formatted either as "/c" or "card1:"
<nexp1> Specifies the number of entries in the root directory (only applicable when <sexp> = "card1:" and "A" flag is set).
   Default: 208 entries.
<nexp2> Specifies the number of bytes per sector (only applicable when <sexp> = "card1:" and "A" flag is set).
   Default: 512 bytes per sector.

Remarks
FORMAT "/c"
Formats the printer's permanent memory partially or completely. System files are distinguished by a leading period character, for example .setup. saved. This makes it possible to format the permanent memory without removing the system files.

If no “A” flag is included in the statement, all files excluding those starting with a period character (.) will be removed (“soft” formatting).

If an “A” flag is included in the statement, all files including those starting with a period character (.) will be removed (“hard” formatting).

Be careful. There is no way to undo a FORMAT operation.

FORMAT "card1:"
Formats a CompactFlash card, which is inserted in the printer's optional memory card adapter, to MS-DOS format. Optionally, the number of entries in the root directory (that is number of files on the card) and the number of bytes per sector can be specified, provided an “A” flag is included in the statement (“hard” formatting).

When a FORMAT statement is executed, any existing data or previous formatting in the card will be erased. After formatting, such a memory card can be OPENed for INPUT/OUTPUT/APPEND or RANDOM access and can also be used in a PC for storing MS-DOS files. The DOS-formatted memory card is referred to as device "card1:".
FORMAT, cont.

Examples

Issuing the statement FILES before and after a FORMAT "/c" statement shows how the memory is affected. Note that system files starting with a period character are not removed, since the FORMAT statement does not contain any "A" flag:

FILES "/c",A

yields for example:

Files on /c

./ 0  ../ 0
APPLICATION 1  boot/ 0
ADMIN/ 0 .setup.saved 222
STDIO 4

2222080 bytes free 227 bytes used

Ok

FORMAT "/c"
Ok

FILES "/c",A

yields for example:

Files on /c

./ 0  ../ 0
boot/ 0
ADMIN/ 0
/setup.saved 222

224128 bytes free 222 bytes used

In the following statement, a CompactFlash memory card is formatted to MS-DOS format in the immediate mode. The number of entries is increased from 208 (default) to 500 and the size of the sectors in decreased from 512 bps (default) to 256 in order to make the card better suited for more but smaller files. The "A" flag specifies "hard" formatting.

FORMAT "card1:\",500,256,A
**FORMAT DATE$**

**Purpose**

Statement for specifying the format of the string returned by DATE$("F") and DATEADD$(...,"F") instructions.

**Syntax**

```
FORMAT DATE$<sexp>
```

- `<sexp>` is a string representing the order between year, month and date plus possible separating characters.
- "Y" represents Year (one digit per Y).
- "M" represents Month (one digit per M).
- "D" represents Day (one digit per D).

Default: YYMMDD
Reset to default by: Empty string ("")

**Remarks**

DATE$ and DATEADD$ will only return formatted dates if these functions include the flag "F".

In the FORMAT DATE$ statement, each Y, M or D character generates one digit from the number of the year, month or day respectively, starting from the end. If the number of Y’s exceeds 4, or the number of M’s or D’s exceeds 2, the exceeding characters generate leading space characters.

Examples (the year is 2003):
- `Y` generates 3
- `YY` generates 03
- `YYY` generates 003
- `YYYY` generates 2003
- `YYYYY` generates ↔2003 (↔ represents a space)

Separating characters are returned as entered in the string. Any character except Y, M, or D are regarded as separators.

The date format is saved in the temporary memory and has to be transmitted to the printer after each power-up.

**Examples**

Changing the date format according to British standard:
```
FORMAT DATE$ "DD/MM/YY"
```

Changing date format back to default (YYMMD): 
```
FORMAT DATE$ ""
```

Changing the date format to Swedish standard:
```
FORMAT DATE$ "YY-MM-DD"
```
FORMAT INPUT

Purpose
Statement for specifying separators for the LAYOUT RUN statement used in the Intermec Direct Protocol.

Syntax
FORMAT INPUT<sexp₁>,[<sexp₂>,[<sexp₃>,[<sexp₄>]]>

<sexp₁> is the start-of-text separator, default STX (ASCII 02 dec.).
<sexp₂> is the end-of-text separator, default EOT (ASCII 04 dec.).
<sexp₃> is the field separator, default CR (ASCII 13 dec.).
<sexp₄> is a string of characters to be filtered out.

Remarks
The LAYOUT RUN statement is used in the Intermec Direct Protocol to transmit variable data to a predefined layout. By default, the string of input data to the various layout fields starts with a STX character and ends with a EOT character. The various fields are separated by CR (carriage return) characters.

To provide full compatibility with various protocols and computer systems, these separators can be changed at will by means of the FORMAT INPUT statement. Each separator can have a maximum length of 10 characters.

As an option, it is possible to specify a string of max. 10 characters to be filtered out. By default, the string is empty and will be reset to default if a new FORMAT INPUT with less than four arguments is issued.

There is a timeout if ETX is not found within 60 seconds after STX has been received.

Always execute the FORMAT INPUT statement in the Immediate Mode. If you are using the Intermec Direct Protocol, exit it using an INPUT OFF statement before changing the separators using a FORMAT INPUT statement. Then you can enter the Intermec Direct Protocol again using an INPUT ON statement.

An error will occur if you, for some reason, issue a FORMAT INPUT statement where one, two or three separators are identical to those already in effect without leaving the Intermec Direct Protocol.

If a certain separating character cannot be produced by the keyboard of the host, use a CHR$ function to specify the character by its ASCII value.

The separators are stored in the temporary memory and must to be transmitted to the printer after each power-up.

Example
Changing the start-of-text separator to #, the end-of-text separator to LF (linefeed), and the field separator to @ after having temporarily switched to the Immediate Mode.

INPUT OFF
FORMAT INPUT "#",CHR(10),"@" INPUT ON
Chapter 2—Program Instructions

 FORMAT TIME$

Purpose
Statement for specifying the format of the string returned by TIMES($F”) and TIMEADD$($F”) instructions.

Syntax

<table>
<thead>
<tr>
<th>FORMAT TIME$&lt;sexp&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;sexp&gt; is a string representing the order between hours, minutes and seconds plus possible separating characters.</td>
</tr>
<tr>
<td>“H” represents hours in a 24 hour cycle (one digit per H).</td>
</tr>
<tr>
<td>“h” represents hours in a 12 hour cycle (one digit per h).</td>
</tr>
<tr>
<td>“M” represents minutes (one digit per M).</td>
</tr>
<tr>
<td>“S” represents seconds (one digit per S).</td>
</tr>
<tr>
<td>“P” represents AM/PM in connection with a 12 hour cycle.</td>
</tr>
<tr>
<td>“p” represents am/pm in connection with a 12 hour cycle.</td>
</tr>
<tr>
<td>All other character produce separator characters.</td>
</tr>
</tbody>
</table>

Default: HHMMSS
Reset to default by: Empty string

Remarks
Each H, h, M, and S character generates one digit. If the number of each character exceeds 2, leading space characters are inserted. Each uppercase or lowercase P character generates one character of AM/PM or am/pm respectively, when a 12-hour cycle is selected.

Hour, minute and second fields are right-justified, whereas am/pm and AM/PM fields are left-justified.

Example (the hour is 8 o'clock in the morning):

h generates 8  P generates A
hh generates 08  PP generates AM
hhhh generates 08  pp generates am

To get 12-hour cycle, all hour format characters must be lowercase “h”.

Separating characters are returned as entered in the string. Any character but H, h, M, S, P, or p are regarded as separators.

The time format is saved in the temporary memory and has to be transmitted to the printer after each power-up.

Examples
Changing the time format according to Swedish standard:
FORMAT TIME$ "HH.MM.SS"

Changing the date format to British standard:
FORMAT TIME$ "hh:MM pp"
**FORMAT$**

**Purpose**
Function for formatting a number represented by a string.

**Syntax**
```
FORMAT$(<sexp>,"<nexp>b"|"d")
```
- `<sexp>` is a string of integers or ASCII characters.
- `<nexp>` is the number of bytes to output in integer to ASCII conversion.

```
FORMAT$(<sexp>,<sexp>)
```
- `<sexp>` is the string of numerals, optionally with decimals, which is to be formatted.
- `<sexp2>` specifies the format of the string.

**Remarks**
FORMAT$ can be used for conversion of strings from ASCII to integer numbers and vice versa when used as in the first syntax above.

- `<nexp>b` = Specifies conversion from integer to ASCII format.
  This converts `<sexp>` from a number in integer format to the corresponding characters in ASCII format. `<nexp>` specifies the number of bytes to output. For example `FORMAT"(1380337988","4b")` yields RFID.

- `d` = Specifies conversion from ASCII format to integer.
  The string `<sexp>` is translated from an ASCII string to the same number in integer format. For example, `FORMAT$"(A","d")` yields 65.

FORMAT$ is also be used to convert a number to a specific display format. The original string `<sexp>` is a string of digits, optionally with a decimal point to separate decimals from integers. It can also contain leading plus (+), minus (-), and space characters. Space characters are ignored, whereas the usual mathematical rules apply to plus and minus signs. All other characters (or plus, minus, and space characters in other positions than leading) generate errors. The format is specified by a string `<sexp2>`. The string can contain any characters, but some have special meanings. Note the explanation of the following characters.

- `0` = Digit place holder, display a digit or zero.
  If the input number has fewer digits than there are zeros (on either side of the decimal separator) in the format string, leading or trailing zeros are displayed. If the number has more digits to the left of the decimal separator than there are zeros to the left of the separator in the format string the digits will be displayed. If the number has more digits to the right of the separator than there are zeros to the right of the decimal separator in the format string, the decimal will be truncated to as many decimal places as there are zeros.

- `#` = Digit placeholder, display a digit or nothing.
  If there is a digit in the expression being formatted in the position where the # appears in the format string, display the digit or otherwise display nothing in that position. If the number has more digits to the left side of
the decimal separator than there are # to the left side of the separator in the format string the digits will be displayed.

. = Decimal separator, to separate the integer and the decimal digits.
,
  = Decimal separator, to separate the integer and the decimal digits. \
  = Display the next character in the format string.

The backslash itself is not displayed. To display a \, use two backslashes.
The only character, which will be displayed in the formatted string without a backslash is space.

**space** = Space
A space will be displayed as literal character wherever it is in the expression format.

• An empty format string is equivalent to "0 .######".

• 0 and # cannot be mixed in every way. Before the decimal separator, use # first and then 0. After the decimal separator, use 0 first and then #. For example: ####00.000### is OK and #00###0.##0#00 is not.

• A point or a comma separates integers and decimals. The decimal separator used in the format is the one that will be the returned separator type. Independent of the separator type in the number the format type will control the return type. Default type is a point.

• A format can consist of separators as space between thousands either a unit as $. For example: "$ ### ### 000.00".

• The attached number string will be truncated to the quantity of decimal in the format.

• Characters will not be displayed on the left side of the decimal separator if there is a # on the left side of the characters and the string to be formatted do not have a digit in the same position as the #. On the right side of the decimal separator, characters will not be displayed if there is a # on the right side of the characters and the string to be formatted do not have a digit in the same position as the #. For example:

<table>
<thead>
<tr>
<th>Format string:</th>
<th>$#\t\e\x\t0.00t\e\x\t\f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>String to be formatted:</td>
<td>1.1 $</td>
</tr>
<tr>
<td>Returned strings:</td>
<td>$1.1$</td>
</tr>
<tr>
<td>Input number:</td>
<td>&quot;$5&quot;</td>
</tr>
<tr>
<td>Input format: &quot;#&quot; = &gt;</td>
<td>5</td>
</tr>
<tr>
<td>&quot;0.00&quot; = &gt;</td>
<td>5.00</td>
</tr>
<tr>
<td>&quot;$0.0&quot; = &gt;</td>
<td>5.0</td>
</tr>
<tr>
<td>&quot;0.#&quot; = &gt;</td>
<td>5.0</td>
</tr>
<tr>
<td>&quot;##,000.0&quot; = &gt;</td>
<td>005.0</td>
</tr>
<tr>
<td>&quot;# 0.0.0&quot; = &gt;</td>
<td>0 5.0</td>
</tr>
</tbody>
</table>
FORMAT$, cont.

Examples

**Addition**

B$="234.9"
C$="1001"
D$="# ##0.##"
A$=FLOATCALC$(B$,"+",C$,15)
PRINT A$

"1235.900000000000000"
yields:

PRINT FORMAT$(A$,D$)

"1 235.9"
yields:

**Subtraction**

A$=FLOATCALC$("234.90","-","100.013",2)
PRINT A$

"134.88"
yields:

PRINT FORMAT$(A$,"\$ 0,000#")

"$ 134,880"
Note: If a higher precision is used in FLOATCALC$, A$ will yield "$134,887".

**Multiplication**

B$="3"
A$=FLOATCALC$("100", "*", B$, 1)
PRINT A$

"300.0"
yields:

C$="0 0 0,00###"
PRINT FORMAT$(A$,C$)

"3 0 0,00"
yields:

**Division**

B$="1.0"
A$=FLOATCALC$(B$,"/","3.0")
PRINT A$

"0.3333333333"
yields:

PRINT FORMAT$(A$,"\$ 000.00###")

"$ 000.33333"
FORMFEED (FF)

Purpose
Statement for feeding out or pulling back a certain length of media.

Syntax
FORMFEED|FF[<nexp>]

Remarks
If no value is entered after the FORMFEED statement, the printer will feed out one single label, ticket, tag, or a portion of continuous stock according to the printer’s setup. See start- and stopadjustments and media type in the User’s Guide for the printer model in question.

If a value is entered after the FORMFEED statement, the media will be fed out or pulled back the corresponding number of dots:

- A positive number of dots makes the printer feed out the specified length of media.
- A negative number of dots makes the printer pull back the specified length of media. Be careful not to enter a value larger than the length of the label to avoid causing a media jam.

It is important whether a FORMFEED statement is issued before or after a PRINTFEED statement:

- FORMFEED statement issued before PRINTFEED affects the position of the origin on the first copy to be printed.
- FORMFEED statement issued after PRINTFEED does not affect the position of the origin on the first copy, but next copy will be affected.

Do not use FORMFEED as a replacement for start- and stopadjustments in the Setup Mode or in connection with batch printing.

Examples
Printing a line of text and feeding out an extra 60 dots of media after printing:
10 FONT "Swiss 721 BT"
20 PRPOS 30,200
30 PRTXT "HELLO"
40 PRINTFEED
50 FORMFEED 60
RUN

Pulling back the media 20 dots before printing:
10 FORMFEED -20
20 FONT "Swiss 721 BT"
30 PRPOS 30,200
40 PRTXT "HELLO"
50 PRINTFEED
RUN
In this case, the positioning of the text line will be performed after the media has been pulled back.
FRE

Purpose

Function returning the number of free bytes in specified part of the printer's memory.

Syntax

FRE(<sexp>|<nexp>>)

<sexp> is the designation of the part of the printer's memory from which the number of free bytes should be returned, for example "/c", "tmp:", "card1:..

<nexp> is a dummy argument. Returns the number of free bytes in the printer's temporary memory ("tmp:").

Remarks

The firmware looks for a colon (:) sign in the argument for the FRE function. If the argument is valid name of a memory device, the number of free bytes in that device is returned.

If the argument specifies device "card1:", but no card is inserted, Error 1039, “Not mounted” will occur.

If the name of a device, that is not a part of the printer's memory (for example "uart1:” or "console:"), is entered as an argument, the FRE function will return 0.

Refer to DEVICES for more information on memory and non-memory devices.

If the argument contains a colon, but is not a valid name of any device (for example "QWERTY:"), Error 1013, “Device not found” will occur.

Any argument, that does not include a colon sign (for example “7” or "QWERTY"), will return the amount of free bytes in the printer's temporary memory ("tmp:").

Example

PRINT FRE("tmp:"

yields for example:

2382384

PRINT FRE("uart1:"

yields:

0

PRINT FRE(1)

yields for example:

2382384
FUNCTEST

Purpose
Statement for performing various hardware tests.

Syntax
FUNCTEST<sexp>,<svar>

<sexp> is the type of test to be performed:
"CARD"
"HEAD" (only "HEAD" yields a meaningful response)
"KERNEL"
"ROMn"

<svar> is the variable in which the result will be placed.

Remarks
The test has a number of possible responses:

<sexp> = "CARD"
NOT IMPLEMENTED Not supported.

<sexp> = "HEAD"
HEAD OK, SIZE:n DOTS The test was successful.
n is the number of dots on the print-head.
HEAD LIFTED Printhead is lifted and must be lowered before test can be performed.
FAULTY PRINTHEAD One or more dots on the printhead are not working.
Note that the voltage for the printhead is not checked. Use the HEAD function for additional printhead tests.

<sexp> = "KERNEL"
NOT IMPLEMENTED Not supported.

<sexp> = "ROMn"
NOT APPLICABLE Not supported.

Any other input to <sexp> yields an empty string.

Example
This example shows how a test program using the FUNCTEST statement may be composed:
10  FUNCTION "HEAD", A$
20  PRINT "HEADTEST: ", A$
RUN

yields for example:

HEADTEST: HEAD OK, SIZE: 832 DOTS

Ok
**FUNCTEST$**

**Purpose**
Function returning the result of various hardware tests.

**Syntax**

```
FUNCTEST$(<sexp>)
```

- `<sexp>` is the type of test to be performed:
  - "CARD"
  - "HEAD" (only "HEAD" yields a meaningful response)
  - "KERNEL"
  - "ROMn"

**Remarks**
The test has a number of possible responses:

- `<sexp> = "CARD"`
  - NOT IMPLEMENTED
  - Not supported.

- `<sexp> = "HEAD"`
  - HEAD OK, SIZE:n DOTS
    - The test was successful.
    - n is the number of dots on the printhead.
  - HEAD LIFTED
    - Printhead is lifted and must be lowered before test can be performed.
  - FAULTY PRINTHED
    - One or more dots on the printhead are not working.
    - Note that the voltage for the printhead is not checked. Use the HEAD function for additional printhead tests.

- `<sexp> = "KERNEL"`
  - NOT IMPLEMENTED
  - Not supported.

- `<sexp> = "ROMn"`
  - NOT APPLICABLE
  - Not supported

Any other input to `<sexp>` yields an empty string.

**Example**
This example shows how a test program using the FUNCTEST$ function may be composed (compare with the example for FUNCTEST statement):

```
PRINT "HEADTEST: ", FUNCTEST$ ("HEAD")
```

yields for example:

```
HEADTEST: HEAD OK, SIZE:1280 DOTS
```

Ok


**Chapter 2—Program Instructions**

## GET

**Purpose**
Statement for reading a record from a random file to a random buffer.

**Syntax**
```
GET[#]<nexp1>,<nexp2>
```
- `#` indicates that whatever follows is a number. Optional.
- `<nexp1>` is the number assigned to the file when it was OPENed.
- `<nexp2>` is the number of the record. Must be ≠ 0.

**Remarks**
The GET statement is used to read a certain record in a certain random file to a buffer, where the record will be assigned to variables according to the FIELD statement given for the buffer. After the GET statement has been executed, you can use references to the variables defined by the FIELD statement to read the characters in the random buffer.

Numeric expressions, which have been converted to string expressions by STR$ functions before being put into the buffer, can be converted back to numeric expressions using VAL functions.

**Example**
```
10 OPEN "PHONELIST" AS #8 LEN=26
20 FIELD#8,8 AS F1$, 8 AS F2$, 10 AS F3$
30 SNAME$="SMITH"
40 CNAME$="JOHN"
50 PHONES="12345630"
60 LSET F1$=SNAME$
70 LSET F2$=CNAME$
80 RSET F3$=PHONES$
90 PUT #8,1
100 CLOSE#8
RUN

SAVE "PROGRAM 1.PRG"

NEW
10 OPEN "PHONELIST" AS #8 LEN=26
20 FIELD#8,8 AS F1$, 8 AS F2$, 10 AS F3$
30 GET #8,1
40 PRINT F1$,F2$,F3$
RUN
```

yields:
```
SMITH — — — JOHN — — — — — — 12345630
```
GETASSOC$

Purpose
Function for getting a value from a string association.

Syntax
GETASSOC$ (<sexp_1>, <sexp_2>)

<sexp_1> is the name of the association (case-sensitive).
<sexp_2> is the name of a tuple in the association.

Remarks
An association is an array of tuples, where each tuple consists of a name and a value.

Example
This example shows how a string, including three stringnames associated with three start values, will be defined and one of them (time) will be changed:

10  QUERYSTRING$=
    "time=UNKNOWN&label=321&desc=DEF"
20  MAKEASSOC"QARRAY", QUERYSTRING$, "HTTP"
30  QTIME$=GETASSOC$ ("QARRAY", "time")
40  QLABELS%=VAL(GETASSOC$ ("QARRAY", "label"))
50  QDESC$=GETASSOC$ ("QARRAY", "desc")
60  PRINT "time="; QTIME$; "LABEL="; QLABELS%;
    "DESCRIPTION="; QDESC$
70  SETASSOC"QARRAY", "time", time$
80  PRINT "time="; GETASSOC$ ("QARRAY", "time")
RUN

yields:

time=UNKNOWN  LABEL=321  DESCRIPTION=DEF

RUN

time=153355
GETASSOCNAME$

Purpose
Function for traversing the tuples of a string association.

Syntax

\[
\text{GETASSOCNAME}(\text{<sexp>}, \text{<nexp>})
\]

- \text{<sexp>} is the association to be traversed (case-sensitive).
- \text{<nexp>} specifies the tuple in the association.
  - \text{<nvar> = 0} specifies first tuple.
  - \text{<nvar> ≠ 0} specifies next tuple.

Remarks
An association is an array of tuples, where each tuple consists of a name and a value. To get the first position in the string association, \text{<nvar>} should be zero. Consecutive calls to \text{GETASSOCNAME} with \text{<nvar>} non-zero will traverse all variables in an undefined order. When a blank string ("") is returned, the last variable has been traversed.

Example
This example shows how “QARRAY” is traversed (run example from GETASSOC first):

```
10 LVAL$=GETASSOCNAME("QARRAY", 0)
20 WHILE LVAL$<>"
30 RVAL$=GETASSOC("QARRAY",LVAL$)
40 PRINT LVAL$;"=";RVAL$  
50 LVAL$=GETASSOCNAME("QARRAY",1)
60 WEND
RUN
```

yields:

```
label=321
desc=DEF
time=153355
```
GETPFSVAR

Purpose

Function for recovering saved variables.

Syntax

GETPFSVAR(<sexp>[,D])

<sexp> is the name of the variable (uppercase characters only).
D optionally specifies that the variable is to be deleted after recovery.

Remarks

This function is used to recover variables registered to be saved at power failure by means of a SETPFSVAR statement and returns either -1 on success or 0 at failure.

If a D flag is included, the variable is deleted after it has been recovered. This can be used to make sure that the variable is up to date and that no old obsolete value is recovered.

The variable name is limited to a length of twenty characters.

Related instructions are SETPFSVAR, DELETPFSVAR, and LISTPFSVAR.

Example

10 IF NOT GETPFSVAR("QS$") THEN QS$ = "<this is the default value, set a new one>
20 IF NOT GETPFSVAR("QCPS%") THEN PRINT "No copies available":END
30 QSTATUS%=GETPFSVAR("AWE$",D):IF QSTATUS% THEN PRINT "Recovered successfully!"
40 SETPFSVAR "QCPS%"
50 'Build label
60 ..... 99 PRINTFEED; QCPS%=QCPS%
100 .....
GOSUB

Purpose

Statement for branching to a subroutine.

Syntax

GOSUB<ncon>|<line label>

<ncon>|<line label> is the number or label of the first line in the desired subroutine.

Remarks

After branching, the subroutine will be executed line by line until a RETURN statement is encountered.

The same subroutine can be branched to many times from different lines in the main program. GOSUB always remembers where the branching took place, which makes it possible to return to the correct line in the main program after the subroutine has been executed.

Subroutines may be nested, which means that a subroutine may contain a GOSUB statement for branching to a secondary subroutine and so on.

Subroutines are normally placed on program lines with higher numbers than the main program. The main program should be appended by an END statement to avoid unintentional execution of subroutines.

Example

This example makes use of line numbers:

```
10    PRINT "This is the main program"
20    GOSUB 1000
30    PRINT "You’re back in the main program"
40    END
1000   PRINT "This is subroutine 1"
1010   GOSUB 2000
1020   PRINT "You’re back from subroutine 2 to 1"
1030   RETURN
2000   PRINT "This is subroutine 2"
2010   GOSUB 3000
2020   PRINT "You’re back from subroutine 3 to 2"
2030   RETURN
3000   PRINT "This is subroutine 3"
3010   PRINT "You’re leaving subroutine 3"
3020   RETURN
RUN
```

yields:

This is the main program
This is subroutine 1
This is subroutine 2
This is subroutine 3
You’re leaving subroutine 3
You’re back from subroutine 3 to 2
You’re back from subroutine 2 to 1
You’re back in the main program

Ok
GOSUB, cont.

In this examples, line numbers have been omitted and line labels are used to make the program branch to subroutines:

```
IMMEDIATE OFF
PRINT "This is the main program"
GOSUB SUB1
PRINT "You’re back in the main program"
END

SUB1: PRINT "This is subroutine 1"
GOSUB SUB2
PRINT "You’re back from subroutine 2 to 1"
RETURN

SUB2: PRINT "This is subroutine 2"
GOSUB SUB3
PRINT "You’re back from subroutine 3 to 2"
RETURN

SUB3: PRINT "This is subroutine 3"
PRINT "You’re leaving subroutine 3"
RETURN
```
Chapter 2—Program Instructions

GOTO

Purpose
Statement for branching unconditionally to a specified line.

Syntax
\[ \text{GOTO}<\text{ncon}|<\text{line}\label> \]

\(<\text{ncon}|<\text{line}\label>\) is the number or label of the line to be branched to.

Remarks
If the specified line contains an executable statement, both that statement and all that follows will be executed. If the specified line does not exist, an error condition will occur.

The GOTO statement can also be used in the immediate mode to resume execution of a program, which has been terminated using a STOP statement, at a specified program line.

Example
In this example the first bar of the tune "Colonel Boogie" will be played only if the title is entered correctly. Otherwise the message "Try again" will be displayed until you manage to type the right name.

10 \(A\$="\text{COLONEL BOOGIE}"\)
20 \(B\$="\text{TRY AGAIN}"\)
30 INPUT "TITLE"; C$
40 IF C$=A$ GOTO 100 ELSE PRINT B$
50 GOTO 30
60 END
100 SOUND 392,15
110 SOUND 330,20
120 SOUND 330,15
130 SOUND 349,15
140 SOUND 392,15
150 SOUND 659,25
160 SOUND 659,20
170 SOUND 523,25
180 GOTO 60
RUN

yields:

TITLE?

Note the way GOTO is used in line 50 to create a loop, which makes the printer await the condition specified in line 40 before the execution is resumed. Instead of line numbers, line labels can be used following the same principles as illustrated in the second example for GOSUB statement.
**HEAD**

**Purpose**
Function returning the result of a thermal printhead check.

**Syntax**

```
HEAD(<nexpi,>)
```

- `<nexpi,> ≥ 0` specifies the number of a dot for which the resistance in ohms will be returned.
- `<nexpi,> = -1` printhead check: Returns -1 (true) if OK
  Returns 0 (false) if error
- `<nexpi,> = -7` returns mean printhead resistance in ohms.

```
<nexpi,> = HEAD(<sexp>)
```

- `<nexpi,>` returns the number (quantity) of faulty dots.
- `<sexp>` returns the dot number and resistance for each faulty dot.

**Remarks**

This function allows you to examine the printhead in regard of dot resistance. There is no guarantee that all defect “dots” will be detected by the HEAD function, since only the resistance is checked. For example, dirty or cracked dots can only be detected visually.

**Note:** The EasyCoder PD41 does not have resistance measuring. Running the HEAD command on this platform will not return actual values.

The detection of a possibly faulty dot or printhead by means of the dot sensing facility does not automatically imply that the printhead is defect and that replacement will be covered by the warranty. Intermec reserves the right of physical examination of the printhead before any replacement free of charge can be discussed.

- `<nexpi,> ≥ 0`
  A positive value specifies a single dot on the printhead and returns its resistance value as a number of ohms. A dot resistance value that deviates considerably from the mean resistance value of the printhead (see below) indicates that the dot may be faulty. The dot numbering starts at 0 (zero), that is, in a 832 dots printhead, the dots are numbered 0-831. On the EasyCoder PD41 the nominal resistance of the dot will be returned.

- `<nexpi,> = -1`
  A check of the complete printhead is performed. PD41 always returns -1.

  HEAD(-1)= -1  The printhead is within the allowed limits (no dot is more than ±15% from the mean resistance value). This does not guarantee the printout quality.

  HEAD(-1)= 0  A possible error has been detected.

- `<nexpi,> = -7`
  The mean resistance value in ohms of all dots of the printhead is returned. The PD41 will return the nominal resistance value of the dots.
**HEAD, cont.**

The second version of the HEAD function measures the dot resistance for every dot in the printhead and faulty dots are reported to the system, so you do not need to use a SET FAULTY DOT statement to report bad dots one at the time. The PD41 will return 0, and <sexp> will be empty.

**Examples**

Read the resistance value of dot No. 5:
```
PRINT HEAD(5)
```

Perform a printhead check:
```
PRINT HEAD(-1)
```

Read the printhead’s mean resistance value:
```
PRINT HEAD(-7)
```

Check printhead for faulty dots and their respective resistance values:
```
A%=HEAD(B$)
```

yields for example:
```
Ok
PRINT A%
5

Ok
PRINT B$
25, 2944
42, 2944
106, 2944
107, 2944
140, 2944

Ok
```
**IF...THEN...(ELSE)**

**Purpose**
Statement for conditional execution controlled by the result of a numeric expression.

**Syntax**

```
IF<nexp>[,<]THEN<stmt1>[ELSE<stmt2>]
```

- `<nexp>` is a numeric expression, which is either true or false.
- `<stmt1>` is the statement or list of statements telling the program what to do, should the IF-condition be true.
- `<stmt2>` is an optional statement or list of statements specifying what will happen, should the IF-condition be false.

**Remarks**

THEN and ELSE statements may be nested.

Multiple THEN and ELSE statements can alternatively be entered on separate lines. If so, the instruction should be appended by ENDIF. See second example below.

**Examples**

These two examples illustrates the different syntaxes:

```plaintext
10 A%=100:B%=20
20 C$="A LARGER THAN B"
30 D$="A NOT LARGER THAN B"
40 IF A%>B% THEN PRINT C$ ELSE PRINT D$
RUN
```

yields:

```
A LARGER THAN B
```

```plaintext
10 A%=VAL(TIME$)
20 IF A%>120000 THEN
30 PRINT "TIME IS ";TIME$; ". ";
40 PRINT "GO TO LUNCH!"
50 ELSE
60 PRINT "CARRY ON - ";
70 PRINT "THERE’S MORE WORK TO DO!"
80 ENDIF
RUN
```

yields for example:

```
TIME IS 121500. GO TO LUNCH!
```
IF...THEN...(ELSE), cont.

IF ... THEN are often used in connection with GOTO. In this example, line numbering is used. Also see the example for the GOTO statement.

```
10  A%=100
20  B%=50
30  IF A%=B% THEN GOTO 50 ELSE PRINT "NOT EQUAL"
40  END
50  PRINT "EQUAL":END
```

```
RUN
```

yields:

```
NOT EQUAL
```

This example corresponds to the preceding example, but line labels are used instead of line numbers.

```
IMMEDIATE OFF
A%=100
B%=50
IF A%=B% THEN GOTO QQQ ELSE PRINT "NOT EQUAL"
END
QQQ: PRINT "EQUAL":END
IMMEDIATE ON
```

```
RUN
```

yields:

```
NOT EQUAL
```
**IMAGE BUFFER MIRROR**

**Purpose**
Statement for mirror the print image around the Y-axis.

**Syntax**
```
IMAGE BUFFER MIRROR
```

**Remarks**
This statement mirrors the current defined image buffer around the Y-axis, that is, the feed direction. Fields defined after the IMAGE BUFFER MIRROR statement is executed are rendered normally. The image buffer width is always 8-bit aligned, even when the X-start parameter in the setup is not. Thus, it is recommended to test that the mirrored image is printed sidewise where intended. In some cases, a small correction using the PRPOS statement or the X-start parameter could be necessary.

**Example**
```
NEW
10 PRPOS 50,300
20 FONT "Swiss 721 BT",40
30 PRTXT "MIRROR"
40 IMAGE BUFFER MIRROR
50 PRPOS 50,100
60 PRTXT "NORMAL"
70 PRINTFEED
```
IMAGE BUFFER SAVE

Purpose
Statement for saving the content of the image buffer as a file.

Syntax
IMAGE BUFFER SAVE<sexp>

<sexp> is the desired name of the file, optionally with a reference to the device where the file should be saved.

Remarks
This statement saves the current content of the print buffer as an image file in RLL format. After saving, the file is automatically installed as an image, that can be printed using a PRIMAGE statement in DIR 1 and DIR 3. Thereby, you can create label templates, to which variable data easily can be added at will.

The size of the print buffer image depends on the size of the print image at the moment the buffer is saved. The width is decided by the Media, Media Size, Width setup value with the first pixel according to the Media, Media Size, Xstart setup value. The height is decided by the actual height in y-dimension of the print image. Note that space characters or invisible "white" parts of an image are included in the height of the print image, even if they are not visible on the printed label.

Example
IMAGE BUFFER SAVE "TEMPLATE7"
**IMAGE LOAD**

**Purpose**
Statement for receiving, converting and installing image and font files.

**Syntax**
```
IMAGE LOAD[<nexp1>,]<sexp1>,<nexp2>[,<sexp2>[,<nexp3>]]
```

- `<nexp1>` is optionally the number of bytes to skip before starting to read the data.
- `<sexp1>` is the desired name of the image or font to be created.
- `<nexp2>` is the size of the original file in number of bytes.
- `<sexp2>` is an optional flag:
  - "S" specifies that the image or font will be saved in the printer’s permanent memory ("/c"). Avoid this option (slow).
  - An empty string ("") specifies that the image or font will be stored in the printer’s temporary memory ("tmp:").
- `<nexp3>` optionally specifies a communication channel OPENed for INPUT by the number assigned to the device.
  (Default: Std IN channel.)

**Remarks**
This statement prepares the printer to receive a .PCX image file, an image file in the internal Intermec Fingerprint bitmap format, or a font file on the standard IN channel (see SETSTDIO statement) or on another communication channel OPEN for INPUT. When the file is received, it will automatically be converted to an image in the internal bitmap format of Intermec Fingerprint or to a scaleable font respectively.

The optional first parameter makes it possible to use this statement in MS-DOS (CR/LF problem).

The name of an image may consist of max. 30 characters including possible extension. The image will have the same direction as the original image file and can only be rotated 180° using a DIR statement. We therefore recommend that you include the extension .1 or .2 to indicate for which print directions the image is intended. Font file names are only restricted to 30 characters. The size of the original file should be given in bytes according to its size in the host.

Before IMAGE LOAD can be used on a serial channel, the setup must be changed to 8 characters, CTS/RTS handshake. When an IMAGE LOAD statement is executed, the execution stops and waits for the number of bytes specified in the statement to be received. During the transfer of image file data to the printer, there is a 25 seconds timeout between characters. If a new character has not been received within the timeout limit, Error 80 “Download timeout” occurs. When the specified number of characters have been received, the execution is resumed.

If the downloading was successful, the downloaded image or font will be installed automatically and can be used without any rebooting.

**Example**
```
IMAGE LOAD "Logotype.1",400,""
```
**IMAGENAME$**

**Purpose**
Function returning the names of the images stored in the printer's memory.

**Syntax**

```
IMAGENAME$(<nexp>)
```

- `<nexp>` is the result of the expression which is either false or true:
  - False (0) indicates first image.
  - True (≠0) indicates next image.

**Remarks**
This function can be used to produce a list of all images (another method is to use the IMAGES statement).

Image files downloaded by means of a TRANSFER KERMIT statement will not be returned, since the software will regard them as files rather than images.

- `IMAGENAME$(0)` produces the first name in the memory.
- `IMAGENAME$(≠0)` produces next name. Can be repeated as long there are any image names left.

**Example**
Use a program like this to list all image names:

```
10  A$=IMAGENAME$(0)
20  IF A$=""THEN END
30   PRINT A$
40  A$=IMAGENAME$(-1)
50  GOTO 20
RUN
```

yields for example:

```
CHESS2X2.1
CHESS4X4.1
DIAMONDS.1
GLOBE.1
```

Ok
IMAGES

Purpose
Statement for returning the names of all images stored in the printer’s memory to the standard OUT channel.

Syntax
IMAGES

Remarks
This statement can be used to list all image names (another method is to use an IMAGENAME$ function).
Image files downloaded by means of a TRANSFER KERMIT statement will not be printed, since the firmware will regard them as files rather than images.

Example
A list of images stored in the printer’s memory may look like this:
IMAGES
 yields for example:
CHESS2X2.1
DIAMONDS.1
CHESS4X4.1
GLOBE.1

3568692 bytes free  1717812 bytes used
Ok
IMMEDIATE

Purpose
Statement for enabling or disabling the immediate mode of Intermec Fingerprint in connection with program editing without line numbers, for reading the current mode, or for reading the current standard IN and OUT channels.

Syntax

<table>
<thead>
<tr>
<th>IMMEDIATE</th>
<th>ON</th>
<th>OFF</th>
<th>MODE</th>
<th>STDIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON</td>
<td>Enables the Immediate Mode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OFF</td>
<td>Disables the Immediate Mode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODE</td>
<td>Prints a line to the STDOUT port with information on the current status of the following modes (ON or OFF):</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Execution</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Immediate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Input</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Layout Input</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Debug STDIO (dbstdio)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STDIO</td>
<td>Prints two lines to the STDOUT port with information on current settings for the STDIN and STDOUT channels.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Remarks

IMMEDIATE ON|OFF
Before starting to write a program without line numbers, the immediate mode must be disabled by means of an IMMEDIATE OFF statement. If not, each line will be executed immediately.

After an IMMEDIATE OFF statement, program lines can be entered without any leading line numbers. References to lines are done using “line labels”, which are called in GOTO or GOSUB and related statements.

A line label is a name followed by a colon (:). The label must not interfere with any keywords or start with a digit and the line must start with the line label. When a line label is used as a reference to another line, for example within a GOTO statement, the colon should be omitted.

The program should be appended by a IMMEDIATE ON statement. At the execution of this statement, the program lines will be numbered automatically in ten-step incremental order, starting with the first line (10-20-30-40-50...). These line numbers will not appear on the screen until the program is LISTed, LOADed, or MERGEd. Line labels will not be converted to line numbers.

Do not issue a RUN statement before the IMMEDIATE ON statement, or an error will occur.
IMMEDIATE, cont.

IMMEDIATE MODE
Execution On/Off indicates if a Fingerprint program is running or not.
Immediate On/Off indicates whether the Immediate Mode is enabled or
disabled as specified by IMMEDIATE ON/OFF.
Input On/Off indicates whether the Direct Protocol is enabled or disabled
as specified by INPUT ON/OFF.
Layout Input On/Off indicates whether or not a layout is being recorded
in the Direct Protocol as specified by LAYOUT INPUT and LAYOUT
END.
Dbstdio On/Off indicates whether the debug standard I/O is active or not.
The following conditions are not reported:
- Running a Fingerprint application.
- Execution of a TRANSFER KERMIT, FILE& LOAD, IMAGE
  LOAD, LOAD, and STORE INPUT instruction.
- Running external commands (ush), for example RUN"rz......"
- Running the Setup Mode or execution of a SETUP statement.

IMMEDIATE STDIO
Two lines will be transmitted on the STDOUT port with information on
the current STDIN and STDOUT channels regarding port, baud rate,
character length, parity, and stop bits.

Examples
A program can be written without using any line numbers, as illustrated by
this short example. QQQ is used as a line label:
IMMEDIATE OFF

    Ok
    PRINT "LINE 1"
    GOSUB QQQ
    END
    QQQ: PRINT "LINE 2"
    RETURN
    IMMEDIATE ON
    Ok
    RUN

    LINE 1
    LINE 2
    Ok

    yields:
    LINE 1
    LINE 2
    Ok
    yields:
IMMEDIATE, cont.

This example shows how the status of the various modes are checked:

**IMMEDIATE MODE**

```plaintext
execution=OFF, immediate=ON, input=OFF, layout input = Off
```

This example shows how the status of the STDIN and STDOUT channels are checked:

**IMMEDIATE STDIO**

```plaintext
stdin=uart1:, 9600, 8, NONE, 1
stdout=uart1:, 9600, 8, NONE, 1
```
**INKEY$**

**Purpose**
Function reading the first character in the receive buffer of the standard IN channel.

**Syntax**

```
INKEY$
```

**Remarks**
For information on standard I/O channels, see SETSTDIO statement.

As opposed to the INPUT statement, INKEY$ does not interrupt the program flow to wait for input data, unless a loop is created by means of a GOTO statement, see line 20 in the example below.

INKEY$ is useful when the host computer is unable to end the input data with a “Carriage Return” (CR; ASCII 13 decimal), but must use some other character, for example “End of Text” (ETX; ASCII 3 decimal). Then a routine, which interprets the substitute character as a carriage return, can be created.

**Example**

In this example, none of the characters received on the standard IN channel will be printed on the host screen until a # character (ASCII 35 decimal) is encountered.

```
10   A$ = INKEY$
20   IF A$ = "" GOTO 10
30   IF A$ = CHR$(35) THEN PRINT B$
40   IF A$ = CHR$(35) THEN END
50   B$ = B$ + A$
60   GOTO 10
RUN
```

Type a number of characters on the keyboard of the host. They will not be printed on the host screen until you type a # character. Then all the characters will appear simultaneously, except for the #-sign.

Note the loop between line 10 and 20, which makes the program wait for you to activate a key.
INPUT (IP)

Purpose
Statement for receiving input data via the standard IN channel during the execution of a program.

Syntax
```
INPUT|IP[<scon><;|,>]<<nvar>|<svar>>[,<nvar>|<svar>>...]
```

- `<scon><;|,>` is an optional prompt string, followed by a semicolon or comma.
- `<<nvar>|<svar>>` are variables to which the input data will be assigned.

Remarks
For information on standard I/O channel, see SETSTDIO statement.

During the execution of a program, an INPUT statement will interrupt the execution. A question mark and/or a prompt will be displayed on the screen of the host to indicate that the program is expecting additional data to be entered. The prompt can be used to tell the operator what type of data he or she is expected to enter.

The prompt will be appended by a question mark if a semicolon (:) is entered after the prompt string. If a comma (,) is used in that position, the printing of the question mark will be suppressed.

If a prompt is not used, the question mark will always be displayed.

Do not enter any comma or semicolon directly after the keyword, only after the prompt, or in order to separate variables.

The input data should be assigned to one or several variables. Each item of data should be separated from next item by a comma. The number of data items entered must correspond to the number of variables in the list, or else an error condition will occur. The variables may be any mix of string variables and numeric variables, but the type of input data must agree with the type of the variable, to which the data is assigned.

Input can also be done directly to the system variables TIME$, DATE$, and SYSVAR.

The maximum number of characters that can be read using an INPUT statement is 32,767 characters.

Note that INPUT filters out any incoming ASCII 00 dec. characters (NUL).

INPUT does not support autohunting (see SETSTDIO).
INPUT (IP), cont.

Examples

This example shows input to one numeric variable and one string variable:

```
10   INPUT "ADDRESS"; A%, B$
20   PRINT A%; " "; B$
30   IF A% > 0 THEN GOTO 50
40   GOTO 10
50   END
RUN
```

yields:

```
ADDRESS?
```

When the prompt “ADDRESS?” appears on the screen, you can type the input data on the terminal’s keyboard, for example:

```
999, HILL STREET
```

Note the separating comma.

If the input text data contains a comma, which shall be printed, you must enclose the input data with quotation marks (“....”), for example:

```
999, "HILL STREET, HILLSBOROUGH"
```

Numeric input data must not include any decimal points.

This example shows how the date can be set directly from the keyboard of the host:

```
INPUT "Enter date: ", DATE$
```

yields:

```
Enter date:
```

When the prompt “Enter date:” appears on the screen of the host, you can type the date as a six-digit combination of year, month and day (see DATE$ variable). Time can also be set using the same method.
**INPUT ON/OFF**

**Purpose**
Statement enabling or disabling the Intermec Direct Protocol.

**Syntax**

<table>
<thead>
<tr>
<th>INPUT ON/OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
</tr>
</tbody>
</table>

**Remarks**
These statements are used to enter or leave the Intermec Direct Protocol.

**INPUT ON** Enables the Intermec Direct Protocol:
- Enables reception of input data to a stored layout
- Starts the error handler
- Sets the verbosity to off (SYSVAR (18) = 0)
- Shows “Direct Protocol 8.XX.X” in the display

**INPUT OFF** Disables the Intermec Direct Protocol:
- Disables reception of input data to a stored layout
- Stops the error handler
- Resets the verbosity to the level selected before last INPUT ON was executed
- Shows “Fingerprint 8.XX.X” in the display

The following instructions will only work with the Intermec Direct Protocol, that is after an INPUT ON statement has been executed:
- COUNT&
- ERROR
- FORMAT INPUT
- INPUT OFF
- LAYOUT END
- LAYOUT INPUT
- LAYOUT RUN

**Example**
This example illustrates how the Intermec Direct Protocol is enabled, how new separators are specified, how a layout is stored in the printer’s memory, how variable data are combined with the layout, and how a label is printed. Finally, the Intermec Direct Protocol is disabled:

```
INPUT ON
FORMAT INPUT "#", "@", ",&" LAYOUT INPUT "tmp:LABEL1" LAYOUT RUN
FT "Swiss 721 BT" PP 100,250 PT VAR1$ PP 100,200 PT VAR2$ LAYOUT END LAYOUT RUN "tmp:LABEL1" PF #Line number 1&Line number 2@ INPUT OFF
```
**INPUT#**

**Purpose**
Statement for reading a string of data from an OPEN device or sequential file.

**Syntax**

```
INPUT#<nexp>,<<nvar>|<svar>>[,<<nvar>|<svar>>...]
```

- `<nexp>` is the number assigned to the file or device when it was OPENed.
- `<<nvar>|<svar>>` is the variable to which the input data will be assigned.

**Remarks**
This statement resembles the INPUT statement, but allows the input to come from other devices than the standard IN channel or from various files. Like the INPUT statement, commas can be used to assign different portions of the input to different variables. INPUT# does not allow prompts to be used.

When reading from a sequential file, the records can be read one after the other by the repeated issuing of INPUT# statements with the same file reference.

Once a file record has been read, it cannot be read again until the file is CLOSED and then OPENed again.

The maximum number of characters that can be read using an INPUT# statement is 32,767 characters.

Note that INPUT# filters out any incoming ASCII 00 dec. characters (NUL).

**Example**

This example assigns data from the first record in the sequential file "Addresses" to the three string variables A$, B$, and C$ and from the second record in the same file to the string variables D$ and E$:

```
. . . . .
. . . . .
100 OPEN "ADDRESSES" FOR INPUT AS #5
110 INPUT#5, A$, B$, C$
120 INPUT#5, D$, E$
. . . . .
. . . . .
```
**INPUT$**

**Purpose**
Function returning a string of data, limited in regard of number of characters, from the standard IN channel, or optionally from an OPENed file or device.

**Syntax**

```
INPUT$(<nexp_1>[,<nexp_2>])
```

- `<nexp_1>` is the number of characters to be read.
- `<nexp_2>` optionally specifies a file or device using the number assigned to it when it was OPENed.

**Remarks**
If no file or device is specified, the input will come from the standard I/O channel, see SETSTDIO statement.

Otherwise, it will come from the specified file or device. The execution will be held until the specified number of characters has been received from the keyboard console, file, or communication channel. If a file does not contain the specified number of characters, the execution will be resumed as soon as all available characters in the file have been received.

The maximum number of characters that can be returned using an INPUT$ statement is 65,536 characters.

**Examples**

This example reads a sequence of 25 characters from the printer’s built-in keyboard and assigns them to a string variable named Z$:

```
. . . . .
. . . . .
1000 OPEN "CONSOLE:" FOR INPUT AS #1
1010 Z$=INPUT$(25,1)
. . . . .
```

In this example, 10 characters are read from the standard IN channel and assigned to a variable.

```
10 A$=INPUT$(10)
```
**INSTR**

**Purpose**
Function searching a specified string for a certain character, or sequence of characters, and returning its position in relation to the start of the string.

**Syntax**

```
INSTR([<nexp>,]<sexp1>,<sexp2>)
```

- `<nexp>` is, optionally, the position where the search will start.
- `<sexp1>` is the string to be searched.
- `<sexp2>` is the character(s) for which the string will be searched.

**Remarks**
INSTR allows you to search a string for some particular character(s) and return the position of the character, or the first character in the sequence, as a number of characters positions measured from the start of the string.

As an option, it is possible to specify a certain position in the string from which the search will start. If no start position is specified, the search will start at the beginning of the string.

The result will be zero if:
- the start position value exceeds the length of the string.
- the string is empty.
- the searched combination of characters cannot be found.

**Examples**

In this example, the string "INTERMEC_PRINTER_AB" is searched for the character combination "AB". No start position is specified.

```
10  A$="INTERMEC PRINTER AB"
20  B$="AB"
30  PRINT INSTR(A$,B$)
```

RUN

yields:

```
18
```

In next example, the string "INTERMEC_PRINTER_AB" is searched for the character "I" and the start position is specified as 4.

```
10  A$="INTERMEC PRINTER AB"
20  B$="I"
30  PRINT INSTR(4,A$,B$)
```

RUN

yields:

```
12
```
**INVIMAGE (II)**

**Purpose**
Statement for inversing the printing of text and images from “black-on-white” to “white-on-black.”

**Syntax**

```
INVIMAGE | II
```

| Default: | NORIMAGE |
| Reset to default by: | PRINTFEED execution |

**Remarks**

This statement can only be used in connection with the printing of text and images (PRTXT and PRIMAGE). In the matrix of the font or image, all “white” dots will be black and all black dots will be “white.” (Obviously, “white” means that the dots will not be subjected to heat and the media therefore will retain its original color, whereas “black” means the color of the printing.)

This implies that most fonts will be printed on a black background which ascends and descends slightly more than most of the characters. Not all fonts are suited for inverse printing. Thin lines, serifs, and ornaments may be difficult to distinguish. There may also be an imbalance between the ascending and descending black background.

The same principles apply to images. The normally invisible background may be larger than expected or be less favourably balanced. Small “white” details tend to be blurred out by the black background. Therefore, before using an inverse image, make a printout sample.

INVIMAGE will be revoked by a NORIMAGE statement.

**Example**

```
10 PRPOS 30,300
20 DIR 1
30 ALIGN 4
40 INVIMAGE
50 FONT "Swiss 721 BT"
60 PRTXT "Inverse printing"
70 PRINTFEED
RUN
```
**KEY BEEP**

**Purpose**
Statement for resetting the frequency and duration of the sound produced by the beeper, when any of the keys on the printer’s keyboard is pressed down.

**Syntax**

\[
\text{KEY}_{\leftrightarrow} \text{BEEP}<\text{nexp}_1>,<\text{nexp}_2>
\]

- \(<\text{nexp}_1>\) is the frequency of the sound in Hz.
- \(<\text{nexp}_2>\) is the duration of the sound in periods of 0.020 seconds each (max. 15,0000 = 5 minutes).

Default:
- Frequency: 1200 Hz
- Duration: 0.030 sec.

**Remarks**
This statement sets the response for all keys of the printer. To turn off the audible key response, set the frequency to a value higher than 9999.

Note that the beeper is disabled during printing.

The table below illustrates the relation between frequencies and the musical scale (same as in the SOUND statement).

<table>
<thead>
<tr>
<th>Key</th>
<th>Hz</th>
<th>Key</th>
<th>Hz</th>
<th>Key</th>
<th>Hz</th>
<th>Key</th>
<th>Hz</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>131</td>
<td>C</td>
<td>262</td>
<td>C</td>
<td>523</td>
<td>C</td>
<td>1047</td>
</tr>
<tr>
<td>C#</td>
<td>138</td>
<td>C#</td>
<td>277</td>
<td>C#</td>
<td>554</td>
<td>C#</td>
<td>1109</td>
</tr>
<tr>
<td>D</td>
<td>147</td>
<td>D</td>
<td>294</td>
<td>D</td>
<td>587</td>
<td>D</td>
<td>1175</td>
</tr>
<tr>
<td>D#</td>
<td>155</td>
<td>D#</td>
<td>311</td>
<td>D#</td>
<td>622</td>
<td>D#</td>
<td>1245</td>
</tr>
<tr>
<td>E</td>
<td>165</td>
<td>E</td>
<td>330</td>
<td>E</td>
<td>659</td>
<td>E</td>
<td>1319</td>
</tr>
<tr>
<td>F</td>
<td>175</td>
<td>F</td>
<td>349</td>
<td>F</td>
<td>699</td>
<td>F</td>
<td>1397</td>
</tr>
<tr>
<td>F#</td>
<td>185</td>
<td>F#</td>
<td>370</td>
<td>F#</td>
<td>740</td>
<td>F#</td>
<td>1480</td>
</tr>
<tr>
<td>G</td>
<td>196</td>
<td>G</td>
<td>392</td>
<td>G</td>
<td>784</td>
<td>G</td>
<td>1568</td>
</tr>
<tr>
<td>G#</td>
<td>208</td>
<td>G#</td>
<td>415</td>
<td>G#</td>
<td>831</td>
<td>G#</td>
<td>1662</td>
</tr>
<tr>
<td>A</td>
<td>220</td>
<td>A</td>
<td>440</td>
<td>A</td>
<td>880</td>
<td>A</td>
<td>1760</td>
</tr>
<tr>
<td>A#</td>
<td>233</td>
<td>A#</td>
<td>466</td>
<td>A#</td>
<td>933</td>
<td>A#</td>
<td>1865</td>
</tr>
<tr>
<td>B</td>
<td>247</td>
<td>B</td>
<td>494</td>
<td>B</td>
<td>988</td>
<td>B</td>
<td>1976</td>
</tr>
</tbody>
</table>

**Example**
In this example, the beeper will produce an A in the one-line octave for 1 second each time a key is pressed down.

10 KEY BEEP 440,50

. . . . .

. . . . .
**KEY ON/OFF**

**Purpose**
Statement enabling or disabling a specified key on the printer’s front panel to be used in connection with an ON KEY...GOSUB statement.

**Syntax**
```plaintext
KEY(<nexp>)OFF|ON
```

- `<nexp>` is the id. number of one of the keys on the printer’s front panel
- OFF|ON disables|enables the specified key.

**Remarks**
Using an ON KEY... GOSUB statement, any key (except the <Shift> key) can be assigned to make the program branch to a subroutine. The keys are enabled/disabled individually and are specified by means of their respective id. numbers in unshifted and/or shifted position, see Appendix A. To specify a shifted key, add 100 to the unshifted id. number the key.

The EasyCoder PD41 only has one button, with id. number 17. The button generates event 117 when released.

Please note the difference between the id. numbers of the keys and the ASCII values they are able to produce (see KEYBMAP$).

**Example**
In this example, the ⏏/F1 key (id. No. 10) is first enabled, then used for branching to a subroutine and finally disabled.

```
10 KEY (10) ON
20 ON KEY (10) GOSUB 1000
30 KEY (10) OFF
```
**KEYBMAP$**

**Purpose**

Variable returning or setting the keyboard map table.

**Syntax**

**Read the map table:**

\[
<\text{svar}> = \text{KEYBMAP$}(<\text{nexp}>)
\]

- `<svar>` returns the keyboard mapping
- `<nexp>` is the type of string to be returned:
  - 0 = Shift or Alt key not activated (64 characters)
  - 1 = Shift key activated (64 characters)
  - 5 = Alt key activated (64 characters) (Alphanumeric keyboard on EasyCoder PX4i and PX6i only)

**Set the map table:**

\[
\text{KEYBMAP$}(<\text{nexp}>) = <\text{sexp}>
\]

- `<nexp>` is the type of string to be remapped:
  - 0 = Shift or Alt key not activated (64 characters)
  - 1 = Shift key activated (64 characters)
  - 5 = Alt key activated (64 characters) (Alphanumeric keyboard on EasyCoder PX4i and PX6i only)
- `<sexp>` is the string specifying the ASCII value for each key position in the selected type of string.

**Remarks**

In the KEYBMAP$ statement, each key on the printer’s front panel has two characteristics:

- The physical location of the key (position number).
  (This is not the same thing as the key’s Id. No, see KEY ON/OFF or ON KEY GOSUB.)
- The ASCII decimal value that will be produced when the key is pressed.
  (Compare with BREAK.)

Refer to Appendix A for illustrations of position numbers, id. numbers, and ASCII values for the various printer models and keyboard types.

In principle, each physical key can produce two or three different ASCII values, one in unshifted position, another in shifted position, and—in case of an alphanumeric keyboard—a third when the `<Alt>` key is activated. One key is appointed `<Shift>` key. When the `<Shift>` key is pressed at the same time as another key, the unshifted ASCII value of the latter will be increased by 128.

You can use the KEYBMAP$ instruction in two ways:

**Reading the keyboard mapping**

You can read how the keyboard is mapped depending on if the `<Shift>` or `<Alt>` is activated or not. The printer will return a string of ASCII values in ascending key position number. Because many keys return non-printable ASCII values (ASCII 00-31 dec.), all will not be returned to the screen of the host or printed on a label.
KEYBMAP$, cont.

Changing the keyboard mapping
You can change the mapping of the keyboard, so a key will produce another ASCII value than before. To do that, you must create a string which specifies the ASCII value for each of all 64 unshifted, shifted, or Alt-initiated key positions in ascending order. Regardless of what the keyboard looks like, there are always 64 theoretical key positions.

Note: The exception being the EasyCoder PD41, whose keyboard map is only 6 bytes long. Index zero indicates the Print button, and 1-5 being reserved.

Characters, that cannot be produced by the keyboard of the host, can be substituted by CHR$ functions, where the character is specified by its ASCII decimal value according to the selected character set (see NASC statement.) The same applies to special characters. Key positions which should be disabled or are not included in the physical keyboard can be mapped as NUL, using the function CHR$(0)$. Note that the position of the <Shift> key cannot be remapped.

When a key is remapped, its id. number will follow the key to its new position. Most keys on the alphanumeric keyboard do not have id. numbers.

There is no default mapping for the alphanumeric keyboard, so the printer assumes that a normal keyboard is fitted and reacts accordingly. To make full use of the alphanumeric keyboard for EasyCoder PX4i and PX6i, all keys have to be mapped, see Appendix A.

Note: In the Setup Mode, the keys have fixed positions and are not affected by any KEYBMAP$ statement. KEYBMAP$ only affects the keys when used outside the Setup Mode.

Examples

The following example illustrates the mapping of the keyboard for EasyCoder PF4i (unshifted keys only).

10  B$=CHR$(1)+STRING$(4,0)+CHR$(2)+STRING$(4,0)+CHR$(3)
20  B$=B$+STRING$(4,0)+CHR$(4)+STRING$(4,0)+CHR$(5)+STRING$(9,0)
30  B$=B$+CHR$(13)+CHR$(28)+CHR$(29)+CHR$(30)+STRING$(6,0)
40  B$=B$+".147"+CHR$(0)+"0258"+CHR$(0)+CHR$(8)+"369"+CHR$(0)+(CHR$(31)+STRING$(8,0)
50  KEYBMAP$(0)=B$
RUN
**KILL**

**Purpose**

Statement for deleting a file, directory, or complete directory sub-trees from the printer’s memory or from a CompactFlash memory card inserted in the memory card adapter.

**Syntax**

```
KILL<sexp>[R,A]
```

- `<sexp>` is the file or directory to be deleted.
- `R` recursively removes all non-system files in the specified sub-tree and then removes all empty directories in the same sub-tree.
- `A` optionally specifies that system files also should be removed.

**Remarks**

The name of the file to be deleted must match the name given when the file was saved, see SAVE statement. The name must include the extension. If no extension was entered manually by the operator when the file was SAVED, the extension "-.PRG" was added automatically.

To KILL a file residing in another directory than the current one (see CHDIR statement), you must include a reference to the directory in question when you specify the file, for example "card1:<filename>.XYZ".

KILL cannot be used for files residing in "rom:", "storage:", or "lock:".

A directory cannot be removed if it contains files or directories unless the R flag is included in the KILL statement. Otherwise error 1073, "Directory not empty" will occur.

A trailing slash character (/) may be added to directory names, but is not necessary.

The A and R flags are only applicable when removing directories, doing otherwise will result in error 1034, “Not a directory”.

The root directory of any device cannot be removed.

Note the symmetry with FILES. FILES<sexp>,R and FILES<sexp>,R,A list files and directories that will be removed using KILL<sexp>,R and KILL<sexp>,R,A respectively.

The current directory may be removed (for example KILL CURDIR$,R). The current directory is not changed after such a command, but is invalid and a successful CHDIR statement is necessary to restore the current directory to one that exists (CHDIR".." may not work).

Also see CHDIR, FILES, and MKDIR
**KILL, cont.**

**Example**

```plaintext
10   ON ERROR GOTO 1000
20   CHDIR("/c")
30   MKDIR "DIR1"                Create the directory DIR1
40   FILES
50   COPY "STDIO", "DIR1"       Copy STDIO into DIR1
60   FILES "DIR1"               List files in DIR1
70   KILL "DIR1"                Try to remove DIR1
80   KILL "DIR1",R              Remove the directory recursively
90   FILES
100  END
1000 PRINT "error number "; ERR;"in line ";ERL
1010 RESUME NEXT
RUN
```

```plaintext
yields for example:
```

Files on /c

```

<table>
<thead>
<tr>
<th>Directory</th>
<th>Application</th>
<th>Admin</th>
</tr>
</thead>
<tbody>
<tr>
<td>boot/</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>STDIO</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

```

22210562 bytes free  4 bytes used

```

STDIO

```

2220032 bytes free  4 bytes used

Error number 1073 in line 70

Files on /c

```

<table>
<thead>
<tr>
<th>Directory</th>
<th>Application</th>
<th>Boot</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLICATION</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ADMIN/</td>
<td>0</td>
<td>STDIO</td>
</tr>
</tbody>
</table>

```

2222080 bytes free  4 bytes used
LAYOUT

Purpose

Statement for handling of layout files.

Syntax

\[ \text{LAYOUT}[F,] <\text{sexp}_1>,<\text{sexp}_2>,<\text{sexp}_3>,<\text{svar}>|<\text{sexp}_3>,<\text{nvar}>|<\text{sexp}_4> \]

- \( F, \) optionally allows use of data and error files instead of arrays
- \(<\text{sexp}_1>\) is the layout file.
- \(<\text{sexp}_2>\) is the logotype name file.
- \(<\text{svar}>\) is the data array (\(<\text{svar}>\)) or data file (\(<\text{sexp}_3>\)).
- \(<\text{nvar}>\) is the error array (\(<\text{nvar}>\)) or error file (\(<\text{sexp}_4>\)).

Remarks

<\text{sexp}_1>: Layout file format sorted in ascending order (Records 1-n, 52 bytes each)
Input: \( H = \) hex digit, \( D = \) Numeric digit, \( C = \) Alpha character

<table>
<thead>
<tr>
<th>Byte #</th>
<th>Parameter</th>
<th>Layout Type</th>
<th>Input</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Element number</td>
<td>HH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Layout type</td>
<td>C</td>
<td>D</td>
<td>Note 1</td>
</tr>
<tr>
<td></td>
<td>A = Logotype by name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B = Bar code</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C = Text</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>E = Bar code extended field</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>H = Barfont on/off</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>J = Baradjust (corresponds to BARADJUST stmt)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>L = Logotype by number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S = Line</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X = Box</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Direction</td>
<td>A,B,C,L,S,X,or X</td>
<td>Dddd</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Barfont on/off (0=off; 1=on)</td>
<td>H</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Security</td>
<td>E</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Alignment</td>
<td>A,B,C,L,S,X</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Aspect height ratio</td>
<td>E</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>5-8</td>
<td>X-position</td>
<td>A,B,C,L,S,X</td>
<td>DDDD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Aspect width ratio</td>
<td>E</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Baradjust left</td>
<td>J</td>
<td>DDDD</td>
<td></td>
</tr>
<tr>
<td>9-12</td>
<td>Y-position</td>
<td>A,B,C,L,S,X</td>
<td>DDDD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rows in bar code</td>
<td>E</td>
<td>DD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Baradjust right</td>
<td>J</td>
<td>DDDD</td>
<td></td>
</tr>
<tr>
<td>13-22</td>
<td>Font name</td>
<td>C</td>
<td>C_{1-10}</td>
<td>Note 2</td>
</tr>
<tr>
<td></td>
<td>Logotype name</td>
<td>A</td>
<td>C_{1-10}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bar code name</td>
<td>B</td>
<td>C_{1-10}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Barfont name</td>
<td>H</td>
<td>C_{1-10}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Line length</td>
<td>S</td>
<td>DDDD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Box width</td>
<td>X</td>
<td>DDDD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Columns in bar code</td>
<td>E</td>
<td>DD</td>
<td>Byte 13-14</td>
</tr>
<tr>
<td></td>
<td>Truncate according to code spec’s</td>
<td>E</td>
<td>D</td>
<td>Byte 15</td>
</tr>
<tr>
<td>23-42</td>
<td>Fixed text or alphanumeric data</td>
<td>B or C</td>
<td>C_{1-10}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fixed numeric data</td>
<td>B</td>
<td>D_{1-10}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Logotype number</td>
<td>L</td>
<td>DD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Box height</td>
<td>X</td>
<td>DDDD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Line thickness</td>
<td>S</td>
<td>DDDD</td>
<td></td>
</tr>
<tr>
<td>43-44</td>
<td>No of char. to print (of byte 23-42)</td>
<td>B or C</td>
<td>DD</td>
<td></td>
</tr>
<tr>
<td>45-46</td>
<td>Image type (I = inverse image)</td>
<td>A,C, or L</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bar code ratio (wide/narrow bars)</td>
<td>B</td>
<td>DD</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>Vertical magnification</td>
<td>A,C, or L</td>
<td>D or C</td>
<td>Note 3</td>
</tr>
<tr>
<td></td>
<td>Bar code magnification</td>
<td>B</td>
<td>D or C</td>
<td>Note 3</td>
</tr>
<tr>
<td>48</td>
<td>Horizontal magnification</td>
<td>A,C, or L</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>49-51</td>
<td>Bar code height</td>
<td>B</td>
<td>DDD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Line thickness</td>
<td>X</td>
<td>DDDD</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 2—Program Instructions

LAYOUT, cont.

Note 1: The bar code extended field record (E) corresponds to the six last parameters in the BARSET statement. Must have a lower element number than the corresponding bar code record (B), which specifies the other bar code parameters.

Note 2: The maximum font name length in the LAYOUT statement is 10 characters. Most font names in Intermec Fingerprint are longer. A workaround method is to use font name aliases with a maximum length of 10 characters, See Chapter 6, “Fonts”.

Note 3: If a magnification of 0-9 is sufficient, enter a numeric digit. If a higher magnification than 9 is required, enter the character with the ASCII decimal number minus 48 that corresponds to the desired magnification, that is, if magnification 10 is desired, enter the character : (colon, ASCII 58 dec).

Logotype name file format #1:
(no embedded spaces in name)
Record 1–n, 10 bytes each.

| C1...C10 | Name for logotype No. 1 |
| ... | ... |
| C1...C10 | Name for logotype No. n |

Logotype name file format #2:
(Records sorted in ascending logotype number order)
Record 1–n, 13 bytes each.

| DD | Logotype number (2 digits) |
| C1...C10 | Name of logotype (10 characters) |

Note: Logotype name file formats #1 and #2 are alternative.

Data array/file format:
(sorted in ascending order)
One array position/One file line.

| HH | Element number |
| C1...Cn | Data |

If a data element cannot be used in the layout, an error will occur and the index of the unused element and error code -1 is placed in the error array/file.

Error array/file format:
(sorted in ascending order)

| Array position/File line No. 0: | Record number for error 1 |
| Array position/File line No. 1: | Error number for error 1 |
| ... | ... |
| Array position/File line No. 2n-2: | Record number for error n |
| Array position/File line No. 2n-1: | Error number for error n |
LAYOUT, cont.

To improve the performance, it is strongly recommended to create the layout and logotype name files in the printer's temporary memory ("tmp:"). Once they have been created in "tmp:“, they could be copied to the printer’s permanent memory to avoid losing them at power off.

Do not confuse this statement with the statements LAYOUT INPUT, LAYOUT END, and LAYOUT RUN.

Example

Note that the 10 characters available to define a font in the LAYOUT statement in most cases cannot accommodate modern outline font names. Instead, use font aliases as described in Chapter 6. In the example below, the font aliases are indicated by lowercase italic typing (lines 90–120, 150).

```plaintext
10  DIM QERR%(10)
20  LAYDATA$(0)="01DAY"
30  LAYDATA$(1)="04123456789012"
40  QERR%(0)=0
50  OPEN "tmp:LOGNAME.DAT" FOR OUTPUT AS 19
60  PRINT# 19,"DIAMONDS.1";
70  CLOSE 19
80  OPEN "tmp:LAYOUT.DAT" FOR OUTPUT AS 6
90  PRINT# 6,"01C11100 10 font alias 00I 11 ";
100 PRINT# 6,"01C11100 40 font alias 00 22 ";
110 PRINT# 6,"01C11100 100 font aliasWEDNES 06I 11 ";
120 PRINT# 6,"01C11100 130 font aliasSATURNUS 05I 11 ";
130 PRINT# 6,"02L11300 70 1 33 ";
140 PRINT# 6,"03S11100 210 300 3 ";
150 PRINT# 6,"04H1 font alias ";
160 PRINT# 6,"04B14100 300 EAN13 0 312 100";
170 CLOSE 6
180 LAYOUT "tmp:LAYOUT.DAT","tmp:LOGNAME.DAT",LAYDATA$,QERR%
190 IF QERR%(1) = 0 THEN GOTO 260
200 PRINT "-ERROR- LAYOUT 1"
210 I%=0
220 IF QERR%(I%)=0 THEN GOTO 260
230 PRINT " ERROR ";QERR%(I%+1);" in record ";QERR%(I%)
240 I%=I%+2
250 GOTO 220
260 PRINTFEED
```
Chapter 2—Program Instructions

**LAYOUT END**

**Purpose**
Statement for stopping the recording of a layout description and saving the layout (Intermec Direct Protocol only).

**Syntax**
```
LAYOUT END
```

**Remarks**
This statement can only be used in the Intermec Direct Protocol after a layout has been recorded by means of a LAYOUT INPUT statement. After a LAYOUT END statement has been executed, no more data will be added to the layout.

By default, the layout will be saved in the printer’s permanent memory (“/c”). To speed up the execution it can, as an alternative, be saved in the temporary memory (see LAYOUT INPUT statement). The layout can be copied and killed as any other program file.

**Example**
This example illustrates how the Intermec Direct Protocol is enabled, how new separators are specified, how a layout is stored in the printer’s temporary memory, how variable data are combined with the layout, and how a label is printed. Finally, the Intermec Direct Protocol is disabled:
```
INPUT ON
FORM INPUT "#","@","&"
LAYOUT INPUT "tmp:LABEL1"
FT "Swiss 721 BT"
PP 100,250
PT VAR1$
PP 100,200
PT VAR2$
LAYOUT END
LAYOUT RUN "tmp:LABEL1"
#Line number 1&Line number 2&@
PF
INPUT OFF
```
LAYOUT INPUT

Purpose
Statement for starting the recording of a layout description (Intermec Direct Protocol only).

Syntax

\[
\text{LAYOUT INPUT } <\text{sexp}>
\]

\(<\text{sexp}>\) is the desired name of the layout (max. 30 characters) including name of the device where the layout is to be stored.

Remarks
This statement can only be used in the Intermec Direct Protocol and starts the recording of a layout. All formatting instructions, such as PRPOS, MAG, FONT, BARFONT, BARSET, PRTXT, PRBAR, PRIMAGE, PRBOX, PRLINE, etc., which are transmitted to the printer on the standard IN channel after a LAYOUT INPUT statement and before a LAYOUT END statement, will be included in the layout.

Layouts cannot be created in "/c" (which by default is the current directory), but must be created in the printer's temporary memory ("tmp:"), or possibly in a CompactFlash card ("card1:"). Once a layout has been created in the temporary memory ("tmp:"), it can be copied to either "/c" or "card1:" so it will not be lost at power-off or reboot.

Variable input data to text, bar code, and image fields can be provided separately using a LAYOUT RUN statement. Such variable data are indicated in the layout by string variables \texttt{VARn$} where \texttt{"n"} is the number of the field in the LAYOUT RUN string of data. For example, the statement PRTXT "Hello" in the layout results in a fixed text, whereas the statement PRTXT VAR1$ results in a variable text, which is provided by the first field in a LAYOUT RUN string.

The layout must not contain any PRINTFEED statements. The layout will not be saved until a LAYOUT END statement is executed.

Example
In this example, the Intermec Direct Protocol is enabled, new separators are specified, a layout is stored in "tmp:”, data are combined with the layout, and a label is printed. Finally, the Direct Protocol is disabled:

\begin{verbatim}
INPUT ON
FORMAT INPUT "#", "@", "&" ↓
LAYOUT INPUT "tmp:LABEL1" ↓
FT "Swiss 721 BT" ↓
PP 100,250 ↓
PT VAR1$ ↓
PP 100,200 ↓
PT VAR2$ ↓
LAYOUT END ↓
LAYOUT RUN "tmp:LABEL1" ↓
#Line number 1&Line number 2@ ↓
PF ↓
INPUT OFF ↓
\end{verbatim}
Chapter 2—Program Instructions

LAYOUT RUN

Purpose

Statement for providing variable input data to a predefined layout (Intermec Direct Protocol only).

Syntax

```
LAYOUT RUN <sexp>
```

- `<sexp>` is the name of the layout as specified in the LAYOUT INPUT statement.

Remarks

This instruction can only be used in the Intermec Direct Protocol and is used to select a predefined layout in a specified part of the printer’s memory (see LAYOUT INPUT statement) and provide input to string variables in the layout. Such variables are indicated by VARn$, where “n” indicates a field in the string of data that should follow the LAYOUT RUN statement.

The string of input data should be composed according to the following syntax, where <STX> is the start-of-text separator, <CR> is the field separator and <EOT> is the end-of-text separator (see FORMAT INPUT statement):

```
<STX><input to VAR1$><CR><input to VAR2$><CR>...<input to VARn$><CR><EOT>
```

Before reverting to “normal” Fingerprint printing after having used variable data (LAYOUT INPUT, LAYOUT END, and LAYOUT RUN), the data must be cleared using LAYOUT RUN with an empty string (LAYOUT RUN "").

Example

This example illustrates how the Intermec Direct Protocol is enabled, how new separators are specified, how a layout is stored in the printer’s temporary memory, how variable data are combined with the layout, and how a label is printed. Finally, the Intermec Direct Protocol is disabled:

```
INPUT ON
FORMAT INPUT "#","@","&" LOCAL
LAYOUT INPUT "tmp:LABEL1" LOCAL
FT "Swiss 721 BT" LOCAL
PP 100,250 LOCAL
PT VAR1$ LOCAL
PP 100,200 LOCAL
PT VAR2$ LOCAL
LAYOUT END LOCAL
LAYOUT RUN "tmp:LABEL1" LOCAL
#Line number 1&Line number 2&@ LOCAL
PF LOCAL
INPUT OFF LOCAL
```
**LBLCOND**

**Purpose**
Statement for overriding the media feed setup.

**Syntax**

```
LBLCOND<nexp_1>,<nexp_2>|<nexp_3>
```

- `<nexp_1>` specifies the type of action:
  - 0 = Overriding the stop adjust.
  - 1 = Overriding the start adjust.
  - 2 = Turning off the Label Stop Sensor/Black Mark Sensor.
  - 3 = Selecting the mode specified by `<nexp_2>`

- `<nexp_2>` specifies `<nexp_1> = 0, 1, or 2` as a number of dots.

- `<nexp_3>` specifies one of the following modes:
  - 0 = Legacy Mode
  - 1 = IPL Mode
  - 2 = Gap Truncate Mode
  
  Default: LBLCOND 3,2

**Remarks**

This instruction allows you to override the printer’s feed-adjust setup or to temporarily disable the label stop sensor or black mark sensor:

- `<nexp_1> = 0` temporarily sets the stop adjust to the value specified by `<nexp_2>`.

- `<nexp_1> = 1` temporarily sets the start adjust to the value specified by `<nexp_2>`.

- `<nexp_1> = 2` makes the label stop sensor (LSS) or black mark sensor temporarily ignore any gaps or marks detected within the length of media feed specified by `<nexp_2>`. However, the label length must be greater than than the distance between the LSS and the tear bar (if not, use LBLCOND 3,xx). This allows the use of labels of such shapes that would make the LSS react prematurely, or tickets with preprint at the back of the media that would interfere with the detection of the black mark.

- `<nexp_1> = 3` is useful as an alternative to LBLCOND 2,xx when the length of the label or ticket is shorter than the distance between the LSS and the tear bar. It makes it possible to select one of the modes specified by `<nexp_3>`.

**Legacy Mode ( `<nexp_3> = 0` )**

If the print image is longer than the physical length of the label or ticket, the print image will extend into the next label until the media feed stops according to the stop adjust setup (for example when the gap becomes aligned with the tear bar). This means that the print image may be truncated, the next label may have to be discarded, and some of the print image may coincide with a gap or slot. This mode was called “Default Mode” in earlier versions of Intermec Fingerprint.
**LBLCOND, cont.**

**IPL Mode** \((\text{nexp}_3 = 1)\)

If the print image is longer than the physical length of the label or ticket, the print image will extend into the following label(s) until the entire print image has been printed. Then the media is fed out to the next gap or mark according to the stop adjust setup. This means that the print image will not be truncated but may extend into one or more consecutive labels, and some of the print image may coincide with gaps or slots.

**Gap Truncate Mode** \((\text{nexp}_3 = 2)\)

If the print image is longer than the physical length of the label or ticket, only the part of the print image that fits on the label or ticket will be printed and the remainder will be ignored. This means that some of the print image may not be printed at all, but the following labels will not be affected.

Verifying a start adjust or stop adjust value in the Setup Mode by pressing key No. 16 (normally labeled “Enter”), or setting the value using a setup file or setup string, will revoke any LBLCOND statement for the parameter in question.

The label stop sensor will be returned to normal operation by the statement: LBLCOND 2,0

All current LBLCOND statements will be revoked at startup or the execution of a REBOOT statement. This means that the start and stop adjust will be decided by the setup and the label stop sensor will work normally.

**Example**

In this example, the start adjust value in the setup mode is overridden and the label stop sensor is set to ignore any gaps in the web within 20 mm (160 dots at 8 dots/mm; 240 dots at 12 dots/mm) of media feed:

```
10   LBLCOND 1,5: LBLCOND 2,160
20   FONT "Swiss 721 BT"
30   PRTXT "Hello"
40   PRINTFEED
RUN
```
LED ON/OFF

Purpose
Statement for controlling the dual-color “Status” indicator.

Syntax

\[
\text{LED<nexp>ON|OFF|BLINK [,DATAIN]}
\]

\(<\text{nexp}>\) specifies the color of the Status indicator:

\(0\) controls the green LED (default ON).
\(1\) controls the red LED (default OFF).

Remarks
All present Intermec Fingerprint printers are equipped with three LED (Light Emitting Diode) control lamps on the front panel. The center one of the LEDs (normally marked “Status” on the keyboard overlay) can be used to indicate, for example, when an error occurs, when the printer is ready, or when data is received.

Under the Direct Protocol, the “Status” LED starts blinking green, when data are received on the standard input channel, and switches to solid green, when the channel has been silent for 0.8 seconds. Under the Direct Protocol, the Status LED may also be affected by the error handler, see ERROR statement.

BLINK mode means that the “Status” LED is lit for 0.4 seconds at an interval of 0.4 seconds.

If the DATA IN flag is set with the BLINK mode, reception of data on the standard input channel controls whether the “Status” LED shall blink or be switched off, exactly as the behavior under the Direct Protocol.

If you set LED 0 ON and LED 1 ON, the “Status” LED will get solid red.

Note: If the printer has a console that does not includes the blue Intermec Ready-to-Work Indicator, please refer to Intermec Fingerprint v8.10, Programmer’s Reference Manual for a description of the LED ON/OFF statement.

Example
In this example, the “Status” LED will be solid red if you, for example, attempt to run the program with a raised printhead. Lower the printhead and a label will be fed out. The “Status” LED switches to solid green.

10 LED 0 ON
20 LED 1 OFF
30 ON ERROR GOTO 1000
40 PRPOS 30,300
50 FONT "Swiss 721 BT"
60 PRTXT "OK!"
70 PRINTFEED
80 LED 0 ON
90 LED 1 OFF
100 END

.......

1000 LED 0 OFF
1010 LED 1 ON
1020 RESUME
**LEFT$**

**Purpose**
Function returning a specified number of characters from a given string starting from the extreme left side of the string, that is from the start.

**Syntax**

LEFT$(<sexp>,<nexp>)

- `<sexp>` is the string from which the characters will be returned.
- `<nexp>` is the number of characters to be returned.

**Remarks**
This function is the complementary function for RIGHT$, which returns the characters starting from the extreme right side, that is from the end.

If the number of characters to be returned is greater than the number of characters in the string, then the entire string will be returned. If the number of characters is set to zero, a null string will be returned.

**Examples**

```plaintext
10   PRINT LEFT$("THERMAL PRINTER", 7)
RUN
     yields:
     THERMAL

10   A$="THERMAL PRINTER":B$="LABEL"
20   PRINT LEFT$(A$, 8);LEFT$(B$, 10);"S"
RUN
     yields:
     THERMAL LABELS
```
LEN

**Purpose**

Function returning the number of character positions in a string.

**Syntax**

LEN(<sexp>)

<sexp> is the string from which the number of characters will be returned.

**Remarks**

The number of characters to be returned includes unprintable characters, but the quotation marks enclosing the string expression are not included.

**Examples**

In this example, lines 40 and 50 illustrate two ways of using the LEN function, when the number of characters from several string expressions are to be added up.

```plaintext
10  A$="INTERMEC"         (8 char.)
20  B$="THERMAL"           (7 char.)
30  C$="PRINTERS"          (8 char.)
40  PRINT LEN(A$+B$+C$)      (8 char.)
50  PRINT LEN(A$)+LEN(B$)+LEN(C$)
   RUN
```

yields:

23
23

This example illustrates that unprintable characters, for example space characters, are included in the value returned by the LEN function:

```plaintext
PRINT LEN("INTERMEC THERMAL PRINTERS")
```

yields:

25
**LET**

**Purpose**
Statement for assigning the value of an expression to a variable.

**Syntax**

```
[LET]<nvar>=<nexp>|<svar>=<sexp>
```

- `<nvar>` is the numeric variable to which a value will be assigned.
- `<nexp>` is the numeric expression from which the value will be assigned to the numeric variable.

or...

- `<svar>` is the string variable to which the content of the string expression will be assigned.
- `<sexp>` is the string expression from which the content will be assigned to the string variable.

**Remarks**
The keyword LET is not necessary, but retained for compatibility with old versions of Intermec Fingerprint. The equal sign (=) is sufficient to make the assignment. Both the expression and the variable must be either string or numeric.

**Example**

```
10  LET A%=100          (numeric variable)
20  B%=150
30  LET C$="INTERMEC"  (string variable)
40  D$="THERMAL PRINTERS" (string variable)
50  PRINT A%+B%,C$+" "+D$  
RUN

yields:

250 INTERMEC THERMAL PRINTERS
```
**LINE INPUT**

**Purpose**
Statement for assigning an entire line, including punctuation marks, from the standard IN channel to a single string variable.

**Syntax**

```plaintext
LINE INPUT[<scon>;]<svar>
```

<**scon**>; is an optional prompt plus a semicolon
<svar> is the string variable to which the input line is assigned.

**Remarks**
For information on standard I/O channel, see SETSTDIO statement.
LINE INPUT differs from INPUT in that an entire line of max. 32,767 characters will be read. Possible commas will appear as punctuation marks in the string instead of dividing the line into portions.

During the execution of a program, a LINE INPUT statement will interrupt the execution. You can make a prompt being displayed on the host screen to notify the operator that the program is expecting additional data to be entered. The input is terminated and the program execution is resumed when a carriage return character (ASCII 13 decimal) is encountered. The carriage return character will not be included in the input line.

Note that LINE INPUT filters out any incoming ASCII 00 dec. characters (NUL).

**Example**
Print your own visiting card like this:

```plaintext
10  LINE INPUT "ENTER NAME: ";A$
20  LINE INPUT "ENTER STREET: ";B$
30  LINE INPUT "ENTER CITY: ";C$
40  LINE INPUT "ENTER STATE + ZIPCODE: ";D$
50  LINE INPUT "ENTER PHONE NO: ";E$
60  FONT "Swiss 721 BT", 8
70  ALIGN 5
80  PRPOS 160,300:PRTXT A$
90  PRPOS 160,250:PRTXT B$
100 PRPOS 160,200:PRTXT C$
110 PRPOS 160,150:PRTXT D$
120 PRPOS 160,100:PRTXT "Phone: "+E$
130 PRINTFEED
RUN
```
**LINE INPUT#**

**Purpose**
Statement for assigning an entire line, including punctuation marks, from a sequential file or a device to a single string variable.

**Syntax**

```
LINE <nexp>,<svar>
```

| `<nexp>` | is the number assigned to the file when it was OPENed. |
| `<svar>` | is the string variable to which the input line is assigned. |

**Remarks**

This statement differs from the INPUT# statement in that an entire line of max. 32,767 characters will be read, and possible commas in the line will be included in the string as punctuation marks instead of dividing it into portions.

When reading from a sequential file, the lines can be read one after the other by the repeated issuing of LINE INPUT# statements, using the same file reference.

Once a line has been read, it cannot be read again until the file is CLOSED and then OPENed again.

The LINE INPUT# statement is useful when the lines in a file has been broken into fields.

Note that LINE INPUT# filters out any incoming ASCII 00 dec. characters (NUL).

**Example**

This example assigns data from the three first lines of the file "Addresses" to the string variables A$, B$, and C$ respectively:

```
100 OPEN "ADDRESSES" FOR INPUT AS #5
110 LINE INPUT# 5, A$
120 LINE INPUT# 5, B$
130 LINE INPUT# 5, C$
. . . . .
```

LIST

Purpose
Statement for listing the current program completely or partially, or listing all variables, to the standard OUT channel.

Syntax
LIST([<ncon>,-[<ncont>]],V,B]

<ncon> is a single line, or the first line number in a range of lines.
<ncont> is optionally the last line number in a range of lines.
,V lists all variables.
,B lists all breakpoints.

Remarks
This instruction is useful after LOADing a program, or if you during programming have changed any program lines, renumbered the lines, or added new lines and want to bring some order in the presentation on the screen of the host. LIST also removes unnecessary characters and adds assumed keywords. The instruction is usually given in the immediate mode, that is on a line without any preceding line number.

The LIST statement can be used in seven different ways:

• If no line number is entered after LIST, the entire current program will be listed. In case the program has been written without line numbers (see IMMEDIATE ON/OFF statements), the lines will be automatically numbered with 10-step incrementation starting with line number 10 (10-20-30-40-50....).

• If a single line number is entered after LIST, only the specified line will be listed.

• If a line number followed by a hyphen (-) is entered after LIST, all lines from the specified line to the end of the program will be listed.

• If a hyphen (-) followed by line number is entered after LIST, all lines from the start of the program through the specified line will be listed.

• If two line numbers are entered after LIST, they will specify the first and last line in a range of lines to be listed.

• If LIST,V is entered, all integer variables, integer array variables, string variables, and string array variables in the printer’s memory will be listed.

• If LIST,B is entered, all breakpoints of the Fingerprint Debugger (see DBBREAK) will be printed in line number order. Line labels that have not been updated, which occurs at program execution, may be misplaced.
**LIST, cont.**

**Examples**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST</td>
<td>Lists all lines in the program.</td>
</tr>
<tr>
<td>LIST 100</td>
<td>Lists line No. 100 only.</td>
</tr>
<tr>
<td>LIST 100-</td>
<td>Lists all lines from line No 100 to the end of the program.</td>
</tr>
<tr>
<td>LIST -500</td>
<td>Lists all lines from the start of the program through line No. 500.</td>
</tr>
<tr>
<td>LIST 100-500</td>
<td>Lists all lines from line 100 through line 500.</td>
</tr>
<tr>
<td>LIST,V</td>
<td>Lists all variables.</td>
</tr>
<tr>
<td>LIST,B</td>
<td>Lists all breakpoints.</td>
</tr>
</tbody>
</table>
LISTPFSVAR

Purpose
Statement for listing variables saved at power failure.

Syntax
LISTPFSVAR

Remarks
Related instructions are SETPFSVAR, GETPFSVAR, and DELETPFSVAR.

Example
LISTPFSVAR

yields for example:

QS$
QCPS%
A%
LOAD

Purpose

Statement for loading a copy of a program, residing in the current directory or in another specified directory, into the printer's working memory.

Syntax

\texttt{LOAD<scon>}

\texttt{<scon>} is the program to be loaded into the working memory.

Remarks

If the program has the extension .PRG, the name of the program can be given with or without any extension. Otherwise, the extension must be included in the name. If the program resides in another directory than the current one (see CHDIR statement), the name must also contain a reference to the directory in question.

LOAD closes any open files and deletes all program lines and variables residing in the working memory before loading the specified program. If the previous program in the working memory has not been saved, see SAVE statement, it will be lost and cannot be retrieved.

While the program is loaded, a syntax check is performed. If a syntax error is detected, the loading will be interrupted and an error message will be transmitted on the standard OUT channel.

Examples

Load the program "LABEL127.PRG" from the current directory:
\texttt{LOAD "LABEL127"}

or
\texttt{LOAD "LABEL127.PRG"}

When “Ok” appears on the screen, the loading is completed. Use a LIST statement to display the program on the screen of your terminal.

You may also load a program stored in another directory than the current one, for example the read-only memory (/rom) or a CompactFlash memory card ("card1:"). Start the file name by specifying the directory, for example:
\texttt{LOAD "/rom/MKAUTO.PRG"}

or
\texttt{LOAD "card1:PROGRAM1.PRG"}

This will create a copy, which you can list or change and then save under a new name.
LOC

Purpose
Function returning the current position in an OPEN file or the status of
the buffers in an OPEN communication channel.

Syntax
LOC(<nexp>)

<nexp> is the number assigned to the file or communication channel
when it was OPENed.

Remarks
In a random file, LOC will return the number of the last record read or
written by the use of GET or PUT statements respectively.

In a sequential file, the number of 128-byte blocks, that have been read or
written since the file was OPENed, will be returned.

LOC can also be used to check the receive or transmit buffer of the speci-
fied communication channel:
• If the channel is OPENed for INPUT, the remaining number of charac-
ters (bytes) to be read from the receive buffer is returned.
• If the channel is OPENed for OUTPUT, the remaining free space
(bytes) in the transmit buffer is returned.

The number of bytes includes characters that will be MPIped as NUL.

Examples
This example closes the file "addresses" when record No. 100 has been read
from the file:
10 OPEN "ADDRESSES" FOR INPUT AS #1
....
....
....
200 IF LOC(1)=100 THEN CLOSE #1
....
....

This example reads the number of bytes which remains to be received from
the receive buffer of "uart2:"
100 OPEN "uart2:" FOR INPUT AS #2
110 A%=LOC(2)
120 PRINT A%
LOF

Purpose
Function returning the length in bytes of an OPEN sequential or random file, or returning the status of the buffers in an OPEN communication channel.

Syntax
LOF(<nexp>)

(<nexp>) is the number assigned to the file or communication channel when it was OPENed.

Remarks
LOF can also be used to check the receive or transmit buffer of the specified communication channel:
- If a channel is OPENed for INPUT, the remaining free space (bytes) in the receive buffer is returned.
- If a channel is OPENed for OUTPUT, the remaining number of characters to be transmitted from the transmit buffer is returned.

Examples
The first example illustrates how the length of the file "Pricelist" is returned:

10 OPEN "PRICELIST" AS #5
20 A%=LOF(5)
30 PRINT A%
. . . .
. . . .

The second example shows how the number of free bytes in the receive buffer of communication channel "uart2:" is calculated:

100 OPEN "uart2:" FOR INPUT AS #2
110 A%=LOF(2)
120 PRINT A%
Chapter 2—Program Instructions

LSET

Purpose

Statement for placing data left-justified into a field in a random file buffer.

Syntax

\[ \text{LSET}<\text{svar}>=\text{<sexp}> \]

- \text{<svar>} is the string variable assigned to the field by a FIELD statement.
- \text{<sexp>} holds the input data.

Remarks

After having OPENed a file and formatted it using a FIELD statement, you can enter data into the random file buffer using the LSET and RSET statements (RSET right-justifies the data).

The input data can only be stored in the buffer as string expressions. Therefore, a numeric expression must be converted to string format by the use of an \text{STR$} function before an LSET or RSET statement is executed.

If the length of the input data is less than the length of the field, the data will be left justified and the remaining number of bytes will be printed as space characters.

If the length of the input data exceeds the length of the field, the input data will be truncated on the right side.

Example

```
10  OPEN "PHONELIST" AS #8 LEN=26
20  FIELD#8,8 AS F1$, 8 AS F2$, 10 AS F3$
30  SNAME$="SMITH"
40  CNAME$="JOHN"
50  PHONE$="12345630"
60  LSET F1$=SNAME$
70  LSET F2$=CNAME$
80  RSET F3$=PHONE$
90  PUT #8,1
100 CLOSE#8
RUN
SAVE "PROGRAM 1.PRG"
NEW
10  OPEN "PHONELIST" AS #8 LEN=26
20  FIELD#8,8 AS F1$, 8 AS F2$, 10 AS F3$
30  GET #8,1
40  PRINT F1$,F2$,F3$
RUN      
```

yields:

SMITH — — — JOHN — — — — — — 12345630
**LTS& ON/OFF**

**Purpose**
Statement for enabling or disabling the label taken sensor.

**Syntax**

```
LTS& ON|OFF
```

Default: LTS& OFF

**Remarks**
The label taken sensor (LTS) is a photoelectric device that can be fitted in the vicinity of the printer’s label outfeed slot and detects if a printed label or ticket has been removed or not. (Usually, a self-adhesive label is not fed out completely, but will remain partly stuck to the liner so it will not fall off.)

Using the LTS ON statement, you can order the printer to stop the execution at next PRINTFEED statement until the LTS no longer detects any label. Then the PRINTFEED is executed. This is most useful when printing batches of labels or tickets. As soon as a label is taken, the next one is printed and awaits being taken care of.

The same result can also be obtained in a more cumbersome way by a program based on the PRSTAT(2) function.

LTS& OFF revokes LTS& ON.

**Example**

```
10 LTS& ON
20 FOR A%=1 TO 5
30 B$=STR$(A%)
40 FONT "Swiss 721 BT"
50 PRPOS 200,200
60 PRTXT B$
70 PRINTFEED
80 NEXT
RUN
```
Chapter 2—Program Instructions

MAG

Purpose
Statement for magnifying a font, barfont, or image up to four times separately in regard of height and width.

Syntax

\[ \text{MAG<nexp_1>,<nexp_2>} \]

- \(<nexp_1>\) is the magnification in regard of height (1, 2, 3, or 4).
- \(<nexp_2>\) is the magnification in regard of width (1, 2, 3, or 4).

Default value: 1,1
Reset to default by: PRINTFEED execution

Remarks
Magnification makes the object grow in directions away from the selected anchor point, see ALIGN statement.

The MAG statement has become more or less obsolete for fonts and bar fonts with the implementation of scaleable fonts. Even if MAG works for such fonts, the printout quality will be much better by using a larger font size rather than magnifying a smaller one. However, the MAG statement is retained to allow compatibility with programs originally written for older Intermec Fingerprint versions.

The MAG statement also works with images. However, since the MAG statement simply enlarges the bitmap pattern of an image, it gives a better printout quality to download and use a larger version of an image rather than magnifying a smaller one.

Note that the MAG statement cannot be used for bar code patterns (use BARHEIGHT and BARMAG statement for that purpose).

Example
This example illustrates how the image "GLOBE.1" is printed both with its original size and magnified 4 times. Note the jagged edges of the curves in the enlarged image.

```
10 ALIGN 2
20 PRPOS 300,50
30 FONT "Swiss 721 BT"
40 PRTXT "Normal Size"
50 PRPOS 300,125
60 PRIMAGE "GLOBE.1"
70 PRPOS 300,300
80 PRTXT "Enlarged 4X"
90 PRPOS 300,375
100 MAG 4,4
110 PRIMAGE "GLOBE.1"
120 PRINTFEED
RUN
```
MAKEASSOC

Purpose
Statement for creating an association.

Syntax
\[
\text{MAKEASSOC} \ <\text{sexp}_1>, <\text{sexp}_2>, <\text{sexp}_3> \\
\]
\(<\text{sexp}_1>\) specifies the name of the association to be created (case-sensitive).
\(<\text{sexp}_2>\) contains an argument list of parameter tuples according to the convention in \(<\text{sexp}_3>\).
\(<\text{sexp}_3>\) should always be "HTTP" (case sensitive).

Remarks
HTTP implies that the argument list in \(<\text{sexp}_2>\) is encoded in "x-www-url-encoding."

Example
This example shows how a string, including three stringnames associated with three start values, will be defined and one of them (time) will be changed:

```plaintext
10  QUERYSTRING$ = "time=UNKNOWN&label=321&desc=DEF"
20  MAKEASSOC "QARRAY", QUERYSTRING$, "HTTP"
30  QTIME$ = GETASSOC$("QARRAY", "time")
40  QLABELS% = VAL(GETASSOC$("QARRAY","label"))
50  QDESC$ = GETASSOC$("QARRAY", "desc")
60  PRINT "time=";QTIME$, "LABEL=";QLABELS%, "DESCRIPTION=";QDESC$
70  SETASSOC "QARRAY", "time", time$
80  PRINT "time="; GETASSOC$("QARRAY", "time")
RUN
```

yields:

time=UNKNOWN LABEL=321 DESCRIPTION=DEF

time=153355
MAP

Purpose
Statement for changing the ASCII value of a character when received on the standard IN channel, or optionally on another specified communication channel.

Syntax
MAP\[<nexp_1>,<nexp_2>,<nexp_3>\]

\(<nexp_1>\) optionally specifies a communication channel:
0 = "console:"
1 = "uart1:"
2 = "uart2:"
3 = "uart3:"
4 = "centronics:"
5 = "net1:"
6 = "usb1:"
7 = "uart4:"
8 = "uart5:"

Default: Standard IN channel.

<\(nexp_2>\) is the original ASCII decimal value.
<\(nexp_3>\) is the new ASCII decimal value after mapping.

Remarks
This statement is used to modify a character set (see NASC and NASCD statements) or to filter out undesired character. If you for example want a “Q” (ASCII 81 dec.) to be printed as the letter “Z” (ASCII 90 dec.), the MAP statement should be entered as: MAP 81, 90

The mapping interprets any ASCII 81 dec. value received on the standard IN channel as ASCII 90 dec., that is when you press “Q” on the keyboard of the host, the character “Z” will be printed (see note). However, pressing “Z” will still produce a “Z”, because that character has not been remapped.

To reset the mapping performed above, map the character back to its original ASCII value like this: MAP 81, 81

When a character is received by the printer, it is processed in regard of possible MAP statements before it “enters” the Intermec Fingerprint firmware. That allows you to filter out undesired control characters, which may confuse the Intermec Fingerprint firmware, for example by mapping them as NUL (ASCII 0 decimal).

After processing, the selected character set (see NASC and NASCD statements) controls how characters will be printed or displayed. If none of the character sets meets your demands completely, use MAP statements to modify the set that comes closest. Note that MAP statements will be processed before any COMSET or ON KEY..GOSUB strings are checked. NASC and NASCD statements will be processed last.
**Chapter 2—Program Instructions**

**MAP, cont.**

Do not map any characters to ASCII values occupied by characters used in Fingerprint instructions, for example keywords, operators, %, $, #, and certain punctuation marks. Mapping will be reset to normal at power-up or reboot.

**Examples**

You can check what characters the host produces using a simple program. Pressing different keys on the host should produce the corresponding characters both on the label and on the screen of the host. If not, try another character set (see NASC). In this example we presume that the keyboard produces ASCII 81 dec. and ASCII 90 dec. when you press the Q and Z keys respectively. Should any unexpected characters be printed on the labels or the screen, check the manuals of the host for information on what ASCII values will be produced by the various keys and how the screen will present various ASCII values received from the printer.

```
10 FONT "Swiss 721 BT"
20 PRPOS 30,100
30 INPUT "Enter character";A$
40 PRTXT A$
50 PRINTFEED
```

By adding a MAP statement in line 5, you can test what happens. In this case we remap the character Q to be printed as Z, as in the explanation on the previous page. After printing, we map the character Q back to its original position.

```
5  MAP 81,90
10 FONT "Swiss 721 BT"
20 PRPOS 30,100
30 INPUT "Enter character";A$
40 PRTXT A$
50 PRINTFEED
60 MAP 81,81
```

Assume that a device connected to "uart2:" produces strings always starting with the control character STX (ASCII 2 decimal). STX can be filtered out by mapping it as NUL (ASCII 0 decimal):

```
10 MAP 2,2,0
```

Should "uart2:" be appointed standard IN channel (see SETSTDIO), the first parameter can be omitted from the example above:

```
10 MAP 2,0
```
**MERGE**

**Purpose**
Statement for merging a program in the printer’s current directory, or optionally in another specified directory, with the program currently residing in the printer’s working memory.

**Syntax**

```
MERGE<scon>
```

- `<scon>` is the name (optionally including a reference to another directory than the current one) of the program, which is to be merged with the program currently residing in the printer’s working memory.

**Remarks**
MERGE creates a copy of a program stored in the current directory (see CHDIR statement), or optionally in a specified other directory, and blends its lines into the program currently residing in the printer’s working memory.

If there are lines with the same numbers in both programs, the lines in the program currently residing in the working memory will be replaced by the corresponding lines in the MERGED program. This also applies to programs written without line numbers, since they will automatically be assigned hidden line numbers (10-20-30... etc.) at the execution of the IMMEDIATE ON statement. In order to avoid overwriting any lines, you may SAVE a program without line numbers using a SAVE `<scon>`, L statement. When MERGED, it will be appended to the current program and assigned line numbers that start with the number of the last line of the current program plus 10. For safety reasons, a backup copy of the current program is recommended before issuing a MERGE statement.

MERGE makes it possible to store blocks of program instructions, which are frequently used, and include them into new programs. The printer’s ROM memory contains a number of useful programs, which also can be MERGED into programs of your own creation.

**Examples**
The program “XYZ.PRG” will be merged with the current program. If there are identical line numbers in both programs, the lines from “XYZ.PRG” will replace those in the current program.
```
MERGE "XYZ.PRG"          (from current directory)
MERGE "/c/XYZ.PRG"        (from permanent memory)
MERGE "tmp:XYZ.PRG"      (from temporary memory)
MERGE "card1:XYZ.PRG"    (from memory card)
```
### MIBVAR&

**Purpose**
Statement to read or write SNMP MIB variables.

**Syntax**

```
MIBVAR&(<nexp>)
```

<nexp> is the number of the variable to access. Allowed values: 0-9.

**Remarks**

SNMP (Simple Netowrk Management Protocol) MIB (Management Information Base) variables are normally used for network management tasks. These variables can be set from the outside through SNMP, and are available to Fingerprint programs through the MIBVAR& command. The SNMP variables must be of string type (maximum 128 characters) and are placed under the SNMP node: enterprises.1963.20.15.21.5.1.1., indexes 0-9. The public community is read-only whereas the “pass” community is read-write. There are no read or write restrictions from Fingerprint.

Related instruction is ON MIBVAR& GOSUB.

**Example**

```
MIBVAR&(1) = "ON"
A$ = MIBVAR&(1)
PRINT A$
```

yields:

ON
**MID$**

**Purpose**
Function returning a specified part of a string.

**Syntax**
```
MID$(<sexp>,<nexp1>[,<nexp2>])
```

- `<sexp>` is the original string.
- `<nexp1>` is the start position in the original string.
- `[,<nexp2>]` is the number of characters to be returned (optional).

**Remarks**
- `<sexp>` is the original string from which a specified part is to be returned.
- `<nexp1>` specifies which character position in the original string is to be the first character in the part to be returned.
- `<nexp2>` restricts the number of characters to be returned. This information is optional. If omitted, all characters from the start position specified by `<nexp1>` to the end of the string will be returned.
- If the value of `<nexp1>` is less than or equal to zero, then Error 44, “Parameter out of range” will occur.
- If the value of `<nexp2>` is less than zero, then Error 44, “Parameter out of range” will occur.
- If the value of `<nexp1>` exceeds the length of the original string, an empty string will be returned, but no error condition will occur.
- If the value of `<nexp1>` does not exceed the length of the original string, but the sum of `<nexp1>` and `<nexp2>` exceeds the length of the original string, the remainder of the original string will be returned.

**Examples**
```
10  A$=MID$("INTERMEC PRINTERS", 6, 3)
20  PRINT A$
RUN                                   yields:
MEC

10  A$="INTERMEC PRINTERS"
20  B%=10
30  C%=7
40  D$=MID$(A$, B%, C%)
50  PRINT D$
RUN                                   yields:
PRINTER
```
**MKDIR**

**Purpose**
Statement for creating a directory.

**Syntax**

```
MKDIR <sexp>
```

<sexp> specifies the directory to be created.

**Remarks**
<sexp> can end with a slash (/) character, but it is not necessary. Only the device /c (or "c:" ) supports creating directories.

**Example**

```
NEW
MKDIR "DIR1"
SAVE "DIR1/PROGRAM.PRG"
FILES "/c/DIR1"
```

yields:

```
FILES on /c/DIR1
PROGRAM.PRG 2
2220032 bytes free 2 bytes used
```
NAME DATE$

Purpose

Statement for formatting the month parameter in return strings of DATE$("F") and DATEADD$(...,"F").

Syntax

```
NAME DATE$ <nexp>, <sexp>
```

- `<nexp>` is the month number (1-12).
- `<sexp>` is the desired name of the month.

Remarks

This statement allows you to assign names to the different months in any form and language you like. The names will be returned instead of the corresponding numbers in connection with DATE$("F") and DATEADD$("F") instructions, provided that a FORMAT DATE$ statement has been executed.

The number of characters assigned to represent months in the FORMAT DATE$ statement decides how much of the names, as specified in the NAME DATE$ statement, will be returned. The names will be truncated at the left side. For example:

```
FORMAT DATE$ "YY.MMM:DD"
NAME DATE$ 1,"JANUARY"
PRINT DATE$("F")
```

```
yields for example:
03.ARY.06
```

Usually, it is best to restrict the month parameter in the FORMAT DATE$ statement to 2 or 3 characters (MM or MMM) and enter the names of the months in the NAME DATE$ statement accordingly.

Example

This example shows how to make the printer return dates in accordance with British standard:

```
10  DATE$="030115"
20  NAME DATE$ 1, "JAN"
30  NAME DATE$ 2, "FEB"
40  NAME DATE$ 3, "MAR"
50  NAME DATE$ 4, "APR"
60  NAME DATE$ 5, "MAY"
70  NAME DATE$ 6, "JUN"
80  NAME DATE$ 7, "JUL"
. . . . .
140  FORMAT DATE$ "MMM DD, YYYY"
150  PRINT DATE$("F")
RUN
```

```
yields:
JAN 15, 2003
```
NAME WEEKDAY$

Purpose
Statement for formatting the day parameter in return strings of WEEKDAY$.

Syntax
```
NAME WEEKDAY$ <nexp>, <sexp>
```
-
- `<nexp>` is the number of the weekday according to the WEEKDAY$ function syntax (Monday = 1... Sunday = 7).
- `<sexp>` is the desired name of the weekday.
  Default: Full English name in lowercase characters, that is Monday, Tuesday, etc.

Remarks
This statement allows you to assign names to the different weekdays in any form and language you like. The names will be returned instead of the corresponding numbers in connection with WEEKDAY$ function.

Example
This example shows how to make the printer return the name of the weekday as an English 3-letter abbreviation:
```
10  FORMAT DATE$ "", MM/DD/YY"
20  DATE$="031201"
30  NAME WEEKDAY$ 1, "Mon"
40  NAME WEEKDAY$ 2, "Tue"
50  NAME WEEKDAY$ 3, "Wed"
60  NAME WEEKDAY$ 4, "Thu"
70  NAME WEEKDAY$ 5, "Fri"
80  NAME WEEKDAY$ 6, "Sat"
90  NAME WEEKDAY$ 7, "Sun"
100 PRINT WEEKDAY$ (DATE$) + DATE$("F")
RUN
```
yields:
Fri, 12/01/03
NASC

Purpose

Statement for selecting a single-byte character set, alternatively the multi-byte character set UTF-8.

Syntax

```
NASC<nexp>|<sexp>
```

- `<nexp>` is the reference number of a character set:
  - 1 = Roman 8 (default)
  - 8 = UTF-8
  - 33 = French
  - 34 = Spanish
  - 39 = Italian
  - 44 = English (UK)
  - 46 = Swedish
  - 47 = Norwegian
  - 49 = German
  - 81 = Japanese Latin (romaji)
  - 351 = Portuguese
  - -1 = PCMAP
  - -2 = ANSI (same as 1252)
  - 850 = MS-DOS Latin 1
  - 851 = MS-DOS Greek 1
  - 852 = MS-DOS Latin 2
  - 855 = MS-DOS Cyrillic
  - 857 = MS-DOS Turkish
  - 1250 = Windows Latin 2 (Central Europe)
  - 1251 = Windows Cyrillic (Slavic)
  - 1252 = Windows Latin 1 (ANSI, same as -2)
  - 1253 = Windows Greek
  - 1254 = Windows Latin 5 (Turkish)
  - 1257 = Windows Baltic Rim

- `<sexp>` “UTF-8” = UTF-8
  “file name” = NSC file character set.

Remarks

Please refer to Chapter 4 for complete character set tables.

By default, after processing of possible MAP statements, the Intermec Fingerprint firmware will print and, when applicable, display all characters according to the Roman 8 character set. However, the Intermec Fingerprint firmware contains a number of other character sets, which allows you to print and display such characters that are characteristic for a number of countries or language areas, or to adapt the printer for the operating system of the host.

That implies that a certain ASCII code received by the printer may result in a different character is printed or displayed depending on which character set has been selected.

If none of the character sets available contains the desired character(s), use a MAP statement to reMAP the character set that comes closest to your needs. Note that MAP statements are processed before NASC statements.
NASC, cont.

You can also specify the multi-byte character set UTF-8. For more information on using UTF-8, see Chapter 4.

A NASC statement will have the following consequences:

Font
The font used will be the one last specified by the FONT command. If none has been specified, the default (Swiss 721 BT) will be used.

Text printing
Text on labels etc. will be printed according to the selected character set. However, parts of the label, that already has been processed and stored in the print buffer before the NASC statement is executed, will not be affected. This implies that labels may be multi-lingual.

LCD Display
New messages in the display will be affected by a NASC statement. However, a message that is already displayed will not be updated automatically. The display is, for all practical reasons, able to show all printable characters. In the Setup Mode, all characters are mapped according to US-ASCII standard.

Communication
Data transmitted via any of the communication channels will not be affected as the data is defined as ASCII values, not as alphanumeric characters. The active character set of the receiving unit will decide the graphic presentation of the input data, for example the screen of the host.

Bar Code Printing
The pattern of the bars reflects the ASCII values of the input data and is not affected by a NASC statement. The bar code interpretation (the human readable characters below the bar pattern) is affected by a NASC statement. However, the interpretation of bar codes, that have been processed and are stored in the print buffer, will not be affected.

Example

This example selects the Italian character set, prints the character corresponding to 123 dec. in that set, changes the set to Swedish, and prints the character corresponding to 123 dec.:

```
10  NASC 39
20  PRTXT CHR$(123)
30  NASC 46
40  PRTXT CHR$(123)
50  PRINTFEED
```

yields:

àå
NASCD

Purpose

Statement for selecting a double-byte character set, alternatively the multi-byte character set UTF-8.

Syntax

```
NASCD <sexp>
```

<sexp> is the name of the character set or the file containing the character map.
Default: "" (disables double-byte interpretation).

Remarks

When a double-byte character set has been selected, the firmware will usually treat all characters from ASCII 161 dec. to ASCII 254 dec (ASCII A1-FE hex) as the first part of a two-byte character. Next character byte received will specify the second part. However, the selected double-byte character set may specify some other ASCII value as the breaking point between single and double byte character sets.

There are various ways to produce double-byte characters from the keyboard of the computer. By selecting the proper character set using a NASCD statement, the typed-in ASCII values will be translated to the corresponding Unicode values, so the desired glyph will be printed. A font must have been selected using the FONTD command in order to print according to the selected double-byte character set.

UTF-8 character mapping is pre-installed in the firmware, and is selected with the command NASCD "UTF-8" (can also be selected with NASC). See Chapter 4, UTF-8, for more information. Other double-byte character set tables and fonts are available from Intermec on special request.

Example

The following text contains both single- and double-byte fonts. The double-byte font and its character set are stored in a Font Install Card:

```
10 NASC 46
20 FONT "Swiss 721 BT", 24, 10
30 FONTD "Song"
40 NASCD "card1:GB2312.NCD"
50 PRTXT CHR$(65);CHR$(164);CHR$(180)
60 PRINTFEED
RUN
```

This program yields a printed text line that starts with the Latin character A (ASCII 65 dec.) followed by the character in the Song font that corresponds to the address 164+180 dec. in the character set “GB2312.NCD.”
NEW

Purpose
Statement for clearing the printer’s working memory in order to allow a new program to be created.

Syntax
NEW

Remarks
The NEW statement will delete the program currently residing in the printer’s working memory, close all files, and clear all variables and breakpoints. If the current program has not been saved (see SAVE statement), it will be lost and cannot be restored.

In the Intermec Direct Protocol, all counters will be removed when a NEW statement is executed.

Note that clearing the printer’s working memory does not imply that the host screen will be cleared too. The lines of the previous program will remain on the screen until gradually being replaced by new lines.

Example
NEW
**NORIMAGE (NI)**

**Purpose**
Statement for returning to normal printing after an INVIMAGE statement has been issued.

**Syntax**

```
NORIMAGE|NI
```

**Remarks**
Normal image is the default type of printing and means that text and images will be printed in black-on-white.

Using an INVIMAGE statement, the printing of text and images can be inverted. Such inverse printing will be discontinued for all PRTXT and PRIMAGE statements that follows the encounter of a NORIMAGE statement.

**Example**
In this example, the first line is printed in inverted fashion and the second line in the normal fashion:

```
10  PRPOS 30,300
20  ALIGN 4
30  INVIMAGE
40  FONT "Swiss 721 BT"
50  PRTXT "INVERSE PRINTING"
60  PRPOS 30, 200
70  NORIMAGE
80  PRTXT "NORMAL PRINTING"
90  PRINTFEED
RUN
```
**ON BREAK GOSUB**

**Purpose**
Statement for branching to a subroutine, when break interrupt instruction is received.

**Syntax**

```
ON <nexp> BREAK <ncon>|<line label> GOSUB <ncon>|<line label>
```

- `<nexp>` is one of the following communication channels:
  - 0 = "console:"
  - 1 = "uart1:"
  - 2 = "uart2:"
  - 3 = "uart3:"
  - 4 = "centronics:"
  - 7 = "uart4:"
  - 8 = "uart5:"

- `<ncon>|<line label>` is the number or label of the program line to be branched to.

**Remarks**
This statement is closely related BREAK and BREAK ON/OFF. When break interrupt is enabled (see BREAK ON) and the operator issues a break interrupt instruction (see BREAK), the execution of the currently running program will be interrupted and branched to a specified line in a subroutine. Break from the console is not supported by the EasyCoder PD41.

**Examples**
In this example, the printer emits a special signal when a break interrupt is issued from the printer's keyboard:

```
10 ON BREAK 0 GOSUB 1000
20 GOTO 20
... ... ...
1000 FOR A%=1 TO 3
1010 SOUND 440,50
1020 SOUND 349,50
1030 NEXT A%
1040 END
```

The same example without line numbers will look like this:

```
IMMEDIATE OFF
ON BREAK 0 GOSUB QQQ
WWW: GOTO WWW
... ... ...
QQQ: FOR A%=1 TO 3
SOUND 440,50
SOUND 349,50
NEXT A%
END
IMMEDIATE ON
```
ON COMSET GOSUB

Purpose
Statement for branching to a subroutine, when the background reception of data on the specified communication channel is interrupted.

Syntax
\[
\text{ON} \leftrightarrow \text{COMSET}<\text{nexp}_1>\text{GOSUB}<\text{nexp}_2>|<\text{line label}>
\]

\(<\text{nexp}_1>\) is one of the following communication channels:
0 = "console:"
1 = "uart1:"
2 = "uart2:"
3 = "uart3:"
4 = "centronics:"
6 = "usb1:"
7 = "uart4:"
8 = "uart5:"

\(<\text{nexp}_2>\)/<line label> is number or label of the program line to be branched to.

Remarks
This statement is closely related to COMSET, COMSTAT, COMSET ON, COMSET OFF, COM ERROR ON/OFF, and COMBUF$. It is used to branch to a subroutine when one of the following conditions occur:
- End character is received.
- Attention string received.
- Max. number of characters received.

These three parameters are set for the specified communication channel by a COMSET statement.

Examples
In this example, the program branches to a subroutine for reading the buffer of the communication channel:

1 REM Exit program with #STOP&
10 COMSET1,"#","&","ZYX","=".,50
20 ON COMSET 1 GOSUB 2000
30 COMSET 1 ON
40 IF A$ <> "STOP" THEN GOTO 40
50 COMSET 1 OFF
.....
1000 END
2000 A$= COMBUF$(1)
2010 PRINT A$
2020 COMSET 1 ON
2030 RETURN
ON COMSET GOSUB, cont.

The same example written without line numbers would look like this:

```basic
IMMEDIATE OFF
REM Exit program with #STOP&
COMSET1, "#", ",", "ZYX", ",", 50
ON COMSET 1 GOSUB QQQ
COMSET 1 ON
WWW: IF A$ <> "STOP" THEN GOTO WWW
COMSET 1 OFF
.
.
.
END
QQQ: A$ = COMBUF$(1)
PRINT A$
COMSET 1 ON
RETURN
IMMEDIATE ON
```
ON ERROR GOTO

Purpose
Statement for branching to an error-handling subroutine when an error occurs.

Syntax
\[ \text{ON} \rightarrow \text{ERROR} \rightarrow \text{GOTO}<\text{ncon}>|<\text{line label}> \]

\(<\text{ncon}>\) is the number or label of the line to which the program should branch when an error condition occurs.

Remarks
If any kind of error condition occurs after this statement has been encountered, the standard error-trapping routine will be ignored and the program will branch to the specified line, which should be the first line in an error-handling subroutine.

If the line number is 0, the standard error-trapping routine will be enabled and no error-branching within the current program will be executed.

Examples
If you try to run this example with the printhead raised (or if any other error occurs), a warning signal will sound and the error LED will be lighted.

10 LED 0 ON:LED 1 OFF
20 ON ERROR GOTO 1000
30 FONT "Swiss 721 BT"
40 PRTXT "HELLO"
50 PRINTFEED
60 END

. . . . .
1000 LED 0 OFF:LED 1 ON
1010 FOR A%=1 TO 3
1020 SOUND 440,50
1030 SOUND 359,50
1040 NEXT A%
1050 RESUME NEXT

The same example written without line numbers would look like this:
IMMEDIATE OFF
LED 0 ON:LED 1 OFF
ON ERROR GOTO QQQ
FONT "Swiss 721 BT"
PRTXT "HELLO"
PRINTFEED
END

. . . . .
QQQ: LED 0 OFF:LED 1 ON
FOR A%=1 TO 3
SOUND 440,50
SOUND 359,50
NEXT A%
RESUME NEXT
IMMEDIATE ON
ON GOSUB

Purpose

Statement for conditional branching to one of several subroutines.

Syntax

ON<nexp>GOSUB<ncon>|<line label>[, <ncon>|<line label>…]

<nexp> is a numeric expression that determines which line the program should branch to.

<ncon>/<line label> is the number or label of the line, or list of lines, to which the program should branch.

Remarks

This statement is closely related to the ON GOTO statement. The numeric expression may result in any positive value. The expression is truncated to an integer value before the statement is executed. If the resulting value is negative, 0, or larger than the number of subroutines, the statement will be ignored.

The value of the numeric expression determines which of the subroutines the program should branch to. For example, if the the value of the numeric expression is 2, the program will branch to the second subroutine in the list.

Examples

In this example, different texts will be printed on the screen depending on which of the keys 1-3 you press on the keyboard of the host.

```
10 INPUT "PRESS KEY 1-3 ", A%
20 ON A% GOSUB 1000,2000,3000
30 END
1000 PRINT "You have pressed key 1"
1010 RETURN
2000 PRINT "You have pressed key 2"
2010 RETURN
3000 PRINT "You have pressed key 3"
3010 RETURN
```

The same example written without line numbers would look like this:

```
IMMEDIATE OFF
INPUT "PRESS KEY 1-3 ", A%
ON A% GOSUB QQQ,WWW,ZZZ
END
QQQ: PRINT "You have pressed key 1"
RETURN
WWW: PRINT "You have pressed key 2"
RETURN
ZZZ: PRINT "You have pressed key 3"
RETURN
IMMEDIATE ON
```
## ON GOTO

### Purpose
Statement for conditional branching to one of several lines.

### Syntax

\[ \text{ON}<\text{nexp}>\text{GOTO}<\text{ncon}>|<\text{line label}>[,<\text{ncon}>|<\text{line label}>...] \]

- \(<\text{nexp}>\) is a numeric expression that determines which line the program should branch to.
- \(<\text{ncon}>/\text{line label}>\) is the number or label of the line, or list of lines, to which the program should branch.

### Remarks
This statement is closely related to the ON GOSUB statement. The numeric expression may result in any positive value. The expression is truncated to an integer value before the statement is executed. If the resulting value is negative, 0, or larger than the number of lines, the statement will be ignored.

The value of the numeric expression determines which of the lines the program should branch to. For example, if the value of the numeric expression is 2, the program will branch to the second line in the list.

### Examples
In this example, different texts will be printed on the screen depending on which of the keys 1-3 you press on the keyboard of the host.

```
10 INPUT "PRESS KEY 1-3 ", A%
20 ON A% GOTO 1000,2000,3000
30 END
1000 PRINT "You have pressed key 1"
1010 GOTO 30
2000 PRINT "You have pressed key 2"
2010 GOTO 30
3000 PRINT "You have pressed key 3"
3010 GOTO 30
```

The same example written without line numbers would look like this:

```
IMMEDIATE OFF
INPUT "PRESS KEY 1-3 ", A%
ON A% GOSUB QQQ,WWW,ZZZ
YYY: END
QQQ: PRINT "You have pressed key 1"
GOTO YYY
WWW: PRINT "You have pressed key 2"
GOTO YYY
ZZZ: PRINT "You have pressed key 3"
GOTO YYY
IMMEDIATE ON
```
**ON HTTP GOTO**

**Purpose**
Statement for branching to a subroutine when a request for an application CGI is received.

**Syntax**

```
ON HTTP GOTO<ncon>|<line label>
```

<ncon>/<line label> is the number or label of the line to which the program will branch when the CGI request is received.

**Remarks**
This statement is used in connection with EasyLAN 100i and defines a Fingerprint subroutine that handles the CGI-request. Setting the handler’s line number or line label to 0 disables the handler.

When a request for an application CGI is received, the current execution point will be pushed on to the stack and then the execution will commence in the handler with stdin and stdout redirected from/to the Web browser.

Related instruction: RESUME HTTP.
ON KEY GOSUB

Purpose
Statement for branching to a subroutine when a specified key on the printer's front panel is activated.

Syntax
\[
\text{ON KEY}(<nexp>)\text{GOSUB}<ncon>|<line label>
\]

\(<nexp>\)
the id. number of one of the keys on the printer's front panel (see illustration below).

\(<ncon>/<line label>\)
the number or label of the line to which the program will branch when the specified key is pressed down.

Remarks
Intermec Fingerprint-compatible printer models are fitted with a membrane-switch keyboard (not EasyCoder PD4). Each key can be enabled individually using its id. number in a KEY ON statement. Then the key can be assigned, alone or in combination with the \(<\text{Shift}>\) key, to make the program branch to a subroutine using an ON KEY... GOSUB statement. The \(<\text{Shift}>\) key adds 100 to the unshifted id. number of each key. See Appendix A for illustrations of the id. numbers of the keyboards for the various printer models.

The EasyCoder PD41 works in the same way, but has only one key with id. number 17 when pressed, and 117 when released.

Note the difference between the id. numbers of the keys and the ASCII values they are able to produce (see for example BREAK).

Note that BREAK takes precedence over any ON KEY statement, provided that break interrupt is not disabled for the "console:" by a BREAK 0 OFF statement.

Examples
This example illustrates how activating the \(<\text{F1}>\) key (id. No. 10) will make the program branch to a subroutine, which contains the PRINTFEED statement. Note line 30 where the execution will wait for the key to be pressed.

```
10   ON KEY (10) GOSUB 1000
20   KEY (10) ON
30   GOTO 30
....
....
....
1000  FONT "Swiss 721 BT"
1010  PRPOS 30,100
1020  PRTXT "HELLO"
1030  PRINTFEED
1040  END
RUN
```
ON KEY GOSUB, cont.

The same example can be written without line numbers this way:

```plaintext
IMMEDIATE OFF
ON KEY (10) GOSUB QQQ
KEY (10) ON
WWW: GOTO WWW
.......
.......
.......

QQQ: FONT "Swiss 721 BT"
PRPOS 30,100
PRTXT "HELLO"
PRINTFEED
END
IMMEDIATE ON
RUN
```
Chapter 2—Program Instructions

ON/OFF LINE

Purpose
Statement controlling the SELECT signal on the "centronics:" communication channel.

Syntax

\[ \text{ON|OFF \ LINE<nexp>} \]

\(<nexp>\) specifies the communication channel:
- 4 = "centronics:"
- 6 = "usb1:"

Remarks
Pin 13 in the Centronics/IEEE 1284 interface connector contains the SELECT signal:
- ON LINE 4 sets the SELECT signal high.
- OFF LINE 4 sets the SELECT signal low.

If no ON/OFF LINE statement is issued, the SELECT signal will be high, that is the Centronics channel will be ON LINE.

ON LINE/OFF LINE for the serial channel "usb1:" is implemented according to USB Device Class for Printing Devices v1.09, January 2000.

Example
In this example, the "centronics:" communication channel is disabled, while a new setup is performed on the printer by means of a setup file, and then enabled:

10 OFF LINE 4
20 SETUP "New Setup.SYS"
30 ON LINE 4

...
ON MIBVAR& GOSUB

Purpose
Statement for branching to a subroutine when a specified SNMP MIB variable is changed.

Syntax
ON MIBVAR&(<nexp>)GOSUB<ncon>|<line label>

<nexp> is the number of a MIBVAR& variable. Allowed values 0-9.
<ncon>/<line label> is the number or label of the line to which the program will branch when the specified MIB variable is modified.

Remarks
This statement makes the Fingerprint program branch to a subroutine whenever the specified MIB variable is modified. This values is then maintained during the entire subroutine execution. External changes during the subroutine execution will cause a new subroutine call at return. Related instruction is MIBVAR&.

Example
This example will print the value of MIBVAR&(1) on a label every time its value is changed. Note that the program is idle (on line 20) as long as the MIBVAR& variable remains unchanged.

```
10 ON MIBVAR&(1) GOSUB 1000
20 GOTO 20
       .......
1000 FONT "Swiss 721 BT"
1010 PRPOS 30,100
1020 A$ = MIBVAR&(1)
1030 PRTXT A$
1040 PRINTFEED
1050 RETURN
RUN
```
OPEN

Purpose
Statement for opening a file, device (for example a network connection)—or creating a new file—for input, output, or append, allocating a buffer, and specifying the mode of access.

Syntax
\[
\text{OPEN}\langle\text{sexp}\rangle[\text{FOR }↔\langle\text{INPUT}|\text{OUTPUT}|\text{APPEND}\rangle↔]\text{AS }[#]\langle\text{nexp}_1\rangle\\[\text{LEN=}\langle\text{nexp}_2\rangle]\\
\]

- \langle\text{sexp}\rangle: is the file or device to be opened, or the file to be created. File names must not contain any colon character (:). A network connection can be specified.
- #: indicates that whatever follows is a number. Optional.
- \langle\text{nexp}_1\rangle: is a designation number for the OPENed file or device.
- \langle\text{nexp}_2\rangle: is, optionally, the length of the record in bytes (default 128 bytes).

Remarks
An OPEN statement must be executed before a file or device can be used for input, output, and/or append. A maximum of 25 files and/or devices can be open at the same time.

Sequential Access Mode
The access mode can optionally be specified as sequential INPUT, OUTPUT, or APPEND:
- INPUT: Sequential input from the file/device, replacing existing data. Existing files/devices only.
- OUTPUT: Sequential output to the file/device, replacing existing data.
- APPEND: Sequential output to the file/device, where new data will be appended without replacing existing data.

Random Access Mode
If no access mode is specified in the statement, the file/device is opened for both input and output (RANDOM access mode). FIELD, LSET, RSET, PUT, and GET can only be used on records in files OPENed in the RANDOM access mode.

Please refer to the DEVICES statement for information on which devices can be opened for the different modes of access.

Lists of the files stored in the various parts of your printer’s memory can be obtained by the use of the FILES statements.
OPEN, cont.

Network connections
Special syntax can be used for the <sexp> expression to open the Net1 device. The syntax used is:

```
OPEN "net1:<host>[,:port][,:timeout]" AS #<nexp>
```

The <host> parameter is an IP address, alternatively a DNS name. The default port number is 9100, allowed values range from 0 to 65535. The timeout is expressed in ticks, range 10-6000, default about 4500 (45 seconds). Only one Net1 connection can be opened at a time. Once the device is open, information can be sent or received according to the corresponding protocol.

Examples

Open and write to a file:

```
10 OPEN "TEST.TXT" FOR OUTPUT AS 1
20 PRINT#1:"AAAA"
30 PRINT#1,"1234"
40 CLOSE 1;
```

Open the file and print it line by line:

```
10 OPEN "TEST.TXT" FOR INPUT AS #2
20 INPUT#2:A$
30 INPUT#2,B$
40 CLOSE#2;
50 PRINT A$
60 PRINT B$
RUN
```

You can allow sequential output to the printer's display using the OPEN statement this way. The method is the same as for files:

```
10 OPEN "console:" FOR OUTPUT AS #1
20 PRINT#1:PRINT#1
30 PRINT#1, "GONE TO LUNCH"
40 PRINT#1, "BACK SOON";
RUN
```

The text will appear on the printer's display as:

```
GONE TO LUNCH
BACK SOON
```

Open the file "PRICELIST" for random access with the reference number #8 and a record length of 254 bytes:

```
10 OPEN "PRICELIST" AS #8 LEN=254
```
The following example opens the Net1 device to connect to the NIST Internet Time Service (ITS):

```
10    OPEN "net1:time.nist.gov:13" AS 1
20    PRINT #1, "time?"
30    QTICKS% = TICKS
40    WHILE TICKS-QTICKS% < 100
50    LINE INPUT #1, A$
60    IF A$<>"" THEN PRINT A$
70    WEND
80    CLOSE 1
```

In the example above, socket port 13 is specified in row 10. Rows 30-40 create a delay to allow the result to be received as A$ within the specified and reasonable amount of time, 100 ticks.
**OPTIMIZE BATCH ON/OFF**

**Purpose**
Statement for enabling/disabling optimizing for batch printing.

**Syntax**

```
OPTIMIZE "BATCH"_ON|OFF
```

<table>
<thead>
<tr>
<th>ON</th>
<th>OFF</th>
<th>enables/disables optimizing respectively.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>Disabled(OFF), Enabled(ON) in Direct Protocol</td>
<td></td>
</tr>
</tbody>
</table>

**Remarks**
This facility is intended to speed up batch printing, which means the uninterrupted printing of large numbers of identical or very similar labels.

The program execution will not wait for the printing of the label to be completed, but proceeds executing next label image into the other of the two image buffers as soon as possible.

OPTIMIZE BATCH is automatically enabled (ON) during a batch (PRINTFEED more than one) if the following conditions are fulfilled:
1. LTS& OFF (default)
2. CUT OFF (default)

In Intermec Direct Protocol, the default value is enabled (ON).

**Examples**
Run these two examples and watch the differences in the printer’s performance:

```plaintext
10  OPTIMIZE "BATCH" ON
20  FOR I%=1 TO 10
30  PRTTXT I%
40  PRINT "Before printfeed"
50  PRINTFEED
60  PRINT "After printfeed"
70  NEXT
RUN
```

```plaintext
10  OPTIMIZE "BATCH" OFF
20  FOR I%=1 TO 10
30  PRTTXT I%
40  PRINT "Before printfeed"
50  PRINTFEED
60  PRINT "After printfeed"
70  NEXT
RUN
```
PORTIN

Purpose
Function reading the status of a port on a Serial/Industrial Interface Board.

Syntax

\[
\text{PORTIN}(<\text{nexp}>)
\]

Where:
- \(<\text{nexp}>\) is the number of the port to be read:
  - IN ports (optical): 101-108 (301-308)
  - OUT ports (relay): 201-204 (401-404)
  - OUT ports (optical): 221-228 (421-428)

Remarks
This function works with the Serial/Industrial Interface Board and is able to read the status of 8 IN ports with optocouplers, 8 OUT ports with optocouplers, and 4 OUT ports with relays. For information on how to set the OUT ports, please refer to the PORTOUT statement.

A current can be lead through an optocoupler in each IN port:
- If the current is on, the PORTIN function returns the value -1 (true).
- If the current is off, the PORTIN function returns the value 0 (false).

This feature is intended to allow the execution of the Intermec Fingerprint to be controlled by various types of external sensors or non-digital switches.

The status of the OUT ports, as set by PORTOUT statements, can also be read by PORTIN functions.

Some printers, like EasyCoder PM4i can carry two Serial/Industrial Interface boards. In this case, the ports on the inner board (that is, the board closest to the CPU board) are specified by the low numbers (101-108, 201-204, and 221-228) while the ports on the outer board are specified by the high numbers (301-308, 401-404, and 421-428).

Please refer to the documentation of the Serial/Industrial Interface Board for more information.

Example
The status of IN port 101 on a Serial/Industrial Interface Board decides when a label is to be printed. The printing will be held until the current is switched off:

\[
\begin{align*}
10 & \quad \text{FONT} \ "\text{Swiss 721 BT}"\\
20 & \quad \text{PRTXT} \ "\text{POWER IS OFF}"\\
30 & \quad \text{IF} \ \text{PORTIN} \ (101) \ \text{THEN} \ \text{GOTO} \ 30\\n40 & \quad \text{PRINTFEED}\\n50 & \quad \text{END}
\end{align*}
\]
Chapter 2—Program Instructions

PORTOUT ON/OFF

Purpose
Statement for setting one of four relay port or one of eight optical ports on a Serial/Industrial Interface Board to either on or off.

Syntax

```
PORTOUT (<nexp>) ON|OFF
```

<nexp> is the number of the port to be set:
- OUT ports (relay): 201-204 (401-404)
- OUT ports (optical): 221-228 (421-428)

Remarks
This statement works with the Serial/Industrial Interface Board and is able to control 8 IN ports with optocouplers, 8 OUT ports with optocouplers, and 4 OUT ports with relays. For information on how to read the status of the various ports, please refer to the PORTIN function.

This feature is intended to allow the execution of the Intermec Fingerprint program to control various external units like gates, lamps, or conveyor belts.

Some printers, like EasyCoder PM4i can carry two Serial/Industrial Interface boards. In this case, the ports on the inner board (that is, the board closest to the CPU board) are specified by the low numbers (201-204 and 221-228) while the ports on the outer board are specified by the high numbers (401-404 and 421-428).

Please refer to the documentation of the Serial/Industrial Interface Board for more information.

Example
The relay of OUT port 201 on a Serial/Industrial Interface Board is Opened and then Closed like this:

```
1000 PORTOUT (201) ON
2000 PORTOUT (201) OFF
```

1000 PORTOUT (201) ON
2000 PORTOUT (201) OFF
PRBAR (PB)

Purpose
Statement for providing input data to a bar code.

Syntax

\[
\text{PRBAR|PB<<sexp>|<nexp>>}
\]

\(<\text{sexp}|<\text{nexp}>>\) is the input data to the bar code generator.

Remarks
The bar code must be defined by BARSET, BARTYPE, BARRATIO, BARHEIGHT, BARMAG, BARFONT, and/or BARFONT ON/OFF statements, or by the corresponding default values.

Make sure that the type of input data (numeric or string) and the number of characters agree with the specification for the selected bar code type. Information on some of the most commonly used bar codes are provided at the end of this manual.

Example
Two different bar codes, one with numeric input data and one with string input data, can be generated this way. The input data could also have been entered in the form of variables:

```
10  BARFONT "Swiss 721 BT", 8 ON
20  PRPOS 50,400
30  ALIGN 7
40  BARSET "INT2OF5",2,1,3,120
50  PRBAR 45673
60  PRPOS 50,200
70  BARSET "CODE39",3,1,2,100
80  PRBAR "ABC"
90  PRINTFEED
RUN
```
### PRBOX (PX)

**Purpose**
Statement for creating a box, optionally containing a single text line or a frame of multiple hyphenated text lines.

**Syntax**
```
PRBOX|PX<nexp1>,<nexp2>,<nexp3>[,<sexp1>[,<nexp4>[,<nexp5>[,<sexp2>[,<sexp3>]]]]]]
```

- `<nexp1>` is the height of the box in dots (1-6000).
- `<nexp2>` is the width of the box in dots (1-6000).
- `<nexp3>` is the line weight in dots (0/1-6000).
- `<sexp1>` is the framed text to be written inside the box (max. 300 char./line, max. 20 lines). Single-byte fonts only.
- `<nexp4>` is the horizontal distance between inner edge of the box line and the text frame (-100 to 100 dots). Default: 0.
- `<nexp5>` is the vertical distance between the inner edge of the box line and text frame and also between each line of text in the frame (-100 to 100 dots). Default: Same value as `<nexp4>`.
- `<sexp2>` is a line delimiter (max. 9 characters), which replaces the default delimiter string CHRS(10) or CHRS(13). Each time this delimiter is encountered in the text string `<sexp1>`, the rest of the text is wrapped to the next line.
- `<sexp3>` is a control string for hyphen delimiter and replacement, see Remarks.

**Remarks**
This statement has two purposes: to create a rectangular white box surrounded by a line with a certain thickness, or to specify a text frame that can contain up to 20 lines of hyphenated text. These two purposes can be combined so a text frame is surrounded by a black box.

**Creating a simple box:**
In this case you only need to specify the first three parameters, that is height, width, and line weight (thickness). The box will be drawn with its anchor point (see ALIGN) at the insertion point, as specified by the nearest preceding PRPOS statement. A box can be aligned left, right, or center along its baseline.

The print direction specifies how the box is rotated in relation to its anchor point.

The line weight (thickness) grows inward from the anchor point. The heavier the line, the less white area inside the box. Thus, it is possible to create a black area using a box with very heavy lines. For a simple box without any text field, the line weight must be >0. The white area inside a box can be used for printing. Boxes, lines, and text may cross (also see XORMODE ON/OFF).
Creating a multiline text field

The PRBOX statement can also be used to create an area in which a field of wrapped and hyphenated text can be printed. As opposed to the PRTXT statement, there is no need to specify each line of text separately. The text field can be framed by the box (line weight > 0), or the box can be invisible (line weight = 0). The maximum number of characters on each line is 300 and the maximum number of lines is 20.

The position of the text frame inside the box is affected by the direction (see DIR statement), the alignment (see ALIGN statement), and by two parameters in the PRBOX statement (<nexp4> and <nexp5>.

The direction rotates the box with its text field around the anchor point as specified by the alignment. The alignment specifies the anchor point of the box itself as left-, right-, or center-aligned (see ALIGN), and at the the same time also decides how the field will be aligned inside the box (9 possible positions) and if the text lines will be left, right, or center justified.

In the following description, horizontal and vertical should be understood in relation to how the text is printed. (That means that in directions 2&4, horizontal and vertical have opposite meanings than in directions 1&3).

The horizontal distance between the inner edge of the box line and the borders of the text field is specified by <nexp4>:
- In case of ALIGN 1, 4, or 7, it decides the distance between the inner edge of the left side box line and the left-hand edge of the text field.
- In case of ALIGN 3, 6, or 9, it decides the distance between the inner edge of the right side box line and the right-hand edge of the text field.
- In case of ALIGN 2, 5, or 8, this parameter has no consequence.
The vertical distance between the inner edge of the box line and the borders of the text field as well as the empty vertical space between the character cells of two adjacent lines (line spacing) is specified by <nexp5>:
- In case of ALIGN 1, 2, or 3, it decides the distance between the inner edge of the bottom box line and the bottom edge of the text field as well as line spacing.
- In case of ALIGN 7, 8, or 9, it decides the distance between the inner edge of the right side box line and the right-hand edge of the text field as well as line spacing.
- In case of ALIGN 4, 5, or 6, this parameter only decides line spacing.

See the illustration on next page for examples of how the alignment affects the location of multi-line text.

If the text in <sexp1> is entered as a continuous string of characters without any spaces, linefeeds, or carriage returns, the text will wrap to the next line when there is no room left for any more characters on a line.

If any combination of a carriage return (CR = ASCII 13 dec.) and a line-feed (LF = ASCII 10 dec.) is encountered, the remaining text will be wrapped once to the next line.

Space characters (ASCII 32 dec.) will also initiate a line wrap. If there are more than one space character, the wrapping will be at the last one that fits into the line in question.

You can replace the default line delimiters (CR, LF, and CR/LF) with another line delimiter specified in a string of max. 9 characters (<sexp2>). This delimiter will not be printed, even if it is a printable character. Each time the delimiter is encountered, the text will wrap to a new line.

Hyphenation Support
In <sexp3> you can modify the way hyphenation will be performed using a special syntax described later on.

You can put “invisible” hyphen delimiters in the text string at suitable wrap-around positions. The delimiter is by default a hyphen sign (ASCII 45 dec.). However, you can use a string of any characters up to nine characters long instead, but be careful so it will not be confused with the text.

If a wrap-around is performed, the corresponding hyphen delimiter will by default be printed as a hyphen sign (ASCII 45 dec.), whereas hyphen delimiters not used for wrap-around will not be printed.

If you for some reason would like to print some other character(s) than hyphens, you can specify a string of hyphen replacement characters. It is possible to use a string up to nine characters long, but the shorter the string the lesser risk that a line will wrap outside the box.

If you have a text string with long words and have not inserted all necessary line delimiters, a line-wrap may occur unexpectedly. You can optionally specify a hyphen delimeters for this case as well. Default: None.
PRBOX, cont.

This is line number 1, and this is line 2, and now comes line 3 followed by No. 4.

ALIGN 7

ALIGN 8

ALIGN 9

ALIGN 4

ALIGN 5

ALIGN 6

ALIGN 1

ALIGN 2

ALIGN 3
Chapter 2—Program Instructions

PRBOX, cont.

Specify the parameter <sexp₃> in PRBOX using the following syntax:

\[ <\text{sexp}_3> = <\text{sexp}_{3a}>[\text{space}<\text{sexp}_{3b}>[\text{space}<\text{sexp}_{3c}>]] \]

- \(<\text{sexp}_{3a}>>\): is a soft hyphen delimiter. If the text does not have enough room on one line, the rest of the text will be wrapped from the last space or from the position marked by the soft hyphen delimiter. Exception: Two adjacent soft hyphen delimiters revoke each other.
  
  Default: Normal hyphen (-).
  
  Max length: 9 characters.

- \(<\text{sexp}_{3b}>>\): is a string delimiter with the value CHRS(32).

- \(<\text{sexp}_{3c}>>\): is one or more characters, that will be printed at the end of a line which has been hyphenated according to a hyphen delimiter (see \(<\text{sexp}_{3a}>>\)).
  
  Default: Normal hyphen (-).
  
  Max length: 9 characters (less is preferred).

- \(<\text{sexp}_{3d}>>\): is a string of hyphen extension characters, used on single words which are too long to be printed on one line and have no hyphen delimiter specified. The hyphen extension character(s) will be printed at the right end of line and the remainder of the word will be printed on the next line.
  
  Default: No character.
  
  Max length: 9 characters.

If no \(<\text{sexp}_{3}>\) is specified, the rule for hyphen delimiter and replacement will be the same as for printing hyphens in text. Two adjacent hyphens will be printed as one.

Examples

This example draws a rectangle without any text:

```
10    PRPOS 50,50
20    PRBOX 200,400,5
30    PRINTFEED
RUN
```

This program illustrates a multi-line text field with line wrap, where “&S” is the soft hyphen delimiter:

```
10    DIR 1
20    ALIGN 8
30    R$="Hyphen&Sated words will be divided into syllableSes."
40    NL$="NEWLINE"
50    S$="&S&Special Cases and EXTRAORDINARY long words."
60    T$=R$+NL$+S$
70    PRPOS 300,300
80    PRBOX 700,500,20,T$+25,1,NL$,"&S - +"
90    PRINTFEED
RUN
```
**PRBUF**

**Purpose**
Statement for receiving and printing bitmap image data using the PRBUF protocol.

**Syntax**

```
PRBUF<nexp1>[,<nexp2]<newline><image data>
```

- `<nexp1>` is the number of bytes of the image in PRBUF protocol.
- `<nexp2>` is, optionally, a timeout between characters in TICKS (0.01 sec).
- Default ≈ 12.7 sec./character.
- `<newline>` is any combination of CR, CR/LF, or LF.
- `<image data>` is the image according to the PRBUF protocol.

**Remarks**

This statement is useful for receiving and printing bitmap images from, for example, a Windows printer driver. It is more effective and requires less memory than using a STORE IMAGE...PRIMAGE sequence. The bitmap image is printed directly and is not saved anywhere in the printer’s memory after the image buffer has been cleared.

At the PRBUF statement, the printer waits for image data to be received on the standard IN channel. PRBUF only works with binary transfers, that is XON/XOFF must be disabled. You can optionally set a timeout between characters (default 12.7 sec.) When the specified number of bytes according to the PRBUF protocol have been received, the image data are processed directly into the printer’s image buffer and printed without requiring any more Fingerprint instructions.

PRBUF does not work if `<nexp1>` bytes cannot be allocated. If memory is low, it is possible to download the bitmap image in two or more blocks.

The field settings (alignment, clipping, direction, xor mode, inverse image, magnification, x-position, and y-position) are handled by the current protocol, but the basic rule is that x- and y-positions, field clipping, and xor mode are handled and the other attributes are ignored.

If PRPOS x,y, then the real print position will be PRPOS x,y+1.

The PRBUF protocol is described in Chapter 3, “Image Transfer.”

The `<newline>` is not part of the statement, but any combination of carriage return (ASCII 13 dec.) and/or linefeed (ASCII 10 dec.) is allowed without interfering with the PRBUF protocol.

**Example**

This example shows how the printer is instructed to receive and print 1,424 bytes of image data according to the PRBUF protocol:

```
PRBUF 1424
<binary image data>
```
PRIMAGE (PM)

Purpose
Statement for selecting an image stored in the printer’s memory.

Syntax
PRIMAGE|PM<sexp>

<sexp> is the full name of the desired image including extension.

Remarks
An image is positioned according to the preceding PRPOS, DIR, and ALIGN statements. It can be magnified by means of a MAG statement.

For the best printout quality, create and download a larger version of the image rather than magnifying a smaller one.

All images provided by Intermec have an extension which indicates for which directions the image is intended:

- Extension .1 indicates print directions 1 & 3.
- Extension .2 indicates print directions 2 & 4.

Even if the Intermec Fingerprint firmware does not require such an extension, we strongly recommend you to follow the same convention when creating your own images as to make it easier to select the correct image.

Example
This example illustrates the printing of a label containing an image printed “upside down”:

10 PRPOS 200,200
20 DIR 3
30 ALIGN 5
40 PRIMAGE "GLOBE.1"
50 PRINTFEED
RUN
PRINT (?)

**Purpose**

Statement for printing data to the standard OUT channel.

**Syntax**

```
PRINT?<exp>[,<exp>|<sexp>>[<,|;><<nexp>|<sexp>>...][;]]
```

<<nexp>|<sexp>> are string or numeric expressions, which will be printed to the standard OUT channel.

**Remarks**

If no expressions are specified after the PRINT statement, it will yield a blank line. If one or more expressions are listed, the expression(s) will be processed and the resulting values will be presented on standard OUT channel (see SETSTDIO statement), for example usually on the screen of the host. The shorthand form of PRINT is a question mark.

Do not confuse the PRINT statement with the PRINTFEED statement.

Each line is divided into zones of 10 character positions each. These zones can be used for positioning the values:

- A comma sign (,) between the expressions causes next value to be printed at the beginning of next zone.
- A semicolon sign (;) between the expressions causes next value to be printed immediately after the last value.
- A plus sign (+) between two string expressions also causes next value to be printed immediately after the last value. (Plus signs cannot be used between numeric expressions.)
- If the list of expressions is terminated by a semicolon, the next PRINT statement will be added on the same line. Otherwise, a carriage return is performed at the end of the line. If the printed line is wider than the screen, the software will automatically wrap to a new line and go on printing.

Printed numbers are always followed by a space character.

Printed negative numbers are preceded by a space character.

**Example**

```
10 LET X%=10
20 LET A$="A"
30 PRINT X%;X%+1,X%+5;X%-25
40 PRINT A$+A$;A$,A$
50 PRINT X%;
60 ? "PIECES"
RUN
```

yields:

```
11 15 -15
AAA A
10 PIECES
```
## PRINT KEY ON/OFF

### Purpose
Statement for enabling or disabling printing of a label by pressing the Print key.

### Syntax
```
PRINT KEY ON|OFF
```
Default: PRINT KEY OFF

### Remarks
In the Immediate Mode and in the Intermec Direct Protocol, the <Print> key can be enabled to issue printing commands, corresponding to PRINT-FEED statements. This implies that each time the <Print> key is pressed, one single label, ticket, tag, or portion of continuous stock will be printed and fed out.

Note that this statement cannot be entered in the Programming Mode (use KEY ON and ON KEY GOSUB statements instead).

### Example
This example shows how the Print key is enabled in the Intermec Direct Protocol and a label is printed (abbreviated instructions are used whenever available):
```
INPUT ON
PRINT KEY ON
PP 100,100
FT "Swiss 721 BT"
PT "TEST LABEL"

[Press the <Print> key]

INPUT OFF
```
PRINT#

Purpose
Statement for printing data to a specified OPENed device or sequential file.

Syntax
PRINT#<nexp1>[,<nexp2>|<sexp1>[<,|;><nexp3>|<sexp2>...[;]]]

<nexp1>
the number assigned to the file or device when it was OPENed.

<<nexp2-n>|<sexp1-n>>
are the string or numeric expressions, which will be printed to the specified file or device.

Remarks
Expressions can be separated by commas or semicolons according to the same rules as for the PRINT statement. It is important that the expressions are separated properly, so they can be read back when needed, or be presented correctly on the printer's LCD display.

PRINT# can only be used to print to sequential files, not to random files.

When sending data to the printer's display ("console:"), PRINT# will work same way as PRINT does on the standard OUT channel. The display can for example be cleared by sending PRINT#<nexp> twice (see line 20 in the example below).

Example
The display on the printer's keyboard console is able to show two lines with 16 characters each. Before sending any text, the device must be OPENed (line 10) and both lines on the display must be cleared (line 20).

Note the trailing semicolon on line 40!

10 OPEN "console:" FOR OUTPUT AS #1
20 PRINT# 1:PRINT# 1
30 PRINT# 1,"OUT OF LABELS"
40 PRINT# 1,"PLEASE RELOAD!";
50 CLOSE# 1
RUN

Since the last line was appended by a semicolon, there will be no carriage return and the text will appear on both line on the printer's display as:

OUT OF LABELS
PLEASE RELOAD!

An alternative method is to send all the data to the display in a single PRINT# statement. Character No. 1-16 will be displayed on the upper line and character No. 17-33 will be displayed on the lower line, whereas character No. 17 will be ignored. Note the trailing semicolon on line 30! (The double-headed arrows in line 30 represent space characters.)

10 OPEN "console:" FOR OUTPUT AS #1
20 PRINT# 1:PRINT# 1
30 PRINT# 1,"OUT OF LABELS _PLEASE_ RELOAD!";
40 CLOSE# 1
RUN
Chapter 2—Program Instructions

PRINTFEED (PF)

Purpose
Statement for printing and feeding out one or a specified number of labels, tickets, tags, or portions of strip, according to the printer's setup.

Syntax

```
PRINTFEED|PF [<nexp1>,] |[-1,<nexp2>,]
```

- `<nexp1>` specifies the number of copies to be printed.
- `-1,<nexp2>` specifies that `<nexp2>` number of identical copies of the last printed label should be reprinted (cannot be used with Intermec Direct Protocol).

Remarks

Each time a PRINTFEED statement without any appending value is executed, one new label, ticket, tag, or portion of continuous stock will be printed.

The PRINTFEED statement can optionally be appended by a numeric expression, which specifies the number of copies to be printed. In the Intermec Direct Protocol, possible counter, time, and date values will be updated between copies printed using a predefined layout. Note that you must never include any PRINTFEED statements in layouts in the Intermec Direct Protocol.

If the number of copies is >1 and LTS& and CUT are disabled (= LTS& OFF and CUT OFF), the BATCH optimizing strategy is automatically enabled, which corresponds to an OPTIMIZE BATCH ON statement. When theses conditions are no longer fulfilled, BATCH optimizing strategy is automatically disabled, which corresponds to an OPTIMIZE BATCH OFF statement.

It is also possible to reprint a specified number of copies of the last printed label, for example after an out-of-media condition (also see PRSTAT).

The execution of a PRINTFEED statement clears the following statements to their default values:

- ALIGN
- BARFONnT
- BARSET
- DIR
- FONT
- BARRATIO
- MAG
- PRPOS
- XORMODE
- BARHEIGHT
- BARHEIGHT
- FONTD
- FONTD

Fields defined by statements, that have been executed before the PRINTFEED statement, are not affected. Note that, when using a PRINTFEED in a loop, all formatting parameters are reset to default each time the PRINTFEED statement is executed and must therefore be included inside the loop.
PRINTFEED (PF), cont.

The length of media to be fed out at execution of a PRINTFEED statement is decided by the choice of media type in the printer's setup (label w gaps, ticket w gaps, fix length strip, or var length strip) and globally by the start and stop adjustment setup (positive or negative). Refer to the User’s Guide for more information. The length of media to be fed out can be further modified by an additional positive or negative FORMFEED statement, either before or after the PRINTFEED statement.

Examples

Printing a single label with one line of text:
10  FONT "Swiss 721 BT"
20  PRTXT "Hello!"
30  PRINTFEED
RUN

Printing five identical labels with one line of text:
10  FONT "Swiss 721 BT"
20  PRTXT "Hello!"
30  PRINTFEED 5
RUN

Printing five labels using a FOR...NEXT loop. Note that formatting parameters are placed inside the loop:
10  FOR A%=1 TO 5
20  FONT "Swiss 721 BT"
30  PRPOS 200, 100
40  DIR 3
50  ALIGN 5
60  PRTXT "Hello!"
70  PRINTFEED
80  NEXT A%
RUN

Printing of five labels in the Intermec Direct Protocol, illustrating how the TICKS value is updated between labels, provided a predefined layout is used (1 TICK = 0.01 sec):
INPUT ON ⇧
FORMAT INPUT "#","@","&" ⇧
LAYOUT INPUT "tmp:LABEL1" ⇧
FT "Swiss 721 BT" ⇧
PP 100,100 ⇧
PT TICKS ⇧
PP 100,200 ⇧
PT VAR1$ ⇧
LAYOUT END ⇧
LAYOUT RUN "tmp:LABEL1" ⇧
#See how time flies&@ ⇧
PF 5 ⇧
INPUT OFF ⇧
PRINTONE

Purpose
Statement for printing characters, specified by their ASCII values, to the standard OUT channel.

Syntax
PRINTONE<nexp>[<,|;><nexp>...][;]

<nexp> is the ASCII decimal value of a character, which will be printed to the standard OUT channel.

Remarks
When, for some reason, certain characters cannot be produced by the host computer, they can be substituted by the corresponding ASCII decimal values using the PRINTONE statement. The characters will be printed, according to the currently selected character set (see NASC statement), to the standard OUT channel, that is, usually to the screen of the host. PRINTONE is very similar to the PRINT statement and the use of commas and semicolons follows the same rules.

Example
PRINTONE 80;82;73;67;69;58;36;52;57;46;57;53

yields:
PRICE: $49.95
PRINTONE#

**Purpose**
Statement for printing characters specified by their ASCII values to a device or sequential file.

**Syntax**
```
PRINTONE#<nexp1>[,<nexp2>][<,|;><nexp3>...][;]
```
- `<nexp1>` is the number assigned to the file or device when it was OPENed.
- `<nexp2>` is the ASCII decimal value of the character, which is to be printed to the specified file or device.

**Remarks**
This statement is useful, when the host for some reason cannot produce certain characters. The ASCII values entered will produce characters according to the currently selected character set, see NASC. The ASCII values can be separated by commas or semicolons according to the same rules as for the PRINT# statement.

PRINTONE# can only be used to print to sequential files, not to random files.

When sending data to the printer’s display, PRINTONE# will work in a way similar to PRINT#. The display can be cleared by sending PRINT#<nexp> twice (see line 20 in the example below).

**Example**
The display on the printer’s keyboard console is able to show two lines with 16 characters each. Before sending any text, the device must be OPENed and the display be cleared. Note the trailing semicolon sign on line 40.

```
10 OPEN "console:" FOR OUTPUT AS #1
20 PRINT# 1:PRINT# 1
30 PRINT# 1,80;82;69;83;83
40 PRINTONE# 1,69;78;84;69;82;
50 CLOSE #1
RUN
```

Since the last line was appended by a semicolon, there will be no carriage return and the text will appear on both line on the printer’s display as:

PRESS
ENTER
**PRLINE (PL)**

**Purpose**
Statement for creating a line.

**Syntax**

```
PRLINE|PL<nexp₁>,<nexp₂>
```

<nexp₁> is the length of the line in dots (max. 6000).

<nexp₂> is the line weight in dots (max. 6000).

**Remarks**
The line will be drawn from the insertion point and away according to the nearest preceding DIR and ALIGN statements (that is the line runs in parallel with any text printed in the selected direction).

A line can be ALIGNed left, right or center. The anchor points are situated at the bottom of the line, which means that with an increasing line weight (thickness), the line will the grow upward in relation to the selected direction. In the illustration below, all lines are aligned left. Lines may cross (see XORMODE ON/OFF statement).

**Example**
This example draws a 2.5 cm (1 inch) long and 10 dots thick line across the media in an 8 dots/mm printer:

```
10   PRPOS 50,100
20   PRLINE 200,10
30   PRINTFEED
RUN
```
PRPOS (PP)

Purpose
Statement for specifying the insertion point for a line of text, a bar code, an image, a box, or a line.

Syntax
```
PRPOS|PP<nexp1>,<nexp2>
```

- `<nexp1>` is the X-coordinate (number of dots from the origin).
- `<nexp2>` is the Y-coordinate (number of dots from the origin).

Default value: 0,0
Reset to default by: PRINTFEED execution.

Remarks
When the printer is set up, a “print window” is created. This involves specifying the location of the origin along the X-axis, setting the max. print width along the X-axis from origin, and setting the max. print length along the Y-axis from origin.

The X-coordinate goes across the media path and the Y-coordinate along the media feed direction, as illustrated below. They are set in relation to the origin on the printhead, not in relation to the media. Thus, the position where an object actually will be printed depends on the relation between printhead and media at the moment when the printing starts.
PRPOS (PP), cont.

The insertion point must be selected so the field in question will fit inside the print window. This implies that the print direction, the size of the field including “invisible” parts of for example an image, the alignment, and other formatting instructions must be considered. A field that does not fit entirely inside the print window will cause Error 1003, “Field out of label”, except when a CLIP ON statement is issued.

To find out the present insertion point, use the PRSTAT function.

Examples

Programming and printing a line of text:

```
10  FONT "Swiss 721 BT"
20  PRPOS 30,200
30  PRTXT "HELLO"
40  PRINTFEED
RUN
```

Each text line is normally positioned separately by its own PRPOS statement. If no position is given for a printable statement, it will be printed immediately after the preceding printable statement.

```
10  FONT "Swiss 721 BT"
20  PRPOS 30,200
30  PRTXT "SUMMER"
40  PRTXT "TIME"
50  PRINTFEED
RUN
```

yields a label with the text:

```
SUMMERTIME
```

A program for fixed line-spacing of text may be composed this way (another way is to use the extended PRBOX statement):

```
10  FONT "Swiss 721 BT"
20  X%=30:Y%=500
30  INPUT A$
40  PRPOS X%,Y%
50  PRTXT A$
60  Y%=Y%+-50
70  IF Y%>=50 GOTO 30
80  PRINTFEED
90  END
RUN
```

Enter the text for each line after the question mark shown on the screen of the host. The Y-coordinate will be decremented by 50 dots for each new line until it reaches the value 50, which means that ten lines will be printed.
**PRSTAT**

**Purpose**

Function returning the printer’s current status or, optionally, the current position of the insertion point.

**Syntax**

```plaintext
PRSTAT[<nexp>]
```

- `<nexp> = 1` returns the X-position for the insertion point at DIR 1&3.
- `<nexp> = 2` returns the Y-position for the insertion point at DIR 2&4.
- `<nexp> = 3` returns the X-position of the corner with the lowest coordinates of the last object.
- `<nexp> = 4` returns the Y-position of the corner with the lowest coordinates of the last object.
- `<nexp> = 5` returns the width along the X-axis of the last object.
- `<nexp> = 6` returns the height along the Y-axis of the last object.
- `<nexp> = 7` returns the print job identifier.
- `<nexp> = 8` returns the print job state (see table below).
- `<nexp> = 9` returns the print job error code.
- `<nexp> = 10` returns the remaining number of copies to be printed in a batch print job.

**Remarks**

PRSTAT

Returns a numeric expression, which is the sum of the values given by the following conditions, at the moment when the PRSTAT function is executed:

- OK .................................................................................................................................................. 0
- Printhead lifted ................................................................. 1
- Label not removed  (see note) .............................................. 2
- Label stop sensor (LSS) detects no label .................. 4
- Printer out of transfer ribbon (TTR) or ribbon installed (DT) .......... 8
- Printhead voltage too high ............................................. 16
- Printer is feeding ................................................................. 32
- RESERVED........................................................................... 64
- Printer out of media............................................................. 128

*Note:* Always returns 0 in printers not fitted with a label taken sensor.

If two error conditions occur at the same time, for example the printhead is lifted and the printer is out of transfer ribbon, the sum will be (1+8) = 9. Every combination of errors will result in a unique sum. You can use it to branch to a subroutine which notifies the operator, interrupts the program or whatever you like. The Label stop sensor will detect no label if a gap or black mark is in front of the sensor, as well as when the printer is out of media.

**PRSTAT(1) & PRSTAT(2)**

The current position of the insertion point in regard of either the X or the Y position can be returned, depending on the selected print direction. This is useful for example measuring the length of a text or a bar code.
Chapter 2—Program Instructions

PRSTAT, cont.

PRSTAT(3)-PRSTAT(6)
These functions are used to return the position and size of the last object regardless of RENDER ON/OFF. Their values are not updated by the execution of a PRBUF statement.

PRSTAT(7)-PRSTAT(10)
These functions are used to detect if a print job has been interrupted, so steps can be taken to reprint missing copies (see PRINTFEED).

PRSTAT (7) returns a print job identifier that is automatically assigned to the print job by the firmware.

PRSTAT (8) returns the state of the print job as a numeric expression, which is the sum of the values given by the following conditions:
- Print cycle not set up for printing, perhaps due to out-of-ribbon ..........0
- The previous print cycle never ended (timeout).................................1
- Print cycle has started....................................................................2
- All lines successfully printed............................................................4
- Printing truncated (media shorter than print image).........................8
- Printhead strobing error or label length exceeded..........................16
- Ribbon low ......................................................................................32

PRSTAT (8) = 6 or 22 indicates a successfully printed label (in the latter case error “next label not found” may have been detected).

PRSTAT (9) returns the error code (see Chapter 7, "Error Messages") detected by the print engine during printfeed. It is used together with PRSTAT(8) to determine the error cause when using OPTIMIZE "BATCH" ON.

PRSTAT (10) returns the number of copies that remains to be printed in an interrupted batch print job.

Examples

This examples shows how two error conditions are checked:
10 A% = PRSTAT
20 IF A% AND 1 THEN GOSUB 1000
30 IF A% AND 128 THEN GOSUB 1010
40 END
.....
1000 PRINT "Printhead is lifted";RETURN
1010 PRINT "Printer out of media";RETURN
RUN

This example illustrates how you can check the length of a text:
10 PRPOS 100,100: FONT "Swiss 721 BT"
20 PRTXT "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
30 PRINT PRSTAT(1)
RUN

yields:
519
PRTXT (PT)

Purpose
Statement for providing the input data for a text field.

Syntax
```
PRTXT|PT<<nexp>|<sexp>>[;<<nexp>|<sexp>>...][]]
```

<<nexp>|<sexp>> specifies one line of text (max. 300 characters)

Remarks
A text field consists of one line of text. The text field must be defined in regard of FONT or FONTD and may be further defined and positioned by DIR, ALIGN, MAG, PRPOS, INVIMAGE, or NORIMAGE statements or their respective default values.

Two or more expressions can be combined to form a text line. They must be separated by semicolons (;) and will be printed adjacently. Plus signs can also be used for the same purpose, but only between string expressions.

String constants must be enclosed by quotation marks, whereas numeric constants or any kind of variables must not.

Examples
Programming and printing a line of text:
```
10 FONT "Swiss 721 BT"
20 PRPOS 30,300
30 PRTXT "How do you do?"
40 PRINTFEED
RUN
```

Several string constants and string variables can be combined into one line of text by the use of plus signs or semicolons:
```
10 FONT "Swiss 721 BT"
20 PRPOS 30,300
30 PRTXT "SUN";"SHINE"
40 A$="MOON"
50 B$="LIGHT"
60 PRPOS 30,200
70 PRTXT A$+B$
80 PRINTFEED
RUN
```
yields a label with the text:

SUNSHINE
MOONLIGHT
PRTXT (PT), cont.

Numeric constants and numeric variables can be combined by the use of semicolons, but plus signs cannot be used in connection with numeric expressions:

```plaintext
10   FONT "Swiss 721 BT"
20   PRPOS 30,300
30   PRTXT 123;456
40   A%=222
50   B%=555
60   PRPOS 30,200
70   PRTXT A%;B%
80   PRINTFEED
RUN
```

yields a label with the text:

```
123456
222555
```

Numeric and string expressions can be mixed on the same line, for example:

```plaintext
10   FONT "Swiss 721 BT"
20   PRPOS 30,300
30   A$="December"
40   B%=27
50   PRTXT A$;" ";B$;" ";"2003"
80   PRINTFEED
RUN
```

yields a label with the text:

```
December 27 2003
```

Two program lines of text will be printed on the same line if the first program line is appended by a semicolon:

```plaintext
10   FONT "Swiss 721 BT"
20   PRPOS 30,300
30   PRTXT "HAPPY"+" ";
40   PRTXT "BIRTHDAY"
50   PRINTFEED
RUN
```

yields a label with the text:

```
HAPPY BIRTHDAY
```
**PUT**

**Purpose**
Statement for writing a given record from the random buffer to a given random file.

**Syntax**

\[
\text{PUT}[\#]<nexp_1>,<nexp_2>
\]

- \# indicates that whatever follows is a number. Optional.
- \(<nexp_1>\) is the number assigned to the file when it was OPENed.
- \(<nexp_2>\) is the number of the record. Must be \(\geq 1\).

**Remarks**
Use LSET or RSET statements to place data in the random buffer before issuing the PUT statement.

**Example**

```plaintext
10   OPEN "PHONELIST" AS #8 LEN=26
20   FIELD#8,8 AS F1$, 8 AS F2$, 10 AS F3$
30   SNAME$="SMITH"
40   CNAME$="JOHN"
50   PHONE$="12345630"
60   LSET F1$=SNAME$
70   LSET F2$=CNAME$
80   RSET F3$=PHONE$
90   PUT #8,1
100  CLOSE#8
RUN

SAVE "PROGRAM 1.PRG "

NEW
10   OPEN "PHONELIST" AS #8 LEN=26
20   FIELD#8,8 AS F1$, 8 AS F2$, 10 AS F3$
30   GET #8,1
40   PRINT F1$,F2$,F3$
RUN

yields:

SMITH <-> JOHN <-> 12345630
```

*Intermec Fingerprint v8.70.0/v10.0.0 Programmer’s Reference Manual*
**RANDOM**

**Purpose**
Function generating a random integer within a specified interval.

**Syntax**
\[
\text{RANDOM}(\text{<nexp_1>},\text{<nexp_2>})
\]

- \text{<nexp_1>} is the first integer in the interval.
- \text{<nexp_2>} is the last integer in the interval.

**Remarks**
\text{<nexp_1>} \leq \text{random integer} \leq \text{<nexp_2>}, that is, the random integer will be:

Equal to or greater than \text{<nexp_1>}
Equal to or less than \text{<nexp_2>}

**Example**
The following example will produce ten random integers between 1 and 100:

```
10  PRINT A% = RANDOM (1, 100)
20  PRINT A%
30  NEXT I%
40  RUN
```

 yields for example:

```
31
45
82
1
13
16
41
77
20
70
```
**RANDOMIZE**

**Purpose**
Statement for reseeding the random number generator, optionally with a specified value.

**Syntax**
```
RANDOMIZE[<nexp>]
```

- `<nexp>` is the integer (0 - 99999999) with which the random number generator will be reseeded.

**Remarks**
If no value is specified, a message will appear asking you to enter a value between 0 and 99,999,999.

**Examples**
In the following example, no reseeding integer is specified in the program. Thus a prompt will appear, asking you to do so:

```
10  RANDOMIZE
20  A%=RANDOM1,100)
30  PRINT A%
RUN
Random Number Seed (0 to 99999999) ?
Enter 555
```
yields for example:

```
36
```

When the reseeding integer is specified, no prompt will appear:

```
10  RANDOMIZE 556
20  A%=RANDOM(1,100)
30  PRINT A%
RUN
```
yields for example:

```
68
```

A higher degree of randomization will be obtained in the random integer generator is reseeded with a more or less random integer, for example provided by a TICKS function:

```
10  A%=TICKS
20  RANDOMIZE A%
30  B%=RANDOM (1,100)
40  PRINT B%
RUN
```
yields for example:

```
42
```
**READY**

**Purpose**
Statement for ordering a ready signal, for example XON, CTS/RTS or PE, to be transmitted from the printer on the specified communication channel.

**Syntax**
```
READY[<nexp>]
```

<nexp> optionally specifies a communication channel:
- 1 = "uart1:"
- 2 = "uart2:"
- 3 = "uart3:"
- 4 = "centronics:"
- 6 = "usb1:"
- 8 = "uart4:"
- 9 = "uart5:"

**Remarks**
The selected communication protocol usually contains some “ready” signal, which tells the host computer that the printer is ready to receive more data. The READY statement allows you to order a ready signal to be transmitted on the specified communication channel. If no channel is specified, the signal will be transmitted on the standard OUT channel (see SETSTDIO statement).

The READY signal is used to revoke a previously transmitted BUSY signal. However, the printer may still be unable to receive more data, for example because of a full receive buffer.

For the "centronics:" communication channel, BUSY/READY controls the PE (paper end) signal on pin 12 according to an error-trapping routine (READY = PE low).

**Example**
You may, for example, want to allow the printer to receive more data on "uart2:" after the process of printing a label is completed. (Running this example may require an optional interface board to be fitted):  
```
10  FONT "Swiss 721 BT"
20  PRTEXT "HELLO!"
30  BUSY2
40  PRINTFEED
50  READY2
RUN
```
**REBOOT**

**Purpose**
Statement for restarting the printer.

**Syntax**
REBOOT

**Remarks**
This statement has exactly the same effect as switching off and on the power to the printer.
REDIRECT OUT

Purpose
Statement for redirecting the output data to a created file.

Syntax
REDIRECT_OUT[<sexp>]

<sexp> is, optionally, the name of the file to be created and where the output will be stored.

Remarks
Normally the output data will be transmitted on the standard output channel (see SETSTDIO statement). In most cases, this means the screen of the host. However, by means of a REDIRECT OUT <sexp> statement, a file can be created to which the output will be redirected. That implies that no data will be echoed back to the host. Normal operation, with the output being transmitted on the standard output channel again, will be resumed when a REDIRECT OUT statement without any appending file name is executed.

Example
In this example, a file ("LIST.DAT") is created to which the names of the files in the printer's permanent memory is redirected. The redirection is then terminated (line No. 30) and the file is OPENed for input.

10 REDIRECT OUT "LIST.DAT"
20 FILES "/c"
30 REDIRECT OUT
40 OPEN "LIST.DAT" FOR INPUT AS #1

. . . . .
. . . . .
. . . . .
**REM (')**

**Purpose**
Statement for adding headlines and comments to the program without including them in the execution.

**Syntax**
```
REM|'<remark>
```

<remark> is a text inserted in the program for explanatory purpose. Max. 32,767 characters per line.

**Remarks**
A REM statement may either be entered on a program line of its own or be inserted at the end of a line containing another instruction. In the latter case, REM must be preceded by a colon ("::REM").

A shorthand form for REM is an apostrophe (ASCII 39 dec.).

It is possible to branch to a line of REM statement. Execution will then continue at the first executable line after the REM line.

REM statements slow down execution and transfer of data and also take up valuable memory space. Therefore, use REM statements with judgement.

**Example**
A program containing REM statements:
```
10 'Label format No. 1
20 FONT "Swiss 721 BT"
30 PRPOS 30,100
40 DIR 1 :REM Print across web
50 ALIGN 4 :REM Aligned left/baseline
60 MAG 2,2 :'Double height and width
70 PRTXT "HELLO"
80 PRINTFEED
RUN
```
**REMOVE IMAGE**

**Purpose**
Statement for removing a specified image from the printer's memory.

**Syntax**

```
REMOVE IMAGE <sexp>
```

<sexp> is the full name including extension of the image to be removed.

**Remarks**
Useful for removing obsolete or faulty images from the printer's memory in order to save valuable memory space.

Note that there is a distinction between on one hand images and on the other hand image files (compare with IMAGES and FILES statements). This implies that REMOVE IMAGE statements can only be used for images downloaded by means of a STORE statement (see STORE and STORE IMAGE). Image files downloaded using for example a TRANSFER KERMIT statement should be removed the same way as other files using a KILL statement.

Be careful, REMOVE IMAGE is irreversible!

**Example**

```
10   REMOVE IMAGE "LOGOTYPE.1"
RUN
```
**RENDER ON/OFF**

**Purpose**
Statement for enabling/disabling rendering of text, bar code, image, box, and line fields.

**Syntax**

```
RENDER ON|OFF
```

<table>
<thead>
<tr>
<th>ON</th>
<th>enables rendering (default).</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>disables rendering.</td>
</tr>
</tbody>
</table>

**Remarks**

These statements are intended to get information regarding size and position of a field without actually rendering it, that is the field will not be printed when the program is executed. The information on the field is retrieved using PRSTAT functions.

RENDER OFF disables the rendering, which means that PRTXT, PRBAR, PRIMAGE, PRLINE, and PRBOX statements will not give any result when a PRINTFEED statement is executed. Any other statements than PRPOS will not update the insertion point. Field numbers (see FIELDNO) will not be updated. Statement such as CLIP ON/OFF, XORMODE ON/OFF, or BARSET will retain their usual meanings. PRBUF will render a field regardless of RENDER ON/OFF.

RENDER ON enables field rendering after a RENDER OFF statement.

Duplicate statement have no effect, that is if a RENDER OFF statement has been executed, another RENDER OFF statement will be ignored. The same applies to RENDER ON.

**Example**

This examples retrieves information on the size of a text field which was not rendered. (The actual result may vary depending on font, font size, and printer type.)

```
10  RENDER OFF
20  PRTXT "Render off"
30  PRINT "Width:",PRSTAT(5),"Height:",PRSTAT(6)
40  RENDER ON
50  PRINTFEED
RUN
```

```
yields for example:
```
```
Width: 153 Height: 46
No field to print in line 50
Ok
```
**RENUM**

**Purpose**
Statement for renumbering the lines of the program currently residing in the printer's working memory.

**Syntax**

```
RENUM[<ncon_1>][, [<ncon_2>][, <ncon_3>]]
```

- `<ncon_1>` is the first line number of the new sequence.
- `<ncon_2>` is the line in the current program at which renumbering is to start.
- `<ncon_3>` is the desired increment between line numbers in the new sequence.

Default values: 10, 1, 10

**Remarks**
This statement is useful for providing space for more program lines when expanding an existing program, and for renumbering programs written without line numbers, for example after being LISTed, LOADed, or MERGEd. Line references following GOTO statements will be renumbered accordingly. Use a LIST statement to print the new numbers on the screen.

**Example**
A program may be renumbered like this:

```
10   FONT "Swiss 721 BT"
20   PRPOS 30,100
30   PRTXT "HELLO"
40   A%=A%+1
50   PRINTFEED
60   IF A%<3 GOTO 40
70   END
RENUM 100,20,50
LIST
```

yields:

```
10   FONT "Swiss 721 BT"
100  PRPOS 30,100
150  PRTXT "HELLO"
200  A%=A%+1
250  PRINTFEED
300  IF A%<3 GOTO 200
350  END
```

Note that the line number in the GOTO statement on line 300 has changed. Line 10 is not renumbered, since line 20 was specified as starting point. The new increment is 50.
### REPRINT ON/OFF

**Purpose**

Statement for enabling/disabling reprinting of a label in the Direct Protocol.

**Syntax**

```
REPRINT <ON|OFF>
```

- **ON**  Enables reprinting (default)
- **OFF** Disables reprinting

**Remarks**

The REPRINT ON/OFF statement is used to enable or disable reprinting of a label. REPRINT OFF also affects and overrides the behaviour of the PRINT KEY ON statement. If REPRINT OFF is entered before PRINT KEY ON, no error message is shown but PRINT KEY is set to ON and an empty label is printed when the <Print> key is pressed.

If REPRINT is set to OFF, there is no way to reprint an old print job. If a PRINTFEED statement is sent to the printer after a print job has been completed, a blank label is fed out and the error 1006 “No field to print” occurs. However, the REPRINT OFF statement does not clear the print buffer, which only occurs after a PRINTFEED statement has been executed (see example 2 below).

Leaving and re-entering the Direct Protocol does not reset the REPRINT status. A reboot resets the REPRINT status to its default value (ON).

A REPRINT OFF statement prevents automatic reprinting after the error has been cleared for the following errors:

- 1005 “Out of paper”
- 1022 “Head lifted”
- 1031 “Next label not found”
- 1058 “Transfer ribbon is installed”

Error 1027 “Out of transfer ribbon” does not generate the display prompt “Continue-Reprint” if REPRINT is set to OFF.

**Examples**

This example disables reprinting:

```
INPUT ON
REPRINT OFF
```

This example shows a special case, when the first and second PRINTFEED statements generates a printed label. After the second PRINTFEED, the print buffer is cleared and a “No field to print” error occurs.

```
INPUT ON
PRTXT "PRINT1"
PRINTFEED
REPRINT OFF
PRINTFEED  Yields a printed label, clears print buffer
PRINTFEED  Yields a blank label, generates an error
```
RESUME

Purpose
Statement for resuming program execution after an error-handling subroutine has been executed.

Syntax
```
RESUME[<<ncon>|<line label>|<NEXT>|<0>|<HTTP>]
```

- `<ncon>` is the number or label of the line to which the program should return.

Remarks
RESUME must only be used in connection with error-handling subroutines (see ON ERROR GOTO).

There are five ways of using RESUME:
- `RESUME` Execution is resumed at the statement where the error occurred.
- `RESUME 0` Same as RESUME.
- `RESUME NEXT` Execution is resumed at the statement immediately following the one that caused the error.
- `RESUME <ncon>` Execution is resumed at the specified line.
- `RESUME <line label>` Execution is resumed at the specified line label.
- `RESUME <HTTP>` Execution is resumed at the point where it was branched by an ON HTTP GOTO statement. Stdin and stdout are restored to their original values.

Examples
This short program is the basis for two examples of alternative subroutines:

```
10 ON ERROR GOTO 1000
20 FONT "Swiss 721 BT"
30 PRPOS 100,100
40 PRTXT "HELLO"
50 PRPOS 100, 300
60 PRIMAGE "GLOBE.1"
70 PRINTFEED
80 END
```

1. A font is selected automatically and execution is resumed from the line after where the error occurred. If another error than the specified error condition occurs, the execution is terminated.

```
1000 IF ERR=15 THEN FONT "Swiss 721 BT":RESUME NEXT
1010 RESUME 80
```

2. An error message is displayed and the execution goes on from the line following the one where the error occurred.

```
1000 IF ERR=15 THEN PRINT "Font not found"
1010 RESUME NEXT
```
RETURN

Purpose
Statement for returning to the main program after having branched to a subroutine because of a GOSUB statement.

Syntax

```
RETURN[<ncon>|<line label>]
```

<ncon> is optionally the number or label of a line in the main program to return to.

Remarks
When the statement RETURN is encountered during the execution of a subroutine, the execution will return to the main program. Execution will continue from the statement immediately following the most recently executed GOSUB or from an optionally specified line.

If a RETURN statement is encountered without a GOSUB statement having been previously executed, Error 28, “Return without Gosub” will occur.

Example

```
10 PRINT "This is the main program"
20 GOSUB 1000
30 PRINT "You’re back in the main program"
40 END
1000 PRINT "This is subroutine 1"
1010 GOSUB 2000
1020 PRINT "You’re back in subroutine 1"
1030 RETURN
2000 PRINT "This is subroutine 2"
2010 GOSUB 3000
2020 PRINT "You’re back in subroutine 2"
2030 RETURN
3000 PRINT "This is subroutine 3"
3010 PRINT "You’re leaving subroutine 3"
3020 RETURN
RUN
```

yields:

```
This is the main program
This is subroutine 1
This is subroutine 2
This is subroutine 3
You’re leaving subroutine 3
You’re back in subroutine 2
You’re back in subroutine 1
You’re back in the main program
```


**REWINDCONTROL**

**Purpose**

Function to control the internal rewind motor in EasyCoder PX-series printers.

**Syntax**

```
REWINDCONTROL <nexp>
```

<\text{nexp}> Positive values: Specifies for how long the internal rewinder will pull liner before paper feed motor starts. Maximum value is 5000 dots.

Negative values: Specifies the number of dots the paper will be fed before the internal rewind motor starts pulling liner. A value of -1 turns the rewind motor off. Maximum value is label length.

**Remarks**

EasyCoder PX-series printers may have two separate motors, one for feeding paper and one for rewinding liner. REWINDCONTROL allows for control of the internal rewind motor. A positive value stretches the liner before printing to ensure a better dispensing of labels. A positive value might cause stepper motor to stall for certain liner surfaces. In such cases, contact your local Intermec support organization. A negative value allows the label to be rewound with the liner using the self-strip mode. A value of -1 turns the internal rewind motor off, which makes for more silent operation when the printer runs without self-strip. This command cannot be used when REWINDEVOID ON is enabled.

**Examples**

```
REWINDCONTROL 200
REWINDCONTROL -100
```
**REWINDVOID**

**Purpose**
Function to rewind VOID labels with the liner in EasyCoder PX-series printers with RFID enabled.

**Syntax**
REWINDVOID OFF|ON[<nexp>]

<nexp> specifies the number of dots the paper will be fed before the internal rewind motor will start pulling liner. Default: 100. Maximum value is label length.

**Remarks**
EasyCoder PX-series printers may have two separate motors, one for feeding paper and one for rewinding liner. This function causes a VOID tag/label to be rewound with the liner paper when using the printer in self-strip mode, avoiding a label being from being dispensed (i.e. to an applicator). The value for <nexp> must be tested for each specific media and label length.

**Example**
REWINDVOID ON 200
**RIGHT$**

**Purpose**
Function returning a specified number of characters from a given string starting from the extreme right side (end) of the string.

**Syntax**

```
RIGHT$(<sexp>,<nexp>)
```

- `<sexp>` is the string from which the characters will be returned.
- `<nexp>` specifies the number of characters to be returned.

**Remarks**
This function is the complementary function for LEFT$, which returns the characters starting from the extreme left side, that is from the start.

If the number of characters to be returned is greater than the number of characters in the string, then the entire string will be returned. If the number of characters is set to zero, a null string will be returned.

**Examples**

```
PRINT RIGHT$("THERMAL_PRINTER", 7)
```

yields:

```
PRINTER
```

```
10   A$="THERMAL_PRINTER" ; B$ = "LABEL"
20   PRINT RIGHT$(B$, 5); RIGHT$(A$, 8); "S"
RUN
```

yields:

```
LABEL_PRINTERS
```
RSET

Purpose
Statement for placing data right-justified into a field in a random file buffer.

Syntax
RSET<svar>=<sexp>

<svar> is the string variable assigned to the field by a FIELD statement.
<sexp> holds the input data.

Remarks
After having OPENed a file and formatted it using a FIELD statement, you can enter data into the random file buffer using the RSET and LSET statements (LSET left-justifies the data).

The input data can only be stored in the buffer as string expressions. Therefore, a numeric expression must be converted to string by the use of a STR$ function before an LSET or RSET statement is executed.

If the length of the input data is less than the field, the data will be right justified and the remaining number of bytes will be printed as space characters.

If the length of the input data exceeds the length of the field, the input data will be truncated on the left side.

Example

```plaintext
10 OPEN "PHONELIST" AS #8 LEN=26
20 FIELD#8,8 AS F1$, 8 AS F2$, 10 AS F3$
30 SNAME$="SMITH"
40 CNAME$="JOHN"
50 PHONE$="12345630"
60 LSET F1$=SNAME$
70 LSET F2$=CNAME$
80 RSET F3$=PHONE$
90 PUT #8,1
100 CLOSE#8
RUN

SAVE "PROGRAM 1.PRG"

NEW
10 OPEN "PHONELIST" AS #8 LEN=26
20 FIELD#8,8 AS F1$, 8 AS F2$, 10 AS F3$
30 GET #8,1
40 PRINT F1$,F2$,F3$
RUN
```
yields:

SMITH ↔ ↔ ↔ JOHNNN12345630
**RUN**

**Purpose**

Statement for starting the execution of a program.

**Syntax**

```
RUN[<<scon>|<ncon>>]
```

- `<scon>` optionally specifies an existing program to be run.
- `<ncon>` optionally specifies the number of a line in the current program where the execution will start.

**Remarks**

The RUN statement starts the execution of the program currently residing in the printer’s working memory, or optionally of a specified program residing elsewhere. The execution will begin at the line with the lowest number, or optionally from a specified line in the current program.

If a program stored in another directory than the current one (see CHDIR statement), and has not been LOADed, its designation must be preceded by a reference to that device ("/c", "tmp:", "/rom", or "card1:", see the last example).

Never use RUN on a numbered line or in a line without number in the Programming Mode, or Error 40, “Run statement in program” will occur.

A RUN statement executed in the Intermec Direct Protocol will make the printer switch to the Immediate Mode, that is it has the same effect as an INPUT OFF statement.

**Examples**

Order the execution of a program this way:

```plaintext
RUN
Executes the current program from its first line.

RUN 40
Executes the current program, starting from line 40.

RUN "TEST"
Executes the program “TEST.PRG” from its first line.

RUN "TEST.PRG"
Executes the program “TEST.PRG” from its first line.

RUN "/rom/FILELIST.PRG"
Executes the program “FILELIST.PRG”, which is stored in the read-only memory, from its first line.
```
**SAVE**

**Purpose**
Statement for saving a file in the printer’s memory or optionally in a memory card.

**Syntax**

```
SAVE<scon>[,P|L]
```

- `<scon>` is the name of the file, optionally starting with a reference to a directory (see DEVICES).
  - Allowed input: Max. 30 characters incl. extension.
  - Max. 26 characters excl. extension
- P optionally protects the file.
- L optionally saves the file without line numbers.

**Remarks**

When a file is SAVEd, it must be given a designation consisting of max. 30 characters including extension. By default, the program will automatically append the name with the extension .PRG and convert all lowercase characters to uppercase. The name must not contain any quotation marks ("."). By starting the file name with a period character (.), you can avoid having it removed at a soft formatting operation, see FORMAT statement. Such a file will also be listed differently, see FILES statement.

Files can only be SAVEd in the printer’s permanent memory ("/c"), the printer’s temporary memory ("tmp:"), or in an optional CompactFlash memory card ("card1:"). If a file with the selected name already exists in the selected directory, that file will be deleted and replaced by the new file without any warning. You can continue to work with a file after saving it, until a NEW, LOAD, KILL, or REBOOT instruction is issued.

A protected file (SAVE <filename>,P) is encrypted at saving and cannot be LISTed after being LOADed. Program lines cannot be removed, changed, or added. Once a file has been protected, it cannot be deprotected again. Therefore, it is advisable to save an unprotected copy, should a programming error be detected later on. If you are going to use an electronic key to prevent unauthorized access to a file, you should protect it.

A SAVEd program can be MERGEd with the program currently residing in the printer’s working memory. To avoid line numbers automatically assigned to the program from interfering with the line numbers in the current program, you can choose to SAVE the program without line numbers (SAVE <filename>,L). See MERGE instruction.

**Examples**

```
SAVE "Label14"
```
saves the file as “LABEL 14.PRG” in current directory.

```
SAVE "/c/Label14",P
```
saves and protects the file "LABEL14.PRG" in the permanent memory.

```
SAVE "card1:Label14",L
```
saves "LABEL14.PRG” without line numbers on a CompactFlash card.
**SET FAULTY DOT**

**Purpose**
Statement for marking one or several dots on the printhead as faulty, or marking all faulty dots as correct.

**Syntax**

```
SET Faulty Dot <nexp>[, <nexpn>...]
```

- `<nexp>` is the number of the dot to be marked as faulty. Successive executions add more faulty dots.
- `<nexpn> = -1` marks all dots as correct (default).

**Remarks**
This statement is closely related to the HEAD function and the BARADJUST statement. You can check the printhead for possible faulty dots by means of the HEAD function and mark them as faulty, using the SET FAULTY DOT statement. Using the BARADJUST statement, you can allow the firmware to automatically reposition horizontal barcodes sideways so as to place the faulty dots between the bars, where no harm to the readability will be done.

This command will work on the EasyCoder PD41, yet the HEAD command cannot be used to identify faulty dots.

Once a number a dot has been marked faulty by a SET FAULTY DOT statement, it will remain so until all dots are marked as correct by a SET FAULTY DOT -1 statement.

Note that the HEAD function makes it possible to mark all faulty dots using a single instruction instead of specifying each faulty dot in a SET FAULTY DOT.

**Example**
This example illustrates how a bar code is repositioned by means of BARADJUST when a number of dots are marked as faulty by a SET FAULTY DOTS statement. Type RUN and send various numbers of faulty dots from the host a few times and see how the bar code moves sideways across the label.

```
10 INPUT "No. of faulty dots" ; A%
20 FOR B% = 1 TO A%
30 C% = C% + 1
40 SET FAULTY DOT C%
50 NEXT
60 D% = A%+2
70 BARADJUST D%, D%
80 PRPOS 0, 30
90 BARTYPE "CODE39"
100 PRBAR "ABC"
110 SET FAULTY DOT -1
120 PRINTFEED
RUN
```
SETASSOC

Purpose
Statement for setting a value for a tuple in a string association.

Syntax

\[
\text{SETASSOC } <\text{sexp}_1>, <\text{sexp}_2>, <\text{sexp}_3>
\]

- \(<\text{sexp}_1>\) is the name of the association (case-sensitive).
- \(<\text{sexp}_2>\) is the name of the tuple
- \(<\text{sexp}_3>\) is the value of the tuple.

Remarks
An association is an array of tuples, where each tuple consists of a name and a value.

Example
This example shows how a string, including three string names associated with three start values, will be defined and one of them (time) will be changed:

```sql
10  QUERYSTRING$="time=UNKNOWN&label=321&desc=DEF"
20  MAKEASSOC "QARRAY",QUERYSTRING$,"HTTP"
30  QTIME$=GETASSOC$("QARRAY","time")
40  QLABELS%=VAL(GETASSOC$("QARRAY","label"))
50  QDESC$=GETASSOC$("QARRAY","desc")
60  PRINT"time=";QTIME$,"LABEL=";QLABELS$,
   "DESCRIPTION=";QDESC$
70  SETASSOC "QARRAY","time",time$
80  PRINT "time=";GETASSOC$("QARRAY","time")
RUN
```

yields:

```
time=UNKNOWN  LABEL=321  DESCRIPTION=DEF
time=153355
```
SETPFSVAR

Purpose
Statement for registering variable to be saved at power failure.

Syntax
```
SETPFSVAR<sexp>,<nexp>
```

- `<sexp>` is the name of a numeric or string variable (uppercase characters only).
- `<nexp>` is the size in bytes of a string variable (max. 230).

Remarks
When a program is loaded, it is copied to and executed in the printer’s temporary memory (“tmp:”). Should an unexpected power failure occur, the printer tries to save as much data as possible in the short time available before all power is lost. To minimize the risk of lose important variable values at a power failure, you can register numeric and string variables to be saved. There is 2176 bytes (incl. overhead) available for this purpose.

However, should the power failure occur while the printer is printing, there will not be any power left to save the current variables.

When you register a string variable, you must also specify its size in bytes.

The variable name is limited to a length of twenty characters.

**Note:** The EasyCoder PD41 lacks power-fail signal, and thus it cannot save variables at unexpected power failures.

Related instructions are GETPFSVAR, DELETPFSVAR, and LISTPFSVAR.

Examples

**Example with string variable:**
```
100   IF QA$="" THEN QA$="Hello":QA%=LEN(QA$)
110   SETPFSVAR "QA$",QA%
```

**Example with numeric variable:**
```
200   SETPFSVAR"QCPS%"
```
**SETSTDIO**

**Purpose**
Statement for selecting standard IN and OUT communication channel.

**Syntax**

```
SETSTDIO<nexp_1>[,<nexp_2>]
```

- `<nexp_1>` is the desired input/output channel:
  - `100 = autohunting enabled (default)`
  - `0 = "console:"`
  - `1 = "uart1:"`
  - `2 = "uart2:"`
  - `3 = "uart3:"`
  - `4 = "centronics:"`
  - `5 = "net1:"`
  - `6 = "usb1:"`
  - `7 = "uart4:"`
  - `8 = "uart5"`

- `<nexp_2>` optionally specifies an output channel other than the input channel:
  - `0 = "console:"`
  - `1 = "uart1:"`
  - `2 = "uart2:"`
  - `3 = "uart3:"`
  - `5 = "net1:"`
  - `6 = "usb1:"`
  - `7 = "uart4:"`
  - `8 = "uart5"

**Remarks**
The printer is controlled from its host via a communication channel. By default, autohunting is selected. Autohunting means that all available channels are continuously scanned for input. When data is received on a channel, it is regarded as standard input/output channel. If no data is received on the present standard input channel within a 2 second timeout period, the firmware scans all other existing channels (except "console:" ) looking for input data. The channel where input data is first found will now be appointed the new stdin/stdout channel. The same procedure is repeated infinitely as long as autohunting is enabled.

There are some restrictions that apply to autohunting:
- If "centronics:" is used as input channel and autohunting is enabled, "uart1:" is selected stdout channel.
- Autohunting does not work with "console:".
- Autohunting does not work with COMSET, INKEY$, INPUT$, or LINE INPUT.
- When using autohunting in the programming mode, the detected port stays as STDIO until the program breaks.
- It is also possible to specify a certain channel as permanent stdin and/or stdout channel. If only one channel is specified, it will serve as both standard input (stdin) and standard output (stdout) channel. Alternatively, different channels can be selected for stdin and stdout.
Chapter 2—Program Instructions

SETSTDIO, cont.

For programming, it is recommended to use "uart1:" both as stdin and stdout channel. If another channel is selected, use the same serial channel for both input and output. The "centronics:" channel can only be used for input to the printer and is thus not suited for programming.

Example

This example selects the "uart2:" communication channel as the standard input and output channel:

```
10    SETSTDIO 2
.
.
.
.
```

This example enables autohunting for input and "uart1:" for output:

```
10    SETSTDIO 100,1
.
.
.
.
```


**SETUP**

**Purpose**

Statement for entering the printer’s Setup Mode or changing the setup.

**Syntax**

```plaintext
SETUP

If no parameter is specified, the printer enters the Setup Mode (no effect on EasyCoder PD41).

**SETUP <sexp>**

<sexp> is the name of an existing setup file that will be used to change the printer’s entire current setup, or a string used to change a single parameter in the printer’s current setup.

**SETUP <sexp>,<sexp>**

<sexp> is the name of a setup section (EasyLAN User’s Guide).
<sexp> is the name of a file that will be used to change the specified setup section.

**SETUP <sexp>,<sexp>,<sexp>**

<sexp> is the name of a setup section (see EasyLAN User’s Guide). Not implemented for "prt".
<sexp> is the name of the setup object (see EasyLAN User’s Guide).
<sexp> specifies the new value (see EasyLAN User’s Guide).

**Remarks**

The SETUP statement can be used for several purposes as illustrated above. Related instructions are SETUP GET and SETUP WRITE.

By default, the setup parameters are saved as a file in the printer’s permanent memory. However, using SYSVAR (35) it is possible to decide that any new change will not be saved (volatile). See SYSVAR.

The methods of manual setup via the printer’s built-in keyboard are described in the User’s Guides manuals for the various printer models. You can also use setup files and setup strings to change the setup as a part of the program execution, or to change the setup remotely from the host.

A setup file may contain new values for one or several setup parameters, whereas a setup string only can change a single parameter. Another difference is that, while the creation of setup files requires several operations, setup strings can be created in a single operation which makes them suitable for use with the Intermec Direct protocol.

When a SETUP<sexp> statement is encountered, the setup will be changed accordingly, then the program execution will be resumed. Note that some printing instructions (ALIGN, DIR, FONT, and PRPOS) may be changed when test labels are printed.
**SETUP, cont.**

Setup files or setup strings have a special syntax for each parameter that must be followed exactly. Variable numeric input data are indicated by "n" – "nnnnn", string data by "sssss", alternative data are indicated by bold characters separated by vertical bars (|). Compulsory space characters are indicated by double-headed arrows (↔). Do not type these. Note that some parameters listed below may only apply to a certain printer model or an optional device.

```
"SER-COM,UART1|UART2|UART3|UART4|UART5,BAUDRATE,300|600|1200|2400|4800|9600|19200|38400|57600|115200"
"SER-COM,UART1|UART2|UART3|UART4|UART5,CHAR_LENGTH,7|8"
"SER-COM,UART1|UART2|UART3|UART4|UART5,PARITY,NONE|EVEN|ODD|MARK|SPACE"
"SER-COM,UART1|UART2|UART3|UART4|UART5,STOPBITS,1|2"
"SER-COM,UART1|UART2|UART3|UART4|UART5,FLOWCONTROL,RTSCTS,ENABLE|DISABLE"
"SER-COM,UART1|UART2|UART3|UART4|UART5,FLOWCONTROL,ENQACK,ENABLE|DISABLE"
"SER-COM,UART1|UART2|UART3|UART4|UART5,FLOWCONTROL,XON/XOFF,DATA_FROM_HOST,ENABLE|DISABLE"
"SER-COM,UART1|UART2|UART3|UART4|UART5,FLOWCONTROL,XON/XOFF,DATA_TO_HOST,ENABLE|DISABLE"
"SER-COM,UART2,PROTOCOL_ADDR,ENABLE|DISABLE"
"SER-COM,UART1|UART2|UART3|UART4|UART5,NEW_LINE,CR/LF|LF|CR"
"SER-COM,UART1|UART2|UART3|UART4|UART5,REC_BUF,nnnnn"
"SER-COM,UART1|UART2|UART3|UART4|UART5,TRANS_BUF,nnnnn"
"SER-COM,UART2,PROTOCOL_ADDR,nn"
"NET-COM,NET1,NEW_LINE,CR/LF|LF|CR"
"NETWORK,IP_SELECTION,DHCP+BOOTP|MANUAL|DHCP|BOOTP"
"NETWORK,IP_ADDRESS,nnn.nnn.nnn.nnn"
"NETWORK,NETMASK,nnn.nnn.n.n"
"NETWORK,DEFAULT_ROUTER,nnn.nnn.nnn.nnn"
"NETWORK,NAME_SERVER,nnn.n.n.n"
"RFID,MODE,ENABLE|DISABLE"
"RFID,TAGADJUST,nnnnn" (negative value allowed)
"RFID,RETRIES,nn"
"RFID,VOIDTEXT,sssss"
"FEEDADJ,STARTADJ,nnnnn" (negative value allowed)
"FEEDADJ,STOPADJ,nnnnn" (negative value allowed)
"MEDIA,MEDIA_SIZE,XSTART,nnnnn"
"MEDIA,MEDIA_SIZE,WIDTH,nnnnn"
"MEDIA,MEDIA_SIZE,LENGTH,nnnnn"
"MEDIA,MEDIA_TYPE,LABEL,(w,GAPS)|TICKET,(w,MARK)|TICKET,(w,GAPS)|FIX_LENGTH_STRIP|VAR_LENGTH_STRIP"
"MEDIA,PAPER_TYPE,TRANSFER,DIRECT,THERMAL"
"MEDIA,PAPER_TYPE,DIRECT,Thermal LABEL_CONSTANT,nnnnn"
"MEDIA,PAPER_TYPE,DIRECT,Thermal LABEL_FACTOR,nnn"
"MEDIA,PAPER_TYPE,TRANSFER,CONSTANT,nnnnn"
"MEDIA,PAPER_TYPE,TRANSFER,FACTOR,nnnnn"
"MEDIA,PAPER_TYPE,TRANSFER,LABEL_OFFSET,nnnnn"
"MEDIA,PAPER_TYPE,TRANSFER,LOW_DIAMETER,nnnnn"
"MEDIA,CONTRAST,-10%|-8%|-6%|-4%|-2%|+0%|+2%|+4%|+6%|+8%|+10%"
"MEDIA,TESTFEED_MODE,FAST|SLOW"
"MEDIA,LEN(SLOW_MODE),nnn"
"MEDIA,PAPER_LOW_DIAMETER,nnn"
"PRINT_DEFS,PRINT_SPEED,nnn"
"PRINT_DEFS,PRINT_LTS_VALUE,nnn"
```
SETUP, cont.

Examples
This example enables a key for branching to the Setup Mode:

```
10  ON KEY(18) GOSUB 1000
20  KEY(18) ON
.....
1000 SETUP
1010 RETURN
```

This example shows how a new file is OPENed for output and each parameter in the setup is changed by means of PRINT# statements. Then the file is CLOSEd. Any lines, except the first and the last line in the example, may be omitted. Finally, the printer’s setup is changed using this file.

```
10  OPEN " /tmp/SETUP.SYS" FOR OUTPUT AS #1
20  PRINT#1, "SER-COM,UART1,BAUDRATE,19200"
30  PRINT#1, "SER-COM,UART1,CHAR LENGTH,7"
40  PRINT#1, "SER-COM,UART1,PARITY,EVEN"
50  PRINT#1, "SER-COM,UART1,STOPBITS,2"
60  PRINT#1, "SER-COM,UART1,FLOWCONTROL,RTS/CTS,ENABLE"
70  PRINT#1, "SER-COM,UART1,FLOWCONTROL,ENQ/ACK,ENABLE"
80  PRINT#1, "SER-COM,UART1,FLOWCONTROL,XON/XOFF,DATA FROM HOST,ENABLE"
90  PRINT#1, "SER-COM,UART1,FLOWCONTROL,XON/XOFF,DATA TO HOST,ENABLE"
100 PRINT#1, "SER-COM,UART1,NEW LINE,CR"
110 PRINT#1, "SER-COM,UART1,REC BUF,800"
120 PRINT#1, "SER-COM,UART1,TRANS BUF,800"
130 PRINT#1, "FEEDADJ,STARTADJ,-135"
140 PRINT#1, "FEEDADJ,STOPADJ,-36"
150 PRINT#1, "MEDIA,MEDIA SIZE,XSTART,50"
160 PRINT#1, "MEDIA,MEDIA SIZE,WIDTH,1000"
170 PRINT#1, "MEDIA,MEDIA SIZE,LENGTH,2000"
180 PRINT#1, "MEDIA,MEDIA TYPE,LABEL (w GAPS)"
190 PRINT#1, "MEDIA,PAPER TYPE,TRANSFER"
200 PRINT#1, "MEDIA,PAPER TYPE,TRANSFER,RIBBON CONSTANT,110"
210 PRINT#1, "MEDIA,PAPER TYPE,TRANSFER,RIBBON FACTOR,25"
220 PRINT#1, "MEDIA,PAPER TYPE,TRANSFER,LABEL OFFSET,00"
230 PRINT#1, "MEDIA,PAPER TYPE,TRANSFER,LOW DIAMETER,30"
240 PRINT#1, "MEDIA,CONTRAST,-4%"
250 PRINT#1, "PRINT DEFS,PRINT SPEED,200"
260 CLOSE
```

This example shows how a setup parameter is changed in the Immediate Mode or the Intermec Direct Protocol, using a setup string.

```
SETUP"MEDIA,MEDIA TYPE,VAR LENGTH STRIP"  ↓↓
```

This method can also be used in the Programming Mode, for example:

```
10 SETUP"MEDIA,MEDIA TYPE,VAR LENGTH STRIP"
```
Chapter 2—Program Instructions

SETUP GET

Purpose
Statement for getting the current setting for a single setup object.

Syntax
```
SETUP GET <sexp1>,<sexp2>,<sexp3>
```

- `<sexp1>` specifies the setup section.
- `<sexp2>` specifies the setup object.
- `<sexp3>` stores the result.

Remarks
Refer to *Intermec EasyLAN User's Guide* for a list of setup sections and objects.

Examples
```
SETUP GET "lan1","RTEL_PR1",A$
SETUP GET "prt","MEDIA,MEDIA TYPE",B$
SETUP GET "alerts","lts",C$
```
SETUP KEY

Purpose
Statement for enabling or disabling the access to the SETUP mode from the printer’s console.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETUP KEY ON</td>
<td>enables entering Setup mode through console.</td>
</tr>
<tr>
<td>SETUP KEY OFF</td>
<td>disables entering Setup mode through console.</td>
</tr>
</tbody>
</table>

Remarks
It may be desired to prevent operators or non-authorized personnel to change printer settings from the printer's console. SETUP KEY OFF allows the administrator to disable the SETUP key on the printer’s console, effectively disallowing changing of settings from the console. SETUP KEY ON reenables the SETUP key and access to the setup mode. The setup mode can always be entered via a SETUP command.


**SETUP WRITE**

**Purpose**

Statement for creating a file containing the printer's current setup or for returning it on a specified communication channel.

**Syntax**

\[
\text{SETUP WRITE} [<\text{sexp}_1>,<\text{sexp}_2>] \\
\]

- `<\text{sexp}_1>` is an optional parameter specifying the setup section.
- `<\text{sexp}_2>` is the name of a file or device to which the printer's current setup is to be written.

**Remarks**

The SETUP WRITE statement is useful when you want to return to the printer's current setup at a later moment. You can make a copy of the current setup using SETUP WRITE <filename>, change the setup using a SETUP <filename> statement, and, when so is required, return to the original setup by issuing a new SETUP <filename> statement containing the name of the file created by the SETUP WRITE <filename> statement.

It is strongly recommended to create the file in the printer's temporary memory ("tmp:"), for example SETUP WRITE "tmp:OLDSETUP". Once it has been created in "tmp:" it can be copied to the printer's permanent memory "/c" so it will not be lost at power off.

Another application of SETUP WRITE is transmitting the printer's current setup on a serial communication channel, for example SETUP WRITE "uart1:"

Setup sections are used in connection with EasyLAN. Refer to the *EasyLAN User's Guide* for a list of setup sections.

SETUP WRITE returns the printer’s setup in the following order (the example shows a standard EasyCoder PF4i printer):

```plaintext
SETUP WRITE "uart1:" yields:

SER-COM, UART1, Baudrate, 9600
SER-COM, UART1, Char Length, 8
SER-COM, UART1, Parity, None
SER-COM, UART1, StopBits, 1
SER-COM, UART1, Flowcontrol, RTS/CTS, Disable
SER-COM, UART1, Flowcontrol, ENQ/ACK, Disable
SER-COM, UART1, Flowcontrol, XON/XOFF, Data FROM Host, Disable
SER-COM, UART1, Flowcontrol, XON/XOFF, Data TO Host, Disable
SER-COM, UART1, New Line, CR/LF
SER-COM, UART1, Rec Buf, 1024
SER-COM, UART1, Trans Buf, 1024
FEEDADJ, STARTADJ, 0
FEEDADJ, STOPADJ, 0
MEDIA, Media Size, XStart, 24
MEDIA, Media Size, Width, 832
MEDIA, Media Size, Length, 1200
MEDIA, Media Type, Label (w Gaps)
```
Chapter 2—Program Instructions

SETUP WRITE, cont.

MEDIA, PAPER TYPE, TRANSFER
MEDIA, PAPER TYPE, DIRECT THERMAL, LABEL CONSTANT, 85
MEDIA, PAPER TYPE, DIRECT THERMAL, LABEL FACTOR, 40
MEDIA, PAPER TYPE, TRANSFER, RIBBON CONSTANT, 95
MEDIA, PAPER TYPE, TRANSFER, RIBBON FACTOR, 25
MEDIA, PAPER TYPE, TRANSFER, LABEL OFFSET, 0
MEDIA, PAPER TYPE, TRANSFER, RIBBON SENSOR, 14
MEDIA, PAPER TYPE, TRANSFER, LOW DIAMETER, 36
MEDIA, CONTRAST, +0%
#MEDIA, TESTFEED, 26 28 0 10
MEDIA, TESTFEED MODE, FAST
MEDIA, LEN (SLOW MODE), 0
PRINT DEFS, HEAD RESIST, 702
PRINT DEFS, PRINT SPEED, 100

Note that when a SETUP WRITE file is used to change the setup, the printer’s present TESTFEED adjustment is not affected.

Examples

In this example, the current setup is saved in the printer’s temporary memory under the name “SETUP1.SYS”. Then the start adjustment is changed to “200” by the creation of a new setup file named “SETUP2.SYS.” The setup file is finally used to change the printer’s setup.

10  SETUP WRITE "tmp:SETUP1.SYS"
20  OPEN "tmp:SETUP2.SYS" FOR OUTPUT AS #1
30  PRINT#1,"FEEDADJ,STARTADJ,200"
40  CLOSE
50  SETUP "tmp:SETUP2.SYS"

In this example, the setup section ”prt” is returned on the serial channel ”uart1:”:

SETUP WRITE "prt","uart1:"
SGN

Purpose
Function returning the sign (positive, zero, or negative) of a specified numeric expression.

Syntax
SGN(<nexp>)

<nexp> is the numeric expression from which the sign will be returned.

Remarks
The sign will be returned in this form:
SGN(<nexp>) = -1 (negative)
SGN(<nexp>) = 0 (zero)
SGN(<nexp>) = 1 (positive)

Examples
Positive numeric expression:
10   A% = (5+5)
20   PRINT SGN(A%)
RUN

yields:
1

Negative numeric expression:
10   A% = (5-10)
20   PRINT SGN(A%)
RUN

yields:
-1

Zero numeric expression:
10   A% = (5-5)
20   PRINT SGN(A%)
RUN

yields:
0
**SORT**

**Purpose**
Statement for sorting a one-dimensional array.

**Syntax**
```
SORT <<nvar>|<svar>>, <nexp1>, <nexp2>, <nexp3>
```
- `<<nvar>|<svar>>` is the array to be sorted.
- `<nexp1>` is the number of the first element.
- `<nexp2>` is the number of the last element.
- `<nexp3>` > 0: Ascending sorting
  - < 0: Descending sorting
  - = 0: Illegal value

In a string array, the value specifies the position according to which the array will be sorted.

**Remarks**
A numeric or string array can be sorted, in its entity or within a specified range of elements in ASCII value order.

The 4th parameter (`<nexp3>`) is used differently for numeric and string arrays. The sign always specifies ascending or descending order. For numeric arrays, the value is of no consequence, but for string arrays, the value specifies for which character position the elements will be sorted. `<nexp3>` = 0 results in Error 41, “Parameter out of range.”

**Example**
One numeric and one string array are sorted in descending order. The string array is sorted in ascending according to the third character position in each string:
```
10  ARRAY% (0) = 1001
20  ARRAY% (1) = 1002
30  ARRAY% (2) = 1003
40  ARRAY% (3) = 1004
50  ARRAY$ (0) = "ALPHA"
60  ARRAY$ (1) = "BETA"
70  ARRAY$ (2) = "GAMMA"
80  ARRAY$ (3) = "DELTA"
90  SORT ARRAY%, 0, 3, -1
100 SORT ARRAY$, 0, 3, 3
110 FOR I% = 0 TO 3
120 PRINT ARRAY% (I%), ARRAY$ (I%)
130 NEXT
RUN
```
yields:
```
1004 DELTA
1003 GAMMA
1002 ALPHA
1001 BETA
```
Chapter 2—Program Instructions

SOUND

Purpose
Statement for making the printer’s beeper produce a sound specified in regard of frequency and duration.

Syntax
```
SOUND<nexp1>,<nexp2>
```

- `<nexp1>` is the frequency of the sound in Hz.
- `<nexp2>` is the duration of the sound in periods of 0.020 sec. each (max. 15,0000 = 5 minutes).

Remarks
This statement allows you to include significant sound signals in your programs, for example to notify the operator that various errors have occurred. A sound with approximately the specified frequency will be produced for the specified duration. If the program encounters a new SOUND statement, it will not be executed until the previous sound has been on for the specified duration. No sound will come from the EasyCoder PD41, yet a time delay equal to the duration specified will occur.

The SOUND statement even allows you to make melodies, although the musical quality may be somewhat limited. The following table illustrates the frequencies corresponding to the notes in the musical scale. To create a period of silence, set the frequency to value higher than 9,999 Hz.

<table>
<thead>
<tr>
<th>Key</th>
<th>Hz</th>
<th>Key</th>
<th>Hz</th>
<th>Key</th>
<th>Hz</th>
<th>Key</th>
<th>Hz</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>131</td>
<td>C#</td>
<td>138</td>
<td>C</td>
<td>262</td>
<td>C#</td>
<td>277</td>
</tr>
<tr>
<td>C#</td>
<td>138</td>
<td>C#</td>
<td>138</td>
<td>D</td>
<td>294</td>
<td>D#</td>
<td>311</td>
</tr>
<tr>
<td>D</td>
<td>147</td>
<td>D#</td>
<td>155</td>
<td>E</td>
<td>330</td>
<td>E#</td>
<td>370</td>
</tr>
<tr>
<td>E</td>
<td>165</td>
<td>E#</td>
<td>165</td>
<td>F</td>
<td>349</td>
<td>F#</td>
<td>370</td>
</tr>
<tr>
<td>F</td>
<td>175</td>
<td>F#</td>
<td>185</td>
<td>G</td>
<td>392</td>
<td>G#</td>
<td>415</td>
</tr>
<tr>
<td>G</td>
<td>196</td>
<td>G#</td>
<td>208</td>
<td>A</td>
<td>440</td>
<td>A#</td>
<td>466</td>
</tr>
<tr>
<td>A</td>
<td>220</td>
<td>A#</td>
<td>233</td>
<td>B</td>
<td>494</td>
<td>B</td>
<td>523</td>
</tr>
<tr>
<td>A#</td>
<td>233</td>
<td></td>
<td></td>
<td>(small octave)</td>
<td>(one-line octave)</td>
<td>(two-line octave)</td>
<td>(three-line octave)</td>
</tr>
</tbody>
</table>

Example
The tune “Colonel Boogie” starts like this:
```
10 SOUND 392,10
20 SOUND 330,15
30 SOUND 330,10
40 SOUND 349,10
50 SOUND 392,10
60 SOUND 659,18
70 SOUND 659,18
80 SOUND 523,25
```
**SPACE$**

**Purpose**
Function returning a specified number of space characters.

**Syntax**

```
SPACE$(<nexp>)
```

- `<nexp>` is the number of space characters to be returned.

**Remarks**
This function is useful for more complicated spacing, for example in tables.

**Examples**
Printing of two left-justified columns on the screen:

```
10 FOR Q%=1 TO 6
20 VERBOFF:INPUT "",A$
30 VERBON:PRINT A$
40 VERBOFF:INPUT "",B$
50 VERBON
60 C$=SPACE$(25-LEN(A$))
70 PRINT C$+B$
80 NEXT Q%
90 END
RUN
```

Enter:
January
February
March
April
May
June
July
August
September
October
November
December

```
yields:
January                  February
March                    April
May                      June
July                     August
September                October
November                 December
```

*Intermec Fingerprint v8.70.0/v10.0.0 Programmer’s Reference Manual*
**SPLIT**

**Purpose**
Function splitting a string into an array according to the position of a specified separator character and returning the number of elements in the array.

**Syntax**

```
SPLIT(<sexp1>,<sexp2>,<nexp>)
```

- `<sexp1>` is the string to be split.
- `<sexp2>` is the string array in which the parts of the split string should be put.
- `<nexp>` specifies the ASCII value for the separator.

**Remarks**
The string is divided by a specified separating character which may found an infinite number of times in the string. Each part of the string will become an element in the string array, but the separator character itself will not be included in the array.

Should the string be split into more than four elements, Error 57, “Subscript out of range” will occur. To avoid this error, issue a DIM statement to create a larger array before the string is split.

**Example**
In this example a string is divided into five parts by the separator character # (ASCII 35 decimal). The result will be an array of five elements numbered 0-4 as specified by a DIM statement. Finally, the number of elements is also printed on the screen.

```
10  A$="ONE#TWO#THREE#FOUR#FIVE"
20  B$="ARRAY$"
30  DIM ARRAY$(5)
40  C%=SPLIT(A$,B$,35)
50  PRINT ARRAY$(0)
60  PRINT ARRAY$(1)
70  PRINT ARRAY$(2)
80  PRINT ARRAY$(3)
90  PRINT ARRAY$(4)
100 PRINT C%
RUN
```

```
yields:
ONE
TWO
THREE
FOUR
FIVE
5
```
**STOP**

**Purpose**
Statement for terminating execution of a program and to return to immediate mode.

**Syntax**
```
STOP
```

**Remarks**
When a STOP statement is encountered, the following message is returned to the Debug STDOUT channel (default is "uart1."):  

Break in line <line number>

You can resume execution where it was stopped by means of a CONT statement or at a specified program line using a GOTO statement in the immediate mode.

STOP is usually used in conjunction with CONT for debugging. When execution is stopped, you can examine or change the values of variables using direct mode statements. You may then use CONT to resume execution. CONT is invalid if the program has been edited during the break.

Related instructions are CONT and DBSTDIO.

**Example**

```
10   A%=100
20   B%=50
30   IF A%=B% THEN GOTO QQQ ELSE STOP
40   GOTO 30
50   QQQ:PRINT "Equal"
```

Ok
RUN
Break in line 30

Ok
PRINT A%
100

Ok
PRINT B%
50

Ok
B%=100

OK
CONT
Equal

Ok
Chapter 2—Program Instructions

STORE IMAGE

Purpose
Statement for setting up parameters for storing an image in the printer’s memory.

Syntax

\[
\text{STORE } \langle \text{sexp}_1 \rangle, \langle \text{nexp}_1 \rangle, \langle \text{nexp}_2 \rangle, \langle \text{nexp}_3 \rangle, \langle \text{sexp}_2 \rangle \\
\text{[RLL]} \quad \text{optionally indicates RLL compression.} \\
\text{[KILL]} \quad \text{optionally specifies that the image will be erased from the temporary memory at startup (recommended).} \\
\langle \text{sexp}_1 \rangle \quad \text{is the name of the image (max 30 char. incl. extension).} \\
\langle \text{nexp}_1 \rangle \quad \text{is the width of the image in bits (=dots).} \\
\langle \text{nexp}_2 \rangle \quad \text{is the height of the image in bits (=dots).} \\
\langle \text{nexp}_3 \rangle \quad \text{is the size of the images in bytes (RLL only).} \\
\langle \text{sexp}_2 \rangle \quad \text{is the name of the protocol:} \\
\text{"INTELHEX"} \\
\text{"UBI00"} \\
\text{"UBI01"} \\
\text{"UBI02"} \\
\text{"UBI03"} \\
\text{"UBI10"}
\]

Remarks
The name of the protocol must be entered in one sequence (for example "INTELHEX"). Upper- or lowercase letter can be used at will. Refer to the Chapter 3, “Image Transfer” for information on protocols.

STORE IMAGE RLL is used when the image to be received is compressed into RLL format. In this case the size of the image must be included in the list of parameters (\langle \text{nexp}_3 \rangle).

STORE IMAGE KILL implies that the image will be stored in the printer’s temporary memory, which is erased at power off or REBOOT. It is strongly recommended to use this option to improve the performance. If you need to store the image permanently, copy it from the temporary memory (‘tmp.’) to the permanent memory (‘/c’) after the download is completed.

A STORE IMAGE statement must precede any STORE INPUT statement.

Example
This example shows how an Intelhex file is received via the standard input channel and stored in the printer’s temporary memory:

\begin{verbatim}
10  STORE OFF
20  INPUT "Name:", N$
30  INPUT "Width:", W$
40  INPUT "Height:", H$
50  INPUT "Protocol:", P$
60  STORE IMAGE N$, W$, H$, P$
70  INPUT ",", F$
80  STORE F$
90  IF MID$(F$, 8, 2)<>"01" THEN GOTO 70
100 STORE OFF
\end{verbatim}
**STORE INPUT**

**Purpose**
Statement for receiving and storing protocol frames of image data in the printer’s memory.

**Syntax**

```
STORE \_<nexp_1> [,<nexp_2>]>
```

- `<nexp_1>` is the timeout in ticks (0.01 sec.) before next character is received.
- `<nexp_2>` is, optionally, the number assigned to a device when it was OPENed for INPUT. Default: Standard IN channel.

**Remarks**
The STORE INPUT statement receives and stores a protocol frame of image data as specified by preceding INPUT and STORE IMAGE statements. It also performs an end frame check. (STORE INPUT substitutes the old STORE statement (not documented in this manual.)

STORE INPUT works differently for various types of protocol:
- **INTELHEX** Receives and stores frames until timeout or end frame is received.
- **UBI00-03** Receives and stores frames until timeout or required number of bytes are received.
- **UBI10** Receives and stores frames until timeout or end frame is received.

**Examples**

This example shows how an Intelhex file is stored using the STORE IMAGE statement. The number of input parameters may vary depending on type of protocol, see STORE INPUT statement.

```
10  STORE OFF
20  INPUT "Name:" , N$
30  INPUT "Width:" , W%
40  INPUT "Height:" , H%
50  INPUT "Protocol:" , P$
60  STORE IMAGE N$, W%, H%, P$
70  STORE INPUT 100
80  STORE OFF
```

To receive the input from another channel than std IN channel, the device must be OPENed for INPUT and a reference be included in the STORE INPUT statement.

```
10  STORE OFF
20  OPEN "uart2:" FOR INPUT AS #9
30  INPUT "Name:" , N$
40  INPUT "Width:" , W%
50  INPUT "Height:" , H%
60  INPUT "Protocol:" , P$
70  STORE IMAGE N$, W%, H%, P$
80  STORE INPUT 100,9
90  CLOSE #9
100 STORE OFF
```
STORE OFF

Purpose
Statement for terminating the storing of an image and resetting the storing parameters.

Syntax
STORE OFF

Remarks
After having stored all protocol frames of an image, the storing must be terminated by a STORE OFF statement. Even if you want to store another image, you must still issue a STORE OFF statement before the parameters for the new image can be set up using a new STORE IMAGE statement.

It is recommended always to start an image storing procedure by issuing a STORE OFF statement to clear the parameters of any existing STORE IMAGE statement.

Example
This example shows how an Intelhex file is received via the standard IN channel and stored in the printer's memory:

10   STORE OFF
20   INPUT "Name:", N$
30   INPUT "Width:", W%
40   INPUT "Height:", H%
50   INPUT "Protocol:", P$
60   STORE IMAGE N$, W%, H%, P$
70   STORE INPUT 100
80   STORE OFF
**STR$**

**Purpose**
Function returning the string representation of a numeric expression.

**Syntax**

\[ \text{STR$(<nexp>)} \]

- `<nexp>` is the numeric expression from which the string representation will be returned.

**Remarks**
This is the complementary function for the VAL function.

**Example**
In this example, the value of the numeric variable A% is converted to string representation and assigned to the string variable A$:

```
10  A%=123
20  A$=STR$(A%)
30  PRINT A%+A%
40  PRINT A$+A$
RUN
```

yields:

```
246
123123
```
**STRING$**

**Purpose**
Function repeatedly returning the character of a specified ASCII value, or the first character in a specified string.

**Syntax**

```
STRING$(<nexp1>,<<nexp2>|<sexp>>)
```

- `<nexp1>` is the number of times the specified character should be repeated.
- `<nexp2>` is the ASCII decimal code of the character to be repeated.
- `<sexp>` is a string expression, from which the first character will be repeated.

**Remarks**
The character to be repeated is specified either by its ASCII decimal code according to the selected character set (see NASC), or as the first character in a specified string expression.

**Example**
In this example, both ways of using STRING$ are illustrated. The character “*” is ASCII 42 decimal:

```
10   A$="*INTERMEC*
20   LEADING$ = STRING$(10,42)
30   TRAILING$ = STRING$(10,A$)
40   PRINT LEADING$; A$; TRAILING$
```

Yields:

```
***********INTERMEC***********
```
**SYSHEALTH**

**Purpose**
Variable for setting or getting the Fingerprint application’s view of the system health and control the Intermec Ready-to-Work Indicator on the printer’s front panel.

**Syntax**

**Setting the system health:** `SYSHEALTH=<nexp>`

<nexp> sets the Fingerprint application’s view of the system health and controls the Intermec Ready-to-Work Indicator:

1. Indicator off
2. Indicator blink
3. Indicator on (default)

**Getting the system health:** `<nvar>=SYSHEALTH`

<nvar> returns the current system health status as shown by the Ready-to-Work Indicator:

1. translates to Indicator off
2. translates to Indicator blink
3. translates to Indicator on

**Remarks**

The readiness of the printer, individually or as a part of a solution, is indicated by the blue Intermec Ready-to-Work Indicator.

If the indicator blinks or is switched off, the printer is not ready. Further information can be obtained in the display window by pressing the `<F5/i>` key (not available on the EasyCoder PD41). In case of several errors or similar conditions occurring simultaneously, only the most significant error is displayed. Once this error has been cleared, next remaining error is displayed.

Provided the printer is connected to a network, all conditions that prevent printing are reported to the Easy ADC Console. The Easy ADC Console is a PC-based software which allows a supervisor to monitor all connected devices that have an Intermec Ready-to-Work Indicator, including handheld computers, access points, and printers.

The SYSHEALTH variable offers the programmer the opportunity to add more functions to the Ready-to-Work Indicator in an application than offered by the printer’s system (see the User’s Guide of the printer in question). However, it does not override the standard indicator handling; that is, the worst case is always reported regardless if it is a system error or an application error.
SYSHEALTH, cont.

Example

This example shows how the Ready-to-Work Indicator can be made to show a “Connection refused” condition:

```
10  ON ERROR GOTO 1000
50  TRANSFER NET
    "ftp://wrong.server.com/file","c/myfile"
60  PRINT "XXX"
100  END
1000 IF ERR=1833 THEN SYSHEALTH=2 ELSE
     SYSHEALTH=3
1010  RETURN
```

You can find out the health of the system this way:

```
A%=SYSHEALTH
PRINT A%
```
SYSHEALTH$

**Purpose**
Function for returning the error causing the current system health status.

**Syntax**

```
<svar>=SYSHEALTH$
```

<svar> returns the error causing the current status, for example “Head lifted.”

**Remarks**
If SYSHEALTH = 3, SYSHEALTH$ returns “Operational.”

**Example**

```
A$=SYSHEALTH$
PRINT A$
```

yields for example

Out of paper
## SYSVAR

**Purpose**

System array for reading or setting various system variables.

**Syntax**

SYSVAR(<nexp>)

- **<nexp>** is the reference number of the system variable:
  0-13 Reserved, obsolete or not implemented.
  14 Read errors since power on
  15 Read errors since last SYSVAR(15)
  16 Read number of bytes received at execution of STORE INPUT
  17 Read number of frames received at execution of STORE INPUT
  18 Read or Set verbosity level
  19 Read or Set type of error message
  20 Read direct or transfer mode
  21 Read printhead density (dots/mm)
  22 Read number of printhead dots
  23 Read status of transfer ribbon sensor
  24 Read if startup has occurred since last SYSVAR(24)
  25 Obsolete
  26 Read ribbon low condition
  27 Set condition for label reprinting at out-of-ribbon error
  28 Set or read media feed data erase at headlift
  29 Read DSR condition on "uart2:"
  30 Read DSR condition on "uart3:"
  31 Read last sent ACK, NAK, or CAN character in the MUSE protocol.
  32 Read odometer value
  33 Read DSR condition on "uart1:"
  34 Read or Set positioning mode for TrueType characters
  35 Setup saving (non-volatile/volatile)
  36 Print changes of program modes
  37 Set minimum gap length
  38 Obsolete
  39 Enable/disable slack compensation-Force startadjust
  40 Not implemented
  41 "Next label not found" at predefined feed length
  42 Stop media feed in the middle of label gaps
  43 Enable/disable file name conversion
  44 Enable/disable filtering of NUL characters in background communication.
  45 Read printhead resolution
  46 Read status of paper low sensor
  47 Enable/disable use of startadjust-stopadjust together with positive and/or negative formfeeds.
  48 Enable/disable direct commands
  49 Set temporary lower speed after negative startadjust.
  50 Set lower speed for given length after lowering printhead.
  51 Set enabled limit for function of SYSVAR(49) and (50).
SYSVAR, cont.

**Remarks**

0-13.
Reserved, obsolete, or not implemented.

14. Errors since power up
(Read only)
Reads number of errors detected since last power up.

15. Errors since last SYSVAR(15)
(Read only)
Reads number of errors detected since last executed SYSVAR(15).

16. Number of bytes received
(Read only)
Reads the number of bytes received after the execution of a STORE INPUT statement. Reset by the execution of a STORE IMAGE statement.

17. Number of frames received
(Read only)
Reads the number of frames received after the execution of a STORE INPUT statement. Reset by the execution of a STORE IMAGE statement.

18. Verbosity level
(Set or Read)
The verbosity level can be set or read.
In the Immediate and Programming Modes, all levels are enabled by default.
In the Intermec Direct Protocol, all levels are disabled by default.

Different verbosity levels can be selected:

SYSVAR (18) = -1 All levels enabled (= VERBON)
SYSVAR (18) = 0 No verbosity (= VERBOFF)
SYSVAR (18) = 1 Echo received characters
SYSVAR (18) = 2 “Ok” after correct command lines
SYSVAR (18) = 4 Echo input characters from comm. port
SYSVAR (18) = 8 Error after failed lines

The levels can be combined, so for example SYSVAR(18)=3 means both “Echo received characters” and “Ok after correct command line.”

The presently selected verbosity level can also be read and is returned as a numeric value, for example by PRINT SYSVAR(18).

19. Type of error message
(Set or Read)
Four types of error messages can be selected:

SYSVAR(19) = 1 <string> in line <line> (default)
for example “Invalid font in line 10”

SYSVAR(19) = 2 Error <number> in line <line>: <string>
for example “Error 19 in line 10: Invalid font”

SYSVAR(19) = 3 E<number>
for example “E19”

SYSVAR(19) = 4 Error <number> in line <line>
for example “Error 19 in line 10”

The presently selected type of error message can also be read and is returned as a numeric value (1-4), for example by PRINT SYSVAR(19).
SYSVAR, cont.

20. **Direct or transfer mode** (Read only)
SYSVAR(20) allows you to read if the printer is set up for direct thermal printing or thermal transfer printing, which is decided by your choice of paper type in the printer’s setup.

The printer returns:
0 = Direct thermal printing
1 = Thermal transfer printing

21. **Printhead density** (Read only)
SYSVAR(21) allows you to read the density of the printer’s printhead, expressed as number of dots per millimeter.

22. **Number of dots** (Read only)
SYSVAR(22) allows you to read the number of dots in the printer’s printhead.

23. **Transfer ribbon sensor** (Read only)
SYSVAR(23) allows you to read the status of the transfer ribbon sensor in thermal transfer printers.

The printer returns:
0 = No ribbon detected
1 = Ribbon detected

24. **Power up since last SYSVAR(24)** (Read only)
This system variable is important when using the Intermec Direct Protocol. At power up, all data not saved as programs, files, fonts or images will be deleted, and most instructions will be reset to their respective default values. SYSVAR(24) allows the host to poll the printer to see if a power up has occurred, for example because of a power failure and, if so, download new data and new instructions.

The printer returns:
0 = No power up since last SYSVAR(24)
1 = Power up has occurred since last SYSVAR(24)

25. **Obsolete**

26: **Ribbon low condition** (Read only)
This parameter allows you to read the status of the ribbon low sensor, assuming that the printer is fitted with a thermal transfer mechanism. In the Setup Mode (Media/Paper Type/Transfer/Low Diameter), you can specify a diameter in mm of the ribbon supply roll, when SYSVAR(26) will switch from 0 to 1.

The printer returns:
0 = Ribbon not low
1 = Ribbon low
By default, the Low Diameter is set to 0, which disables the ribbon low function. However, if the Low Diameter is set to a higher value than 0 and SYSVAR(26) returns 1, the error condition 1083 “Ribbon Low” occurs at every tenth PRINTFEED operation. Further actions must be taken care of by the running Fingerprint program.

**27: Condition for label reprinting at out-of-ribbon**  (Set or Read)
When printing a batch of labels using thermal transfer printing (OPTIMIZE "BATCH" ON or PRINTFEED<n>), a label is deemed erroneous and thus eligible for reprinting if the ribbon has been empty for a distance longer than specified in dots by SYSVAR(27). Default is 0. Non-negative integers only.

**28. Erase media feed data at headlift**  (Set or Read)
The firmware keeps track of all labels (or similar) between the label stop sensor and the dot line of the printhead. If the printhead is lifted, there is a large risk that the media is moved, so the media feed will not work correctly before those labels have been fed out. This parameter allows you to decide or read whether these data should be cleared or not when the printhead is lifted.

SYSVAR(28) = 0  Media feed data are not cleared at headlift
SYSVAR(28) = 1  Media feed data are cleared at headlift (default)
SYSVAR(28) = 2  Media feed data are cleared at headlift and the firmware looks for the first gap or mark and adjusts the media feed using the same data as before the head was lifted.

**29: DSR condition on "uart2:"**  (Read only)
This parameter allows you to read the DSR (Data Send Ready) condition on the serial channel "uart2:". The printer returns:

0 = No
1 = Yes

**30: DSR condition on "uart3:"**  (Read only)
This parameter allows you to read the DSR (Data Send Ready) condition on the serial channel "uart3:".

The printer returns:

0 = No
1 = Yes
SYSVAR, cont.

31: Last control character sent (Read only)
This parameter allows you to read the last control character sent from the MUSE protocol (special applications).

The printer returns one of the following alternatives:
- NUL
- ACK
- NAK
- CAN

32: Odometer value (Read only)
Returns the length of media feed past the printhead (the PD41 odometer cannot distinguish printheads, so it counts length for the platform).
Resolution: 10 meters.

33: DSR condition on "uart1:" (Read only)
This parameter allows you to read the DSR (Data Send Ready) condition on the serial channel “uart1:”. Not available on the EasyCoder PD41.

The printer returns:
- 0 = No
- 1 = Yes

34: TrueType character positioning mode (Set or Read)
This parameter allows you to select one of three modes for the positioning of TrueType characters and also to read for which mode the printer is set.

The modes are:
- 0 = Standard mode (default)
  This mode was introduced with Intermec Fingerprint 7.2.
- 1 = Compatible mode
  This mode is compatible with Intermec Fingerprint 7.xx earlier than version 7.2.
- 2 = Adjusted mode
  This mode was introduced with Intermec Fingerprint 7.2.

35: Setup Saving (Set or Read)
This parameter allows you to decide whether a change in the printer’s setup is to be saved as a file (that is be effective after a reboot or power down) or not be saved (volatile). You can also read for which alternative the printer is set. Note that the SYSVAR (35) setting at the moment when the new setup is entered decides whether it will be saved or not.

The alternatives are:
- 0 = Setup saved to file (non-volatile) Default
- 1 = Setup not saved to file (volatile)
SYSVAR, cont.

36: Print changes of program modes (Set or Read)
This parameter is used with the Fingerprint debugger and controls whether changes of program modes should be printed to the Debug Standard Out port (see DBSTDIO).

The options are:
0 = Disable printout (default)
1 = Enable printout

37: Set Minimum Gap Length (Set or Read)
The media may have perforations or marks that are not intended to be interpreted as gaps or black marks by the LSS. Using this SYSVAR parameters, it is possible to make the LSS ignore gaps or marks that are shorter than a specified value. (In this context, long and short are related to the media feed direction.) The minimum gap length is specified in dots within a range of 1-32. Default value is 1 mm (0.039 inches). Note that SYSVAR(37) affects PRINTFEED and FORMFEED. For TESTFEED, see SYSVAR(38).

38: Set Equal Safe for TESTFEED
This parameter is obsolete and has no effect, even if it does not cause an error if used.

39: Enable/Disable Slack Compensation-Force Startadjust (Set or Read)
Slack compensation:
Label slack compensation is a method of eliminating slack in the belts after having fed the media back. At a negative FORMFEED, the printer will pull back the media slightly more than specified by the FORMFEED statement and then feed the media forward the same distance. If slack compensation is enabled, and FORMFEED -100 is specified, the printer will pull back the media for example -112 dots and then feed the media forward +12 dots to take out the slack.

Force Startadjust:
Performing a positive FORMFEED will normally only feed out paper even if startadjust is set to a negative value. Using narrow labels and liner rewinding may cause the printout position to differ on the next printed label after a FORMFEED. Forcing a startadjust when doing FORMFEED improves printout precision in such cases.

The options are:
0 = Disable slack compensation and disable force startadjust.
1 = Enable slack compensation (default).
2 = Force startadjust at FORMFEED.
3 = Force startadjust at FORMFEED and enable slack compensation.

40: Not Implemented

41: “Next label not found” at Predefined Feed Length (Set or Read)
The automatic detection of the error condition “Next label not found” (error 1031) by the label stop sensor can be overridden by specifying a
fixed length in dots. The length should preferably correspond to at least the
distance between the tops of two consecutive labels. During printing, error
1031 occurs if the media does not come loose from the core (media glued
to core) or if a label is missing on the liner. Especially useful for short labels
(10–40 mm/0.4–1.5 inches long). Default value is 0.

42: Stop Media Feed in the Middle of Label Gaps (Set or Read)
0 = The media feed stops so the middle of a 3 mm (0.12 in) gap becomes
aligned with the tear bar when using labels (w gaps). This is the default
setting.
1 = The media feed stops so the middle of the gap becomes aligned with
the tear bar, regardless of gap size.

43: Enable/Disable File Name Conversion (Set or Read)
File name conversion means that lowercase characters will be converted to
uppercase and the extension .PRG will be added if an extension is missing.
0 = File name conversion is enabled (default)
1 = File name conversion is disabled.

44: Enable/Disable filtering of NUL characters (Set or Read)
SYSVAR(44) controls the filtering of NUL characters in background com-
munication (see COMBUF$).
0 = Enables filtering (default)
1 = Disables filtering

45: Read Printhead Resolution (Read only)
SYSVAR(45) returns the resolution of the printhead expressed in dots per
inch (dpi).

46: Read status of Paper Low Sensor (Read only)
0 = Indicates that the diameter of the media supply is larger than
specified in the Setup Mode.
1 = Indicates that the detected diameter of the media supply roll is equal
or less than the diameter specified in the Setup Mode (Media/Paper/
Low Diameter). The error condition 1084 “Paper low” will occur.
This error does not stop the printing, but interrupts any program that
does not handle it.

47: Enable/disable use of startadjust-stopadjust together with
positive and/or negative formfeeds. (Set or Read)
0 = Use of startadjust/stopadjust OR negative/positive Formfeed values.
Default.
1 = Enable the use of startadjust/stopadjust values together with negative/
positive Formfeed values.

48: Enable/Disable direct commands (Set or Read)
0 = Disables use of direct commands (default)
1 = Scans stdIN channel for direct commands. Can only be set in the
Direct Protocol.
49: Set temporary lower speed after negative startadjust.  (Set or Read)
This is used to avoid hard stretches/jerks at the beginning of a label with a full roll. Specify the percentage of the negative startadjust value that should be fed slower (70 mm/s). For example sysvar(49)=150 will cause the printer to run slower for 150% of the negative startadjust value, before speeding up.

50: Set lower speed after lowering printhead.  (Set or Read)
To avoid hard stretches/jerks after loading new media roll, this can lower the speed (to 70 mm/s) for the specified length after lowering the printhead. The value is given in dots. Default is 0.

51: Set enabled limit for of SYSVAR(49) and (50).  (Set or Read)
Set the value in meters that values for SYSVAR(49) and SYSVAR(50) will be in effect after lowering the printhead.

Examples

Reading the value of a system variable, in this case the transfer ribbon sensor:
PRINT SYSVAR(23)

Setting the value of a system variable. In this case verbosity is disabled:
SYSVAR(18) = 0
TAGFIELD

Purpose
Statement defining a field in the RFID tag’s memory area available for RFID operations.

Syntax
```
TAGFIELD[<sexp1>,][<sexp2>,[<nexp3>,[<nexp4>]]]
```

- `<sexp1>` is the optional name of the field to make available.
- `<sexp2>` is the name of the segment to make available. Allowed values:
  - Class 1 tags (EPC Class 1 Version 1):
    - "@ID" Tag identification segment for EPC data. Default.
  - ISO18000-6B / EPC UCode 1.19 tags:
    - "@ID" Tag identification segment. Default.
    - "@DATA" Optional data segment.
    - "@ALL" Tag’s complete memory structure.
  - Gen 2 tags (EPC Class 1 Generation 2):
    - "@RESERVED" Access and kill password segment.
    - "@EPC" EPC specific data. Default.
    - "@TID" Tag identifier.
    - "@USER" User defined data.
- `<nexp3>` is the field’s starting byte in the chosen segment. Default varies according to RFID tag standard and segment in use.
- `<nexp4>` is the length of the field in bytes. Default varies according to the RFID tag standard and EPCGlobal tag format in use.

Remarks
The TAGFIELD command is used to specify the field available for subsequent RFID commands, such as TAGWRITE or TAGREAD. The memory structure of RFID depends on the tag type. Class 1 tags only allow the "@ID" segment, where EPC data is stored. The ISO18000-6B, or EPC UCode 1.19 tags contains two segments, "@ID" and "@DATA". Gen 2 RFID tags have four segments, "@RESERVED", "@EPC", "@TID", and "@USER". Chapter 6 contains tables with the memory structure of supported tag types.

In order to support legacy programs written for Class 1 or ISO18000-6B tags when migrating to Generation 2 tags, Fingerprint will map the "@ID" and "@DATA" fields to "@EPC" and "@USER" fields respectively.

The `<nexp3>` and `<nexp4>` parameters specify a subset of a segment as the currently available field. It is possible to name the field to use with `<sexp1>`. This name may not start with an @-character. A possible representation of tag segments of a ISO18000-6B tag can be seen in the figure in the example. The default values for `<nexp3>` and `<nexp4>` may be modified by a subsequent TAGFORMAT command, see Chapter 6. Values for `<nexp3>` and `<nexp4>` do not normally need to be set when writing EPC information, as these are automatically adjusted for the EPC tag format specified. The only exception is ISO 18000-6B tags, see Chapter 6, Tag Memory Allocation.

The TAGFIELD command resets the TAGFORMAT to its default. Related commands are TAGFORMAT, TAGREAD and TAGWRITE.
**TAGFIELD, cont.**

**Examples**

The fields A, B, and C in the figure are defined by the commands below. The tag is a possible representation of an ISO18000-6B:

A: \[ \text{TAGFIELD } "@ID" \]
B: \[ \text{TAGFIELD } "@DATA", 0, 8 \]
C: \[ \text{TAGFIELD } "C", "@DATA", 8, 14 \]

The field name "C" can then be reused:

C: \[ \text{TAGFIELD } "C" \]
TAGFORMAT

Purpose
Statement specifying the format of the data to be read from or written to an RFID tag.

Syntax
```
TAGFORMAT <sexp>
```

<sexp> is the format of the data. Available formats are:
- "ASCII" 8-bit ASCII string.
- "HEX" Hexadecimal string. Values 0-9 and a-f allowed. Hex characters must be entered in pairs. Default.
- "NUM" Integer. 0 to 2147483647 allowed. Always uses 4 bytes to represent data, unless a field smaller has been defined, and the number fits in that field. Not allowed in Class1 tag.

**EPCGlobal tag formats:**
- "SGTIN-64" "Filter","Company Prefix","Item Reference","Serial Number"
- "SGTIN-96" "Filter","Company Prefix","Item Reference","Serial Number"
- "SSCC-64" "Filter","Company Prefix","Serial Reference"
- "SSCC-96" "Filter","Company Prefix","Serial Reference"
- "SGLN-64" "Filter","Company Prefix","Location Reference","Serial Number"
- "SGLN-96" "Filter","Company Prefix","Location Reference","Serial Number"
- "GRAI-64" "Filter","Company Prefix","Asset Type","Serial Number"
- "GRAI-96" "Filter","Company Prefix","Asset Type","Serial Number"
- "GIAI-64" "Filter","Company Prefix","Individual Asset Reference"
- "GIAI-96" "Filter","Company Prefix","Individual Asset Reference"
- "GID-96" "General Manager Number","Object Class","Serial Number"
- "USDOD-64" "Filter","Government Managed Identifier","Serial Number"
- "USDOD-96" "Filter","Government Managed Identifier","Serial Number"
- "EPC-HEX64" Hex values for all bits on a 64-bit tag’s memory area.
- "EPC-HEX96" Hex values for all bits on a 96-bit tag’s memory area.
- "EPC-URN" Uniform Resource Name string. Standardized format for entry of EPC identity.

Remarks
The TAGFORMAT command is used to specify the format of the data to be read from or written to a tag with a subsequent TAGREAD or TAGWRITE command. The format applies to the field defined by the most recent TAGFIELD command.

More information on tag formats can be found in Chapter 6.

TAGFORMAT is reset to its default by a TAGFIELD command. Related commands are TAGFIELD, TAGREAD and TAGWRITE.

Examples
```
10  FILTER$ = "3"
20  PREFIX$ = "0614141"
30  ITEM$ = "100734"
40  SERIAL$ = "2"
50  TAGFIELD @EPC
60  TAGFORMAT "SGTIN-96"
70  TAGWRITE FILTER$, PREFIX$, ITEM$, SERIAL$
```
**TAGFORMAT, cont.**

```
10  TAGFIELD "@ID"
20  TAGFORMAT "EPC-URN"
30  TAGREAD MyURI$
40  PRINT MyURI$
RUN
```

yields for example

```
urn:epc:sgtin-96:3.0614141.100734.2
```

```
10  TAGFIELD "@USER",10,4
20  TAGFORMAT "ASCII"
30  TAGWRITE "RFID"
40  TAGREAD A$
50  PRINT A$
RUN
```

yields

```
RFID
```
TAGPROTECT

Purpose
Statement used to protect tag data from being overwritten.

Syntax
TAGPROTECT <sexp1>

<sexp1> specifies the tag protection level. Available options are:
Class 1/ISO18000-6B/EPC 1.19 tags
"ON" Tag is protected.
"OFF" Tag remains unprotected.

Remarks
The TAGPROTECT command is used to protect the tag from being overwritten by an RFID operation. The protection is permanent. The command applies to the latest defined field. The command is executed at the next TAGWRITE operation, since it does not alone align the tag over the antenna.

Related commands are TAGFIELD, TAGFORMAT and TAGWRITE.

Examples
10 TAGFIELD "@ID"
20 TAGFORMAT "SSCC-64"
30 TAGPROTECT "ON"
40 TAGWRITE "1","12345","123456"
**TAGREAD**

**Purpose**
Statement used to read an RFID tag field.

**Syntax**

```
TAGREAD<nvar>|<svar1>[,<svar2>,<svar3>,<svar4>]
```

- `<nvar>` is the numeric variable to store the data. Only used with the "NUM" field in TAGFORMAT.
- `<svar_n>` is the string variable to store the data. The number of arguments depends on TAGFORMAT.

**Remarks**
The TAGREAD command reads the data from the field specified by the latest TAGFIELD command into the variable `<nvar>` or variables `<svar_n>`. The format of the data will be the one defined by the latest TAGFORMAT command. If you state a numeric variable `<nvar>`, an error is returned unless the data is in the "NUM" format.

Related commands are TAGFIELD, TAGFORMAT and TAGWRITE.

**Examples**

Using numeric variables:
```
10 TAGFIELD "@DATA",2,2
20 TAGFORMAT "NUM"
30 TAGWRITE 19562
40 TAGFIELD "@DATA",2,2
50 TAGFORMAT "NUM"
60 TAGREAD A%
RUN
```

yields
```
19562
```

Reading a an SGTIN-96 tag from a Gen 2 tag:
```
10 TAGFIELD "@EPC"
20 TAGFORMAT "SGTIN-96"
30 TAGREAD FILTER$, PREFIX$, ITEM$, SERIAL$
40 PRINT FILTER$, PREFIX$, ITEM$, SERIAL$
RUN
```

yields for example
```
3 0614141 100734 2
```
TAGWRITE

**Purpose**
Statement used to write to an RFID tag field.

**Syntax**

```
TAGWRITE<nvar>|<svar1>[,<svar2>,<svar3>,<svar4>]
```

- `<nvar>` is the numeric variable to be written. Only used with the "NUM" field in TAGFORMAT.
- `<svarN>` is the string variable to be written. The number of arguments depends on TAGFORMAT.

**Remarks**
The TAGWRITE command writes the data from `<nvar>` or `<svarN>` to the field specified by the latest TAGFIELD command. The format of the data will be the one specified by the latest TAGFORMAT command. If the data written is shorter than the field, the field is padded with zeroes. The exception to this rule occurs in some EPC tag formats where the exact number of digits must be entered, even if it means adding non-significant digits (for example writing 00234 instead of 234), see Chapter 6. If the data is too long to fit in the specified field, an error is returned.

Related commands are FORMAT$, TAGFIELD, TAGFORMAT and TAGREAD.

**Examples**
Writing an SGTIN-96 to a Gen 2 tag:
```
10   TAGFIELD "@EPC"
20   TAGFORMAT "SGTIN-96"
30   TAGWRITE "3","0614141","100734","2"
```

Two examples of writing the same SSCC-64 info to an ISO18000-6B tag:
```
10   TAGFIELD "@DATA",10,8
20   TAGFORMAT "SSCC-64"
30   TAGWRITE "1","12345","123456"
```

```
10   TAGFIELD "@DATA",10,8
20   TAGFORMAT "EPC-URN"
30   TAGWRITE "urn:epc:tag:sscc-64:1.12345.123456"
```
**TESTFEED**

**Purpose**

Statement for adjusting the label stop, ribbon end/low, paper low sensors, and RFID module while running the media and ribbon feed mechanisms.

**Syntax**

```
TESTFEED[<nexp>]
```

- `<nexp>` is an optional feed length in dots.

**Remarks**

A TESTFEED statement feeds `<nexp>` dots while calibrating the label stop/black mark sensor (LSS) for the characteristics of the media presently loaded in the printer. The statement is needed to detect media, gaps, black marks, and out-of-paper conditions. It should be done for all media types.

If an RFID module is installed, and RFID is enabled, TESTFEED will attempt to communicate and identify the RFID tag. This TESTFEED is always performed in SLOW MODE, see below.

If `<nexp>` is omitted, it is automatically set to 1.5 times the media length specified in the setup. For the TESTFEED to be successful, at least one gap or black mark must pass the LSS. Best result for “Ticket w Mark” is obtained if entering a `<nexp>` value of 1200 or any other reasonable number.

When a TESTFEED is executed, the ribbon end/low and paper low sensors are also calibrated (if installed). However, this does not apply when the testfeed is ordered using the testfeed option in the Setup Mode.

In the Immediate Mode, a TESTFEED is performed when the `<Shift>` and `<Feed>` keys are pressed simultaneously.

In the setup, TESTFEED MODE can be set to SLOW. This might be necessary when using media with pre-printed lines. This is done with a SETUP command or through the printer’s Setup Mode. When in SLOW mode, TESTFEED will sample the media length plus 10 mm. Alternatively, the length sampled can be set using the MEDIA,LEN (SLOW MODE) option, the minimum being the number of dots corresponding to 10 mm. This value is ignored when TESTFEED MODE is set to FAST.

Since the TESTFEED is essential for a proper media load, some facility for issuing a TESTFEED statement should be included in all custom-made label-printing programs (see the example below).

**Example**

This program performs a TESTFEED statement when the `<Shift>` and `<Feed>` key are pressed simultaneously on the printer’s built-in keyboard:

```
10  ON KEY (119) GOSUB QTESTFEED
20  KEY (119) ON
30  QLOOP:
40  GOTO QLOOP
     . . . . .
1000 QTESTFEED:
1010  TESTFEED
1020  RETURN
```
**TICKS**

**Purpose**
Function returning the time, that has passed since the last power up in the printer, expressed in number of “TICKS” (1 TICK = 0.01 seconds.)

**Syntax**
TICKS

**Remarks**
TICKS allows you to measure time more exactly than the TIME$ variable, which cannot handle time units smaller than 1 second.
The TICKS counter is reset to zero at power up.

**Example**
```
10  A%=TICKS
20  PRINT A%
RUN
```
yields for example:

1081287
The time which has passed since the printer was started is 10812.87 seconds, that is 3 hours 12.87 seconds.
**TIME$**

**Purpose**
Variable for setting or returning the current time.

**Syntax**

**Setting the time:**

\[
\text{TIME$=}<\text{sexp}>
\]

\(<\text{sexp}>\)
sets the current time by a 6-digit number specifying Hour, Minute and Second.

**Reading the time:**

\[
<\text{svar}>=\text{TIME$[<\text{sexp}>]}\]

\(<\text{svar}>\)
returns the current time according to the printer’s clock.
\(<\text{sexp}>\)
is an optional flag "F", indicating that the time will be returned according to the format specified by \text{FORMAT TIME$}.

**Remarks**
This variable works best if a real-time clock circuit (RTC) is fitted on the printer’s CPU board. The RTC is battery backed-up and will keep record of the time even if the power is turned off or lost.

If no RTC is installed, the internal clock will be used. After startup, an error will occur when trying to read the date or time before the internal clock has been manually set using either a \text{DATE$} or a \text{TIME$} variable. If only the date is set, the internal clock starts at 00:00:00 and if only the time is set, the internal clock starts at Jan 01 1980. After setting the internal clock, you can use the \text{DATE$} and \text{TIME$} variables the same way as when an RTC is fitted, until a power off or \text{REBOOT} causes the date and time values to be lost.

Time is always entered and, by default, returned as HHMMSS, where:

- HH = Hour Two digits (00-23)
- MM = Minute Two digits (00-59)
- SS = Second Two digit (00-59)

Time is entered as a 24-hour cycle, for example 8 o’clock pm is entered as "200000".

The clock will be reset at the exact moment, when the appending carriage return character is received, for example when you press the Return key (Immediate Mode and Intermec Direct Protocol), or when the instruction is executed (Programming Mode).

The format for how the printer will return time from a \text{TIME$}("F") variable can be changed using a \text{FORMAT TIME$} statement.

**Example**

Setting and reading the time, then printing it on the screen of the host:

\[
\begin{align*}
10 & \text{ TIME$ = "154300"} \\
20 & \text{ FORMAT TIME$ "HH.MM"} \\
30 & \text{ PRINT "Time is " + TIME$("F")}
\end{align*}
\]

\text{RUN}

yields:

Time is 15.43
**TIMEADD$**

**Purpose**
Function returning a new time after a number of seconds have been added to, or subtracted from, the current time or optionally a specified time.

**Syntax**

```plaintext
TIMEADD$([<sexp1>,]<nexp>[,<sexp2>])
```

- `<sexp1>` is any time given according to the TIME$ format, which a certain number of seconds should be added to or subtracted from.
- `<nexp>` is the number of seconds to be added to (or subtracted from) the current time, or optionally the time specified by `<sexp1>`.
- `<sexp2>` is an optional flag "F", indicating that the time will be returned according to the format specified by FORMAT TIME$.

**Remarks**
The original time (`<sexp1>`) should always be entered according to the TIME$ format (HHMMSS), where:

- **HH** = Hour, Two digits (00-23)
- **MM** = Minute, Two digits (00-59)
- **SS** = Second, Two digits (00-59)

Time is entered as a 24-hour cycle, for example 8 o'clock pm is entered as "200000".

The number of seconds to be added or subtracted from the original time should be specified as a positive or negative numeric expression respectively.

If no "F" flag is included in the TIMEADD$ function, the result will be returned according to the TIME$ format, see above.

If the TIMEADD$ function includes an "F" flag, the result will be returned in the format specified by FORMAT TIME$.

**Examples**

```plaintext
10 A%=30
20 B$=TIMEADD$ ("133050",A%)
30 PRINT B$
RUN
133120
```

```plaintext
10 TIME$="133050"
20 FORMAT TIME$ "hh.mm.ss pp"
30 A% = -40
40 PRINT TIMEADD$(A%,"F")
RUN
01.30.10 pm
```
**TIMEDIFF**

**Purpose**
Function returning the difference between two specified moments of time in number of seconds.

**Syntax**
```
TIMEDIFF(<sexp_1>,<sexp_2>)
```

- `<sexp_1>` is one of two moments of time (time 1).
- `<sexp_2>` is the other of the two moments (time 2).

**Remarks**
To get the result as a positive value, the two moments of time, for which the difference is to be calculated, should be entered with the earlier moment (time 1) first and the later moment (time 2) last, see the first example below.

If the later moment (time 2) is entered first, the resulting value will be negative, see the second example below.

The time should be entered according to the format for the TIME$ variable, that is in the order HHMMSS, where:

- **HH** = Hour Two digits (00-23)
- **MM** = Minute Two digits (00-59)
- **SS** = Second Two digits (00-59)

Time is entered as a 24-hour cycle, for example 8 o’clock pm is entered as "200000".

The resulting difference in seconds will be returned.

**Examples**
```
PRINT TIMEDIFF ("133050","133120")
yields: 30

PRINT TIMEDIFF ("133120","133050")
yields: -30
```
TRANSFER KERMIT

Purpose
Statement for transferring data files using KERMIT communication protocol.

Syntax
```
TRANSFER K[ERMIT]<sexp1>[,<sexp2>[,<sexp3>[,sexp4>]]]
```

- `<sexp1>` specifies the direction of the transmission by the expression "S" (= send) or "R" (= receive).
- `<sexp2>` is, optionally, the name of the file transmitted from the printer (default "KERMIT.FILE").
- `<sexp3>` specifies, optionally, the input device as "uart1:" , "uart2:" , "uart3:" , "uart4:" , or "uart5:" (default: std IN channel).
- `<sexp4>` specifies, optionally, the output device as "uart1:" , "uart2:" , "uart3:" , "uart4:" , or "uart5:" (default: std OUT channel).

Remarks
Kermit is a protocol for serial binary transfer of a complete file between for example a PC and a printer. Kermit is included in Windows HyperTerminal and in many other communication programs.

Warning, tests have shown that Microsoft Windows Terminal, versions 3.0 and 3.1, is unable to receive a file from the printer, even if capable of sending a file to the printer.


TRANSFER KERMIT can only handle one single file at a time.

When transmitting files from the printer to the host, carefully observe possible restrictions concerning the number of characters, etc. in the file name, that is imposed by the operating system of the host.

When receiving a file, you must start the transmission within 30 seconds from completing the TRANSFER KERMIT "R" statement. The printer will store the file in the current directory "/c", "tmp:”, or "card1:”, see CHDIR statement. (Obviously, files cannot be received into "/rom”.) If there already exists a file in the current directory with the same name as the one to be transferred, the existing file will be replaced by the new file. Thus, it is up to you to keep record of the files already stored in the current directory (see FILES statement). Before transfer, give the new file a name that is not already occupied by an existing file, unless you want to replace the existing file. Downloaded fonts and image are auto-installed.

Examples
Setting up the printer for file reception on the standard IN channel:
```
TRANSFER KERMIT "R"
```

Transmission from printer to host of the file "FILE1.TXT" on a channel other than the standard OUT channel:
```
TRANSFER K "S","FILE1.TXT","uart2:","uart2:"
```
Chapter 2—Program Instructions

**TRANSFER NET**

**Purpose**

Statement for transferring files to and from the printer using FTP.

**Syntax**

```plaintext
TRANSFER N[ET] <sexp_1>,<sexp_2>[,<sexp_3>]
```

- `<sexp_1>` is the source. If the source is a local file, this file will be sent from the printer to the destination specified by `<sexp_2>`. If the source is a URI, this file will be fetched from the server and sent to the printer and stored at the location specified by `<sexp_2>`.
- `<sexp_2>` is the destination of the file transfer.
- `<sexp_3>` is an optional account secret.

**Remarks**

**Limitations:**

TRANSFER NET is not a complete ftp client. It only supports file transfer to and from the printer in binary format. Only one file can be transferred per command. File transfer between two local or two remote files is not supported.

**Local files:**

A local file is a path to an existing file (when sending from the printer) or to the file that will be created (when fetching to the printer). If a local file already exists when fetching a file to the printer, the existing file will be replaced, if it is not write-protected. A read-protected file will not be sent. If the destination is a local directory, the fetched file will get the same name as the source file.

**URIs:**

An URI shall be entered in the format

```
ftp://[<user>:<password>@]<server>[:port]/<path>
```

Entries inside square brackets [...] are optional. The following default values are used:

- `user`: `anonymous`
- `password`: `nopass@<ip address>`
- `port`: `21`

If the destination is a URI specifying a directory, the sent file will get the same name as the source file.

**Account Secret:**

If the user does not want to reveal his/her username and password in plain text in a Fingerprint program, the account secret option can be used. The account secret holds the secret information and cannot be read by any user. To create an account secret, use the external command RUN "secret". Listing and deleting accounts secrets are reserved for admin.

**Fonts and Images:**

Downloaded fonts and images will be auto-installed.
TRANSFER NET, cont.

Example

This example shows how the file README.uploads is fetched from sunet's ftp server and stored as UPLOAD.TXT in the current directory. Default user, password and port number are used.

"UPLOAD.TXT"
TRANSFER STATUS

Purpose
Statement for checking last TRANSFER KERMIT or TRANSFER ZMODEM operation.

Syntax
TRANSFER STATUS<nvar>,<svar>

<nvar> is a five-element one-dimensional numeric array where the elements will return:
0: Number of packets. (Kermit only)
1: Number of NAK's. (Kermit only)
2: ASCII value of last status character. (Kermit only)
3: Last error. (Kermit and ZMODEM)
4: Block check type used. (Kermit only)

<svar> is a two-element one-dimensional string array where the elements will return:
0: Type of protocol. ("KERMIT" or "ZMODEM")
1: Last file name received.

Remarks
After a file transfer using the Kermit or ZMODEM protocol has been performed (see TRANSFER KERMIT and TRANSFER ZMODEM statements), you can check how the transfer was performed. Note that the numeric array requires the use of a DIM statement, since the array will contain more than four elements.

Example
10 TRANSFER KERMIT "R"
20 DIM A%(4)
30 TRANSFER STATUS A%, B$
40 PRINT A%(0), A%(1), A%(2), A%(3), A%(4)
50 PRINT B$(0), B$(1)
       ....
       ....
       ....
Chapter 2—Program Instructions

TRANSFER ZMODEM

Purpose
Statement for transferring data files using ZMODEM communication protocol.

Syntax
\[
\text{TRANSFER} \leftrightarrow \text{Z}[\text{MODEM}][<\text{sexp}_1>,[<\text{sexp}_2>,[<\text{sexp}_3>,<\text{sexp}_4>]]]
\]

- \(<\text{sexp}_1>\)
  - specifies the direction of the transmission by the expression "S" (= send) or "R" (= receive).
- \(<\text{sexp}_2>\)
  - is, optionally, the name of the file transmitted from the printer (default "ZMODEM.FILE").
- \(<\text{sexp}_3>\)
  - specifies, optionally, the input device as "uart1:", "uart2:", "uart3:", "uart4:", or "uart5:" (default: std IN channel).
- \(<\text{sexp}_4>\)
  - specifies, optionally, the output device as "uart1:", "uart2:", "uart3:", "uart4:", or "uart5:" (default: std OUT channel).

Remarks
ZMODEM is a protocol for serial transfer of a complete file between for example a PC and a printer. For more information on the ZMODEM protocol, please refer to http://www.omen.com. Related instructions are the external commands RZ (receive data using the ZMODEM protocol) and SZ (send data using the ZMODEM protocol).

TRANSFER ZMODEM can only handle one single file at a time.

When transmitting files from the printer to the host, carefully observe possible restrictions concerning the number of characters etc. in the file name, that is imposed by the operating system of the host.

When receiving a file, you must start the transmission within 30 seconds from completing the TRANSFER ZMODEM "R" statement. The printer will store the file in the current directory "/c", "tmp:“, or "card1:“, see CHDIR statement. (Obviously, files cannot be received into "/rom"). If there already exists a file in the current directory with the same name as the one to be transferred, the existing file will be replaced by the new file. Thus, it is up to you to keep record of the files already stored in the current directory (see FILES statement). Before transfer, give the new file a name that is not already occupied by an existing file, unless you want to replace the existing file. If you use TRANSFER ZMODEM to download a font or image file, the font or image will automatically be installed after the downloading is completed without any need for a reboot.

Examples
Setting up the printer for file reception on the standard IN channel:
TRANSFER ZMODEM "R"

Transmission from printer to host of the file "FILE1.TXT" on a channel other than the standard OUT channel:
TRANSFER Z "S","FILE1.TXT","uart2:","uart2:"
# TRANSFER$ Function

**Purpose**
Function executing a transfer from source to destination as specified by a TRANSFERSET statement.

**Syntax**
```
TRANSFER$(<nexp>)
```

- `<nexp>` is the character time-out in ticks (10 milliseconds).

**Remarks**
The TRANSFER$ function executes the transfer from source to destination as specified by the TRANSFERSET statement. It also checks the transfer and breaks it if no character has been transmitted before the specified time-out has expired or if any break character, as specified by the break character string in the TRANSFERSET statement, is encountered.

- If the transmission was interrupted because a character in the break set was encountered, that character will be returned.
- If the transmission was interrupted because of a time-out error, an empty string will be returned.
- If the transmission was interrupted because of the reception of a character on any other communication channel than the source (as specified by TRANSFERSET statement), an empty string will be returned.

**Example**
The transfer will be executed by the TRANSFER$ function in line 60 and possible interruptions will be indicated by a break character or empty string ("") in the string variable C$.

```
10 OPEN "LABEL1.PRG" FOR INPUT AS #1
20 OPEN "UART1:" FOR OUTPUT AS #2
30 A$=CHR$(13)
40 B$=CHR$(10)
50 TRANSFERSET #1, #2, A$+B$
60 C$=TRANSFER$(100)

.......
.......
```
# TRANSFERSET

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Statement for entering setup for the TRANSFER$ function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>TRANSFERSET[#]&lt;nexp_1&gt;,[#]&lt;nexp_2&gt;,&lt;sexp&gt;[,&lt;nexp_3&gt;]</td>
</tr>
<tr>
<td>#</td>
<td>optional number sign.</td>
</tr>
<tr>
<td>&lt;nexp_1&gt;</td>
<td>is the number of the source (the file or device OPENed for input).</td>
</tr>
<tr>
<td>&lt;nexp_2&gt;</td>
<td>is the number of the destination file (the file or device OPENed for output or append).</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>is a set of break characters.</td>
</tr>
<tr>
<td>&lt;nexp_3&gt;</td>
<td>optionally enables or disables break on any other channel than the source:</td>
</tr>
<tr>
<td></td>
<td>&lt;nexp&gt; = 0    Break disabled</td>
</tr>
<tr>
<td></td>
<td>&lt;nexp&gt; ≠ 0    Break enabled</td>
</tr>
<tr>
<td>Default:</td>
<td>Standard I/O with no break characters.</td>
</tr>
<tr>
<td></td>
<td>Break on any other channel enabled.</td>
</tr>
</tbody>
</table>

| Remarks                  | This statement sets up the transfer of data from a file or device OPENed for input to another file or device OPENed for output or append. The transfer will be interrupted if any character in a string of break characters, specified in this statement, is encountered (optionally on another specified channel). The actual transfer is executed by means of a TRANSFER$ function, that also returns the break character that has caused any possible interruption. |

| Example                  | In this example, the data transfer from a file in the current directory to an external device connected to the communication port "uart1:" will be interrupted as soon as a carriage return or a line feed character is encountered in the file. |
|                         | 10 OPEN "LABEL1.PRG" FOR INPUT AS #1                   |
|                         | 20 OPEN "uart1:" FOR OUTPUT AS #2                     |
|                         | 30 A$=CHR$(13)                                         |
|                         | 40 B$=CHR$(10)                                         |
|                         | 50 TRANSFERSET #1, #2, A$+B$                         |
|                         | 60 C$=TRANSFER$(100)                                  |
|                         | .......                                               |
|                         | .......                                               |
|                         | .......                                               |
Chapter 2—Program Instructions

**TRON/TROFF**

**Purpose**
Statements enabling/disabling tracing of the program execution.

**Syntax**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRON</td>
<td>Enables tracing.</td>
</tr>
<tr>
<td>TROFF</td>
<td>Disables tracing (default)</td>
</tr>
</tbody>
</table>

**Remarks**
Useful for debugging purposes. When tracing is enabled, each line number of the program is displayed on the screen within parentheses as the execution goes on.

Tracing will be disabled when a TROFF statement is executed.

**Example**

```
10 PRINT "HELLO"
20 INPUT"Enter Text"; A$
30 PRINT A$
TRON
RUN
```

Yields:

(10) WORLD
(20) Enter test? (Operator enters "WORLD")
(30) WORLD
Chapter 2—Program Instructions

VAL

Purpose
Function returning the numeric representation of a string expression.

Syntax
\[
\text{VAL(<sexp>)}
\]

<sexp> is the string expression from which the numeric representation will be returned.

Remarks
VAL is the complementary function for STR$.

VAL ignores space characters from the argument string to determine the result.

If the first character in the string expression is anything else but a digit, a plus sign, or a minus sign, the VAL function returns the value 0.

Example
In this example, the values of the string variables A$ and B$ are read and assigned to the numeric variables A% and B%:

```Basic
10 A$ = "123, MAIN STREET"
20 A% = VAL(A$)
30 B$ = "PHONE 123456"
40 B% = VAL(B$)
50 PRINT A$
60 PRINT A%
70 PRINT B$
80 PRINT B%
RUN
```

yields:

```
123, MAIN STREET
123
PHONE 123456
0
```
VERBON/VERBOFF

**Purpose**
Statements for specifying the verbosity level of the communication from the printer on the standard OUT channel (serial communication only).

**Syntax**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERBON</td>
<td>Enables all verbosity levels (default).</td>
</tr>
<tr>
<td>VERBOFF</td>
<td>Disables all verbosity levels.</td>
</tr>
</tbody>
</table>

**Remarks**

**VERBON**
By default, when a character is received on the standard IN channel (see SETSTDIO statement), the corresponding character will be echoed back on the standard OUT channel. As the serial channel "uart1:" is by default selected both standard IN and OUT channel, this implies that when you enter a character on the keyboard of the host, the same character will appear on the screen after having been transmitted to the printer and back.

When an instruction has been successfully executed, “Ok” will be displayed on the screen, else an error message will be returned. Obviously, this requires two-way communication, so verbosity has no meaning in case of the parallel "centronics:" communication protocol.

VERBON corresponds to SYSVAR(18) = -1.
VERBOFF corresponds to SYSVAR(18) = 0.

Other verbosity levels can be selected using SYSVAR(18), and the type of error message can be selected using SYSVAR (19).

**VERBOFF**
All responses will be suppressed, which means that no characters or error messages will be echoed back. VERBOFF statements do not affect question marks and prompts displayed as a result of an INPUT statement. Instructions like DEVICES, FILES, FONTS, IMAGES, LIST, and PRINT will also work normally.

**Example**
This example shows how VERBOFF is used to suppress the printing of INPUT data in lines 20 and 40 during the actual typing on the host, and VERBON to allow the printing of the resulting string variables on the screen:

```
10    FOR Q%=1 TO 6
20    VERBOFF: INPUT "", A$
30    VERBON: PRINT A$;
40    VERBOFF: INPUT "", B$
50    VERBON
60    C$=SPACE$(25-LEN(A$))
70    PRINT C$+B$
80    NEXT Q$
90    END
```
**VERSION$**

**Purpose**
Function returning the version of the firmware, printer family, or type of CPU board.

**Syntax**

\[
\text{VERSION$}([<\text{nexp}>])
\]

\(<\text{nexp}>)\] is, optionally, the type of information to be returned:
- 0 = Version of firmware (default)
- 1 = Printer family
- 2 = Type of CPU board

**Remarks**
The name of the firmware depends on if the printer is running in the Immediate or Programming Modes, or in the Intermec Direct Protocol.

The printer family is returned as one of the following alternatives:
- PD41
- PF2i
- PF4i
- PM4i
- PX4i
- PX6i

The type of CPU-board is returned as a string of text, for example:
- hardware version 4.0

**Examples**

PRINT VERSION$ (0) yields for example:
Fingerprint 8.70.0

PRINT VERSION$ (1) yields for example:
PF4i

PRINT VERSION$ (2) yields for example:
hardware version 4.0
**WEEKDAY**

**Purpose**

Function returning the weekday of a specified date.

**Syntax**

\[\text{WEEKDAY(}<\text{sexp}>\text{)}\]

\(<\text{sexp}>\) is the date in \text{DATES} format from which the weekday will be returned.

**Remarks**

This function returns the weekday as a numeric constant:

1 = Monday  
2 = Tuesday  
3 = Wednesday  
4 = Thursday  
5 = Friday  
6 = Saturday  
7 = Sunday

The date should be entered according to the syntax for the \text{DATE} variable, that is in the following order:

\[\text{YY} = \text{Year} \quad \text{Last two digits} \quad \text{(for example 2003 = 03)}\]
\[\text{MM} = \text{Month} \quad \text{Two digits} \quad \text{(01-12)}\]
\[\text{DD} = \text{Day} \quad \text{Two digits} \quad \text{(01-28|29|30|31)}\]

Example: December 1, 2003 is entered as "031201".

The built-in calendar corrects illegal values for the years 1980-2048, for example the illegal date 031232 will be corrected to 040101.

**Example**

In this example the weekday for the current date is printed on the screen of the host (another way is to use \text{NAME} \text{WEEKDAY}\$ statement and \text{WEEKDAY}\$ function):

```
10 \text{B$=DATE$}\$
20 \text{A$= \text{WEEKDAY (B$)}}$\$
30 \text{IF A$ = 1 \text{ THEN PRINT } "MONDAY"}$\$
40 \text{IF A$ = 2 \text{ THEN PRINT } "TUESDAY"}$\$
50 \text{IF A$ = 3 \text{ THEN PRINT } "WEDNESDAY"}$\$
60 \text{IF A$ = 4 \text{ THEN PRINT } "THURSDAY"}$\$
70 \text{IF A$ = 5 \text{ THEN PRINT } "FRIDAY"}$\$
80 \text{IF A$ = 6 \text{ THEN PRINT } "SATURDAY"}$\$
90 \text{IF A$ = 7 \text{ THEN PRINT } "SUNDAY"}$\$
```

Yields for example:

\text{THURSDAY}
WEEKDAY$

Purpose
Returning the name of the weekday from a specified date.

Syntax
WEEKDAY$(<sexp>)

<sexp> is the date for which the name of the weekday, according to a list of weekday names created by means of NAME WEEKDAY$ statement, will be returned.

Remarks
This function returns the name of the weekday according to a list of weekday names specified by means of NAME WEEKDAY$ statement or—if the name is missing—the full English name in lowercase characters, for example “friday”.

The date should be entered according to the syntax for the DATE$ variable, that is in the following order:

YY = Year Last two digits (for example 2003 = 03)
MM = Month Two digits (01-12)
DD = Day Two digits (01-28|29|30|31)

Example: December 1, 2003 is entered as "031201".

The built-in calendar corrects illegal values for the years 1980-2048, for example the illegal date 031232 will be corrected to 040101.

Example
This example shows how to make the printer return the name of the weekday as a three-letter English abbreviation in connection with a formatted date:

10 FORMAT DATE$ ", MM/DD/YY"
20 DATE$="031201"
30 NAME WEEKDAY$ 1, "Mon"
40 NAME WEEKDAY$ 2, "Tue"
50 NAME WEEKDAY$ 3, "Wed"
60 NAME WEEKDAY$ 4, "Thu"
70 NAME WEEKDAY$ 5, "Fri"
80 NAME WEEKDAY$ 6, "Sat"
90 NAME WEEKDAY$ 7, "Sun"
100 PRINT WEEKDAY$ (DATE$) + DATE$("F")
RUN

yields:

MON, 12/01/03
**WEEKNUMBER**

**Purpose**
Function returning the number of the week for a specified date.

**Syntax**
```plaintext
WEEKNUMBER(<sexp>[,<nexp>])
```
- `<sexp>` is the date for which the week number will be returned (1-53).
- `<nexp>` specifies the calculating function (0-14) as listed below. Default is 0.

**Remarks**
**WEEKNUMBER calculating function:**

<table>
<thead>
<tr>
<th><code>&lt;nexp&gt;</code></th>
<th>Week #1 starts...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>according to ISO 8601 (European standard):</td>
</tr>
<tr>
<td></td>
<td>• week #1 will start on the last Monday at or before the New Year, if January 1 occurs on a Monday, Tuesday, Wednesday, or Thursday.</td>
</tr>
<tr>
<td></td>
<td>• week #1 will start on the first Monday after the New Year, if January 1 occurs on a Friday, Saturday, or Sunday.</td>
</tr>
<tr>
<td>1</td>
<td>at Sunday in the first week with 7 days in the actual year</td>
</tr>
<tr>
<td>2</td>
<td>at January 1:st, with each following week starting on a Sunday</td>
</tr>
<tr>
<td>3</td>
<td>at Monday in the first week with 7 days in the actual year</td>
</tr>
<tr>
<td>4</td>
<td>at January 1:st, with each following week starting on a Monday</td>
</tr>
<tr>
<td>5</td>
<td>at Tuesday in the first week with 7 days in the actual year</td>
</tr>
<tr>
<td>6</td>
<td>at January 1:st, with each following week starting on a Tuesday</td>
</tr>
<tr>
<td>7</td>
<td>at Wednesday in the first week with 7 days in the actual year</td>
</tr>
<tr>
<td>8</td>
<td>at January 1:st, with each following week starting on a Wednesday</td>
</tr>
<tr>
<td>9</td>
<td>at Thursday in the first week with 7 days in the actual year</td>
</tr>
<tr>
<td>10</td>
<td>at January 1:st, with each following week starting on a Thursday</td>
</tr>
<tr>
<td>11</td>
<td>at Friday in the first week with 7 days in the actual year</td>
</tr>
<tr>
<td>12</td>
<td>at January 1:st, with each following week starting on a Friday</td>
</tr>
<tr>
<td>13</td>
<td>at Saturday in the first week with 7 days in the actual year</td>
</tr>
<tr>
<td>14</td>
<td>at January 1:st, with each following week starting on a Saturday</td>
</tr>
</tbody>
</table>

The date should be entered according to the syntax for the DATE$ variable, that is in the following order:

```plaintext
YY = Year Last two digits (for example 2003 = 03)
MM = Month Two digits (01-12)
DD = Day Two digits (01-28|29|30|31)
```

Example: December 1, 2003 is entered as "031201".

The built-in calendar corrects illegal values for the years 1980-2048, for example the illegal date 031232 will be corrected to 040101.

**Examples**
This example returns the week number of December 29, 2002 using calculating function 2:
```plaintext
PRINT WEEKNUMBER ("031229", 2)
```
yields for example:

53
WHILE...WEND

Purpose

Statement for executing a series of statements in a loop providing a given condition is true.

Syntax

WHILE <nexp>
<stmt>
[<stmt>]
WEND

<nexp> is a numeric expression that is either TRUE (-1) or FALSE (0).
<stmt> is a statement, or a list of statements on separate lines, that will be executed provided <nexp> is TRUE.

Remarks

If <nexp> is TRUE, all following statements will be executed successively until a WEND statement is encountered. The program execution then goes back to the WHILE statement and repeats the process, provided <nexp> still is TRUE.

If <nexp> is FALSE, the execution resumes at the statement following the WEND statement.

WHILE...WEND statements can be nested. Each WEND matches the most recent WHILE statement.

Example

In this example, the WHILE...WEND loop will only be executed if the character “Y” (ASCII 89 dec.) is entered on the keyboard of the host.

```
10 B%=0
20 WHILE B%<>89
30 INPUT "Want to exit? Press Y=Yes or N=No ",A$
40 B%=ASC(A$)
50 WEND
60 PRINT "The answer is Yes"
70 PRINT "You will exit the program"
80 END
```

yields:

Want to exit? Press Y=Yes or N=No    N
Want to exit? Press Y=Yes or N=No    Y
The answer is Yes
You will exit the program
**XORMODE ON/OFF**

**Purpose**
Statement for enabling or disabling the xor/flip mode of Intermec Fingerprint in connection with graphical operations.

**Syntax**

```
XORMODE ON|OFF
```

**Remarks**
When XORMODE is set ON, dots are reversed, as opposed to set, by all graphical operations except bar codes. This means that if, for example, two black lines cross, the intersection will be white. If XORMODE is set to OFF, the intersection will be black.

Default is XORMODE OFF. XORMODE is automatically set to default when a PRINTFEED statement is executed or a Fingerprint program has been successfully run.

**Example**
The following program illustrates the difference between XORMODE ON and XORMODE OFF. The two lines to the left are drawn with XORMODE disabled and the lines to the right with XORMODE enabled.

```
10 XORMODE OFF
20 PRPOS 0,50
30 PRLINE 300,30
40 DIR 4
50 PRPOS 100,0
60 PRLINE 200,30
70 XORMODE ON
80 DIR 1
90 PRPOS 400,50
100 PRLINE 300,30
110 DIR 4
120 PRPOS 500,0
130 PRLINE 200,30
140 PRINTFEED
RUN
```
External Command: Account Secret

Purpose
Creating an Account Secret for use with the TRANSFER NET statement.

Syntax

```
secret [-t]<application> <name> <string>  (create a secret)
```

For user /admin/ two more functions are available:

```
secret –rm <application> <name>  (delete a secret)
secret –l  (list all secrets)
```

- `-t` Temporary (will be removed at next reboot). Optional.
- `-rm` Remove
- `-l` List
- `<application>` ftp
- `<name>` is the name of secret.
- `<string>` is the secret string.

Remarks

The user may not want to have his/her username and password in plain text in a Fingerprint program. Instead of writing the account info in the URI, the `<account secret>` parameter of TRANSFER NET can be used. The account secret holds the secret information and cannot be read by any user.

For the application ftp, `<string>` should have the following structure:

```
<string> = <user>[:passwd]@<server> where <server> is:
<server name>|<server name>:<port number>|*
```

If `<server>` is set to “*”, the server name supplied in the URI will be used. This means that this account secret can be used with any server.

Account info, that is, user, password, server name and port number, can be stated in both the account secret and the URI. Any parameter supplied in the account secret will have precedence over parameters supplied in the URI. This means that if, for example, the URI states that port 25 should be used and the account secret says port 21, then port 21 will be used.

Examples

Create a temporary account secret, my_account, and use it to send the file beta1.bin in directory /tmp to the server TheServer. The sent file will get the name beta1.bin and will be put in the home directory of myusername on TheServer.

First of all, set SYSVAR(43) to 1 to avoid file name conversion:

```
SYSVAR(43) = 1
```

RUN "secret –t ftp my_account myusername:mypassword@*"

TRANSFER NET "/tmp/beta1.bin","ftp://TheServer/", "my_account"
External Command; Account Secret, cont.

Create a permanent account secret, frodo, and use it to send the file MY.TXT in the current directory to the server Frodo. The file YOUR.TXT will be put in the directory /absolute/path/.

RUN "secret ftp frodo frodo_username:frodo_password@Frodo"

TRANSFER NET "MY.TXT","ftp://Frodo//absolute/path/YOUR.TXT","frodo"

The server name in the URI will not be used since there is a server name in the account secret. Hence the two following command lines will have the same effect:


and

TRANSFER NET "MY.TXT",ftp://///absolute/path/YOUR.TXT","frodo"
**Chapter 2—Program Instructions**

### External Command; ZMODEM

**Purpose**
External commands for receiving and sending data using the ZMODEM protocol.

**Syntax**

<table>
<thead>
<tr>
<th>RUN &quot;rz [&lt;switches&gt;] [&lt;filename&gt;]&quot;</th>
<th>(receive data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;switches&gt;:</td>
<td></td>
</tr>
<tr>
<td>-c</td>
<td>Forces no crash recovery, even if sender requests ZCRESUM (resume interrupted file transfer).</td>
</tr>
<tr>
<td>-e</td>
<td>Print last error to std OUT channel.</td>
</tr>
<tr>
<td>-l[&lt;logfile&gt;]</td>
<td>Send verbose output to logfile. Default logfile name is &quot;tmp:.zmodemlog&quot;.</td>
</tr>
<tr>
<td>-r</td>
<td>If ZMCLOB is not set and the file already exists, replace file if the transfer is successful.</td>
</tr>
<tr>
<td>-v[&lt;level&gt;]</td>
<td>Set verbosity level. Level is a decimal number. Default level is 1.</td>
</tr>
<tr>
<td>-u</td>
<td>Translate file name to uppercase. If a filename is given as parameter, no translation is done.</td>
</tr>
<tr>
<td>&lt;filename&gt;</td>
<td>is optionally the name under which the file will be saved.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RUN &quot;sz [&lt;switches&gt;] [&lt;filename&gt;]&quot;</th>
<th>(send data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;switches&gt;:</td>
<td></td>
</tr>
<tr>
<td>-l[&lt;logfile&gt;]</td>
<td>Send verbose output to logfile. Default logfile name is &quot;tmp:.zmodemlog&quot;.</td>
</tr>
<tr>
<td>-v[&lt;level&gt;]</td>
<td>Set verbosity level. Level is a decimal number. Default level is 1.</td>
</tr>
<tr>
<td>&lt;filename&gt;</td>
<td>is the name of the file.</td>
</tr>
</tbody>
</table>

**Remarks**
Note that rz and sz must be entered in lowercase characters.

If a file name is given in the rz statement, this name overrides the name given by the transmitting unit.

For more information on the ZMODEM protocol, please refer to http://www.omen.com. Related instruction is TRANSFER ZMODEM.
3 Image Transfer

This chapter describes the various image transfer file protocols used in Intermec Fingerprint v8.70.0 and v10.0.0.
## Protocols

The following five image transfer file protocols are used in connection with the STORE IMAGE statement and use a common format for the image data, as described on next page.

### Intelhex

Intel hex [Intel Hexadecimal Intellec 8/MDS (I_hex) file format] is a well-known standard format for transfer of bitmap images. Please refer to the standard literature on the subject.

Note that:
- Hex digits in Intelhex frames must be uppercase.
- Null frames may be omitted.
- Frames can be received in any order.
- Maximum file size is 64 kbytes.

### UBI00

Each frame contains:

```
<data bytes>
```

- **Binary images. Modulo 2 bytes.**

### UBI01

Each frame of data contains:

```
<data bytes> <checksum>
```

- **Binary images. Modulo 2 bytes.**
- **Modulo 65536 byte-wise sum of what is defined in protocol of “data bytes.”**
- **2 byte binary. MSB, LSB.**

### UBI02

Each frame of data contains:

```
<number of data bytes> <data bytes> <checksum>
```

- **2 bytes binary. MSB, LSB.**
- **Binary images. Modulo 2 bytes.**
- **Modulo 65536 byte-wise sum of what is defined in protocol of “number of data bytes” and “data bytes.”**
- **2 byte binary. MSB, LSB.**

### UBI03

Each frame of data contains:

```
<start of frame id.> <number of data bytes> <data bytes> <checksum>
```

- **1 byte (ASCII 42 dec = “*”).**
- **2 bytes binary. MSB, LSB.**
- **Binary images. Modulo 2 bytes.**
- **Modulo 65536 byte-wise sum of what is defined in protocol of “start of frame id” and “number of data bytes” and “data bytes.”**
- **2 byte binary. MSB, LSB.**
Chapter 3—Image Transfer

Image Format

The following image format is valid for Intelhex, UBI00, UBI01, UBI02, and UBI03 image transfer protocols, but not for the UBI10 protocol, which is a combined image transfer protocol and format.

A bitmap picture can be encoded in one of two ways, as a plain bit representation or encoded with a Run Lenght Limited (RLL) algorithm.

Pictures can be magnified, by the printer, up to four times independently in both x and y directions.

The pictures can be rotated 180 degrees by the printer (that is printed upside-down.) To print a bitmap in all four directions you have to define two bitmaps, one straight and one rotated 90 degrees. To comply with the Intermec Fingerprint convention, use the extension .1 for the straight bitmap and extension .2 for the rotated one.

Bitmap pictures, in both encoding schemes, are printed with the lowest address first, that is the first row of defined data is the first thing out. (This may be somewhat confusing. The only result, if you misinterpret this, is that your picture will come out upside-down.)

**Bitmap pattern, bit representation**

The bitmap picture is encoded word oriented (16 bits), low byte first. The bits in each byte is read from lsb first (bit 0.)

**Bitmap pattern, Run Lenght Limited (RLL)**

RLL encoding is a very efficient way of compressing big bitmaps with relatively big black and/or white areas.

The RLL encoded picture is encoded byte oriented (8 bits.) Each byte represents the number of consecutive black or white dots. The sum of bytes for each row must equal the width of the pattern. The first byte represent white dots, the second black and so on. The last byte must alter the color back to white. If the first dot is black just enter a zero first. Valid values for dot fields is 0 to 127 (0 to 7f hex.) To get a row longer then 127, concatenate two rows with zero, for example to get a row of 240 dots, enter 128,0,112.

The next step in our RLL encoding algoritm is to compress identicals rows, two identical rows are compressed by adding a byte in both ends of the dot row, the valid range for these bytes are -1 to -128 (ff to 80 hex.)
Example 1: Bitmap encoding
To clarify this, let’s try a simple example. X’s represent black dots in the final printout. The pattern shown is 22 bits wide and 28 rows high.

Note:
- The bit order in each byte. Note also word fill to nearest word (16 bit).
- To the right is a hex representation of the pattern, as it would appear in a memory dump.
- To get the pattern to appear as printed on this page with direction one, the last row (row 27) should have the lowest address.

<table>
<thead>
<tr>
<th>byte 3</th>
<th>byte 2</th>
<th>byte 1</th>
<th>byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>76543210765432107654321076543210</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>row</th>
<th>byte 3</th>
<th>byte 2</th>
<th>byte 1</th>
<th>byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>.XXXXXXXXX.xxxxxxxxxxxxxxxxxx</td>
<td>ff, ff, 3f, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>X..................X......X</td>
<td>01, 00, 20, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>X...........X........X...X</td>
<td>61, 00, 20, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>X...........X........X...X</td>
<td>a1, 00, 20, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>X...........X........X...X</td>
<td>21, 01, 20, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>X...........X........X...X</td>
<td>21, 02, 20, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>X...........X........X...X</td>
<td>21, 04, 20, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>X...........X........X...X</td>
<td>21, 08, 20, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>X...........X........X...X</td>
<td>21, 10, 20, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>X...........X........X...X</td>
<td>21, 20, 20, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>X...........X........X...X</td>
<td>21, 40, 20, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>X...........X........X...X</td>
<td>21, 80, 20, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>X...........X........X...X</td>
<td>21, 00, 21, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>X...........X........X...X</td>
<td>21, 00, 22, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>X...........X........X...X</td>
<td>21, 00, 24, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>X...........X........X...X</td>
<td>21, 00, 28, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>X...........X........X...X</td>
<td>fd, ff, 2f, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>X...........X........X...X</td>
<td>21, 00, 20, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>X...........X........X...X</td>
<td>21, 00, 20, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>X...........X........X...X</td>
<td>21, 00, 20, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>X...........X........X...X</td>
<td>21, 00, 20, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>X...........X........X...X</td>
<td>21, 00, 20, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>X...........X........X...X</td>
<td>21, 00, 20, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>X...........X........X...X</td>
<td>21, 00, 20, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>X...........X........X...X</td>
<td>21, 00, 20, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>X...........X........X...X</td>
<td>f9, 03, 20, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>X...........X........X...X</td>
<td>01, 00, 20, 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>.XXXXXXX.xxxxxxxxxxxxxx</td>
<td>ff, ff, 3f, 00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Example 2: RLL Encoding**

To clarify this, let's try a simple example. X's represent black dots in the final print out. The pattern shown is 22 bits wide and 32 rows high.

**Note:**
- Notice the reverse byte order. Count dots from right.
- To the right is a decimal representation of the pattern.
- To get the pattern to appear as printed on this page with direction one, the last row (row 27) should have the lowest address. Row 18 until 24 is repeated by the data in row 17.

<table>
<thead>
<tr>
<th>row</th>
<th>X pattern</th>
<th>Decimal Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>XX........X</td>
<td>0, 22, 0</td>
</tr>
<tr>
<td>1</td>
<td>X...........X</td>
<td>0, 1, 20, 1, 0</td>
</tr>
<tr>
<td>2</td>
<td>X............X</td>
<td>0, 1, 4, 2, 14, 1, 0</td>
</tr>
<tr>
<td>3</td>
<td>X...........X.X</td>
<td>0, 1, 4, 1, 1, 13, 1, 0</td>
</tr>
<tr>
<td>4</td>
<td>X............X.X</td>
<td>0, 1, 4, 1, 2, 1, 12, 1, 0</td>
</tr>
<tr>
<td>5</td>
<td>X............X.X</td>
<td>0, 1, 4, 1, 3, 1, 11, 1, 0</td>
</tr>
<tr>
<td>6</td>
<td>X............X.X</td>
<td>0, 1, 4, 1, 4, 1, 10, 1, 0</td>
</tr>
<tr>
<td>7</td>
<td>X............X.X</td>
<td>0, 1, 4, 1, 5, 1, 9, 1, 0</td>
</tr>
<tr>
<td>8</td>
<td>X............X.X</td>
<td>0, 1, 4, 1, 6, 1, 8, 1, 0</td>
</tr>
<tr>
<td>9</td>
<td>X............X.X</td>
<td>0, 1, 4, 1, 7, 1, 7, 1, 0</td>
</tr>
<tr>
<td>10</td>
<td>X............X.X</td>
<td>0, 1, 4, 1, 8, 1, 6, 1, 0</td>
</tr>
<tr>
<td>11</td>
<td>X............X.X</td>
<td>0, 1, 4, 1, 9, 1, 5, 1, 0</td>
</tr>
<tr>
<td>12</td>
<td>X............X.X</td>
<td>0, 1, 4, 1, 10, 1, 4, 1, 0</td>
</tr>
<tr>
<td>13</td>
<td>X.........XX...X</td>
<td>0, 1, 4, 1, 11, 1, 3, 1, 0</td>
</tr>
<tr>
<td>14</td>
<td>X.........XX...X</td>
<td>0, 1, 4, 1, 12, 1, 2, 1, 0</td>
</tr>
<tr>
<td>15</td>
<td>X.........XX...X</td>
<td>0, 1, 4, 1, 13, 1, 1, 1, 0</td>
</tr>
<tr>
<td>16</td>
<td>X........XXXXX.XX</td>
<td>0, 1, 1, 18, 1, 1, 0</td>
</tr>
<tr>
<td>17</td>
<td>X........XXXXX.XX</td>
<td>-8, 0, 1, 4, 1, 15, 1, 0, -8</td>
</tr>
<tr>
<td>18</td>
<td>X........XXXXX.XX</td>
<td>0, 1, 2, 5, 13, 1, 0</td>
</tr>
<tr>
<td>19</td>
<td>X........XXXXX.XX</td>
<td>0, 1, 20, 1, 0</td>
</tr>
<tr>
<td>20</td>
<td>X........XXXXX.XX</td>
<td>0, 22, 0</td>
</tr>
<tr>
<td>21</td>
<td>X........XXXXX.XX</td>
<td>-4, 11, 1, 10, -4</td>
</tr>
<tr>
<td>22</td>
<td>X........XXXXX.XX</td>
<td>7, 9, 6</td>
</tr>
</tbody>
</table>
**UBI10**

UBI10 is a combined protocol/file format for image transfer, as opposed to Intelhex and UBI00-UBI03 protocols described earlier in this chapter.

**Protocol Description**

- **!BG**
- **!X<pos>A**
- **!Y<pos>A**
- **!X<pos>A | !Y<pos>A !SB<bytes>W<data>**
- **!X<pos>A | !Y<pos>A !SB<bytes>W<data>**
- **!X<pos>A | !Y<pos>A !SB<bytes>W<data>**
- **...**
- **!X<pos>A | !Y<pos>A !SB<bytes>W<data>!EG**
- **!PRINT**

**Frame Definitions**

The width of the image in the STORE IMAGE statement should be given as a multiple of 16 bits.

- **!BG** Begin graphics.
  Always appended by a carriage return character.

  **!X<pos>A** Set absolute x position <pos>.
  The value must be divisible by 8.
  Default value is 0.
  Once set, it will affect all consecutive y-positions in the image, until a new x-position is set.
  Appended by a carriage return character, unless followed by a **!SB<bytes>W<data>** string on the same line.

  **!Y<pos>A** Set absolute y position <pos>.
  Default value is 0.
  Appended by a carriage return character, unless followed by a **!SB<bytes>W<data>** string on the same line.

  **!SB<bytes>W<data>** Send one line of bitmap with <bytes> number of bytes. <data> is bitmap bytes.
  Can be preceded by a new x- and/or y-position.
  If appended by a carriage return character, next !SB set of data will be positioned at the current y-position incremented by 1.
  If no appending carriage return character is used, a new y-position must be specified for next !SB set of data.

  **!EG** End graphics.
  Always appended by a carriage return character.

  **!PRINT** End page (end frame).
  Always appended by a carriage return character.
The image illustrated above contains 2 bytes (= 16 bits) in each horizontal line. By setting the absolute start position to \( x = 8 \), you can start counting from the start of the second byte, that is \( x = 8 \) in the matrix above. The first 3 bits (x-positions) are white, then comes one black bit followed by three white bits, and finally one black bit. Expressed in 0:s and 1:s, where 0 represents a white bit and 1 a black bit, the pattern will be 00000001. This binary number can be expressed as 11 hex. The same pattern is repeated for each y-position from \( y = 1 \) thru \( y = 7 \) with the exception of position \( y = 4 \), where all bits are black except for the leading three, i.e. the pattern is 00011111, which can be expressed as 1F hex. Use this hexadecimal values as input data as shown in the example below.

**Example**

The simplified image above is transmitted to the printer. Do not use XON/XOFF (11 hex/13 hex) protocol, since these characters may coincide with input data. Use RTS/CTS instead. Do not strip LF.

```
10 STORE OFF
20 OPEN "uart1:" FOR INPUT AS #1
30 QNAME$="H.1"
40 QWIDTH%=16
50 QHEIGHT%=10
60 QPRO$="UBI10"
70 STORE IMAGE QNAME$,QWIDTH%,QHEIGHT%,QPROT$
80 STORE INPUT 900,4: 'Timeout 9 sec.
90 CLOSE#1
100 STORE OFF
RUN
```

The input string in line 80 should contain the following data. Carriage returns (\( \downarrow \)) after each \$B set of data increments the y-position by 1 in consecutive order. It may also be sent as a continuous string.

\[
\begin{align*}
!BG \downarrow & \quad \text{(Begin graphic)} \\
!X8A \downarrow & \quad \text{(Set x-position)} \\
!Y1A!SB1W<11 \text{ hex}> \downarrow & \quad \text{(Set y-position + data for } y = 1) \\
!SB1W<11 \text{ hex}> \downarrow & \quad \text{(Data for } y = 2) \\
!SB1W<11 \text{ hex}> \downarrow & \quad \text{(Data for } y = 3) \\
!SB1W<1F \text{ hex}> \downarrow & \quad \text{(Data for } y = 4) \\
!SB1W<11 \text{ hex}> \downarrow & \quad \text{(Data for } y = 5) \\
!SB1W<11 \text{ hex}> \downarrow & \quad \text{(Data for } y = 6) \\
!SB1W<11 \text{ hex}>!EG \downarrow & \quad \text{(Data for } y = 7 \text{ + end graphics)} \\
!PRINT \downarrow & \quad \text{(End frame)}
\end{align*}
\]
Chapter 3—Image Transfer

PRBUF Protocol

The PRBUF Protocol is designed for downloading bitmap print image data directly from an application program, such as a Windows printer driver, directly to the printer’s image buffer in connection with the PRBUF statement.

The protocol consist of a two-byte header and a number of data bytes:

Header
Bye No 1. is always the @-sign (Commercial at; Unicode 0x0040) and indicates start of the protocol header.

Byte No 2 is:
0    Reserved (bitmap format)
1    Reserved (RLL image format)
2    RLL buffer format
3-255  Reserved

RLL Buffer format
The RLL buffer format is optimized for use by Windows drivers. In most cases the performance of the host outruns the performance of the printer, so it is preferred to do most of the processing in the host before sending the job down to the printer.

- Data byte 1 & 2 specifies the pixelwidth (unsigned) of data in BIG Endian format for one line.
- Data byte 3 & 4 specifies the pixelheight (unsigned) of the buffer when it is expanded BIG Endian.
- Data byte 5-nn specifies the bitmap in RLL format.

Example of RLL buffer protocol header, 515x212 pixels hexdump:
40 02 02 03 00 d4

RLL format
The RLL format is good for black and white pixel runs. It compresses data in both dimensions. It works well with one-dimensional bar codes, but grayscale growths in size instead of shrinking. The format is symmetric so that all pixel runs start and end with a white pixel and with line repetitions whenever applicable. This makes the format possible to turn upside down.

The RLL format is specified on the next page.
Specification of the RLL format

- total width of RLL pattern

\[
\begin{align*}
<\text{begin}> & \text{ := } <\text{linerps}>|<\text{small white pixelrun}> \\
<\text{end}> & \text{ := } <\begin{align*}
\text{begin}>|<\text{empty}> \\
<\text{toggling pixelruns}> & \text{ := } <\text{whiteAndBlack pixelruns}>| <\text{blackAndWhite pixelruns}>| <\text{white pixelrun}>|<\text{black pixelrun}>|<\text{small white pixelrun}>|<\text{black pixelrun}> \\
<\text{whiteAndBlack pixelruns}> & \text{ := } <\text{white pixelrun}>|<\text{black pixelrun}>|<\text{small white pixelrun}>|<\text{black pixelrun}> \\
<\text{blackAndWhite pixelruns}> & \text{ := } <\text{black pixelrun}>|<\text{white pixelrun}> \\
<\text{linerps}> & \text{ := } ((-1)*(-128))\times1 \text{ number of equal lines} \\
<\text{small white pixelrun}> & \text{ := } 0-127, \text{ number of white pixels} \\
<\text{black pixelrun}> & \text{ := } 0-255, \text{ number of black pixels} \\
<\text{white pixelrun}> & \text{ := } 0-255, \text{ number of white pixels} \\
<\text{empty}> & \text{ := } \text{empty, extreme if the entire line fits in one pixelrun.} \\
\end{align*}\n\]

If there is no line repetition, there does not have to be any line repeat. If the pixelrun is out of range, it must be split into several runs.

Example of RLL format for an eight bit pattern:

\[
\begin{align*}
-\ast-\ast-\ast & 1,1,1,1,1,1,0 \\
-\ast-\ast-\ast & 0,1,1,1,1,1,1 \\
-\ast-\ast-\ast & 2,2,2,2,0 \\
\ast-\ast-\ast- & 0,2,2,2,0 \\
\ast-\ast-\ast & -32 \\
\end{align*}
\]

Note the last 0 to end with a white pixelrun

begins with a white pixelrun of 0 pixels

repetition, stopped with a white pixelrun of 0 pixels

line and pixel repetitions

Example of coding a black square of 800 dots to valid RLL format:

\[
\begin{align*}
-128,0,255,0,255,0,255,0,35,0,-128 \\
-128,0,255,0,255,0,255,0,35,0,-128 \\
-128,0,255,0,255,0,255,0,35,0,-128 \\
-128,0,255,0,255,0,255,0,35,0,-128 \\
-128,0,255,0,255,0,255,0,35,0,-128 \\
-128,0,255,0,255,0,255,0,35,0,-128 \\
-128,0,255,0,255,0,255,0,35,0,-128 \\
-128,0,255,0,255,0,255,0,35,0,-128 \\
-128,0,255,0,255,0,255,0,35,0,-128 \\
-32,0,255,0,255,0,255,0,35,0,-32 \\
\end{align*}
\]
This chapter contains the various single-byte character sets, that can be selected using the NASC statement. Printouts samples are shown in the font “Swiss 721 BT”. Other fonts may not include all characters listed in the character sets. A short description of the multi-byte character set UTF-8 is included. Other double-byte character sets are not included, but are available separately on special request.

Printout samples are shown for the fifteen scaleable single-byte fonts included in the Intermec Fingerprint v8.70.0/v10.0.0 firmware. It also describes the method of creating font aliases and contains character sets for the OCR-A, OCR-B, and DingDings fonts.

For more information on character sets and fonts, refer to the *Intermec Fingerprint v.8.xx, Font Reference Manual*. 
Chapter 4—Character Sets and Fonts

Character Sets

The following information applies to all single-byte character sets:

• Characters between ASCII 00 decimal and ASCII 31 decimal are unprintable control characters as listed below.

• Characters between ASCII 32 decimal and ASCII 127 decimal can always be printed, regardless of 7-bit or 8-bit communication protocol, provided that the selected font contains the characters in question.

• Characters between ASCII 128 decimal and ASCII 255 decimal can only be printed if the selected font contains the characters in question and an 8-bit communication protocol is used. If you use 7-bit communication, select another national character set (see NASC statement) or use a MAP statement to remap a character set.

• If a character, which does not exist in the selected font, is used, an error condition will occur.

Non-printable control characters (ASCII 00-31 dec)

<table>
<thead>
<tr>
<th>ASCII</th>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>NUL</td>
<td>Null</td>
</tr>
<tr>
<td>01</td>
<td>SOH</td>
<td>Start of heading</td>
</tr>
<tr>
<td>02</td>
<td>STX</td>
<td>Start of text</td>
</tr>
<tr>
<td>03</td>
<td>ETX</td>
<td>End of text</td>
</tr>
<tr>
<td>04</td>
<td>EOT</td>
<td>End of transmission</td>
</tr>
<tr>
<td>05</td>
<td>ENQ</td>
<td>Enquiry</td>
</tr>
<tr>
<td>06</td>
<td>ACK</td>
<td>Acknowledge</td>
</tr>
<tr>
<td>07</td>
<td>BEL</td>
<td>Bell</td>
</tr>
<tr>
<td>08</td>
<td>BS</td>
<td>Backspace</td>
</tr>
<tr>
<td>09</td>
<td>HT</td>
<td>Horizontal tabulation</td>
</tr>
<tr>
<td>10</td>
<td>LF</td>
<td>Line feed</td>
</tr>
<tr>
<td>11</td>
<td>VT</td>
<td>Vertical tabulation</td>
</tr>
<tr>
<td>12</td>
<td>FF</td>
<td>Form feed</td>
</tr>
<tr>
<td>13</td>
<td>CR</td>
<td>Carriage Return</td>
</tr>
<tr>
<td>14</td>
<td>SO</td>
<td>Shift out</td>
</tr>
<tr>
<td>15</td>
<td>SI</td>
<td>Shift in</td>
</tr>
<tr>
<td>16</td>
<td>DLE</td>
<td>Data link escape</td>
</tr>
<tr>
<td>17</td>
<td>DC1</td>
<td>Device control one</td>
</tr>
<tr>
<td>18</td>
<td>DC2</td>
<td>Device control two</td>
</tr>
<tr>
<td>19</td>
<td>DC3</td>
<td>Device control three</td>
</tr>
<tr>
<td>20</td>
<td>DC4</td>
<td>Device control four</td>
</tr>
<tr>
<td>21</td>
<td>NAK</td>
<td>Negative acknowledge</td>
</tr>
<tr>
<td>22</td>
<td>SYN</td>
<td>Synchronous idle</td>
</tr>
<tr>
<td>23</td>
<td>ETB</td>
<td>End of transmission block</td>
</tr>
<tr>
<td>24</td>
<td>CAN</td>
<td>Cancel</td>
</tr>
<tr>
<td>25</td>
<td>EM</td>
<td>End of medium</td>
</tr>
<tr>
<td>26</td>
<td>SUB</td>
<td>Substitute</td>
</tr>
<tr>
<td>27</td>
<td>ESC</td>
<td>Escape</td>
</tr>
<tr>
<td>28</td>
<td>FS</td>
<td>File separator</td>
</tr>
<tr>
<td>29</td>
<td>GS</td>
<td>Group separator</td>
</tr>
<tr>
<td>30</td>
<td>RS</td>
<td>Record separator</td>
</tr>
<tr>
<td>31</td>
<td>US</td>
<td>Unit separator</td>
</tr>
</tbody>
</table>
### Roman 8 Character Set

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>`</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td>(</td>
<td>)</td>
</tr>
<tr>
<td>40</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>50</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
<td>&lt;</td>
<td>=</td>
</tr>
<tr>
<td>60</td>
<td>&gt;</td>
<td>?</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
</tr>
<tr>
<td>70</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
</tr>
<tr>
<td>80</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
</tr>
<tr>
<td>90</td>
<td>\</td>
<td>^</td>
<td>_</td>
<td>`</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>100</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
<td>p</td>
</tr>
<tr>
<td>110</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
</tr>
<tr>
<td>120</td>
<td>{</td>
<td></td>
<td>}</td>
<td>~</td>
<td>\</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>140</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>150</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>160</td>
<td>À</td>
<td>À</td>
<td>À</td>
<td>À</td>
<td>È</td>
<td>È</td>
<td>È</td>
<td>È</td>
<td>Ê</td>
<td>Ê</td>
</tr>
<tr>
<td>170</td>
<td>`</td>
<td>`</td>
<td>`</td>
<td>`</td>
<td>~</td>
<td>Ù</td>
<td>Ù</td>
<td>£</td>
<td>–</td>
<td>Ý</td>
</tr>
<tr>
<td>180</td>
<td>Ç</td>
<td>ç</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ì</td>
<td>Ì</td>
<td>Ì</td>
<td>Ì</td>
<td>£</td>
<td>¥</td>
</tr>
<tr>
<td>190</td>
<td>f</td>
<td>c</td>
<td>â</td>
<td>ê</td>
<td>ô</td>
<td>û</td>
<td>a</td>
<td>é</td>
<td>ò</td>
<td>ó</td>
</tr>
<tr>
<td>200</td>
<td>à</td>
<td>è</td>
<td>ô</td>
<td>û</td>
<td>á</td>
<td>é</td>
<td>ô</td>
<td>ù</td>
<td>Å</td>
<td>Å</td>
</tr>
<tr>
<td>210</td>
<td>Ø</td>
<td>Æ</td>
<td>å</td>
<td>Í</td>
<td>ø</td>
<td>æ</td>
<td>Æ</td>
<td>í</td>
<td>Ö</td>
<td>Ù</td>
</tr>
<tr>
<td>220</td>
<td>É</td>
<td>Ê</td>
<td>Ô</td>
<td>Á</td>
<td>Â</td>
<td>ã</td>
<td>Ð</td>
<td>ð</td>
<td>Í</td>
<td>Í</td>
</tr>
<tr>
<td>230</td>
<td>Ì</td>
<td>Ï</td>
<td>Ô</td>
<td>Õ</td>
<td>Õ</td>
<td>ô</td>
<td>Õ</td>
<td>s</td>
<td>Õ</td>
<td>Õ</td>
</tr>
<tr>
<td>240</td>
<td>Ï</td>
<td>Ï</td>
<td>Ï</td>
<td>Ï</td>
<td>Ï</td>
<td>Ï</td>
<td>Ï</td>
<td>Ï</td>
<td>Ï</td>
<td>Ï</td>
</tr>
<tr>
<td>250</td>
<td>Ø</td>
<td>ø</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
</tr>
</tbody>
</table>
### French Character Set

#### NASC 33

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>!</td>
<td>&quot;</td>
<td>£</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>.</td>
<td>/</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
</tr>
<tr>
<td>60</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
<td>à</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>70</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>80</td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
</tr>
<tr>
<td>90</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>ô</td>
<td>ç</td>
<td>$</td>
<td>^</td>
<td>_</td>
</tr>
<tr>
<td>100</td>
<td>μ</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
</tr>
<tr>
<td>110</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
</tr>
<tr>
<td>120</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>è</td>
</tr>
<tr>
<td>130</td>
<td>ë</td>
<td>€</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>170</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>210</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>220</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>230</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>240</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

*Intermec Fingerprint v8.70.0/v10.0.0 Programmer’s Reference Manual*
## Spanish Character Set

### NASC 34

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>£</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
<td>$</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>70</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>80</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>90</td>
<td>Z</td>
<td>i</td>
<td>Ñ</td>
<td>į</td>
<td>_</td>
<td>`</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
</tr>
<tr>
<td>110</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
</tr>
<tr>
<td>120</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ĕ</td>
<td>ķ</td>
<td>~</td>
</tr>
<tr>
<td>130</td>
<td>€</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Ä</td>
<td>Å</td>
<td>Æ</td>
<td>Ė</td>
</tr>
<tr>
<td>170</td>
<td>^</td>
<td>^</td>
<td>^</td>
<td>_</td>
<td>—</td>
<td></td>
<td></td>
<td></td>
<td>Ù</td>
<td>Û</td>
</tr>
<tr>
<td>180</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>£</td>
<td></td>
<td>£</td>
<td>¥</td>
</tr>
<tr>
<td>190</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Š</td>
<td>£</td>
</tr>
<tr>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>210</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>220</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>230</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>240</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Italian Character Set

NASC 39

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>! &quot; £ $ % &amp; '</td>
<td>( ) * + , - . /</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>: ;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>&lt; = &gt; ? § A B C D E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>F G H I J K L M N O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>P Q R S T U V W X Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>Z ° ç é ^ _ ü a b c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>d e f g h i j k l m</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>n o p q r s t u v w</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>x y z à ò è i ã</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>ç ç Ñ ñ i ã ó ç ý</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>Ç ç Ñ ñ i ã ó ç ý</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>f ĝ ā ê ô û á é é ó ú</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>à è õ ü ä ë ö ü å</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>Å Å Ė Ė Ŗ Ŗ î ȉ ȉ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>170</td>
<td>^ ~ Ù Ù £ - Ý ý</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>Ç ç Ñ ñ i ã ó ç ý</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>f ĝ ā ê ô û á é é ó ú</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>à è õ ü ä ë ö ü å</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>210</td>
<td>Ø Æ å í ø æ Ä ï Ö Ü</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>220</td>
<td>É ï Ô Á ŕ Å ã Ð õ í</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>230</td>
<td>Í Ó Ô ŕ Š ŕ Ý ý</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>240</td>
<td>ß ß · μ ¶ â ¼ – ٤ ١⁄₂ ܐ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>o « ] » ±</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## English (UK) Character Set

### NASC 44

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>!</td>
<td>&quot;</td>
<td>£</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
</tr>
<tr>
<td>60</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>70</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
</tr>
<tr>
<td>80</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
</tr>
<tr>
<td>90</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
<td>`</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>100</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
</tr>
<tr>
<td>110</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
</tr>
<tr>
<td>120</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td></td>
<td>}</td>
<td>-</td>
<td>€</td>
</tr>
<tr>
<td>130</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>170</td>
<td>Å</td>
<td>Å</td>
<td>È</td>
<td>È</td>
<td>Ê</td>
<td>Ê</td>
<td>Ì</td>
<td>Ì</td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>^</td>
<td>~</td>
<td>Ù</td>
<td>Ù</td>
<td>£</td>
<td>-</td>
<td>Ŷ</td>
<td>Ŷ</td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>Ç</td>
<td>ç</td>
<td>Ñ</td>
<td>Ñ</td>
<td>ñ</td>
<td>ñ</td>
<td>€</td>
<td>¥</td>
<td>$</td>
</tr>
<tr>
<td>200</td>
<td>f</td>
<td>c</td>
<td>â</td>
<td>ê</td>
<td>ô</td>
<td>û</td>
<td>â</td>
<td>é</td>
<td>ó</td>
</tr>
<tr>
<td>210</td>
<td>à</td>
<td>è</td>
<td>ò</td>
<td>ù</td>
<td>ä</td>
<td>ë</td>
<td>ö</td>
<td>ü</td>
<td>Å</td>
</tr>
<tr>
<td>220</td>
<td>Ø</td>
<td>Å</td>
<td>á</td>
<td>í</td>
<td>ø</td>
<td>æ</td>
<td>Ä</td>
<td>ï</td>
<td>Ö</td>
</tr>
<tr>
<td>230</td>
<td>É</td>
<td>í</td>
<td>β</td>
<td>Ô</td>
<td>Á</td>
<td>Ā</td>
<td>â</td>
<td>ô</td>
<td>ð</td>
</tr>
<tr>
<td>240</td>
<td>ñ</td>
<td>ò</td>
<td>õ</td>
<td>ô</td>
<td>ñ</td>
<td>š</td>
<td>s</td>
<td>û</td>
<td>Ÿ</td>
</tr>
<tr>
<td>250</td>
<td>p</td>
<td>p</td>
<td>·</td>
<td>μ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Intermec Fingerprint v8.70.0/v10.0.0 Programmer’s Reference Manual
## Swedish Character Set

<table>
<thead>
<tr>
<th>Code</th>
<th>Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>! &quot; # $ % &amp; ' ( ) * + , - . / 0 1</td>
</tr>
<tr>
<td>&lt; = &gt; ? É Á À Ò Ó</td>
<td>F G H I J K L M N O</td>
</tr>
<tr>
<td>P Q R S T U V W X Y</td>
<td>Z Ä Ö Å Ü _ é a b c</td>
</tr>
<tr>
<td>d e f g h i j k l m</td>
<td>n o p q r s t u v w</td>
</tr>
<tr>
<td>x y z à ö å ü €</td>
<td>□</td>
</tr>
<tr>
<td>fè â è ô û à é ô ú</td>
<td>à è ò û â è ò û Å Å</td>
</tr>
<tr>
<td>Ø Æ å í ò á æ ò å</td>
<td>À È É Ë Ì Í Ë Ë</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>À Â È Ë Î Í Ï Ì ́</td>
<td></td>
</tr>
<tr>
<td>^ ~ Ù Ù £ – Ý ý °</td>
<td>Ç ç Ñ ñ i í ï ë £ ¥ $</td>
</tr>
<tr>
<td>f ç â ê ô û à é ô ú</td>
<td>à è ò û â è ò û Å Å</td>
</tr>
<tr>
<td>Ø Æ å í ò á æ ò å</td>
<td>À È É Ë Ì Í Ë Ë</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>240</td>
<td>p b μ ¶ ¾ ¼ ½ a</td>
</tr>
<tr>
<td>250</td>
<td>« ■ » ± □</td>
</tr>
</tbody>
</table>
### Norwegian Character Set

**NASC 47**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
</tr>
<tr>
<td>60</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>70</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>80</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>90</td>
<td>Z</td>
<td>Å</td>
<td>Ø</td>
<td>Â</td>
<td>^</td>
<td>_</td>
<td>`</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>100</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
</tr>
<tr>
<td>110</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
</tr>
<tr>
<td>120</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>æ</td>
<td>ø</td>
<td>å</td>
<td>–</td>
<td>€</td>
<td>☙</td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
</tr>
<tr>
<td>140</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
</tr>
<tr>
<td>150</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
</tr>
<tr>
<td>160</td>
<td>À</td>
<td>Å</td>
<td>È</td>
<td>É</td>
<td>Ê</td>
<td>Ë</td>
<td>Ì</td>
<td>Ì’</td>
<td>’</td>
<td>‘</td>
</tr>
<tr>
<td>170</td>
<td>^</td>
<td>~</td>
<td>Ü</td>
<td>Ü</td>
<td>£</td>
<td>–</td>
<td>Ŷ</td>
<td>Ŷ</td>
<td>°</td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>Ç</td>
<td>ç</td>
<td>Ň</td>
<td>Ň</td>
<td>Ñ</td>
<td>Ñ</td>
<td>i</td>
<td>ì</td>
<td>à</td>
<td>ã</td>
</tr>
<tr>
<td>190</td>
<td>f</td>
<td>c</td>
<td>â</td>
<td>ê</td>
<td>ô</td>
<td>û</td>
<td>á</td>
<td>ê</td>
<td>é</td>
<td>ô</td>
</tr>
<tr>
<td>200</td>
<td>à</td>
<td>è</td>
<td>ô</td>
<td>ù</td>
<td>ä</td>
<td>æ</td>
<td>ö</td>
<td>ü</td>
<td>Å</td>
<td>̈</td>
</tr>
<tr>
<td>210</td>
<td>Ø</td>
<td>Æ</td>
<td>å</td>
<td>í</td>
<td>ø</td>
<td>æ</td>
<td>Ä</td>
<td>ï</td>
<td>Ö</td>
<td>Ù</td>
</tr>
<tr>
<td>220</td>
<td>É</td>
<td>ï</td>
<td>ß</td>
<td>Ô</td>
<td>Á</td>
<td>ã</td>
<td>Ð</td>
<td>Ð</td>
<td>Ò</td>
<td>Ï</td>
</tr>
<tr>
<td>230</td>
<td>Í</td>
<td>Ó</td>
<td>Ô</td>
<td>Ó</td>
<td>Ŝ</td>
<td>Ŝ</td>
<td>Ú</td>
<td>Ý</td>
<td>Ý</td>
<td>̄</td>
</tr>
<tr>
<td>240</td>
<td>ß</td>
<td>b</td>
<td>·</td>
<td>µ</td>
<td>¶</td>
<td>³/₄</td>
<td>—</td>
<td>¼</td>
<td>½</td>
<td>a</td>
</tr>
<tr>
<td>250</td>
<td>ø</td>
<td>«</td>
<td>✈</td>
<td>»</td>
<td>±</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
<td>☙</td>
</tr>
</tbody>
</table>
## German Character Set

### NASC 49

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>40</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>60</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>70</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>90</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>100</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>110</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>120</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>130</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>140</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>150</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>160</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>170</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>180</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>190</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>200</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>210</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>220</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>230</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>240</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>250</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
### Japanese Latin Character Set

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td>(</td>
<td>)</td>
<td>*</td>
</tr>
<tr>
<td>+</td>
<td>,</td>
<td>.</td>
<td>/</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
</tr>
<tr>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
</tr>
<tr>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
</tr>
<tr>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>]</td>
<td>^</td>
</tr>
<tr>
<td>_</td>
<td>`</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
</tr>
<tr>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
</tr>
<tr>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td>}</td>
</tr>
<tr>
<td>~</td>
<td>€</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
<tr>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
<td>⌚</td>
</tr>
</tbody>
</table>
## Portuguese Character Set

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>!</td>
<td>'</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>!!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
</tr>
<tr>
<td>60</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
<td>$</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>70</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
</tr>
<tr>
<td>80</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
</tr>
<tr>
<td>90</td>
<td>Z</td>
<td>Æ</td>
<td>Ç</td>
<td>Œ</td>
<td>Ô</td>
<td>Ò</td>
<td>Â</td>
<td>Ê</td>
<td>Ë</td>
</tr>
<tr>
<td>100</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
</tr>
<tr>
<td>110</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
</tr>
<tr>
<td>120</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>á</td>
<td>ç</td>
<td>ã</td>
<td>Ó</td>
<td>Ó</td>
<td>Ó</td>
</tr>
<tr>
<td>130</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
</tr>
<tr>
<td>140</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
</tr>
<tr>
<td>150</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
</tr>
<tr>
<td>160</td>
<td>À</td>
<td>À</td>
<td>Ê</td>
<td>Ô</td>
<td>Ê</td>
<td>Ê</td>
<td>Ê</td>
<td>Ê</td>
<td>Ê</td>
</tr>
<tr>
<td>170</td>
<td>^</td>
<td>~</td>
<td>Ù</td>
<td>Ù</td>
<td>£</td>
<td>–</td>
<td>Ý</td>
<td>Ý</td>
<td>ç</td>
</tr>
<tr>
<td>180</td>
<td>Ç</td>
<td>ç</td>
<td>Ñ</td>
<td>Ñ</td>
<td>i</td>
<td>i</td>
<td>¥</td>
<td>¥</td>
<td>$</td>
</tr>
<tr>
<td>190</td>
<td>f</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
</tr>
<tr>
<td>200</td>
<td>à</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
</tr>
<tr>
<td>210</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
</tr>
<tr>
<td>220</td>
<td>É</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
</tr>
<tr>
<td>230</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
</tr>
<tr>
<td>240</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
</tr>
<tr>
<td>250</td>
<td>Ï</td>
<td>Ï</td>
<td>Ï</td>
<td>Ï</td>
<td>Ï</td>
<td>Ï</td>
<td>Ï</td>
<td>Ï</td>
<td>Ï</td>
</tr>
</tbody>
</table>

Intermec Fingerprint v8.70.0/v10.0.0 Programmer’s Reference Manual
## PCMAP Character Set

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td>(</td>
<td>)</td>
<td>*</td>
</tr>
<tr>
<td>40</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>50</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
</tr>
<tr>
<td>60</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
</tr>
<tr>
<td>70</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
</tr>
<tr>
<td>80</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>^</td>
</tr>
<tr>
<td>90</td>
<td>`</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
</tr>
<tr>
<td>100</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>110</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>á</td>
<td>ä</td>
<td>à</td>
<td>â</td>
<td>ã</td>
<td>ç</td>
<td>ê</td>
<td>è</td>
<td>ë</td>
<td>ë</td>
</tr>
<tr>
<td>140</td>
<td>ê</td>
<td>è</td>
<td>ë</td>
<td>ë</td>
<td>ë</td>
<td>ë</td>
<td>ë</td>
<td>ë</td>
<td>ñ</td>
<td>ñ</td>
</tr>
<tr>
<td>150</td>
<td>ñ</td>
<td>ñ</td>
<td>ñ</td>
<td>ñ</td>
<td>ñ</td>
<td>ñ</td>
<td>ñ</td>
<td>ñ</td>
<td>ñ</td>
<td>ñ</td>
</tr>
<tr>
<td>160</td>
<td>ñ</td>
<td>ñ</td>
<td>ñ</td>
<td>ñ</td>
<td>ñ</td>
<td>ñ</td>
<td>ñ</td>
<td>ñ</td>
<td>ñ</td>
<td>ñ</td>
</tr>
<tr>
<td>170</td>
<td>^</td>
<td>½</td>
<td>¼</td>
<td>i</td>
<td>«</td>
<td>»</td>
<td>À</td>
<td>Ý</td>
<td>Ý</td>
<td>Ý</td>
</tr>
<tr>
<td>180</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
</tr>
<tr>
<td>190</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
</tr>
<tr>
<td>200</td>
<td>à</td>
<td>ë</td>
<td>ê</td>
<td>ô</td>
<td>û</td>
<td>û</td>
<td>û</td>
<td>û</td>
<td>û</td>
<td>û</td>
</tr>
<tr>
<td>210</td>
<td>ë</td>
<td>ë</td>
<td>ë</td>
<td>ë</td>
<td>ë</td>
<td>ë</td>
<td>ë</td>
<td>ë</td>
<td>ë</td>
<td>ë</td>
</tr>
<tr>
<td>220</td>
<td>û</td>
<td>û</td>
<td>û</td>
<td>û</td>
<td>û</td>
<td>û</td>
<td>û</td>
<td>û</td>
<td>û</td>
<td>û</td>
</tr>
<tr>
<td>230</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
</tr>
<tr>
<td>240</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
</tr>
<tr>
<td>250</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
<td>Ò</td>
</tr>
</tbody>
</table>
## ANSI Character Set

### NASC -2

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>;</td>
</tr>
<tr>
<td>60</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>70</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
</tr>
<tr>
<td>80</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
</tr>
<tr>
<td>90</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>100</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
</tr>
<tr>
<td>110</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
</tr>
<tr>
<td>120</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td>}</td>
<td>~</td>
<td>€</td>
<td>€</td>
<td>€</td>
</tr>
<tr>
<td>130</td>
<td>,</td>
<td>f</td>
<td>”</td>
<td>…</td>
<td>†</td>
<td>†</td>
<td>†</td>
<td>†</td>
<td>†</td>
</tr>
<tr>
<td>140</td>
<td>œ</td>
<td>ŀ</td>
<td>Ŗ</td>
<td>Ū</td>
<td>Œ</td>
<td>Ž</td>
<td>Ŏ</td>
<td>Ŷ</td>
<td>Ÿ</td>
</tr>
<tr>
<td>150</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>♯</td>
<td>Š</td>
<td>Ş</td>
<td>Š</td>
<td>Š</td>
<td>Š</td>
</tr>
<tr>
<td>160</td>
<td>i</td>
<td>Ç</td>
<td>£</td>
<td>¥</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
</tr>
<tr>
<td>170</td>
<td>a</td>
<td>«</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>180</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>190</td>
<td>1/4</td>
<td>¿</td>
<td>À</td>
<td>Á</td>
<td>Â</td>
<td>À</td>
<td>Æ</td>
<td>Ç</td>
<td>È</td>
</tr>
<tr>
<td>200</td>
<td>È</td>
<td>Ë</td>
<td>Ë</td>
<td>Ë</td>
<td>Ë</td>
<td>Ë</td>
<td>Ë</td>
<td>Ë</td>
<td>Ë</td>
</tr>
<tr>
<td>210</td>
<td>Ô</td>
<td>Ô</td>
<td>Ô</td>
<td>Ô</td>
<td>Ô</td>
<td>Ô</td>
<td>Ô</td>
<td>Ô</td>
<td>Ô</td>
</tr>
<tr>
<td>220</td>
<td>Ü</td>
<td>Œ</td>
<td>B</td>
<td>á</td>
<td>â</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
</tr>
<tr>
<td>230</td>
<td>æ</td>
<td>ç</td>
<td>è</td>
<td>é</td>
<td>è</td>
<td>ë</td>
<td>ë</td>
<td>ë</td>
<td>ë</td>
</tr>
<tr>
<td>240</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
</tr>
<tr>
<td>250</td>
<td>û</td>
<td>û</td>
<td>û</td>
<td>û</td>
<td>û</td>
<td>û</td>
<td>û</td>
<td>û</td>
<td>û</td>
</tr>
</tbody>
</table>
### MS-DOS Latin 1 Character Set

<table>
<thead>
<tr>
<th><strong>Code</strong></th>
<th><strong>Character</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>! &quot; # $ % &amp; '</td>
</tr>
<tr>
<td>1</td>
<td>( ) * + , - . / 0 1</td>
</tr>
<tr>
<td>2</td>
<td>2 3 4 5 6 7 8 9 : ;</td>
</tr>
<tr>
<td>3</td>
<td>&lt; = &gt; ? @ A B C D E</td>
</tr>
<tr>
<td>4</td>
<td>F G H I J K L M N O</td>
</tr>
<tr>
<td>5</td>
<td>P Q R S T U V W X Y</td>
</tr>
<tr>
<td>6</td>
<td>Z [ \ ] ^ _ ` a b c</td>
</tr>
<tr>
<td>7</td>
<td>d e f g h i j k l m</td>
</tr>
<tr>
<td>8</td>
<td>n o p q r s t u v w</td>
</tr>
<tr>
<td>9</td>
<td>x y z {</td>
</tr>
<tr>
<td>10</td>
<td>é â à å å ç ê ë ë ì</td>
</tr>
<tr>
<td>11</td>
<td>î i Ì À È Æ æ Æ ô ô ô ô</td>
</tr>
<tr>
<td>12</td>
<td>û ù ý Ö Ü ø £ Ø × f</td>
</tr>
<tr>
<td>13</td>
<td>á í ó ú ñ Ñ ã ã ã</td>
</tr>
<tr>
<td>14</td>
<td>½ ¼ i « »</td>
</tr>
<tr>
<td>15</td>
<td>—</td>
</tr>
<tr>
<td>16</td>
<td>¥ — — — — — — —</td>
</tr>
<tr>
<td>17</td>
<td>â À À © — — — —</td>
</tr>
<tr>
<td>18</td>
<td>¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥</td>
</tr>
<tr>
<td>19</td>
<td>â Â ô ô ô ô ô ô ô</td>
</tr>
<tr>
<td>20</td>
<td>Î Î Î Î Î Î Î Î</td>
</tr>
<tr>
<td>21</td>
<td>Õ Õ Õ Õ Õ Õ Õ Õ Õ</td>
</tr>
<tr>
<td>22</td>
<td>Ô Ô Ô Ô Ô Ô Ô Ô Ô Ô</td>
</tr>
<tr>
<td>23</td>
<td>µ µ µ µ µ µ µ µ µ µ</td>
</tr>
<tr>
<td>24</td>
<td>— — — — — — — — — —</td>
</tr>
<tr>
<td>25</td>
<td>1 3 2</td>
</tr>
</tbody>
</table>
### MS-DOS Greek 1 Character Set

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td>(</td>
<td>)</td>
</tr>
<tr>
<td>40</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>50</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
</tr>
<tr>
<td>60</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>70</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>80</td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
</tr>
<tr>
<td>90</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
</tr>
<tr>
<td>100</td>
<td>`</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
</tr>
<tr>
<td>110</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
</tr>
<tr>
<td>120</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
</tr>
<tr>
<td>130</td>
<td>{</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>~</td>
<td>Ç</td>
<td>û</td>
<td>é</td>
<td>â</td>
<td>ä</td>
<td>å</td>
<td>À</td>
<td>Ç</td>
</tr>
<tr>
<td>150</td>
<td>ç</td>
<td>ê</td>
<td>ê</td>
<td>è</td>
<td>è</td>
<td>è</td>
<td>i</td>
<td>i</td>
<td>i</td>
</tr>
<tr>
<td>160</td>
<td>û</td>
<td>ù</td>
<td>Ø</td>
<td>Ô</td>
<td>Ü</td>
<td>á</td>
<td>£</td>
<td>é</td>
<td>ë</td>
</tr>
<tr>
<td>170</td>
<td>ë</td>
<td>ë</td>
<td>ë</td>
<td>ë</td>
<td>ë</td>
<td>ë</td>
<td>ë</td>
<td>ë</td>
<td>ë</td>
</tr>
<tr>
<td>180</td>
<td>½</td>
<td>Θ</td>
<td>I</td>
<td>«</td>
<td>»</td>
<td>‹</td>
<td>‹</td>
<td>‹</td>
<td>‹</td>
</tr>
<tr>
<td>190</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>210</td>
<td>Ó</td>
<td>Ó</td>
<td>Ó</td>
<td>Ó</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>220</td>
<td>δ</td>
<td>ε</td>
<td>ζ</td>
<td>η</td>
<td>θ</td>
<td>ι</td>
<td>κ</td>
<td>λ</td>
<td>μ</td>
</tr>
<tr>
<td>230</td>
<td>ν</td>
<td>ξ</td>
<td>ο</td>
<td>π</td>
<td>ρ</td>
<td>σ</td>
<td>ς</td>
<td>τ</td>
<td>υ</td>
</tr>
<tr>
<td>240</td>
<td>ψ</td>
<td>χ</td>
<td>ψ</td>
<td>ψ</td>
<td>ψ</td>
<td>ψ</td>
<td>ψ</td>
<td>ψ</td>
<td>ψ</td>
</tr>
<tr>
<td>250</td>
<td>ω</td>
<td>ω</td>
<td>ω</td>
<td>ω</td>
<td>ω</td>
<td>ω</td>
<td>ω</td>
<td>ω</td>
<td>ω</td>
</tr>
</tbody>
</table>
### MS-DOS Latin 2 Character Set

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>.</td>
<td>/</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
</tr>
<tr>
<td>60</td>
<td>&lt;=</td>
<td>=&gt;</td>
<td>?</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>70</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
</tr>
<tr>
<td>80</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
</tr>
<tr>
<td>90</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td></td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
<td>`</td>
</tr>
<tr>
<td>100</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
</tr>
<tr>
<td>110</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
</tr>
<tr>
<td>120</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
</tr>
<tr>
<td>130</td>
<td></td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
<td>`</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>140</td>
<td>é</td>
<td>â</td>
<td>ã</td>
<td>ũ</td>
<td>ü</td>
<td>ü</td>
<td>ö</td>
<td>œ</td>
<td>Œ</td>
</tr>
<tr>
<td>150</td>
<td>Î</td>
<td>Œ</td>
<td>Ç</td>
<td>É</td>
<td>L</td>
<td>Í</td>
<td>ô</td>
<td>Õ</td>
<td>Ô</td>
</tr>
<tr>
<td>160</td>
<td>Í</td>
<td>ß</td>
<td>Ö</td>
<td>Ù</td>
<td>Ù</td>
<td>Ù</td>
<td>Ŭ</td>
<td>Ù</td>
<td>Ù</td>
</tr>
<tr>
<td>170</td>
<td>á</td>
<td>í</td>
<td>ó</td>
<td>ú</td>
<td>À</td>
<td>ŀ</td>
<td>ŀ</td>
<td>ź</td>
<td>Ž</td>
</tr>
<tr>
<td>180</td>
<td>ż</td>
<td>Ć</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>ņ</td>
<td>ņ</td>
<td>ņ</td>
</tr>
<tr>
<td>190</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
</tr>
<tr>
<td>200</td>
<td>Ą</td>
<td>Ą</td>
<td>Ą</td>
<td>Ą</td>
<td>Ą</td>
<td>Ą</td>
<td>Ą</td>
<td>Ą</td>
<td>Ą</td>
</tr>
<tr>
<td>210</td>
<td>Ć</td>
<td>Ć</td>
<td>Ć</td>
<td>Ć</td>
<td>Ć</td>
<td>Ć</td>
<td>Ć</td>
<td>Ć</td>
<td>Ć</td>
</tr>
<tr>
<td>220</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
</tr>
<tr>
<td>230</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
</tr>
<tr>
<td>240</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
</tr>
<tr>
<td>250</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
<td>Ń</td>
</tr>
</tbody>
</table>
## MS-DOS Cyrillic Character Set

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td>(</td>
<td>)</td>
</tr>
<tr>
<td>40</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
</tr>
<tr>
<td>60</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>70</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>80</td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
</tr>
<tr>
<td>90</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>[</td>
<td>\</td>
<td>^</td>
<td>_</td>
<td>`</td>
<td>a</td>
</tr>
<tr>
<td>100</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
</tr>
<tr>
<td>110</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>120</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>~</td>
<td>_</td>
<td>`</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>140</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>j</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
</tr>
<tr>
<td>150</td>
<td>п</td>
<td>л</td>
<td>б</td>
<td>й</td>
<td>ё</td>
<td>ю</td>
<td>ю</td>
<td>ю</td>
<td>ю</td>
</tr>
<tr>
<td>160</td>
<td>а</td>
<td>б</td>
<td>в</td>
<td>г</td>
<td>д</td>
<td>е</td>
<td>ф</td>
<td>г</td>
<td>Ё</td>
</tr>
<tr>
<td>170</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
</tr>
<tr>
<td>180</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
</tr>
<tr>
<td>190</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
</tr>
<tr>
<td>200</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
</tr>
<tr>
<td>210</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
</tr>
<tr>
<td>220</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
</tr>
<tr>
<td>230</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
</tr>
<tr>
<td>240</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
</tr>
<tr>
<td>250</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
<td>Ё</td>
</tr>
</tbody>
</table>
### MS-DOS Turkish Character Set

**NASC 857**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>60</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>70</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>80</td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
</tr>
<tr>
<td>90</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
<td>`</td>
</tr>
<tr>
<td>100</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
</tr>
<tr>
<td>110</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
</tr>
<tr>
<td>120</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
</tr>
<tr>
<td>130</td>
<td>}</td>
<td>~</td>
<td>Ç</td>
<td>Ü</td>
<td>ç</td>
<td>ü</td>
<td>Ė</td>
<td>Ė</td>
<td>Ė</td>
</tr>
<tr>
<td>140</td>
<td>ê</td>
<td>ê</td>
<td>ê</td>
<td>ê</td>
<td>ê</td>
<td>ê</td>
<td>ê</td>
<td>ê</td>
<td>ê</td>
</tr>
<tr>
<td>150</td>
<td>î</td>
<td>î</td>
<td>î</td>
<td>î</td>
<td>î</td>
<td>î</td>
<td>î</td>
<td>î</td>
<td>î</td>
</tr>
<tr>
<td>160</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
<td>ò</td>
</tr>
<tr>
<td>170</td>
<td>½</td>
<td>¼</td>
<td>i</td>
<td>«</td>
<td>»</td>
<td>ć</td>
<td>ç</td>
<td>ç</td>
<td>ç</td>
</tr>
<tr>
<td>180</td>
<td>Å</td>
<td>Å</td>
<td>Å</td>
<td>Å</td>
<td>Å</td>
<td>Å</td>
<td>Å</td>
<td>Å</td>
<td>Å</td>
</tr>
<tr>
<td>190</td>
<td>¥</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>200</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>210</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>220</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>230</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>240</td>
<td>±</td>
<td>⅓</td>
<td>⅓</td>
<td>⅓</td>
<td>⅓</td>
<td>⅓</td>
<td>⅓</td>
<td>⅓</td>
<td>⅓</td>
</tr>
<tr>
<td>250</td>
<td>.</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>----</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>30</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>.</td>
<td>/</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>;</td>
</tr>
<tr>
<td>60</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>70</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
</tr>
<tr>
<td>80</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
</tr>
<tr>
<td>90</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
<td>`</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>100</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
</tr>
<tr>
<td>110</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
</tr>
<tr>
<td>120</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>~</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>Š</td>
<td>T</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
</tr>
<tr>
<td>150</td>
<td>——</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>Š</td>
<td>Š</td>
<td>š</td>
<td>ť</td>
<td>ť</td>
<td>ť</td>
<td>Ŧ</td>
<td>Ŧ</td>
<td>Ũ</td>
</tr>
<tr>
<td>170</td>
<td>Ľ</td>
<td>A</td>
<td>Ľ</td>
<td>Ľ</td>
<td>Ľ</td>
<td>Ľ</td>
<td>Ľ</td>
<td>Ľ</td>
<td>Ľ</td>
</tr>
<tr>
<td>180</td>
<td>μ</td>
<td>Ψ</td>
<td>α</td>
<td>Ψ</td>
<td>Ψ</td>
<td>Ψ</td>
<td>Ψ</td>
<td>Ψ</td>
<td>Ψ</td>
</tr>
<tr>
<td>190</td>
<td>Ł</td>
<td>Ą</td>
<td>Ł</td>
<td>Ą</td>
<td>Ł</td>
<td>Ą</td>
<td>Ł</td>
<td>Ą</td>
<td>Ł</td>
</tr>
<tr>
<td>200</td>
<td>Č</td>
<td>Ė</td>
<td>Ė</td>
<td>Ė</td>
<td>Ė</td>
<td>Ė</td>
<td>Ė</td>
<td>Ė</td>
<td>Ė</td>
</tr>
<tr>
<td>210</td>
<td>Ń</td>
<td>Ō</td>
<td>Ō</td>
<td>Ō</td>
<td>Ō</td>
<td>Ō</td>
<td>Ō</td>
<td>Ō</td>
<td>Ō</td>
</tr>
<tr>
<td>220</td>
<td>Ű</td>
<td>Ŷ</td>
<td>Ų</td>
<td>Ŵ</td>
<td>Ŵ</td>
<td>Ŵ</td>
<td>Ŵ</td>
<td>Ŵ</td>
<td>Ŵ</td>
</tr>
<tr>
<td>230</td>
<td>Ć</td>
<td>Ć</td>
<td>Ć</td>
<td>Ć</td>
<td>Ć</td>
<td>Ć</td>
<td>Ć</td>
<td>Ć</td>
<td>Ć</td>
</tr>
<tr>
<td>240</td>
<td>đ</td>
<td>ň</td>
<td>ň</td>
<td>ň</td>
<td>ň</td>
<td>ň</td>
<td>ň</td>
<td>ň</td>
<td>ň</td>
</tr>
<tr>
<td>250</td>
<td>ũ</td>
<td>ū</td>
<td>ū</td>
<td>ū</td>
<td>ū</td>
<td>ū</td>
<td>ū</td>
<td>ū</td>
<td>ū</td>
</tr>
</tbody>
</table>
## Windows Cyrillic Character Set

**NASC 1251**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>130</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>170</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>210</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>220</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>230</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>240</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- `! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _ ` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~ Ђ ѓ Ⴀ Ⴁ Ⴂ Ⴃ Ⴄ Ⴅ Ⴆ Ⴇ Ⴈ Ⴉ Ⴊ Ⴋ Ⴌ Ⴍ Ⴘ Ⴜ Ⴝ Ⴞ Ⴟ ± 1 i j l m n o p s t u v w x y z a b c d e f g h i j k l m n o p q r s t u v w x y z`
### Windows Latin 1 Character Set

#### NASC 1252

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>.</td>
<td>/</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
</tr>
<tr>
<td>60</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>70</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>80</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>90</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
<td>`</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>100</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
</tr>
<tr>
<td>110</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
</tr>
<tr>
<td>120</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td>}</td>
<td>~</td>
<td></td>
<td>€</td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>,</td>
<td>f</td>
<td>…</td>
<td>†</td>
<td>‡</td>
<td>^</td>
<td>°</td>
<td>^</td>
<td>›</td>
<td>‹</td>
</tr>
<tr>
<td>140</td>
<td>Ç</td>
<td>Ź</td>
<td>¤</td>
<td>‘</td>
<td>’</td>
<td>“</td>
<td>”</td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>–</td>
<td>—</td>
<td>~</td>
<td>™</td>
<td>Ş</td>
<td>œ</td>
<td>Ą</td>
<td>Ż</td>
<td>Ÿ</td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>i</td>
<td>C</td>
<td>£</td>
<td>¥</td>
<td></td>
<td></td>
<td>$</td>
<td></td>
<td>©</td>
<td></td>
</tr>
<tr>
<td>170</td>
<td>a</td>
<td>«</td>
<td>–</td>
<td>©</td>
<td>–</td>
<td>°</td>
<td>±</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>180</td>
<td>′</td>
<td>µ</td>
<td>¶</td>
<td>1</td>
<td>0</td>
<td>»</td>
<td>¼</td>
<td>½</td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>¾</td>
<td>Ć</td>
<td>À</td>
<td>Á</td>
<td>Â</td>
<td>Ä</td>
<td>Å</td>
<td>Æ</td>
<td>Ç</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>È</td>
<td>É</td>
<td>È</td>
<td>Œ</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
<td>Í</td>
<td>Ñ</td>
<td></td>
</tr>
<tr>
<td>210</td>
<td>Õ</td>
<td>Ô</td>
<td>Ô</td>
<td>Õ</td>
<td>×</td>
<td>Ø</td>
<td>Ù</td>
<td>Ú</td>
<td>Ù</td>
<td></td>
</tr>
<tr>
<td>220</td>
<td>Ü</td>
<td>Ý</td>
<td>Ù</td>
<td>Ù</td>
<td>Ù</td>
<td>Ù</td>
<td>Ù</td>
<td>Ù</td>
<td>Ù</td>
<td></td>
</tr>
<tr>
<td>230</td>
<td>æ</td>
<td>ç</td>
<td>ě</td>
<td>Ė</td>
<td>Ė</td>
<td>Ė</td>
<td>Ė</td>
<td>Ė</td>
<td>Ė</td>
<td></td>
</tr>
<tr>
<td>240</td>
<td>ņ</td>
<td>ņ</td>
<td>ņ</td>
<td>ņ</td>
<td>ņ</td>
<td>ņ</td>
<td>ņ</td>
<td>ņ</td>
<td>ņ</td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>ų</td>
<td>ų</td>
<td>ų</td>
<td>ų</td>
<td>ų</td>
<td>ų</td>
<td>ų</td>
<td>ų</td>
<td>ų</td>
<td></td>
</tr>
</tbody>
</table>
# Windows Greek Character Set

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>!</td>
<td>&quot;</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>;</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>70</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>80</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>90</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
<td>`</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>100</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
</tr>
<tr>
<td>110</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
</tr>
<tr>
<td>120</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>~</td>
<td>€</td>
<td>¢</td>
<td>Deutschland</td>
<td>™</td>
<td>©</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>170</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>μ</td>
<td>Π</td>
<td>Ε</td>
<td>Η</td>
<td>Ι</td>
<td>Τ</td>
<td>Λ</td>
<td>Ω</td>
<td>Ι</td>
<td>Ω</td>
</tr>
<tr>
<td>190</td>
<td>Y</td>
<td>Ω</td>
<td>ι</td>
<td>A</td>
<td>B</td>
<td>Γ</td>
<td>Δ</td>
<td>Ε</td>
<td>Z</td>
<td>H</td>
</tr>
<tr>
<td>200</td>
<td>Θ</td>
<td>Ι</td>
<td>K</td>
<td>Λ</td>
<td>Μ</td>
<td>Ν</td>
<td>Ξ</td>
<td>Ο</td>
<td>Π</td>
<td>Ρ</td>
</tr>
<tr>
<td>210</td>
<td>Σ</td>
<td>Τ</td>
<td>Υ</td>
<td>Φ</td>
<td>Χ</td>
<td>Ψ</td>
<td>Ω</td>
<td>Ι</td>
<td>Υ</td>
<td></td>
</tr>
<tr>
<td>220</td>
<td>á</td>
<td>é</td>
<td>ĕ</td>
<td>ů</td>
<td>a</td>
<td>b</td>
<td>γ</td>
<td>δ</td>
<td>ε</td>
<td></td>
</tr>
<tr>
<td>230</td>
<td>ζ</td>
<td>η</td>
<td>θ</td>
<td>ι</td>
<td>κ</td>
<td>λ</td>
<td>μ</td>
<td>ν</td>
<td>ξ</td>
<td>ο</td>
</tr>
<tr>
<td>240</td>
<td>π</td>
<td>ρ</td>
<td>ζ</td>
<td>σ</td>
<td>τ</td>
<td>ξ</td>
<td>η</td>
<td>ψ</td>
<td>ω</td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>ï</td>
<td>ü</td>
<td>ó</td>
<td>ú</td>
<td>ω</td>
<td>ι</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Windows Latin 5 Character Set

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
</tr>
<tr>
<td>60</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>70</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
</tr>
<tr>
<td>80</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
</tr>
<tr>
<td>90</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
<td>`</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>100</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
</tr>
<tr>
<td>110</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
</tr>
<tr>
<td>120</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td>}</td>
<td>~</td>
<td>€</td>
<td>¥</td>
</tr>
<tr>
<td>130</td>
<td>,</td>
<td></td>
<td>f</td>
<td></td>
<td></td>
<td></td>
<td>&quot;</td>
<td>†</td>
<td>‡</td>
</tr>
<tr>
<td>140</td>
<td>®</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>-</td>
<td>--</td>
<td>~</td>
<td>™</td>
<td>Š</td>
<td>∞</td>
<td>©</td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>i</td>
<td>φ</td>
<td>£</td>
<td>¥</td>
<td>!</td>
<td>$</td>
<td>&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>170</td>
<td>a</td>
<td>«</td>
<td>¬</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>±</td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>µ</td>
<td>¶</td>
<td>1</td>
<td>0</td>
<td>»</td>
<td>¼</td>
<td>½</td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>¾</td>
<td>¿</td>
<td>À</td>
<td>Á</td>
<td>Â</td>
<td>Ñ</td>
<td>Ă</td>
<td>Ä</td>
<td>Å</td>
</tr>
<tr>
<td>200</td>
<td>È</td>
<td>É</td>
<td>Ê</td>
<td>Ë</td>
<td>Ì</td>
<td>Í</td>
<td>Î</td>
<td>Ï</td>
<td>Ì</td>
</tr>
<tr>
<td>210</td>
<td>Ö</td>
<td>Ô</td>
<td>Õ</td>
<td>Ö</td>
<td>Ö</td>
<td>X</td>
<td>Ø</td>
<td>Ù</td>
<td>Ú</td>
</tr>
<tr>
<td>220</td>
<td>Ü</td>
<td>Í</td>
<td>Š</td>
<td>ß</td>
<td>à</td>
<td>á</td>
<td>â</td>
<td>ã</td>
<td>à</td>
</tr>
<tr>
<td>230</td>
<td>æ</td>
<td>ç</td>
<td>è</td>
<td>é</td>
<td>ê</td>
<td>ë</td>
<td>ì</td>
<td>ï</td>
<td>ì</td>
</tr>
<tr>
<td>240</td>
<td>ę</td>
<td>ę</td>
<td>é</td>
<td>é</td>
<td>é</td>
<td>é</td>
<td>ì</td>
<td>ì</td>
<td>ì</td>
</tr>
<tr>
<td>250</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
<td>ã</td>
</tr>
</tbody>
</table>

---

Intermec Fingerprint v8.70.0/v10.0.0 Programmer’s Reference Manual
### Windows Baltic Rim Character Set

**NASC 1257**

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>30</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
</tr>
<tr>
<td>60</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>70</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
</tr>
<tr>
<td>80</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
</tr>
<tr>
<td>90</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
<td>`</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>100</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
</tr>
<tr>
<td>110</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
</tr>
<tr>
<td>120</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td></td>
<td>}</td>
<td>~</td>
<td>€</td>
</tr>
<tr>
<td>130</td>
<td>,</td>
<td>.</td>
<td>&quot;</td>
<td>”</td>
<td>…</td>
<td>†</td>
<td>‡</td>
<td>‰</td>
<td>%</td>
</tr>
<tr>
<td>140</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>170</td>
<td>R</td>
<td>«</td>
<td>¬</td>
<td>¬</td>
<td>¬</td>
<td>¬</td>
<td>©</td>
<td>Ø</td>
<td>©</td>
</tr>
<tr>
<td>180</td>
<td>µ</td>
<td>¶</td>
<td>·</td>
<td>Ø</td>
<td>½</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>¼</td>
<td>æ</td>
<td>Å</td>
<td>Ç</td>
<td>Ć</td>
<td>Ă</td>
<td>Ą</td>
<td>Ć</td>
<td>Ę</td>
</tr>
<tr>
<td>200</td>
<td>Č</td>
<td>Ž</td>
<td>Ň</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
</tr>
<tr>
<td>210</td>
<td>N</td>
<td>O</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
</tr>
<tr>
<td>220</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
</tr>
<tr>
<td>230</td>
<td>ḳ</td>
<td>Ė</td>
<td>Ė</td>
<td>Ė</td>
<td>Ė</td>
<td>Ė</td>
<td>Ė</td>
<td>Ė</td>
<td>Ė</td>
</tr>
<tr>
<td>240</td>
<td>š</td>
<td>ň</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
</tr>
<tr>
<td>250</td>
<td>Š</td>
<td>Ť</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
<td>Ž</td>
</tr>
</tbody>
</table>
UTF-8 Character Set

The UTF-8 (Universal character set Transformation Format-8) was devised to be a character set which encodes all Unicode characters while maintaining compatibility with the US-ASCII (0 to 127 dec.) range of characters. The UTF-8 character set is available from Fingerprint v8.50. UTF-8 data is encoded with 1, 2, 3 or 4 bytes, depending on the character number range. The table below shows the UTF-8 binary sequences corresponding to the Unicode character number. Characters requiring a UTF-8 sequence of four bytes are currently not supported.

<table>
<thead>
<tr>
<th>Unicode character number range</th>
<th>UTF-8 Byte sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hex range</td>
<td>Binary</td>
</tr>
<tr>
<td>0000-007F</td>
<td>(x_7x_6x_5x_4x_3x_2x_1)</td>
</tr>
<tr>
<td>0080-07FF</td>
<td>(y_5y_4y_3y_2y_1x_6x_5x_4x_3x_2x_1)</td>
</tr>
<tr>
<td>0800-FFFF</td>
<td>(z_5z_4z_3y_5y_4y_3y_2y_1x_6x_5x_4x_3x_2x_1)</td>
</tr>
<tr>
<td>010000-10FFFF</td>
<td>Not currently supported.</td>
</tr>
</tbody>
</table>

The following table outlines the procedure to convert a Unicode character code in hexadecimal format to the UTF-8 byte decimal value necessary to print the characters.

<table>
<thead>
<tr>
<th>Procedure explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify your desired character and its Unicode Hex value. Identify the range it belongs to in order to find the number of bytes necessary for UTF-8 encoding, see table above.</td>
<td>Cyrillic capital letter ZHE: Ж</td>
</tr>
<tr>
<td>Hex value: 0416</td>
<td>0416 is in the range 0080-07FF and requires thus two bytes to be encoded.</td>
</tr>
<tr>
<td>Convert to binary</td>
<td></td>
</tr>
<tr>
<td>Identify (x) and (y) bits (and (z) if applicable) as shown in the table above. Start from the least significant digits to the right and pad with zeroes to the left if necessary.</td>
<td></td>
</tr>
<tr>
<td>Convert the bytes to decimal format and use these in your print command.</td>
<td>208 150</td>
</tr>
<tr>
<td>prntx chr$(208)+chr$(150)</td>
<td></td>
</tr>
</tbody>
</table>

Note: The conversions between hex-binary-decimal can be done with most scientific calculators (for example the Windows calculator).

The UTF-8 character set is invoked using NASC 8, NASC "UTF-8" or NASCD "UTF-8". When selecting UTF-8 with the NASC command, the font must be selected with the FONT command. Disable UTF-8 encoding by choosing a different character set with the NASC command.

Use FONTD to select the desired font if you invoke UTF-8 with the NASCD command. Having selected UTF-8 with the NASCD command, you must actively disable it with NASCD "" before returning to a single-byte character set.

Note: In order to avoid confusion between active character sets and fonts, it is recommended that you use only the NASC and FONT commands.
when working with UTF-8 unless you have experience with the NASCD and FONTD commands.

When using UTF-8, it is important that the font contains the desired characters. Complete lists of characters included in pre-installed fonts can be found in the *Intermec Fingerprint, Font Reference Manual*. The default font, Swiss 721 BT, contains the largest number of glyphs of the pre-installed fonts. More information on adding fonts to the printer can be found under the FONT command in Chapter 2. Unicode character numbers can be found at the web site of the Unicode organization (www.unicode.org).

It is not recommended to have UTF-8 enabled when printing bar codes since bar code data will use the UTF-8 byte sequence as input, while the human readable uses the UTF-8 mapped character number.

**Note:** Recall that FONT and FONTD commands are reset to their defaults after a PRINTFEED (or CLL) command. That is not the case for NASC and NASCD commands.

**Example**

The following example will print the Hiragana Letter Small A character (Unicode hex 3041), corresponding to the UTF-8 sequence 227 dec. + 129 dec. + 129 dec., in the *Song* font. This is followed by the Cyrillic Capital Letter ZHE (Unicode hex 0416) in the *Swiss 721 BT* font.

```
10  NASC "UTF-8"
20  FONT "Song"
30  PRTXT CHR$(227)+CHR$(129)+CHR$(129)
40  PRTXT " = Hiragana Letter Small A"
50  PRPOS 0,35
60  FONT "Swiss 721 BT"
70  PRTXT CHR$(208)+CHR$(150)
80  PRTXT " = Cyrillic Capital Letter ZHE"
90  PRINTFEED
```
## Scaleable Fonts

The printer comes with 15 pre-installed scaleable fonts. The printout samples below are in 10 point size, no slant, and 100% width. The quality of these samples does not exactly correspond to the printout quality from your printer, which is affected by printhead density, printing method, type of media and ribbon, and a number of other factors.

<table>
<thead>
<tr>
<th>Font Name</th>
<th>Sample Text</th>
<th>Sample Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Century Schoolbook BT</td>
<td>THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG</td>
<td><img src="image" alt="Century Schoolbook BT" /></td>
</tr>
<tr>
<td>DingDings SWA</td>
<td>THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG</td>
<td><img src="image" alt="DingDings SWA" /></td>
</tr>
<tr>
<td>Dutch 801 Roman BT</td>
<td>THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG</td>
<td><img src="image" alt="Dutch 801 Roman BT" /></td>
</tr>
<tr>
<td>Dutch 801 Bold BT</td>
<td>THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG</td>
<td><img src="image" alt="Dutch 801 Bold BT" /></td>
</tr>
<tr>
<td>Futura Light BT</td>
<td>THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG</td>
<td><img src="image" alt="Futura Light BT" /></td>
</tr>
<tr>
<td>Letter Gothic 12 Pitch BT</td>
<td>THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG</td>
<td><img src="image" alt="Letter Gothic 12 Pitch BT" /></td>
</tr>
<tr>
<td>Monospace 821 BT</td>
<td>THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG</td>
<td><img src="image" alt="Monospace 821 BT" /></td>
</tr>
<tr>
<td>Monospace 821 Bold BT</td>
<td>THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG</td>
<td><img src="image" alt="Monospace 821 Bold BT" /></td>
</tr>
<tr>
<td>OCR-A BT</td>
<td>THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG</td>
<td><img src="image" alt="OCR-A BT" /></td>
</tr>
<tr>
<td>OCR-B 10 Pitch BT</td>
<td>THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG</td>
<td><img src="image" alt="OCR-B 10 Pitch BT" /></td>
</tr>
<tr>
<td>Prestige 12 Pitch Bold BT</td>
<td>THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG</td>
<td><img src="image" alt="Prestige 12 Pitch Bold BT" /></td>
</tr>
<tr>
<td>Swiss 721 BT</td>
<td>THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG</td>
<td><img src="image" alt="Swiss 721 BT" /></td>
</tr>
<tr>
<td>Swiss 721 Bold BT</td>
<td>THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG</td>
<td><img src="image" alt="Swiss 721 Bold BT" /></td>
</tr>
<tr>
<td>Swiss 721 Bold Condensed BT</td>
<td>THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG</td>
<td><img src="image" alt="Swiss 721 Bold Condensed BT" /></td>
</tr>
<tr>
<td>Zurich Extra Condensed Bold</td>
<td>THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG</td>
<td><img src="image" alt="Zurich Extra Condensed Bold" /></td>
</tr>
</tbody>
</table>
Chapter 4—Character Sets and Fonts

Bitmap Fonts

It is possible to use fonts in the “old” Intermec .ATF bitmap font format. This feature improves compatibility with custom-made programs originally created in Fingerprint v6.xx or earlier versions.

Downloading an .ATF font (for example XX030RSN.ATF) to the printer produces three fonts in the memory; one without any extension (for example XX030RSN), one with the extension .1 (for example XX030RSN.1), and one with the extension .2 (for example XX030RSN.2). When using bitmap fonts in Fingerprint v8.xx, the relation between print direction and extension is of no consequence.

It is recommended to exclude the font height parameter in the FONT and BARFONT statements and use MAG to enlarge the font. Slant does not work at all with bitmap fonts.

Font Aliases

The standard font names in Intermec Fingerprint are much longer than in earlier versions of Fingerprint and may be cumbersome to use. They are also incompatible with the LAYOUT statement, which restricts the font name to 10 characters.

However, it is possible to create a file containing a list of font aliases. The file should be named exactly as shown here (note the leading period character that specifies it as a system file):

"c: .FONTALIAS"

The format of the file should be:

"<Alias name #1>","<Name of font>"[,<size>,<slant>,,<width>]]
"<Alias name #2>"","<Name of font>"[,<size>,<slant>,,<width>]]
"<Alias name #3>"","<Name of font>"[,<size>,<slant>,,<width>]]

etc., etc.

The file can contain as many fontname aliases as required. The default size is 12 points, the default slant is 0°, and the default width is 100 (%).

A font alias can be used as any other font, but its size, slant, and width cannot be changed.

Note: The printer must be rebooted before the font alias can be used.

Examples:

"BODYTEXT","Century Schoolbook BT",10,0,80
"HEADLINE","Swiss 721 Bold BT",18,0,110
"WARNING","Swiss 721 BT",12

For more information on fonts and character sets, refer to the Intermec Fingerprint v.8.xx, Font Reference Manual.
## OCR-A BT Character Set

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>!</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
</tr>
<tr>
<td>60</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>70</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>80</td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
</tr>
<tr>
<td>90</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>]</td>
<td>^</td>
<td>_</td>
<td>`</td>
</tr>
<tr>
<td>100</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
</tr>
<tr>
<td>110</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
</tr>
<tr>
<td>120</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
</tr>
<tr>
<td>130</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>170</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>210</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>220</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>230</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>240</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 4—Character Sets and Fonts

OCR-B 10 Pitch BT Character Set

0 1 2 3 4 5 6 7 8 9
30 ! " # $ % & ’
40 ( ) * + , - . / 0 1
50 2 3 4 5 6 7 8 9 : ;
60 < = > ? @ A B C D E
70 F G H I J K L M N O
80 P Q R S T U V W X Y
90 Z [ \ ] ^ _ ` a b c
d e f g h i j k l m
n o p q r s t u v w
x y z { | } ~
120
130
140
150
160
170 “ ” “” ‘ “’ “” " ‘’
180 
190
200
210 |
220 ũ ū û ù u u u u u
230 æ æ æ æ æ æ æ æ æ
240 æ æ æ æ æ æ æ æ æ
250
### DingDings SWA Character Set

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 10 |   |   |   |   |   |   |   |   |   |
| 11 |   |   |   |   |   |   |   |   |   |
| 12 |   |   |   |   |   |   |   |   |   |
| 13 |   |   |   |   |   |   |   |   |   |
| 14 |   |   |   |   |   |   |   |   |   |
| 15 |   |   |   |   |   |   |   |   |   |
| 16 |   |   |   |   |   |   |   |   |   |
| 17 |   |   |   |   |   |   |   |   |   |
| 18 |   |   |   |   |   |   |   |   |   |
| 19 |   |   |   |   |   |   |   |   |   |
| 20 |   |   |   |   |   |   |   |   |   |
| 21 |   |   |   |   |   |   |   |   |   |
| 22 |   |   |   |   |   |   |   |   |   |
| 23 |   |   |   |   |   |   |   |   |   |
| 24 |   |   |   |   |   |   |   |   |   |
| 25 |   |   |   |   |   |   |   |   |   |
This chapter lists the bar codes included in the Intermec Fingerprint v8.70.0/v10.0.0 firmware and gives examples of some commonly used bar codes.
Introduction

The printer contains a number of bar code generators, which can produce highly readable bar codes in four different directions.

However, a general rule which applies to all thermal printers is that it is more difficult to print a bar code with the bars across the media path (ladder style) than along the media path (picket fence style.) Therefore, to ensure a highly readable printout, we recommend that you do not use narrow bars less than 3 dots, when printing bar codes with the bars across the media path (ladder style).

No such restrictions apply for bar codes with the bars along the media path (picket fence style).

Another factor, that affects the printout quality of the bar codes, is the print speed. Generally, a lower print speed gives a better quality, especially for ladder style bar codes and at low ambient temperatures. Do not use a higher print speed than necessary and consider the overall print cycle time. In some instances, a lower print speed may actually give better overall performance. We recommend you do your own tests with your unique applications to find the best compromise between printout quality, performance, and media.

Specifications for bar code symbologies can be obtained from organizations like:

EAN International
http://www.ean-int.org

UCC - The Uniform Code Council, Inc. (UCC)
http://www.uc-council.org

AIM International, Inc.
http://www.aimi.org

American National Standard Institute (ANSI)
http://www.ansi.org
## Supported Bar Codes

<table>
<thead>
<tr>
<th>Bar Code</th>
<th>Designation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Codabar</td>
<td>&quot;CODABAR&quot;</td>
</tr>
<tr>
<td>Code 11</td>
<td>&quot;CODE11&quot;</td>
</tr>
<tr>
<td>Code 16K</td>
<td>&quot;CODE16K&quot;</td>
</tr>
<tr>
<td>Code 39</td>
<td>&quot;CODE39&quot;</td>
</tr>
<tr>
<td>Code 39 full ASCII</td>
<td>&quot;CODE39A&quot;</td>
</tr>
<tr>
<td>Code 39 w. checksum</td>
<td>&quot;CODE39C&quot;</td>
</tr>
<tr>
<td>Code 49</td>
<td>&quot;CODE49&quot;</td>
</tr>
<tr>
<td>Code 93</td>
<td>&quot;CODE93&quot;</td>
</tr>
<tr>
<td>Code 128</td>
<td>&quot;CODE128&quot;</td>
</tr>
<tr>
<td>Code 128 subset A</td>
<td>&quot;CODE128A&quot;</td>
</tr>
<tr>
<td>Code 128 subset B</td>
<td>&quot;CODE128B&quot;</td>
</tr>
<tr>
<td>Code 128 subset C</td>
<td>&quot;CODE128C&quot;</td>
</tr>
<tr>
<td>Data Matrix</td>
<td>&quot;DATAMATRIX&quot;</td>
</tr>
<tr>
<td>DUN-14/16</td>
<td>&quot;DUN&quot;</td>
</tr>
<tr>
<td>EAN-8</td>
<td>&quot;EAN8&quot;</td>
</tr>
<tr>
<td>EAN-8 Composite with CC-A or CC-B</td>
<td>&quot;EAN8_CC&quot;</td>
</tr>
<tr>
<td>EAN-13</td>
<td>&quot;EAN13&quot;</td>
</tr>
<tr>
<td>EAN-13 Composite with CC-A or CC-B</td>
<td>&quot;EAN13_CC&quot;</td>
</tr>
<tr>
<td>EAN 128</td>
<td>&quot;EAN128&quot;</td>
</tr>
<tr>
<td>EAN 128 subset A</td>
<td>&quot;EAN128A&quot;</td>
</tr>
<tr>
<td>EAN 128 subset B</td>
<td>&quot;EAN128B&quot;</td>
</tr>
<tr>
<td>EAN 128 subset C</td>
<td>&quot;EAN128C&quot;</td>
</tr>
<tr>
<td>EAN.UCC 128 Composite with CC-A or CC-B</td>
<td>&quot;EAN128_CCAB&quot;</td>
</tr>
<tr>
<td>EAN.UCC 128 Composite with CC-C</td>
<td>&quot;EAN128_CCC&quot;</td>
</tr>
<tr>
<td>Five-Character Supplemental Code</td>
<td>&quot;ADDON5&quot;</td>
</tr>
<tr>
<td>Industrial 2 of 5</td>
<td>&quot;C2OF5IND&quot;</td>
</tr>
<tr>
<td>Industrial 2 of 5 w. checksum</td>
<td>&quot;C2OF5INDC&quot;</td>
</tr>
<tr>
<td>Interleaved 2 of 5</td>
<td>&quot;INT2OF5&quot;</td>
</tr>
<tr>
<td>Interleaved 2 of 5 w. checksum</td>
<td>&quot;INT2OF5C&quot;</td>
</tr>
<tr>
<td>Interleaved 2 of 5 A</td>
<td>&quot;I2OF5A&quot;</td>
</tr>
<tr>
<td>Matrix 2 of 5</td>
<td>&quot;C2OPSMAT&quot;</td>
</tr>
<tr>
<td>MaxiCode</td>
<td>&quot;MAXICODE&quot;</td>
</tr>
<tr>
<td>MicroPDF417</td>
<td>&quot;MICROPDF417&quot;</td>
</tr>
<tr>
<td>MSI (modified Plessey)</td>
<td>&quot;MSI&quot;</td>
</tr>
<tr>
<td>Planet</td>
<td>&quot;PLANET&quot;</td>
</tr>
<tr>
<td>PDF 417</td>
<td>&quot;PDP417&quot;</td>
</tr>
<tr>
<td>Plessey</td>
<td>&quot;PLESSEY&quot;</td>
</tr>
<tr>
<td>Postnet</td>
<td>&quot;POSTNET&quot;</td>
</tr>
<tr>
<td>QR Code</td>
<td>&quot;QRCODE&quot;</td>
</tr>
<tr>
<td>RSS-14</td>
<td>&quot;RSS14&quot;</td>
</tr>
<tr>
<td>RSS-14 Truncated</td>
<td>&quot;RSS14T&quot;</td>
</tr>
<tr>
<td>RSS-14 Stacked</td>
<td>&quot;RSS14S&quot;</td>
</tr>
<tr>
<td>RSS-14 Stacked Omnidirectional</td>
<td>&quot;RSS14SO&quot;</td>
</tr>
<tr>
<td>RSS-14 Limited</td>
<td>&quot;RSS14L&quot;</td>
</tr>
</tbody>
</table>
RSS-14 Expanded
RSS-14 Expanded Stacked
Straight 2 of 5
Two-Character Supplemental Code
UCC-128 Serial Shipping Container Code
UPC-5 digits Add-On Code
UPC-A
UPC-A Composite with CC-A or CC-B
UPC-D1
UPC-D2
UPC-D3
UPC-D4
UPC-D5
UPC-E
UPC-E Composite with CC-A or CC-B
UPC Shipping Container Code
One-and Two-Dimensional Bar Codes

On the following pages, a quick survey of the characteristics of some of the most common one- and two dimensional bar codes will be given. This information is only intended to help you avoid entering unacceptable parameters or input data. For further information, please refer to the standard literature on the subject of bar codes.

For simple bar codes, like EAN and UPC codes, it is recommended to use separate instructions, whereas for more complex bar codes like MicroPDF417 or QR Code, the BARSET statement is required.

The following bar codes are described:

• Code 39
• Code 128
• Data Matrix
• EAN-8
• EAN-13
• EAN 128
• Interleaved 2 of 5
• MaxiCode
• MicroPDF417
• PDF417
• QR Code
• RSS-14
• RSS-14 Truncated
• RSS-14 Stacked
• RSS-14 Stacked Omnidirectional
• RSS-14 Limited
• RSS-14 Expanded
• RSS-14 Expanded Stacked
• UPC-A
• UPC-E

Add-on codes and composite bar codes are described in separate sections later in this chapter.
## Code 39

### Separate Instructions

| BARTYPE: | "CODE39"  
|          | "CODE39A"  
|          | "CODE39C"  
| BARRATIO: | 2:1-3:1. Default: 3:1  
| BARMAG: | No restriction, but if the narrow element is less than 4 dots wide, then the ratio must be larger than 2.25:1 (9:4). Default 2.  
| BARHEIGHT: | No restriction. Default: 100.  
| BARFONT: | No restriction  

### Input Data

| No. of characters: | Unlimited.  
| Check digit: | No  
| Digits: | 0-9  
| Uppercase letters: | A-Z  
| Lowercase letters: | No  
| Punctuation marks: | , . space $ / + %  
| Start character: | * Added automatically  
| Stop character: | * Added automatically  

### Remarks

Code 39 is an alphanumeric self-checking discrete code.

Code39A allows for the entire ASCII set to be encoded (128 characters).

Code39C allows characters that are not in the input set to be sent into the symbol without error. The character will not be printed.
Chapter 5—Bar Codes

Code 128

Separate Instructions

<table>
<thead>
<tr>
<th>BARTYPE:</th>
<th>&quot;CODE128&quot; &quot;CODE128A&quot; &quot;CODE128B&quot; &quot;CODE128C&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARRATIO:</td>
<td>Fixed. BARRATIO statement ignored.</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>≥ 2. Default: 2.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>No restriction. Default: 100.</td>
</tr>
<tr>
<td>BARFONT:</td>
<td>No restriction</td>
</tr>
</tbody>
</table>

BARSET Instruction

<table>
<thead>
<tr>
<th>BARSET</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>[#&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter in the syntax</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>Bar code type</td>
</tr>
<tr>
<td></td>
<td>&quot;CODE128&quot; &quot;CODE128A&quot; &quot;CODE128B&quot; &quot;CODE128C&quot;</td>
</tr>
<tr>
<td>&lt;nexp1&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp2&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp4&gt;</td>
<td>Barheight</td>
</tr>
<tr>
<td></td>
<td>No restriction. Default: 100.</td>
</tr>
<tr>
<td>&lt;nexp5&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp6&gt;</td>
<td>Character exclusion in bar code (does not affect human readable)</td>
</tr>
<tr>
<td></td>
<td>0 = Disables parenthesis and spaces in the bar code data.</td>
</tr>
<tr>
<td></td>
<td>Non-zero = the bar code and human readable field will include exactly the</td>
</tr>
<tr>
<td></td>
<td>same data.</td>
</tr>
<tr>
<td>&lt;nexp7&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp8&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp9&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp10&gt;</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>
Chapter 5—Bar Codes

Input Data

<table>
<thead>
<tr>
<th>No. of characters:</th>
<th>Unlimited</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check digit:</td>
<td>One</td>
</tr>
<tr>
<td>Added automatically.</td>
<td></td>
</tr>
<tr>
<td>Input characters:</td>
<td>ASCII 0-127</td>
</tr>
<tr>
<td>Function characters</td>
<td>FNC1 ASCII 128 dec.)</td>
</tr>
<tr>
<td>FNC2 ASCII 129 dec.)</td>
<td></td>
</tr>
<tr>
<td>FNC3 ASCII 130 dec.)</td>
<td></td>
</tr>
<tr>
<td>FNC4 ASCII 131 dec.)</td>
<td></td>
</tr>
<tr>
<td>See note 1.</td>
<td></td>
</tr>
<tr>
<td>Start characters:</td>
<td>See note 2.</td>
</tr>
<tr>
<td>Code characters:</td>
<td>See note 1 &amp; 2</td>
</tr>
<tr>
<td>Shift characters:</td>
<td>See note 1 &amp; 2</td>
</tr>
<tr>
<td>Stop characters:</td>
<td>Always</td>
</tr>
</tbody>
</table>

Remarks

Note 1:
Function characters FNC1-4, code characters, and shift characters require either an 8-bit communication protocol, remapping to an ASCII value between 0-127 dec., or the use of an CHR$ function.

FNC2-4 are not allowed in Subset C.

Note 2:
Code 128 has automatic selection of start character and character subset (that is, selects optimal start character and handles shift and changes of subset depending on the content of the input data), whereas Code 128A, Code 128B, and Code 128C selects subset A, B, and C respectively. The last character in the bar code name signifies both the start character and the chosen subset.

The selected subset can be changed anywhere in the input string, either for a single character using a Shift character (not for Subset C), or for the remainder of the input string using a Code character (all subsets).

The Shift and Code characters consist of a combination of two characters:

- Two left-pointing double angle quotation marks («) specify a Shift character.
  Shift character: « « (« = ASCII 171 dec.)

- One left-pointing double angle quotation mark («) specifies a Code character. It should be followed by an uppercase letter that specifies the subset:
  Code character: « + A|B|C (« = ASCII 171 dec.)
Data Matrix

Separate Instructions

<table>
<thead>
<tr>
<th>Bar Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARTYPE</td>
<td>&quot;DATAMATRIX&quot;</td>
</tr>
<tr>
<td>BARRATIO</td>
<td>Fixed. Values will be interpreted as BARMAG.</td>
</tr>
<tr>
<td>BARMAG</td>
<td>&lt;128. Default 2.</td>
</tr>
<tr>
<td>BARHEIGHT</td>
<td>Not applicable</td>
</tr>
<tr>
<td>BARFONT</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

BARSET Instruction

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[#&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter in the syntax</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>Bar code type &quot;DATAMATRIX&quot;</td>
</tr>
<tr>
<td>&lt;nexp1&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp2&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp4&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp5&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp6&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp7&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp8&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp9&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp10&gt;</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

Input Data

<table>
<thead>
<tr>
<th>Character Details</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of characters</td>
<td>2,335 ASCII characters Only A-Z &amp; . , - / represented</td>
</tr>
<tr>
<td>Check digit</td>
<td>Yes Added automatically.</td>
</tr>
<tr>
<td>Input characters</td>
<td>ASCII 0-255 decimal</td>
</tr>
</tbody>
</table>

Remarks

Note 1:
ECC 200 type is used for this Datamatrix symbol. ECC 200 data may be encoded using any of the encodation schemes ASCII, C40, Text, X12, EDIFACT or Base 256. ASCII encodation is the basic scheme. All other encodation schemes are invoked from ASCII encodation and return to this scheme.

Note 2:
The number of characters will decide the size of symbol. For example data “123456” will generate a 10 × 10 (rows, columns) Datamatrix symbol, and 72 digits will generate a 24 × 24 Datamatrix symbol. Less characters can be used for all symbols (10 × 10, 12 × 12, . , 144 × 144) if the data includes non-numeric characters.
Chapter 5 — Bar Codes

EAN-8

Separate Instructions

<table>
<thead>
<tr>
<th>BARTYPE:</th>
<th>&quot;EAN8&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARRATIO:</td>
<td>Fixed ratio. BARRATIO statement ignored.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>No restriction. Default: 100.</td>
</tr>
<tr>
<td>BARFONT:</td>
<td>Barfont generated automatically. BARFONT statement ignored. BARFONT ON/OFF work.</td>
</tr>
</tbody>
</table>

Input Data

<table>
<thead>
<tr>
<th>No. of characters:</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check digit:</td>
<td>One Added automatically</td>
</tr>
<tr>
<td>Digits:</td>
<td>0-9</td>
</tr>
<tr>
<td>Uppercase letters:</td>
<td>No</td>
</tr>
<tr>
<td>Lowercase letters:</td>
<td>No</td>
</tr>
<tr>
<td>Punctuation marks:</td>
<td>No</td>
</tr>
<tr>
<td>Start character:</td>
<td>No</td>
</tr>
<tr>
<td>Stop character:</td>
<td>No</td>
</tr>
</tbody>
</table>
EAN-13

Separate Instructions

<table>
<thead>
<tr>
<th><strong>BARTYPE:</strong></th>
<th>&quot;EAN13&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BARRATIO:</strong></td>
<td>Fixed ratio. BARRATIO statement ignored.</td>
</tr>
<tr>
<td><strong>BARMAG:</strong></td>
<td>Maximum 8. Default 2.</td>
</tr>
<tr>
<td><strong>BARHEIGHT:</strong></td>
<td>No restriction. Default: 100.</td>
</tr>
<tr>
<td><strong>BARFONT:</strong></td>
<td>Barfont generated automatically. BARFONT statement ignored. BARFONT ON/OFF work.</td>
</tr>
</tbody>
</table>

Input Data

<table>
<thead>
<tr>
<th><strong>No. of characters:</strong></th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Check digit:</strong></td>
<td>One Added automatically</td>
</tr>
<tr>
<td><strong>Digits:</strong></td>
<td>0-9</td>
</tr>
<tr>
<td><strong>Uppercase letters:</strong></td>
<td>No</td>
</tr>
<tr>
<td><strong>Lowercase letters:</strong></td>
<td>No</td>
</tr>
<tr>
<td><strong>Punctuation marks:</strong></td>
<td>No</td>
</tr>
<tr>
<td><strong>Start character:</strong></td>
<td>No</td>
</tr>
<tr>
<td><strong>Stop character:</strong></td>
<td>No</td>
</tr>
</tbody>
</table>
## EAN 128

### Separate Instructions

<table>
<thead>
<tr>
<th>BARTYPE:</th>
<th>&quot;EAN128&quot;</th>
<th>&quot;EAN128A&quot;</th>
<th>&quot;EAN128B&quot;</th>
<th>&quot;EAN128C&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARRATIO:</td>
<td>Fixed. BARRATIO statement ignored.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BARMAG:</td>
<td>≥ 2. Default: 2.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>No restriction. Default: 100.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BARFONT:</td>
<td>No restriction</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### BARSET Instruction

<table>
<thead>
<tr>
<th>BARSET</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>[#&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter in the syntax</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>Bar code type</td>
</tr>
<tr>
<td>&quot;EAN128&quot;</td>
<td>&quot;EAN128A&quot;</td>
</tr>
<tr>
<td>&quot;EAN128B&quot;</td>
<td>&quot;EAN128C&quot;</td>
</tr>
<tr>
<td>&lt;nexp1&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp2&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp3&gt;</td>
<td>Barmag/Enlargement</td>
</tr>
<tr>
<td>&lt;nexp4&gt;</td>
<td>Barheight</td>
</tr>
<tr>
<td>&lt;nexp5&gt;</td>
<td>No restriction. Default: 100.</td>
</tr>
<tr>
<td>&lt;nexp6&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp7&gt;</td>
<td>Character exclusion in bar code (does not affect human readable)</td>
</tr>
<tr>
<td>0</td>
<td>Disables parenthesis and spaces in the bar code data.</td>
</tr>
<tr>
<td>Non-zero</td>
<td>the bar code and human readable field will include exactly the same data.</td>
</tr>
<tr>
<td>&lt;nexp8&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp9&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp10&gt;</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

### Input Data

| No. of characters: | Unlimited |
| Check digit: | Trailing symbol check character |
| Added automatically. |
| Input characters: | ASCII 0-127 |
| According to Roman 8 character set. |
| Start characters: | See note 2. |
| Code characters: | See note 1 & 2 |
| Shift characters: | See note 1 & 2 |
| Stop characters: | Always |
| Added automatically. |
Remarks

This bar code is identical to Code 128 with the exception that the initial FNC1 function character is generated automatically.

**Note 1:**
Code characters and shift characters require either an 8-bit communication protocol, remapping to an ASCII value between 0-127 dec., or the use of an CHR$ function.

**Note 2:**
EAN 128 has automatic selection of start character and character subset (that is, selects optimal start character and handles shift and changes of subset depending on the content of the input data), whereas EAN 128A, EAN 128B, and EAN 128C selects subset A, B, and C respectively. The last character in the bar code name signifies both the start character and the chosen subset.

The selected subset can be changed anywhere in the input string, either for a single character using a Shift character (not for Subset C), or for the remainder of the input string using a Code character (all subsets).

The Shift and Code characters consist of a combination of two characters:

- Two left-pointing double angle quotation marks («) specify a Shift character.
  - Shift character: «« (« = ASCII 171 dec.)

- One left-pointing double angle quotation mark («) specifies a Code character. It should be followed by an uppercase letter that specifies the subset.
  - Code character: « + A|B|C (« = ASCII 171 dec.)
Chapter 5—Bar Codes

Interleaved 2 of 5

Separate Instructions

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BARTYPE:</td>
<td>&quot;INT2OF5&quot;</td>
</tr>
<tr>
<td>BARRATIO:</td>
<td>2:1-3:1. Default: 3:1</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>No restriction. Default 2.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>No restriction. Default: 100.</td>
</tr>
<tr>
<td>BARFONT:</td>
<td>No restriction</td>
</tr>
</tbody>
</table>

Input Data

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of characters:</td>
<td>Unlimited.</td>
</tr>
<tr>
<td>Check digit:</td>
<td>No</td>
</tr>
<tr>
<td>Digits:</td>
<td>0-9</td>
</tr>
<tr>
<td>Uppercase letters:</td>
<td>No</td>
</tr>
<tr>
<td>Lowercase letters:</td>
<td>No</td>
</tr>
<tr>
<td>Punctuation marks:</td>
<td>No</td>
</tr>
<tr>
<td>Start character:</td>
<td>Yes</td>
</tr>
<tr>
<td>Stop character:</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Added automatically

Remarks

Interleaved 2 of 5 is a numeric code where input digits are encoded in pairs. If an odd number of digits is entered, a leading zero will be added automatically.
MaxiCode

Separate Instructions

<table>
<thead>
<tr>
<th>BARTYPE:</th>
<th>&quot;MAXICODE&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARRATIO:</td>
<td>Not applicable. Input ignored.</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>Not applicable. Input ignored.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>Not applicable. Input ignored.</td>
</tr>
<tr>
<td>BARFONT:</td>
<td>Not applicable. Input ignored.</td>
</tr>
<tr>
<td>BARSET:</td>
<td>Not applicable. Input ignored.</td>
</tr>
</tbody>
</table>

Remarks

MaxiCode requires 8 fields of data separated by a LF character, which is entered as CHR$(10). Regardless of which mode is chosen, all eight fields must contain valid data and must be present in final in data string, see table below

\[ F_n = \text{Field No, (mode n) = decoded by reader} \]

\[ F_1 \text{ (mode 2&3): } \]
5 Characters, numeric (mode 2) or
6 alphanumeric characters (mode 3).

\[ F_2 \text{ (mode 2): } \]
4 digits [0-9999]

\[ F_3 \text{ (mode 2&3): } \]
Country code 3 digits [0-999]

\[ F_4 \text{ (mode 2&3): } \]
Service class, 3 digits [0-999]

\[ F_5 \text{ (LPM mode 2,3,4,:)}: \]
User defined message.

\[ F_6: \]
Mode selector, one digit [2|3|4].

\[ F_7: \]
Position in structured append, one digit [1-8]

\[ F_8: \]
Total number of symbols in structure, one digit [1-8]

F_8 in structured append is the trigger for structured append mode.

If \( F_8 > 1 \), the two first code word in secondary message will be a pad followed by position code word. \( F_8 \) has higher precedence than \( F_7 \).

If \( F_7 > 1 \) when \( F_8 = 1 \), the two first codeword will not signal structured append.

If \( F_8 > 1 \), \( F_7 \) may be \( > F_8 \) without error and structured append codeword will signal given values.

No of characters: Up to 84 on mode 2 & 3 or up to 138 in mode 4.

Check character: Automatically, Reed-Solomon algorithm.

Data type: ASCII 0-255
## Chapter 5—Bar Codes

### Summary

<table>
<thead>
<tr>
<th>Data element</th>
<th>Length</th>
<th>Type</th>
<th>Sample</th>
<th>Mode 2: Structured data message for US Destinations</th>
<th>Mode 3: Structured data message for International Destinations</th>
<th>Mode 4: Standard Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Message</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zip code + 4 digit extension</td>
<td>Mode 2: 9</td>
<td>Numeric (mode 2)</td>
<td>1280</td>
<td>Mandatory</td>
<td>Mandatory</td>
<td>N/A</td>
</tr>
<tr>
<td>Country Code</td>
<td>3</td>
<td>Numeric</td>
<td>840</td>
<td>Mandatory</td>
<td>Mandatory</td>
<td>N/A</td>
</tr>
<tr>
<td>Service Class</td>
<td>3</td>
<td>Numeric</td>
<td>001</td>
<td>Mandatory</td>
<td>Mandatory</td>
<td>N/A</td>
</tr>
<tr>
<td>Secondary Message</td>
<td>84/138</td>
<td>Alphanumeric</td>
<td>This is the secondary message</td>
<td>Optional</td>
<td>Optional</td>
<td>N/A</td>
</tr>
<tr>
<td>Sample data streams as decoded by scanner</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1/ Header is encoded into secondary message.

### Example

This example shows how the following information is used to create a MaxiCode symbology:

- **Zip Code**: 84170
- **Zip Code Extension**: 1280
- **Country Code**: 840
- **Class of Service**: 001
- **Message Header**: []><RS>01
- **Year**: 96
- **Tracking Number**: 1Z12345675
- **SCAC**: UPSN
- **UPS Shipper Number**: 12345E
- **Julian Day of Pickup**: 089
- **Shipment ID**: 1324567
- **Package**: 1/1
- **Weight**: 10.1
- **Address Validation**: Y
- **Ship to Street**: 1 Main ST
- **Ship to City**: PITTSBURGH
- **Ship to State**: PA

```plaintext
10 PRPOS 100,100
20 DIR 1
30 ALIGN 1
40 a$="84170"+CHR$(10)+"1280"+CHR$(10)+"840"+CHR$(10)+"001"+CHR$(10)+"01"+CHR$(29)+"96"+"1Z12345675"+CHR$(29)+"UPSN"+CHR$(29)+"12345E"+CHR$(29)+"089"+CHR$(29)+"1234567"+CHR$(29)+"1/1"+CHR$(29)+"10.1"+CHR$(29)+"Y"+CHR$(29)+"1 MAIN ST"
50 b$= CHR$(29)+"PITTSBURGH"+CHR$(29)+"PA"+CHR$(29)+CHR$(30)+CHR$(4)+CHR$(10)+"2"+CHR$(10)+"1"+CHR$(10)+"1"
60 BARTYPE "MAXICODE"
70 PRBAR a$;b$
80 PRINTFEED
RUN
```
MicroPDF417

MicroPDF417 is a multi-row symbology based on PDF417. A limited set of symbol sizes is available where each size has a fixed level of error correction. Most symbol characteristics such as data character encodation, error correction, and symbol character sets are identical to those of PDF417. Up to 250 alphanumeric characters or 366 numeric digits can be encoded in a symbol.

BARSET Instruction

<table>
<thead>
<tr>
<th>BARSET</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>[#&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter in the syntax</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>Bar code type</td>
</tr>
<tr>
<td>&lt;nexp1&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp2&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp5&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp6&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp7&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp8&gt;</td>
<td>Number of rows 0, 4-44. Default: 0 (=automatic)</td>
</tr>
<tr>
<td>&lt;nexp9&gt;</td>
<td>Number of columns 0-4. Default: 0 (=automatic)</td>
</tr>
<tr>
<td>&lt;nexp10&gt;</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

Remarks

Setting The Number of Rows And Columns
The symbol size is defined by specifying the number of rows and columns. Not all combinations of rows and columns are allowed. The table below illustrates the valid combinations.

<table>
<thead>
<tr>
<th>No. of columns</th>
<th>Valid number of rows for each no. of columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11 14 17 20 24 28 - - - - - - - - - - - -</td>
</tr>
<tr>
<td>2</td>
<td>8 11 14 17 20 23 26 - - - - - - - - - - - -</td>
</tr>
<tr>
<td>3</td>
<td>6 8 10 12 15 20 26 32 38 44 - - - - - - - - - - - -</td>
</tr>
<tr>
<td>4</td>
<td>4 6 8 10 12 15 20 26 32 38 44 - - - - - - - - - - - -</td>
</tr>
</tbody>
</table>

If the number of rows is set to a value that does not match the valid values for the given number of columns, the printer will automatically choose a larger number from the list of valid values.

Automatic Selection
The number of columns and rows can be set automatically by the printer. If the number of columns is set to 0, the printer will set the number of columns as well as the number of rows automatically, regardless of the number of rows specified. The printer will try to fit the given data into a symbol with as few columns as possible. If the number of columns is non-zero and the number of rows is set to 0, the printer will automatically...
set the number of rows to the lowest number required to encode the given data.

**Limitations**
Enhanced applications such as Extended Channel Interpretation (ECI), structured append, reader initialisation, Code 128 emulation, and macro characters are not supported.

**Examples**

This example shows how a MicroPDF417 bar code is specified using the BARTYPE and BARSET statements.

Bar width: 2 dots  
Bar height: 8 dots  
Number of rows: 26  
Number of columns: 3

```
BARTYPE "MICROPDF417"
BARSET #4,2,8,1,1,1,26,3
```

**Note:** The bar width and bar height can also be set using BARMAG and BARHEIGHT respectively.

The number of columns and rows are set using the Fingerprint statement BARSET. Parameters number 9 and 10 are the number of rows and columns respectively. Examples A and B below set the number of rows to 12 and the number of columns to 3. The type of bar code is set to MicroPDF417. Not all parameters of the BARSET command are applicable to the MicroPDF417 implementation. The parameters ignored by the implementation are set to ‘1’ in example B (large bar ratio, small bar ratio, security level, aspect height, aspect width).

Example A (Direct Protocol)
```
BARTYPE "MICROPDF417"
BARSET #9, 12
BARSET #10, 3
```

Example B (Direct Protocol)
```
BARSET "MICROPDF417",1,1,2,8,1,1,1,12,3
```

The example code below prints a small MicroPDF417 bar code containing the string “MicroPDF417.” The number of rows and columns is set by the printer based on the input string since the number of columns is set to 0.

```
10 BARSET "MICROPDF417",1,1,4,8,1,1,1,0,0
20 PRPOS 50, 50
30 PRBAR "MICROPDF417"
40 PRINTFEED
```
PDF417

Separate Instructions

<table>
<thead>
<tr>
<th>BARTYPE:</th>
<th>&quot;PDF417&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARRATIO:</td>
<td>Fixed. Values will be interpreted as BARMAG.</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>2-128. Default 2.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>No restriction. Default: 100.</td>
</tr>
<tr>
<td>BARFONT:</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

BARSET Instruction

<table>
<thead>
<tr>
<th>BARSET</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>[#&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter in the syntax</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>Bar code type &quot;PDF417&quot;</td>
</tr>
<tr>
<td>&lt;nexp_1&gt;</td>
<td>Large bar ratio No restriction. Default: 3.</td>
</tr>
<tr>
<td>&lt;nexp_2&gt;</td>
<td>Narrow bar ratio No restriction. Default: 1.</td>
</tr>
<tr>
<td>&lt;nexp_4&gt;</td>
<td>Barheight &lt; 500. Default: 100.</td>
</tr>
<tr>
<td>&lt;nexp_5&gt;</td>
<td>Security level 1-5. Default: 2.</td>
</tr>
<tr>
<td>&lt;nexp_6&gt;</td>
<td>Aspect height ratio No restriction. Default: 3.</td>
</tr>
<tr>
<td>&lt;nexp_8&gt;</td>
<td>Number of rows No restriction. Default: 0.</td>
</tr>
<tr>
<td>&lt;nexp_9&gt;</td>
<td>Number of columns No restriction. Default: 0.</td>
</tr>
<tr>
<td>&lt;nexp_10&gt;</td>
<td>Truncate flag 0 or ≠ 0. The default 0 will print a normal symbol.</td>
</tr>
</tbody>
</table>

Input Data

<table>
<thead>
<tr>
<th>No. of characters:</th>
<th>1,800 ASCII characters or 2,700 digits.</th>
<th>Depending of level of compactness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check digit:</td>
<td>Yes</td>
<td>Added automatically.</td>
</tr>
<tr>
<td>Input characters</td>
<td>ASCII 0-255 decimal</td>
<td></td>
</tr>
</tbody>
</table>
**Example**

This example shows PDF417 in GM label as per ANSI B-14, with following data:

<table>
<thead>
<tr>
<th>Data Identifier/Separator</th>
<th>Data</th>
<th>Field name</th>
</tr>
</thead>
<tbody>
<tr>
<td>{[} &gt; {RS}</td>
<td></td>
<td>Message Header</td>
</tr>
<tr>
<td>06 {GS}</td>
<td>12345678</td>
<td>Part Number</td>
</tr>
<tr>
<td>[GS]</td>
<td>160</td>
<td>Quantity</td>
</tr>
<tr>
<td>IJ</td>
<td>UN123456789B4C6D8E</td>
<td>License Plate</td>
</tr>
<tr>
<td>[GS]</td>
<td>L6-987</td>
<td>Material Handling Code</td>
</tr>
<tr>
<td>[GS]</td>
<td>54321 ZES</td>
<td>Plant/Dock Code</td>
</tr>
<tr>
<td>K</td>
<td>GM1234</td>
<td>PO Number</td>
</tr>
<tr>
<td>[GS]</td>
<td>G1155</td>
<td>Kanban Number</td>
</tr>
<tr>
<td>[GS]</td>
<td>KLT3214</td>
<td>Container Type</td>
</tr>
<tr>
<td>[GS]</td>
<td>10GT</td>
<td>Gross Weight</td>
</tr>
<tr>
<td>[RS]</td>
<td></td>
<td>Record Separator</td>
</tr>
<tr>
<td>[EOT]</td>
<td></td>
<td>Message Trailer</td>
</tr>
</tbody>
</table>

```plaintext
10  PRPOS 16,1180
20  DIR 4
30  ALIGN 9
40  BARSET "()","CHR$(30)+"06"+CHR$(29)+"P12345678"+CHR$(29)+"Q160"+CHR$(29)+"1JUN123456789A2B4C6D8E"+CHR$(29)+"20LA6-987"+CHR$(29)+"21L54321 ZES"+CHR$(29)+"GM1234"+CHR$(29)+"15KG1155"+CHR$(29)+"BKLT3214"+CHR$(29)+"7Q10GT"+CHR$(30)+CHR$(4)
50  PRINTFEED
60  PRINTFEED
RUN
```
Chapter 5—Bar Codes

QR Code

BARSET Instruction

<table>
<thead>
<tr>
<th>BARSET</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>[&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter in the syntax</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>Bar code type</td>
</tr>
<tr>
<td>&lt;nexp_1&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp_2&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp_3&gt;</td>
<td>Element size in dots</td>
</tr>
<tr>
<td>&lt;nexp_4&gt;</td>
<td>QR Code model</td>
</tr>
<tr>
<td>&lt;nexp_5&gt;</td>
<td>Security level</td>
</tr>
<tr>
<td>&lt;nexp_6&gt;</td>
<td>(Mask pattern selection)</td>
</tr>
<tr>
<td>&lt;nexp_7&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp_8&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp_9&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp_10&gt;</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Security level</th>
<th>Numeric</th>
<th>Alphanumeric</th>
<th>8-bit byte data</th>
<th>Kanji</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>1176</td>
<td>707</td>
<td>486</td>
<td>299</td>
</tr>
<tr>
<td>M</td>
<td>877</td>
<td>531</td>
<td>365</td>
<td>225</td>
</tr>
<tr>
<td>Q</td>
<td>738</td>
<td>447</td>
<td>307</td>
<td>189</td>
</tr>
<tr>
<td>H</td>
<td>498</td>
<td>302</td>
<td>207</td>
<td>127</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Security level</th>
<th>Numeric</th>
<th>Alphanumeric</th>
<th>8-bit byte data</th>
<th>Kanji</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>7089</td>
<td>4296</td>
<td>2953</td>
<td>1817</td>
</tr>
<tr>
<td>M</td>
<td>5596</td>
<td>3391</td>
<td>2331</td>
<td>1435</td>
</tr>
<tr>
<td>Q</td>
<td>3993</td>
<td>2420</td>
<td>1663</td>
<td>1024</td>
</tr>
<tr>
<td>H</td>
<td>3057</td>
<td>1852</td>
<td>1273</td>
<td>784</td>
</tr>
</tbody>
</table>

Remarks

Input data Capacity (Model 1)

Input data Capacity (Model 2)

Error Correction Levels

<table>
<thead>
<tr>
<th>Security level</th>
<th>Numeric</th>
<th>Alphanumeric</th>
<th>8-bit byte data</th>
<th>Kanji</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>7089</td>
<td>4296</td>
<td>2953</td>
<td>1817</td>
</tr>
<tr>
<td>M</td>
<td>5596</td>
<td>3391</td>
<td>2331</td>
<td>1435</td>
</tr>
<tr>
<td>Q</td>
<td>3993</td>
<td>2420</td>
<td>1663</td>
<td>1024</td>
</tr>
<tr>
<td>H</td>
<td>3057</td>
<td>1852</td>
<td>1273</td>
<td>784</td>
</tr>
</tbody>
</table>

The unit is number of characters. Mixed mode is supported for all combinations except for combinations containing both 8-bit byte and Kanji data. The type of data is set automatically by the implementation based on the input characters.

Error Correction Levels

<table>
<thead>
<tr>
<th>Security level</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>7%</td>
</tr>
<tr>
<td>M</td>
<td>15%</td>
</tr>
<tr>
<td>Q</td>
<td>25%</td>
</tr>
<tr>
<td>H</td>
<td>30%</td>
</tr>
</tbody>
</table>
**Chapter 5—Bar Codes**

**Example**

This example shows how BARSET is used in two different ways to create a QR Code Model 2 with element size 4 and security code M:

```plaintext
BARSET "QRCODE",1,1,4,2,2
or
BARSET #4,"QRCODE",4,201,2
```
## RSS-14

### Separate Instructions

<table>
<thead>
<tr>
<th>BARTYPE:</th>
<th>&quot;RSS14&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARRATIO:</td>
<td>Not applicable</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default: 2.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>Specifies the height of the linear bar code. Default: 100.</td>
</tr>
<tr>
<td>BARFONT ON/OFF:</td>
<td>No restriction.</td>
</tr>
</tbody>
</table>

### BARSET Instruction

<table>
<thead>
<tr>
<th>BARSET</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>[#&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter in the syntax</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>Bar code type &quot;RSS14&quot;</td>
</tr>
<tr>
<td>&lt;nexp,&gt;</td>
<td>Placeholders Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp,&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp,&gt;</td>
<td>Barmag/Enlargement See BARMAG.</td>
</tr>
<tr>
<td>&lt;nexp,&gt;</td>
<td>Barheight See BARHEIGHT.</td>
</tr>
<tr>
<td>&lt;nexp,&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp,&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp,&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp,&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp,&gt;</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

### Input Data

<table>
<thead>
<tr>
<th>No. of characters:</th>
<th>13 digits If less than 13 digits are entered, leading zeros will automatically be added so the string will be 13 digits long.</th>
</tr>
</thead>
</table>

1. There are restrictions in the standard for the minimum size for each RSS bar code, even if it is possible to print an RSS bar code in any height. The height should relate to the magnification. RSS Stacked differs, because the bar code rows do not have the same height. BARHEIGHT or BARSET<nexp,> specifies the height of the lower row and height of the upper row is automatically calculated from the height of the lower row.

For RSS-14, the width is 96X and the minimum height is 33X, where X is the width of the most narrow element as specified by BARMAG or BARSET<nexp,>. 
Chapter 5—Bar Codes

RSS-14 Truncated

Separate Instructions

| BARTYPE:       | "RSS14T"              |
| BARRATIO:      | Not applicable        |
| BARMAG:        | Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default: 2. |
| BARHEIGHT:     | Specifies the height of the linear bar code\(^1\). Default: 100. |
| BARFONT ON/    | No restriction        |
| OFF:           |                       |

BARSET Instruction

<table>
<thead>
<tr>
<th>BARSET</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>[#&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter in the syntax</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>Bar code type &quot;RSS14T&quot;</td>
</tr>
<tr>
<td>&lt;nexp_1&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp_2&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp_3&gt;</td>
<td>Barmag/Enlargement See BARMAG.</td>
</tr>
<tr>
<td>&lt;nexp_4&gt;</td>
<td>Barheight See BARHEIGHT.</td>
</tr>
<tr>
<td>&lt;nexp_5&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp_6&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp_7&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp_8&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp_9&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp_10&gt;</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

Input Data

| No. of characters: | 13 digits | If less than 13 digits are entered, leading zeros will automatically be added so the string will be 13 digits long. |

\(^1\) There are restrictions in the standard for the minimum size for each RSS bar code, even if it is possible to print an RSS bar code in any height. The height should relate to the magnification. RSS Stacked differs, because the bar code rows do not have the same height. BARHEIGHT or BARSET<sexp> specifies the height of the lower row and height of the upper row is automatically calculated from the height of the lower row.

For RSS-14 Truncated, the width is 96X and the minimum height is 13X, where X is the width of the most narrow element as specified by BARMAG or BARSET<sexp>.
## RSS-14 Stacked

### Separate Instructions

<table>
<thead>
<tr>
<th>BARTYPE:</th>
<th>&quot;RSS14S&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARRATIO:</td>
<td>Not applicable.</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default: 2.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>Specifies the height of the linear bar code(^1). Default: 100.</td>
</tr>
<tr>
<td>BARTFONT ON/OFF:</td>
<td>No restriction.</td>
</tr>
</tbody>
</table>

### BARSET Instruction

<table>
<thead>
<tr>
<th>BARSET Instruction</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>[#&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter in the syntax</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>Bar code type &quot;RSS14S&quot;</td>
</tr>
<tr>
<td>&lt;nexp,&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp,&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp,&gt;</td>
<td>Barmag/Enlargement See BARMAG.</td>
</tr>
<tr>
<td>&lt;nexp,&gt;</td>
<td>Barheight See BARHEIGHT.</td>
</tr>
<tr>
<td>&lt;nexp,&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp,&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp,&gt;</td>
<td>Height of separator row The height is the same for the pattern row between the 2D bar code and the linear bar code as for the pattern rows between the linear rows. If too low a height is entered, the height will be changed to the smallest legal height for the selected magnification. Minimum 1X, maximum 2X.</td>
</tr>
<tr>
<td>&lt;nexp,&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp,&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp,(_{16})&gt;</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

### Input Data

| No. of characters: | 13 digits | If less than 13 digits are entered, leading zeros will automatically be added so the string will be 13 digits long. |

\(^1\) / There are restrictions in the standard for the minimum size for each RSS bar code, even if it is possible to print an RSS bar code in any height. The height should relate to the magnification. RSS Stacked differs, because the bar code rows do not have the same height. BARHEIGHT or BARSET<sexp,> specifies the height of the lower row and height of the upper row is automatically calculated from the height of the lower row.

For RSS-14S, the width is 50X and the minimum height is 13X (upper 5X + lower 7X + separator 1X min.), where X is the width of the most narrow element as specified by BARMAG or BARSET<sexp,>.
Example

Example of an RSS14S bar code with the following characteristics and with recommended minimum height selected:
Place holder (nexp₁): 1
Place holder (nexp₂): 1
Most narrow element width in dots: 3
Height in dots: 21
Place holder (nexp₃): 1
Place holder (nexp₄): 1
Separator pattern row height: 4
Data: 1234567890123

BARSET "RSS14S",1,1,3,21,1,1,4
PRBAR "1234567890123"
RSS-14 Stacked Omnidirectional

Separate Instructions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARTYPE:</td>
<td>&quot;RSS14SO&quot;</td>
</tr>
<tr>
<td>BARRATIO:</td>
<td>Not applicable.</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default: 2.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>Specifies the height of the linear bar code. Default: 100.</td>
</tr>
<tr>
<td>BARFONT ON/OFF:</td>
<td>No restriction.</td>
</tr>
</tbody>
</table>

BARSET Instruction

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARSET [#&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter in the syntax</td>
</tr>
<tr>
<td>&lt;sexp&gt; Bar code type</td>
<td>&quot;RSS14SO&quot;</td>
</tr>
<tr>
<td>&lt;nexp&gt; Placeholder</td>
<td>Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp&gt; Placeholder</td>
<td>Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp&gt; Barmag/Enlargement</td>
<td>See BARMAG.</td>
</tr>
<tr>
<td>&lt;nexp&gt; Barheight</td>
<td>See BARHEIGHT.</td>
</tr>
<tr>
<td>&lt;nexp&gt; Placeholder</td>
<td>Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp&gt; Placeholder</td>
<td>Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp&gt; Height of separator row</td>
<td>The height is the same for the pattern row between the 2D bar code and the linear bar code as for the pattern rows between the linear rows. If too low a height is entered, the height will be changed to the smallest legal height for the selected magnification. Minimum 1X, maximum 2X.</td>
</tr>
<tr>
<td>&lt;nexp&gt; Not applicable</td>
<td></td>
</tr>
<tr>
<td>&lt;nexp&gt; Not applicable</td>
<td></td>
</tr>
<tr>
<td>&lt;nexp&gt; Not applicable</td>
<td></td>
</tr>
</tbody>
</table>

Input Data

<table>
<thead>
<tr>
<th>No. of characters</th>
<th>13 digits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If less than 13 digits are entered, leading zeros will automatically be added so the string will be 13 digits long.</td>
</tr>
</tbody>
</table>

1/ There are restrictions in the standard for the minimum size for each RSS bar code, even if it is possible to print an RSS bar code in any height. The height should relate to the magnification. RSS Stacked differs, because the bar code rows do not have the same height. BARHEIGHT or BARSET<sexp> specifies the height of the lower row and height of the upper row is automatically calculated from the height of the lower row.

For RSS-14SO, the width is 50X and the minimum height is 69X (upper 33X + lower 33X + separator 3X1X min.).
## RSS-14 Limited

### Separate Instructions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARTYPE:</td>
<td>&quot;RSS14L&quot;</td>
</tr>
<tr>
<td>BARRATIO:</td>
<td>Not applicable.</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default: 2.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>Specifies the height of the linear bar code(^1). Default: 100.</td>
</tr>
<tr>
<td>BARFONT ON/OFF:</td>
<td>No restriction.</td>
</tr>
</tbody>
</table>

### BARSET Instruction

<table>
<thead>
<tr>
<th>BARSET</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>[#&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter in the syntax</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>Bar code type &quot;RSS14L&quot;</td>
</tr>
<tr>
<td>&lt;nexp(_1)&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp(_2)&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp(_3)&gt;</td>
<td>Barmag/Enlargement See BARMAG.</td>
</tr>
<tr>
<td>&lt;nexp(_4)&gt;</td>
<td>Barheight See BARHEIGHT.</td>
</tr>
<tr>
<td>&lt;nexp(_5)&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp(_6)&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp(_7)&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp(_8)&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp(_9)&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp(_10)&gt;</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

### Input Data

<table>
<thead>
<tr>
<th>No. of characters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>13 digits</td>
<td>If less than 13 digits are entered, leading zeros will automatically be added so the string will be 13 digits long.</td>
</tr>
</tbody>
</table>

\(^1\) There are restrictions in the standard for the minimum size for each RSS bar code, even if it is possible to print an RSS bar code in any height. The height should relate to the magnification. RSS Stacked differs, because the bar code rows do not have the same height. BARHEIGHT or BARSET\(<nexp\(_4\)>\) specifies the height of the lower row and height of the upper row is automatically calculated from the height of the lower row.

For RSS-14L, the width is 71X and the minimum height is 10X, where X is the width of the most narrow element as specified by BARMAG or BARSET\(<nexp\(_3\)>\).
**RSS-14 Expanded**

**Separate Instructions**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARTYPE:</td>
<td>&quot;RSS14E&quot;</td>
</tr>
<tr>
<td>BARRATIO:</td>
<td>Not applicable.</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default: 2.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>Specifies the height of the linear bar code. Default: 100.</td>
</tr>
<tr>
<td>BARFONT ON/ OFF:</td>
<td>No restriction.</td>
</tr>
</tbody>
</table>

**BARSET Instruction**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARSET[#&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter in the syntax</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>Bar code type &quot;RSS14E&quot;</td>
</tr>
<tr>
<td>&lt;nexp&gt;</td>
<td>Place holder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp&gt;</td>
<td>Barmag/Enlargement See BARMAG.</td>
</tr>
<tr>
<td>&lt;nexp&gt;</td>
<td>Barheight See BARHEIGHT.</td>
</tr>
<tr>
<td>&lt;nexp&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>&lt;nexp&gt;</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

**Input Data**

<table>
<thead>
<tr>
<th>No. of characters:</th>
<th>Max. 71 numeric or 41 alphanumeric characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allowed characters:</td>
<td>0-9 A-Z a-z ! &quot; % &amp; ' ( ) * + , - . / ; &lt; = &gt; ? _ space FNC1 [CHR$(128)]</td>
</tr>
</tbody>
</table>

1/ There are restrictions in the standard for the minimum size for each RSS bar code, even if it is possible to print an RSS bar code in any height. The height should relate to the magnification. RSS Stacked differs, because the bar code rows do not have the same height. BARHEIGHT or BARSET<nexp> specifies the height of the lower row and height of the upper row is automatically calculated from the height of the lower row.

For RSS-14E, the width is depending on input and the minimum height is 33X, where X is the width of the most narrow element as specified by BARMAG or BARSET<nexp>.

**Remarks**

The RSS-14 Expanded can be used for intelligent encoding of the input data. They can be created with different encoding methods and compressed data fields. To understand how to create intelligent bar codes with RSS-14E, see Chapter 7 “Symbol Requirements for RSS Expanded” in the AIM specification: *International Symbology Specification Reduced Space Symbology (RSS), AIM Inc. ITS/99-0012, Version 1.0 1999-10-29.*
Chapter 5—Bar Codes

RSS-14 Expanded Stacked

Separate Instructions

<table>
<thead>
<tr>
<th>BARTYPE:</th>
<th>&quot;RSS14ES&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARRATIO:</td>
<td>Not applicable.</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default: 2.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>Specifies the height of the linear bar code(^1). Default: 100.</td>
</tr>
<tr>
<td>BARFONT ON/OFF:</td>
<td>No restriction.</td>
</tr>
</tbody>
</table>

BARSET Instruction

<table>
<thead>
<tr>
<th>BARSET</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>[%&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter in the syntax</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>Bar code type &quot;RSS14ES&quot;</td>
</tr>
<tr>
<td>&lt;nexp&gt; Placeholder</td>
<td>Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp&gt; Placeholder</td>
<td>Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp&gt; Barmag/Enlargement</td>
<td>See BARMAG.</td>
</tr>
<tr>
<td>&lt;nexp&gt; Barheight</td>
<td>See BARHEIGHT.</td>
</tr>
<tr>
<td>&lt;nexp&gt; Segments per row</td>
<td>2-22 (default 2). Multiples of 2 only.</td>
</tr>
<tr>
<td>&lt;nexp&gt; Placeholder</td>
<td>Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp&gt; Height of separator row</td>
<td>The height for the pattern rows between the linear rows. If too low a height is entered, the height will be changed to the smallest legal height for the selected magnification. Minimum 1X, maximum 2X.</td>
</tr>
<tr>
<td>&lt;nexp&gt; Not applicable</td>
<td></td>
</tr>
<tr>
<td>&lt;nexp&gt; Not applicable</td>
<td></td>
</tr>
<tr>
<td>&lt;nexp&gt; Not applicable</td>
<td></td>
</tr>
</tbody>
</table>

Input Data

| No. of characters: | Max. 71 numeric or 41 alphanumeric characters | Allowed characters: 0-9 A-Z a-z ! " % & ' ( ) * + , . / : ; < = > ? _ space FNC1 [CHR$(128)] |

\(^1\). There are restrictions in the standard for the minimum size for each RSS bar code, even if it is possible to print an RSS bar code in any height. The height should relate to the magnification. RSS Stacked differs, because the bar code rows do not have the same height. BARHEIGHT or BARSET<nexp> specifies the height of the lower row and height of the upper row is automatically calculated from the height of the lower row.

For RSS-14ES, the width is depending on input and the minimum height is 34X per row + 3*X1X per separator, where X is the width of the most narrow element as specified by BARMAG or BARSET<nexp>.

---

372  Intermec Fingerprint v8.70.0/v10.0.0 Programmer’s Reference Manual
Remarks

The RSS-14 Expanded Stacked can be used for intelligent encoding of the input data. They can be created with different encoding methods and compressed data fields. To understand how to create intelligent bar codes with RSS-14 ES, see Chapter 7 “Symbol Requirements for RSS Expanded” in the AIM specification: *International Symbology Specification Reduced Space Symbology (RSS), AIM Inc. ITS/99-0012, Version 1.0 1999-10-29*.

Example

Example of RSS14 Expanded Stacked for a variable weight item (0,001 kilogram increments) and with recommended minimum height selected:

Start parameter: #4
Most narrow element width in dots: 2
Height in dots: 68
Segments per row: 4
Place holder (nexp): 1
Separator pattern row height: 2
Data: 01900123456789003103001750

Explanation of data: 01 9001234567890 0 3103 001750

Application Identifier (AI): 01
AI 01 item ID: 9001234567890
(In this method, the first digit must be 9.)

Digit: 0
(The value is insignificant but a digit must be entered as a place holder.)

Application Identifier (AI): 3103
AI 3103 variable weight element string: 001750

BARTYPE "RSS14ES"
BARSET #4,2,68,4,1,2
PRBAR "01900123456789003103001750"
Chapter 5—Bar Codes

UPC-A

Separate Instructions

<table>
<thead>
<tr>
<th>BARTYPE:</th>
<th>&quot;UPCA&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARRATIO:</td>
<td>Fixed ratio. BARRATIO statement ignored.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>No restriction. Default: 100.</td>
</tr>
<tr>
<td>BARFONT:</td>
<td>Barfont generated automatically. BARFONT statement ignored. BARFONT ON/OFF work.</td>
</tr>
</tbody>
</table>

Input Data

<table>
<thead>
<tr>
<th>No. of characters:</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check digit:</td>
<td>One Added automatically</td>
</tr>
<tr>
<td>Digits:</td>
<td>0-9</td>
</tr>
<tr>
<td>Uppercase letters:</td>
<td>No</td>
</tr>
<tr>
<td>Lowercase letters:</td>
<td>No</td>
</tr>
<tr>
<td>Punctuation marks:</td>
<td>No</td>
</tr>
<tr>
<td>Start character:</td>
<td>No</td>
</tr>
<tr>
<td>Stop character:</td>
<td>No</td>
</tr>
</tbody>
</table>
UPC-E

Separate Instructions

| BARTYPE: | "UPCE" |
| BARRATIO: | Fixed ratio. BARRATIO statement ignored. |
| BARHEIGHT: | No restriction. Default: 100. |
| BARFONT: | Barfont generated automatically. BARFONT statement ignored. BARFONT ON/OFF work. |

Input Data

| No. of characters: | 6 |
| Check digit: | One Added automatically |
| Digits: | 0-9 |
| Uppercase letters: | No |
| Lowercase letters: | No |
| Punctuation marks: | No |
| Start character: | No |
| Stop character: | No |
AddOn Codes

Summary

The EAN and UPC bar code standards allow for the use of two- and five-character supplemental bar codes to be printed together with the normal code.

From Fingerprint v8.40 it is possible to print these bar codes simply by adding the desired characters to the PRBAR statement, separated by a period (.). The placement of the addon bar code is done automatically.

Examples

The AddOn characters are added to the PRBAR statement by use of a period (.) character as separator.

```
10  PRPOS 100,100
20  BARSET "EAN8"
30  ALIGN 1
40  DIR 1
50  BARFONT ON
60  PRBAR "1234567.12345"
70  PRINTFEED
RUN
```
yields:

![Barcode image]

The AddOn code can be used in combination with any EAN and UPC bar code set. The following example prints a composite bar code, with a two-character AddOn code. More information on composite bar codes can be found later in this chapter.

```
10  PRPOS 100,100
20  BARSET "EAN8_CC"
30  ALIGN 1
40  DIR 1
50  BARFONT ON
60  PRBAR "1234567.12|987654321"
70  PRINTFEED
RUN
```
yields:

![Barcode image]
Composite Bar Codes

Introduction

The EAN.UCC Composite Symbology standard has been published by AIM in the public domain.

An EAN.UCC Composite symbol consists of a linear component associated with an adjacent 2D Composite Component.

The Composite symbol always includes a linear component so that the primary identification is readable by all scanning technologies, and so that 2D imagers can use the linear component as a finder pattern for the adjacent 2D Composite Component.

Refer to the programming example later in this chapter.

Function

EAN.UCC Composite Symbology is depending on several other symbologies. Some minor adjustment of implementation of PDF417 and MicroPDF417, nine new symbologies and a logical way of combining nine linear and three 2D symbologies in different composite combinations have been implemented.

Basic Characteristics

An EAN-UCC Composite symbol consists of a linear component and associated with an adjacent 2D Composite Component.

The Linear Components

The linear bar code component encodes the item’s primary identification. The design is one of the following barcode types:

- UCC/EAN-128,
- EAN/UPC: 8 or 13,
- UPC-A or E,
- EAN/UPC 8, 13, UPC-A or E
- Any RSS-family symbology that includes a separator character between the data for the linear component and the 2D component in the input string.
The Composite Components
The 2D composite barcode component encodes supplementary data, such as a batch number or expiration date. There are three types:

- **CC-A**, a variant of MicroPDF417, designed for efficient encoding of supplemental application identifier data.

- **CC-B**, a MicroPDF417 symbol with codeword 920 in the first data codeword position as a linkage flag denoting EAN.UCC Composite Symbology data compaction.

- **CC-C**, a PDF417 symbol with a codeword 920 in the first data codeword position as a linkage flag denoting EAN.UCC Composite Symbology data compaction.

The 2D Composite Component, abbreviated as CC, is chosen based on the selected linear component and on the amount of complementary data to be encoded. The choice of linear symbol determines the name of the Composite symbol, such as EAN-13 Composite symbol or UCC/EAN-128 Composite symbol.

Human Readable
All combinations of the EAN.UCC Composite Symbology will handle the human readable in the same way. The human readable part will be an optional presentation of the information in the bar code of either data or both data. The linear human readable part will be presented below the linear component, within the same rules as for the single bar code. The human readable for the 2D composite component part will be presented under the human readable for the linear bar code.

EAN/UPC bar codes have fixed fonts for the human readable bar code interpretation.

Supported Component Combinations
Based upon the width of the linear component, a choice of “best fit” 2D Composite Component is specified. The table below lists all of the permissible combinations.

<table>
<thead>
<tr>
<th>Linear Component</th>
<th>CC-A/CC-B</th>
<th>CC-C</th>
<th>No of columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPC-A &amp; EAN-13</td>
<td>Yes/Yes</td>
<td>No</td>
<td>4</td>
</tr>
<tr>
<td>EAN-8</td>
<td>Yes/Yes</td>
<td>No</td>
<td>3</td>
</tr>
<tr>
<td>UPC-E</td>
<td>Yes/Yes</td>
<td>No</td>
<td>2</td>
</tr>
<tr>
<td>UCC/EAN-128</td>
<td>Yes/Yes</td>
<td>Yes</td>
<td>4/variable width (1-30)</td>
</tr>
<tr>
<td>RSS-14</td>
<td>Yes/Yes</td>
<td>No</td>
<td>4</td>
</tr>
<tr>
<td>RSS-14 Stacked</td>
<td>Yes/Yes</td>
<td>No</td>
<td>2</td>
</tr>
<tr>
<td>RSS-14 Stacked omnidirectional</td>
<td>Yes/Yes</td>
<td>No</td>
<td>2</td>
</tr>
<tr>
<td>RSS Limited</td>
<td>Yes/Yes</td>
<td>No</td>
<td>3</td>
</tr>
<tr>
<td>RSS Expanded</td>
<td>Yes/Yes</td>
<td>No</td>
<td>4</td>
</tr>
<tr>
<td>RSS Expanded Stacked</td>
<td>Yes/Yes</td>
<td>No</td>
<td>4</td>
</tr>
</tbody>
</table>

UPC-A Composite Symbol
There is no linkage flag in the UPC-A symbol to indicate the presence of an associated 2D Composite Component. UPC-A linear component may only be linked to four-column CC-A or CC-B components.
EAN-13 Composite Symbol
There is no linkage flag in the EAN-13 symbol to indicate the presence of an associated 2D Composite Component. EAN-13 linear component may only be linked to four-column CC-A or CC-B components.

EAN-8 Composite Symbol
There is no linkage flag in the EAN-8 symbol to indicate the presence of an associated 2D Composite Component. EAN-8 linear component may only be linked to three-column CC-A or CC-B components.

UPC-E Composite Symbol
There is no linkage flag in the UPC-E symbol to indicate the presence of an associated 2D Composite Component. UPC-E linear component may only be linked to two-column CC-A or CC-B components.

UCC/EAN-128 Composite Symbol
UCC/EAN-128 is a Code 128 with a FNC1 character in the first position after the start character. When UCC/EAN-128 is the linear component of an EAN.UCC Composite Symbol, the bar code shall have a Code Set character (see the table below) as a linkage flag in the last symbol character position before the check character. The printer will add the linkage flag automatically.

Code Set characters to be used for linkage flag

<table>
<thead>
<tr>
<th>Active Code Set</th>
<th>Link flag character for CC-A or CC-B</th>
<th>Link flag character for CC-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>CODE B</td>
<td>CODE C</td>
</tr>
<tr>
<td>B</td>
<td>CODE C</td>
<td>CODE A</td>
</tr>
<tr>
<td>C</td>
<td>CODE A</td>
<td>CODE B</td>
</tr>
</tbody>
</table>

UCC/EAN-128 Composite Symbol may be combined with any of the 2D bar code components created for the EAN.UCC Composite Symbology. The choice between using CC-A/B or CC-C is made by the user.

This table describes how an UCC/EAN-128 Composite symbol is built.

<table>
<thead>
<tr>
<th>Start char.</th>
<th>FNC1</th>
<th>Data</th>
<th>Link flag char.</th>
<th>Check digit</th>
<th>Stop char.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Added by the printer.</td>
<td>Added by the printer.</td>
<td>The data is added by the user. If it is necessary to change Code Set the printer add Code- or Shift-characters to change subset.</td>
<td>Added by the printer.</td>
<td>Calculated and added by the printer.</td>
<td>Added by the printer.</td>
</tr>
</tbody>
</table>

It is possible to choose to filter out spaces, parentheses, and Carriage Returns for the bar code data and display them in the human readable field. Carriage return makes it possible to display the human readable field in multiple rows.
Chapter 5—Bar Codes

UCC/EAN-128 with a 2D Composite component CC-A or CC-B:
The length of the data decides if a CC-A or a CC-B will be used. Only a 4
columns wide CC-A or a CC-B is used.

UCC/EAN-128 with a 2D Composite component CC-C:
- The number of columns of the CC-C is selectable by the user.
- The separator pattern is a complement of the linear symbol. The pat-
terns above the UCC/EAN-128 component’s quiet zones are light.
- The 2D bar code is placed above the separator pattern on the following
position: The first interior space module of the CC-C component is
aligned with the second module of the linear components star character.

RSS Composite symbol
When RSS symbols are used as a Composite Component, the encoded
value includes a linkage flag indicating the presence of an adjacent 2D
Composite Component.

RSS-14 Composite Symbol
RSS-14 linear component may only be linked to four-column CC-A or
CC-B components.

RSS-14 Truncated Composite Symbol
RSS-14 Truncated linear component may only be linked to four-column
CC-A or CC-B components.

RSS-14 Stacked Composite Symbol
RSS-14 Stacked linear component may only be linked to two-column CC-
A or CC-B components.

RSS-14 Stacked Omnidirectional Composite Symbol
RSS-14 Stacked Omnidirectional linear component may only be linked to
two-column CC-A or CC-B components.

RSS Limited Composite Symbol
RSS-14 Limited linear component may only be linked to three-column
CC-A or CC-B components.

RSS Expanded Composite Symbol
RSS-14 Expanded linear component may only be linked to four-column
CC-A or CC-B components.

RSS Expanded Stacked Composite Symbol
RSS-14 Expanded Stacked linear component may only be linked to four-
column CC-A or CC-B components. If linked to a 2D Composite Com-
ponent the top row of the linear component must contain at least four
symbol characters.

CC-ABC
CC-A, CC-B and CC-C are the 2D Composite Components in
EAN.UCC Composite Symbology and is almost the same bar code as the
PDF417 bar code (CC-C) and MicroPDF417 (CC-A and CC-B).

PDF417 is a multi-row, variable-length symbol offering high data capacity
and error-collection capability see [3].
MicroPDF is a multi-row symbology based on PDF417. A limited set of symbol sizes is available where each size has a fixed level of error correction.

**CC-A**
CC-A is a variant of MicroPDF417, and the smallest of the 2D Composite Components. CC-A can encode up to 56 digits and has 3 to 12 rows and 2 to 4 columns. CC-A is a variant of MicroPDF417 with a unique combination of row address patterns.

**CC-B**
CC-B is a variant of MicroPDF417 symbol uniquely identified by the codeword 920 as the first codeword in the symbol. Encoding systems select CC-B when the data to be encoded exceeds the capacity for CC-A. CC-B can encode up to 338 digits and has from 10 to 44 rows and 2 to 4 columns.

**CC-C**
CC-C bar code is the same as a PDF417 except for:
- Reserved code word: CC-C has codeword 920 in the second symbol character position (immediately following the Symbol Length Descriptor.)
- High-level data encodation: After the first data codeword 920 the second data codeword is a latch to Byte Compaction mode.
- Structure Append: Is not used in CC-C
- Reader initialization: Codeword 921 do not appear in CC-C.
- Quiet zones: No quiet zones are required above and under the bar code.
- Reference decode algorithm: An incorrect composite 2D bar code.
- Error correction: CC-C meets or exceeds the minimum error correction level recommended for PDF417.
- Symbology identifiers: Special for CC-C
- Composite Component escape mechanism: After the first Byte mode codeword 901 or 924 in the 2D Composite Component, if another codeword greater than 899 occurs, the composite “symbol” is logical terminated at that point. The remaining codewords are encoded and decoded according to specials rules.

A CC-C is only a part of an EAN.UCC Composite symbol with an UCC/EAN-128. The number of columns in the bar codes are possible to set (up to 30 columns). The number of rows are as many as necessary to represent the 2D data with the chosen number of columns.

**Example**
This example corresponds to the illustration at the start of this chapter.

BARFONT ON
BARSET "EAN128_CCC",1,1,5,100,0,0,0,0,0,1
PRBAR "(01)93812345678901|(10)ABCD123456(410)3898765432108"
PRINTFEED
EAN8 Composite with CC-A or CC-B

Separate Instructions

<table>
<thead>
<tr>
<th>BARTYPE:</th>
<th>&quot;EAN8_CC&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARRATIO:</td>
<td>Not applicable.</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default: 2. Maximum: 8.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>Specifies the height of the linear bar code. Default: 100.</td>
</tr>
<tr>
<td>BARFONT ON/</td>
<td>ON: The human readable for the linear bar will be printed under the linear</td>
</tr>
<tr>
<td>OFF:</td>
<td>bar code. If BARSET&lt;exp10&gt; is set to 1, the human readable for the 2D bar code is printed too.</td>
</tr>
<tr>
<td></td>
<td>OFF: No human readable is printed.</td>
</tr>
</tbody>
</table>

BARSET Instruction

<table>
<thead>
<tr>
<th>BARSET</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>[#{&lt;con}&gt;]</td>
<td>Optionally specifies the start parameter</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>Bar code type &quot;EAN8_CC&quot;</td>
</tr>
<tr>
<td>&lt;nexp1&gt;</td>
<td>Placeholder Value insignificant. 0 is not allowed.</td>
</tr>
<tr>
<td>&lt;nexp2&gt;</td>
<td>Placeholder Value insignificant. 0 is not allowed.</td>
</tr>
<tr>
<td>&lt;nexp3&gt;</td>
<td>Barmag/Enlargement See BARMAG.</td>
</tr>
<tr>
<td>&lt;nexp4&gt;</td>
<td>Barheight See BARHEIGHT.</td>
</tr>
<tr>
<td>&lt;nexp5&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp6&gt;</td>
<td>Character exclusion in bar code (does not affect human readable) Always 1 (default). The bar code and the human readable field will include exactly the same data.</td>
</tr>
<tr>
<td>&lt;nexp7&gt;</td>
<td>Height of separator pattern row in dots Value must be 1-2 x BARMAG. For example, if BARMAG = 5, 5-10 is OK</td>
</tr>
<tr>
<td>&lt;nexp8&gt;</td>
<td>Height of each row in 2D bar code in dots Default: 0 (= 3 x BARMAG)</td>
</tr>
<tr>
<td>&lt;nexp9&gt;</td>
<td>Separator character between data to linear bar code and 2D bar code ASCII 2-255 dec. Default: ASCII 124 dec. ASCII 48-57 dec. not allowed.</td>
</tr>
<tr>
<td>&lt;nexp10&gt;</td>
<td>2D human readable On/Off 0 = No human readable 2D field. 1 = Print human readable 2D field.</td>
</tr>
</tbody>
</table>

Input Data

<table>
<thead>
<tr>
<th>No. of characters:</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check digit:</td>
<td>1</td>
</tr>
<tr>
<td>Digits:</td>
<td>0-9</td>
</tr>
<tr>
<td>Uppercase letters:</td>
<td>No</td>
</tr>
<tr>
<td>Lowercase letters:</td>
<td>No</td>
</tr>
<tr>
<td>Punctuation marks:</td>
<td>No</td>
</tr>
<tr>
<td>Start characters:</td>
<td>No</td>
</tr>
<tr>
<td>Stop characters:</td>
<td>No</td>
</tr>
</tbody>
</table>
Chapter 5—Bar Codes

EAN13 Composite with CC-A or CC-B

Separate Instructions

<table>
<thead>
<tr>
<th>BARTYPE:</th>
<th>&quot;EAN13_CC&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARRATIO:</td>
<td>Not applicable.</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default: 2. Maximum: 8.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>Specifies the height of the linear bar code. Default: 100.</td>
</tr>
<tr>
<td>BARFONT ON/</td>
<td></td>
</tr>
<tr>
<td>OFF:</td>
<td></td>
</tr>
<tr>
<td>BARSET</td>
<td>Instruction</td>
</tr>
<tr>
<td>[###&lt;ncon&gt;]&lt;sep&gt;</td>
<td>Optionally specifies the start parameter</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>Bar code type</td>
</tr>
<tr>
<td>&lt;nexp&gt;</td>
<td>Placeholder Value insignificant. 0 is not allowed.</td>
</tr>
<tr>
<td>&lt;nexp&gt;</td>
<td>Placeholder Value insignificant. 0 is not allowed.</td>
</tr>
<tr>
<td>&lt;nexp&gt;</td>
<td>Barmag/Enlargement</td>
</tr>
<tr>
<td>&lt;nexp&gt;</td>
<td>Barheight Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp&gt;</td>
<td>Character exclusion in bar code (does not affect human readables) Always 1 (default). The bar code and the human readable field will include exactly the same data.</td>
</tr>
<tr>
<td>&lt;nexp&gt;</td>
<td>Height of separator pattern row in dots Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK</td>
</tr>
<tr>
<td>&lt;nexp&gt;</td>
<td>Height of each row in 2D bar code in dots Default: 0 (= 3 × BARMAG)</td>
</tr>
<tr>
<td>&lt;nexp&gt;</td>
<td>Separator character between data to linear bar code and 2D bar code ASCII 2-255 dec. Default: ASCII 124 dec. ASCII 48-57 dec. not allowed.</td>
</tr>
<tr>
<td>&lt;nexp&gt;</td>
<td>2D human readables On/Off 0 = No human readable 2D field. 1 = Print human readable 2D field.</td>
</tr>
</tbody>
</table>

Input Data

<table>
<thead>
<tr>
<th>No. of characters:</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check digit:</td>
<td>1</td>
</tr>
<tr>
<td>Digits:</td>
<td>0-9</td>
</tr>
<tr>
<td>Uppercase letters:</td>
<td>No</td>
</tr>
<tr>
<td>Lowercase letters:</td>
<td>No</td>
</tr>
<tr>
<td>Punctuation marks:</td>
<td>No</td>
</tr>
<tr>
<td>Start characters:</td>
<td>No</td>
</tr>
<tr>
<td>Stop characters:</td>
<td>No</td>
</tr>
</tbody>
</table>
## UPC-E Composite with CC-A or CC-B

### Separate Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARTYPE:</td>
<td>&quot;UPCE_CC&quot;</td>
</tr>
<tr>
<td>BARRATIO:</td>
<td>Not applicable.</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default: 2. Maximum: 8.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>Specifies the height of the linear bar code. Default: 100.</td>
</tr>
<tr>
<td>BARFONT ON/OFF:</td>
<td>ON: The human readable for the linear bar will be printed under the linear bar code. If BARSET&lt;exp10&gt; is set to 1, the human readable for the 2D bar code is printed too.</td>
</tr>
<tr>
<td></td>
<td>OFF: No human readable is printed.</td>
</tr>
</tbody>
</table>

### BARSET Instruction

<table>
<thead>
<tr>
<th>BARSET Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[&quot;&lt;ncon&gt;&quot;</td>
<td>Optionally specifies the start parameter</td>
</tr>
<tr>
<td>&lt;sexp&gt; Bar code type</td>
<td>&quot;UPCE_CC&quot;</td>
</tr>
<tr>
<td>&lt;nexp1&gt; Placeholder</td>
<td>Value insignificant. 0 is not allowed.</td>
</tr>
<tr>
<td>&lt;nexp2&gt; Placeholder</td>
<td>Value insignificant. 0 is not allowed.</td>
</tr>
<tr>
<td>&lt;nexp3&gt; Barmag/Enlargement</td>
<td>See BARMAG.</td>
</tr>
<tr>
<td>&lt;nexp4&gt; Barheight</td>
<td>See BARHEIGHT.</td>
</tr>
<tr>
<td>&lt;nexp5&gt; Placeholder</td>
<td>Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp6&gt; Character exclusion in bar code (does not affect human readings)</td>
<td>Always 1 (default). The bar code and the human readable field will include exactly the same data.</td>
</tr>
<tr>
<td>&lt;nexp7&gt; Height of separator pattern row in dots</td>
<td>Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK</td>
</tr>
<tr>
<td>&lt;nexp8&gt; Height of each row in 2D bar code in dots</td>
<td>Default: 0 (= 3 × BARMAG)</td>
</tr>
<tr>
<td>&lt;nexp9&gt; Separator character between data to linear bar code and 2D bar code</td>
<td>ASCII 2-255 dec. Default: ASCII 124 dec. ASCII 48-57 dec. not allowed.</td>
</tr>
<tr>
<td>&lt;nexp10&gt; 2D human readable On/Off</td>
<td>0 = No human readable 2D field. 1 = Print human readable 2D field.</td>
</tr>
</tbody>
</table>

### Input Data

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of characters:</td>
<td>6</td>
</tr>
<tr>
<td>Check digit:</td>
<td>1 Added automatically.</td>
</tr>
<tr>
<td>Digits:</td>
<td>0-9</td>
</tr>
<tr>
<td>Uppercase letters:</td>
<td>No</td>
</tr>
<tr>
<td>Lowercase letters:</td>
<td>No</td>
</tr>
<tr>
<td>Punctuation marks:</td>
<td>No</td>
</tr>
<tr>
<td>Start characters:</td>
<td>No</td>
</tr>
<tr>
<td>Stop characters:</td>
<td>No</td>
</tr>
</tbody>
</table>
Chapter 5—Bar Codes

UPC-A Composite with CC-A or CC-B

Separate Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARTYPE:</td>
<td>&quot;UPCA_CC&quot;</td>
</tr>
<tr>
<td>BARRATIO:</td>
<td>Not applicable.</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default: 2. Maximum: 8.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>Specifies the height of the linear bar code. Default: 100.</td>
</tr>
<tr>
<td>BARFONT ON/OFF:</td>
<td>ON: The human readable for the linear bar will be printed under the linear bar code. If BARSET&lt;sexp&gt; is set to 1, the human readable for the 2D bar code is printed too.</td>
</tr>
<tr>
<td></td>
<td>OFF: No human readable is printed.</td>
</tr>
</tbody>
</table>

BARSET Instruction

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARSET[#&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>Bar code type &quot;UPCA_CC&quot;</td>
</tr>
<tr>
<td>&lt;nexp1&gt;</td>
<td>Placeholder Value insignificant. 0 is not allowed.</td>
</tr>
<tr>
<td>&lt;nexp2&gt;</td>
<td>Placeholder Value insignificant. 0 is not allowed.</td>
</tr>
<tr>
<td>&lt;nexp3&gt;</td>
<td>Barmag/Enlargement See BARMAG.</td>
</tr>
<tr>
<td>&lt;nexp4&gt;</td>
<td>Barheight See BARHEIGHT.</td>
</tr>
<tr>
<td>&lt;nexp5&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp6&gt;</td>
<td>Character exclusion in bar code (does not affect human readables) Always 1 (default). The bar code and the human readable field will include exactly the same data.</td>
</tr>
<tr>
<td>&lt;nexp7&gt;</td>
<td>Height of separator pattern row in dots Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK</td>
</tr>
<tr>
<td>&lt;nexp8&gt;</td>
<td>Height of each row in 2D bar code in dots Default: 0 (= 3 × BARMAG)</td>
</tr>
<tr>
<td>&lt;nexp9&gt;</td>
<td>Separator character between data to linear bar code and 2D bar code ASCII 2-255 dec. Default: ASCII 124 dec. ASCII 48-57 dec. not allowed.</td>
</tr>
<tr>
<td>&lt;nexp10&gt;</td>
<td>2D human readables On/Off 0 = No human readable 2D field. 1 = Print human readable 2D field.</td>
</tr>
</tbody>
</table>

Input Data

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of characters</td>
<td>11</td>
</tr>
<tr>
<td>Check digit</td>
<td>1 Added automatically.</td>
</tr>
<tr>
<td>Digits</td>
<td>0-9</td>
</tr>
<tr>
<td>Uppercase letters</td>
<td>No</td>
</tr>
<tr>
<td>Lowercase letters</td>
<td>No</td>
</tr>
<tr>
<td>Punctuation marks</td>
<td>No</td>
</tr>
<tr>
<td>Start characters</td>
<td>No</td>
</tr>
<tr>
<td>Stop characters</td>
<td>No</td>
</tr>
</tbody>
</table>
### EAN.UCC 128 Composite with CC-C

#### Separate Instructions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARTYPE:</td>
<td>&quot;EAN128_CCC&quot;</td>
</tr>
<tr>
<td>BARRATIO:</td>
<td>Not applicable.</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default: 2.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>Specifies the height of the linear bar code. Default: 100.</td>
</tr>
<tr>
<td>BARFONT ON/OFF:</td>
<td>ON: The human readable for the linear bar will be printed under the linear bar code. If BARSET&lt;sexp&gt; is set to 1, the human readable for the 2D bar code is printed too. OFF: No human readable is printed.</td>
</tr>
</tbody>
</table>

#### BARSET Instruction

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARSET</td>
<td>Instruction</td>
</tr>
<tr>
<td>[#{&lt;ncon}&gt;]</td>
<td>Optionally specifies the start parameter</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>Bar code type &quot;EAN128_CCC&quot;</td>
</tr>
<tr>
<td>&lt;nexp&gt;</td>
<td>Place holder</td>
</tr>
<tr>
<td>&lt;nexp2&gt;</td>
<td>Place holder</td>
</tr>
<tr>
<td>&lt;nexp3&gt;</td>
<td>Barmag/Enlargement</td>
</tr>
<tr>
<td>&lt;nexp4&gt;</td>
<td>Barheight</td>
</tr>
<tr>
<td>&lt;nexp5&gt;</td>
<td>Number of columns/row in 2D bar code 0 (default) = select the largest number of columns for the 2D code to fit within the linear bar code including its quiet zones.</td>
</tr>
<tr>
<td>&lt;nexp6&gt;</td>
<td>Character exclusion in bar code (does not affect human readables) 0: The bar code will not include any of the characters space, (,), and carriage return [CHR$(13)] but they will be printed in the human readable field. Carriage returns make it possible to print human readable in multiple rows. 1 (default): The bar code and the human readable field will include exactly the same data.</td>
</tr>
<tr>
<td>&lt;nexp7&gt;</td>
<td>Height of separator pattern row in dots Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK</td>
</tr>
<tr>
<td>&lt;nexp8&gt;</td>
<td>Height of each row in 2D bar code in dots Default: 0 (= 3 × BARMAG)</td>
</tr>
<tr>
<td>&lt;nexp9&gt;</td>
<td>Separator character between data to linear bar code and 2D bar code ASCII 2-255 dec. Default: ASCII 124 dec. ASCII 48-57 dec. not allowed.</td>
</tr>
<tr>
<td>&lt;nexp10&gt;</td>
<td>2D human readables On/Off 0 = No human readable 2D field. 1 = Print human readable 2D field.</td>
</tr>
</tbody>
</table>
Chapter 5—Bar Codes

Input Data

<table>
<thead>
<tr>
<th>No. of characters:</th>
<th>Unlimited</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check digit:</td>
<td>Trailing symbol check character Added automatically.</td>
</tr>
<tr>
<td>Input characters:</td>
<td>ASCII 0-127 According to Roman 8 character set.</td>
</tr>
<tr>
<td>Start characters:</td>
<td>See note 2.</td>
</tr>
<tr>
<td>Code characters:</td>
<td>See note 1 &amp; 2</td>
</tr>
<tr>
<td>Shift characters:</td>
<td>See note 1 &amp; 2</td>
</tr>
<tr>
<td>Stop characters:</td>
<td>Always Added automatically.</td>
</tr>
</tbody>
</table>

Remarks

EAN 128 is identical to Code 128 with the exception that the initial FNC1 function character is generated automatically.

**Note 1:**
Code characters and shift characters require either an 8-bit communication protocol, remapping to an ASCII value between 0-127 dec., or the use of an CHR$ function.

**Note 2:**
EAN 128 has automatic selection of start character and character subset (that is, selects optimal start character and handles shift and changes of subset depending on the content of the input data), whereas EAN 128A, EAN 128B, and EAN 128C selects subset A, B, and C respectively. The last character in the bar code name signifies both the start character and the chosen subset.

The selected subset can be changed anywhere in the input string, either for a single character using a Shift character (not for Subset C), or for the remainder of the input string using a Code character (all subsets).

The Shift and Code characters consist of a combination of two characters:

- Two left-pointing double angle quotation marks («) specify a Shift character.
  
  \[
  \text{Shift character: } \texttt{««} \quad (\texttt{«} = \text{ASCII 171 dec.})
  \]

- One left-pointing double angle quotation mark («) specifies a Code character. It should be followed by an uppercase letter that specifies the subset.
  
  \[
  \text{Code character: } \texttt{« + A|B|C} \quad (\texttt{«} = \text{ASCII 171 dec.})
  \]
EAN.UCC 128 Composite with CC-A or CC-B

Separate Instructions

<table>
<thead>
<tr>
<th>BARTYPE:</th>
<th>&quot;EAN128_CCAB&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARRATIO:</td>
<td>Not applicable.</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default: 2.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>Specifies the height of the linear bar code. Default: 100.</td>
</tr>
<tr>
<td>BARFONT ON/</td>
<td>ON: The human readable for the linear bar will be printed under the linear bar code. If BARSET&lt;exp10&gt; is set to 1, the human readable for the 2D bar code is printed too.</td>
</tr>
<tr>
<td>OFF:</td>
<td>OFF: No human readable is printed.</td>
</tr>
</tbody>
</table>

BARSET Instruction

<table>
<thead>
<tr>
<th>BARSET</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>[#&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter in the syntax</td>
</tr>
<tr>
<td>&lt;exp&gt;</td>
<td>Bar code type &quot;EAN128_CCAB&quot;</td>
</tr>
<tr>
<td>&lt;exp1&gt;</td>
<td>Placeholder Value insignificant. 0 is not allowed.</td>
</tr>
<tr>
<td>&lt;exp2&gt;</td>
<td>Placeholder Value insignificant. 0 is not allowed.</td>
</tr>
<tr>
<td>&lt;exp3&gt;</td>
<td>Barmag/Enlargement See BARMAG.</td>
</tr>
<tr>
<td>&lt;exp4&gt;</td>
<td>Barheight See BARHEIGHT.</td>
</tr>
<tr>
<td>&lt;exp5&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;exp6&gt;</td>
<td>Character exclusion in bar code (does not affect human readables) 0: The bar code will not include any of the characters space, (,), and carriage return [CHR$(13)] but they will be printed in the human readable field. Carriage returns make it possible to print human readable in multiple rows. 1 (default): The bar code and the human readable field will include exactly the same data.</td>
</tr>
<tr>
<td>&lt;exp7&gt;</td>
<td>Height of separator pattern row in dots Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK</td>
</tr>
<tr>
<td>&lt;exp8&gt;</td>
<td>Height of each row in 2D bar code in dots Default: 0 (= 3 × BARMAG)</td>
</tr>
<tr>
<td>&lt;exp9&gt;</td>
<td>Separator character between data to linear bar code and 2D bar code ASCII 2-255 dec. Default: ASCII 124 dec. ASCII 48-57 dec. not allowed.</td>
</tr>
<tr>
<td>&lt;exp10&gt;</td>
<td>2D human readables On/Off 0 = No human readable 2D field. 1 = Print human readable 2D field.</td>
</tr>
</tbody>
</table>


**Input Data**

<table>
<thead>
<tr>
<th>No. of characters:</th>
<th>Unlimited</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check digit:</td>
<td>Trailing symbol check character Added automatically.</td>
</tr>
<tr>
<td>Input characters:</td>
<td>ASCII 0-127 According to Roman 8 character set.</td>
</tr>
<tr>
<td>Start characters:</td>
<td>See note 2.</td>
</tr>
<tr>
<td>Code characters:</td>
<td>See note 1 &amp; 2</td>
</tr>
<tr>
<td>Shift characters:</td>
<td>See note 1 &amp; 2</td>
</tr>
<tr>
<td>Stop characters:</td>
<td>Always Added automatically.</td>
</tr>
</tbody>
</table>

**Remarks**

EAN 128 is identical to Code 128 with the exception that the initial FNC1 function character is generated automatically.

**Note 1:**
Code characters and shift characters require either an 8-bit communication protocol, remapping to an ASCII value between 0-127 dec., or the use of an CHR$ function.

**Note 2:**
EAN 128 has automatic selection of start character and character subset (that is, selects optimal start character and handles shift and changes of subset depending on the content of the input data), whereas EAN 128A, EAN 128B, and EAN 128C selects subset A, B, and C respectively. The last character in the bar code name signifies both the start character and the chosen subset.

The selected subset can be changed anywhere in the input string, either for a single character using a Shift character (not for Subset C), or for the remainder of the input string using a Code character (all subsets).

The Shift and Code characters consist of a combination of two characters:

- Two left-pointing double angle quotation marks («) specify a Shift character.
  
  **Shift character:** «« \( (« = \text{ASCII 171 dec.}) \)

- One left-pointing double angle quotation mark («) specifies a Code character. It should be followed by an uppercase letter that specifies the subset.
  
  **Code character:** « + A|B|C \( (« = \text{ASCII 171 dec.}) \)
### RSS-14 (Composite)

#### Separate Instructions

<table>
<thead>
<tr>
<th>BARTYPE:</th>
<th>&quot;RSS14&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARRATIO:</td>
<td>Not applicable.</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default: 2.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>Specifies the height of the linear bar code. Default: 100.</td>
</tr>
<tr>
<td>BARFONT ON/OFF:</td>
<td>No restriction.</td>
</tr>
</tbody>
</table>

#### BARSET Instruction

<table>
<thead>
<tr>
<th>BARSET</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>[#&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter in the syntax</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>Bar code type &quot;RSS14&quot;</td>
</tr>
<tr>
<td>&lt;nexp_1&gt;</td>
<td>Placeholder Value insignificant. 0 is not allowed.</td>
</tr>
<tr>
<td>&lt;nexp_2&gt;</td>
<td>Placeholder Value insignificant. 0 is not allowed.</td>
</tr>
<tr>
<td>&lt;nexp_3&gt;</td>
<td>Barmag/Enlargement See BARMAG.</td>
</tr>
<tr>
<td>&lt;nexp_4&gt;</td>
<td>Barheight See BARHEIGHT.</td>
</tr>
<tr>
<td>&lt;nexp_5&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp_6&gt;</td>
<td>Character exclusion in bar code (does not affect human readables) 0: The bar code will not include any of the characters space, (,), and carriage return [CHR$(13)] but they will be printed in the human readable field. Carriage returns make it possible to print human readable in multiple rows. 1 (default): The bar code and the human readable field will include exactly the same data.</td>
</tr>
<tr>
<td>&lt;nexp_7&gt;</td>
<td>Height of separator pattern row in dots Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK</td>
</tr>
<tr>
<td>&lt;nexp_8&gt;</td>
<td>Height of each row in 2D bar code in dots Default: 0 (= 3 × BARMAG)</td>
</tr>
<tr>
<td>&lt;nexp_9&gt;</td>
<td>Separator character between data to linear bar code and 2D bar code ASCII 2-255 dec. Default: ASCII 124 dec. ASCII 48-57 dec. not allowed.</td>
</tr>
<tr>
<td>&lt;nexp_10&gt;</td>
<td>2D human readables On/Off 0 = No human readable 2D field. 1 = Print human readable 2D field.</td>
</tr>
</tbody>
</table>

#### Input Data

<table>
<thead>
<tr>
<th>No. of characters:</th>
<th>13 digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>If less than 13 digits are entered, leading zeros will automatically be added so the string will be 13 digits long.</td>
<td></td>
</tr>
</tbody>
</table>
1/. There are restrictions in the standard for the minimum size for each RSS bar code, even if it is possible to print an RSS bar code in any height. The height should relate to the magnification. RSS Stacked differs, because the bar code rows do not have the same height. BARHEIGHT or BARSET<nexp> specifies the height of the lower row and height of the upper row is automatically calculated from the height of the lower row.

For RSS-14, the width is 96X and the minimum height is 33X, where X is the width of the most narrow element as specified by BARMAG or BARSET<nexp>.
RSS-14 Truncated (Composite)

Separate Instructions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARTYPE:</td>
<td>&quot;RSS14T&quot;</td>
</tr>
<tr>
<td>BARRATIO:</td>
<td>Not applicable.</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default: 2.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>Specifies the height of the linear bar code. Default: 100.</td>
</tr>
<tr>
<td>BARFONT ON/ OFF:</td>
<td>No restriction.</td>
</tr>
</tbody>
</table>

BARSET Instruction

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARSET</td>
<td>Instruction</td>
</tr>
<tr>
<td>[#&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter in the syntax</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>Bar code type &quot;RSS14T&quot;</td>
</tr>
<tr>
<td>&lt;nexp1&gt;</td>
<td>Placeholder Value insignificant. 0 is not allowed.</td>
</tr>
<tr>
<td>&lt;nexp2&gt;</td>
<td>Placeholder Value insignificant. 0 is not allowed.</td>
</tr>
<tr>
<td>&lt;nexp3&gt;</td>
<td>Barmag/Enlargement See BARMAG.</td>
</tr>
<tr>
<td>&lt;nexp4&gt;</td>
<td>Barheight See BARHEIGHT.</td>
</tr>
<tr>
<td>&lt;nexp5&gt;</td>
<td>Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp6&gt;</td>
<td>Character exclusion in bar code (does not affect human readables) 0: The bar code will not include any of the characters space, (,), and carriage return [CHR$(13)] but they will be printed in the human readable field. Carriage returns make it possible to print human readable in multiple rows. 1 (default): The bar code and the human readable field will include exactly the same data.</td>
</tr>
<tr>
<td>&lt;nexp7&gt;</td>
<td>Height of separator pattern row in dots Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK</td>
</tr>
<tr>
<td>&lt;nexp8&gt;</td>
<td>Height of each row in 2D bar code in dots Default: 0 (= 3 × BARMAG)</td>
</tr>
<tr>
<td>&lt;nexp9&gt;</td>
<td>Separator character between data to linear bar code and 2D bar code ASCII 2-255 dec. Default: ASCII 124 dec. ASCII 48-57 dec. not allowed.</td>
</tr>
<tr>
<td>&lt;nexp10&gt;</td>
<td>2D human readables On/Off 0 = No human readable 2D field. 1 = Print human readable 2D field.</td>
</tr>
</tbody>
</table>

Input Data

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of characters:</td>
<td>13 digits If less than 13 digits are entered, leading zeros will automatically be added so the string will be 13 digits long.</td>
</tr>
</tbody>
</table>
There are restrictions in the standard for the minimum size for each RSS bar code, even if it is possible to print an RSS bar code in any height. The height should relate to the magnification. RSS Stacked differs, because the bar code rows do not have the same height. BARHEIGHT or BARSET<nexp> specifies the height of the lower row and height of the upper row is automatically calculated from the height of the lower row.

For RSS-14 Truncated, the width is 96X and the minimum height is 13X, where X is the width of the most narrow element as specified by BARMAG or BARSET<nexp>. 
Chapter 5—Bar Codes

RSS-14 Stacked (Composite)

Separate Instructions

<table>
<thead>
<tr>
<th>BARTYPE:</th>
<th>&quot;RSS14S&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARRATIO:</td>
<td>Not applicable.</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default: 2.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>Specifies the height of the linear bar code(^1). Default: 100.</td>
</tr>
<tr>
<td>BARFONT ON/OFF:</td>
<td>No restriction.</td>
</tr>
</tbody>
</table>

BARSET Instruction

<table>
<thead>
<tr>
<th>BARSET</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>[#&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter in the syntax</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>Bar code type &quot;RSS14S&quot;</td>
</tr>
<tr>
<td>&lt;nexp_1&gt;</td>
<td>Placeholder Value insignificant. 0 is not allowed.</td>
</tr>
<tr>
<td>&lt;nexp_2&gt;</td>
<td>Placeholder Value insignificant. 0 is not allowed.</td>
</tr>
<tr>
<td>&lt;nexp_4&gt;</td>
<td>Barmag/Enlargement See BARMAG.</td>
</tr>
<tr>
<td>&lt;nexp_5&gt;</td>
<td>Barheight See BARHEIGHT.</td>
</tr>
<tr>
<td>&lt;nexp_6&gt;</td>
<td>Character exclusion in bar code (does not affect human readables) 0: The bar code will not include any of the characters space, (, ), and carriage return [CHR$(13)] but they will be printed in the human readable field. Carriage returns make it possible to print human readable in multiple rows. 1 (default): The bar code and the human readable field will include exactly the same data.</td>
</tr>
<tr>
<td>&lt;nexp_7&gt;</td>
<td>Height of separator pattern row in dots Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK</td>
</tr>
<tr>
<td>&lt;nexp_8&gt;</td>
<td>Height of each row in 2D bar code in dots Default: 0 (= 3 × BARMAG)</td>
</tr>
<tr>
<td>&lt;nexp_9&gt;</td>
<td>Separator character between data to linear bar code and 2D bar code ASCII 2-255 dec. Default: ASCII 124 dec. ASCII 48-57 dec. not allowed.</td>
</tr>
<tr>
<td>&lt;nexp_10&gt;</td>
<td>2D human readables On/Off 0 = No human readable 2D field. 1 = Print human readable 2D field.</td>
</tr>
</tbody>
</table>

Input Data

| No. of characters: | 13 digits | If less than 13 digits are entered, leading zeros will automatically be added so the string will be 13 digits long. |
There are restrictions in the standard for the minimum size for each RSS bar code, even if it is possible to print an RSS bar code in any height. The height should relate to the magnification. RSS Stacked differs, because the bar code rows do not have the same height. BARHEIGHT or BARSET<nexp> specifies the height of the lower row and height of the upper row is automatically calculated from the height of the lower row.

For RSS-14S, the width is 50X and the minimum height is 13X (upper 5X + lower 7X + separator 1X min.), where X is the width of the most narrow element as specified by BARMAG or BARSET<nexp>.

**Example**

Example of an RSS14S bar code with the following characteristics and with recommended minimum height selected:

- Place holder (nexp1): 1
- Place holder (nexp2): 1
- Most narrow element width in dots: 3
- Height in dots: 21
- Place holder (nexp3): 1
- Character exclusion (nexp4): 1
- Separator pattern row height (nexp5): 4
- Height of rows in 2D bar code (nexp6): 4
- Separator character (nexp7): 124
- 2D human readables (nexp8): 1
- Data: 1234567890123|987654321

BARSET "RSS14S",1,1,3,21,1,1,4,4,124,1
PRBAR "1234567890123|987654321"
# RSS-14 Stacked Omnidirectional (Composite)

## Separate Instructions

<table>
<thead>
<tr>
<th>BARTYPE:</th>
<th>&quot;RSS14SO&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARRATIO:</td>
<td>Not applicable.</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default: 2.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>Specifies the height of the linear bar code(^1). Default: 100.</td>
</tr>
<tr>
<td>BARFONT ON/OFF:</td>
<td>No restriction.</td>
</tr>
</tbody>
</table>

## BARSET Instruction

<table>
<thead>
<tr>
<th>BARSET Instruction</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>[#&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter in the syntax</td>
</tr>
<tr>
<td>(&lt;sexp&gt;)</td>
<td>Bar code type &quot;RSS14SO&quot;</td>
</tr>
<tr>
<td>(&lt;nexp_1&gt;)</td>
<td>Placeholder Value insignificant. 0 is not allowed.</td>
</tr>
<tr>
<td>(&lt;nexp_2&gt;)</td>
<td>Placeholder Value insignificant. 0 is not allowed.</td>
</tr>
<tr>
<td>(&lt;nexp_3&gt;)</td>
<td>Barmag/Enlargement See BARMAG.</td>
</tr>
<tr>
<td>(&lt;nexp_4&gt;)</td>
<td>Barheight See BARHEIGHT.</td>
</tr>
<tr>
<td>(&lt;nexp_5&gt;)</td>
<td>Value insignificant.</td>
</tr>
<tr>
<td>(&lt;nexp_6&gt;)</td>
<td>Character exclusion in bar code (does not affect human readables) 0: The bar code will not include any of the characters <code>space</code>, <code>(</code>, <code>)</code>, and carriage return [CHR$(13)] but they will be printed in the human readable field. Carriage returns make it possible to print human readable in multiple rows. 1 (default): The bar code and the human readable field will include exactly the same data.</td>
</tr>
<tr>
<td>(&lt;nexp_7&gt;)</td>
<td>Height of separator pattern row in dots Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK</td>
</tr>
<tr>
<td>(&lt;nexp_8&gt;)</td>
<td>Height of each row in 2D bar code in dots Default: 0 (= 3 × BARMAG)</td>
</tr>
<tr>
<td>(&lt;nexp_9&gt;)</td>
<td>Separator character between data to linear bar code and 2D bar code ASCII 2-255 dec. Default: ASCII 124 dec. ASCII 48-57 dec. not allowed.</td>
</tr>
<tr>
<td>(&lt;nexp_{10}&gt;)</td>
<td>2D human readables On/Off 0 = No human readable 2D field. 1 = Print human readable 2D field.</td>
</tr>
</tbody>
</table>

## Input Data

<table>
<thead>
<tr>
<th>No. of characters:</th>
<th>13 digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>If less than 13 digits are entered, leading zeros will automatically be added so the string will be 13 digits long.</td>
<td></td>
</tr>
</tbody>
</table>
There are restrictions in the standard for the minimum size for each RSS bar code, even if it is possible to print an RSS bar code in any height. The height should relate to the magnification. RSS Stacked differs, because the bar code rows do not have the same height. BARHEIGHT or BARSET<nexp> specifies the height of the lower row and height of the upper row is automatically calculated from the height of the lower row.

For RSS-14SO, the width is 50X and the minimum height is 69X (upper 33X + lower 33X + separator 3*1X min.), where X is the width of the most narrow element as specified by BARMAG or BARSET<nexp>. 
Chapter 5—Bar Codes

RSS-14 Limited (Composite)

Separate Instructions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARTYPE:</td>
<td>&quot;RSS14L&quot;</td>
</tr>
<tr>
<td>BARRATIO:</td>
<td>Not applicable.</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default: 2.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>Specifies the height of the linear bar code.</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default: 2.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>Specifies the height of the linear bar code.</td>
</tr>
<tr>
<td>BARFONT ON/</td>
<td>No restriction.</td>
</tr>
<tr>
<td>OFF:</td>
<td></td>
</tr>
</tbody>
</table>

BARSET Instruction

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARSET</td>
<td>Instruction</td>
</tr>
<tr>
<td>[#&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter in the syntax</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>Bar code type</td>
</tr>
<tr>
<td>&lt;nexp_1&gt;</td>
<td>Placeholder</td>
</tr>
<tr>
<td>&lt;nexp_2&gt;</td>
<td>Value insignificant. 0 is not allowed.</td>
</tr>
<tr>
<td>&lt;nexp_3&gt;</td>
<td>Barmag/Enlargement</td>
</tr>
<tr>
<td>&lt;nexp_4&gt;</td>
<td>See BARMAG.</td>
</tr>
<tr>
<td>&lt;nexp_5&gt;</td>
<td>Barheight</td>
</tr>
<tr>
<td>&lt;nexp_6&gt;</td>
<td>See BARHEIGHT.</td>
</tr>
<tr>
<td>&lt;nexp_7&gt;</td>
<td>Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp_8&gt;</td>
<td>Character exclusion in bar code (does not affect human readables)</td>
</tr>
<tr>
<td>&lt;nexp_9&gt;</td>
<td>Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK</td>
</tr>
<tr>
<td>&lt;nexp_10&gt;</td>
<td>Height of each row in 2D bar code in dots</td>
</tr>
<tr>
<td>&lt;nexp_11&gt;</td>
<td>Separator character between data to linear bar code and 2D bar code</td>
</tr>
<tr>
<td>&lt;nexp_12&gt;</td>
<td>2D human readables On/Off</td>
</tr>
</tbody>
</table>

Input Data

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of characters:</td>
<td>13 digits  If less than 13 digits are entered, leading zeros will automatically be added so the string will be 13 digits long.</td>
</tr>
</tbody>
</table>
There are restrictions in the standard for the minimum size for each RSS bar code, even if it is possible to print an RSS bar code in any height. The height should relate to the magnification. RSS Stacked differs, because the bar code rows do not have the same height. BARHEIGHT or BARSET<nexp> specifies the height of the lower row and height of the upper row is automatically calculated from the height of the lower row.

For RSS-14L, the width is 71X and the minimum height is 10X, where X is the width of the most narrow element as specified by BARMAG or BARSET<nexp>. 


### RSS-14 Expanded (Composite)

#### Separate Instructions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARTYPE:</td>
<td>&quot;RSS14E&quot;</td>
</tr>
<tr>
<td>BARRATIO:</td>
<td>Not applicable.</td>
</tr>
<tr>
<td>BARMAG:</td>
<td>Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default: 2.</td>
</tr>
<tr>
<td>BARHEIGHT:</td>
<td>Specifies the height of the linear bar code. Default: 100.</td>
</tr>
<tr>
<td>BARFONT ON/OFF:</td>
<td>No restriction.</td>
</tr>
</tbody>
</table>

#### BARSET Instruction

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARSET</td>
<td>Instruction</td>
</tr>
<tr>
<td>[#&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter in the syntax</td>
</tr>
<tr>
<td>&lt;sexp&gt;</td>
<td>Bar code type &quot;RSS14E&quot;</td>
</tr>
<tr>
<td>&lt;nexp1&gt;</td>
<td>Placeholder Value insignificant. 0 is not allowed.</td>
</tr>
<tr>
<td>&lt;nexp2&gt;</td>
<td>Placeholder Value insignificant. 0 is not allowed.</td>
</tr>
<tr>
<td>&lt;nexp3&gt;</td>
<td>Barmag/Enlargement See BARMAG.</td>
</tr>
<tr>
<td>&lt;nexp4&gt;</td>
<td>Barheight See BARHEIGHT.</td>
</tr>
<tr>
<td>&lt;nexp5&gt;</td>
<td>Placeholder Value insignificant.</td>
</tr>
<tr>
<td>&lt;nexp6&gt;</td>
<td>Character exclusion in bar code (does not affect human readables) 0: The bar code will not include any of the characters space, (,), and carriage return [CHR$(13)] but they will be printed in the human readable field. Carriage returns make it possible to print human readable in multiple rows. 1 (default): The bar code and the human readable field will include exactly the same data.</td>
</tr>
<tr>
<td>&lt;nexp7&gt;</td>
<td>Height of separator pattern row in dots Value must be 1-2 × BARMAG. For example, if BARMAG = 5, 5-10 is OK</td>
</tr>
<tr>
<td>&lt;nexp8&gt;</td>
<td>Height of each row in 2D bar code in dots Default: 0 (= 3 × BARMAG)</td>
</tr>
<tr>
<td>&lt;nexp9&gt;</td>
<td>Separator character between data to linear bar code and 2D bar code ASCII 2-255 dec. Default: ASCII 124 dec. ASCII 48-57 dec. not allowed.</td>
</tr>
<tr>
<td>&lt;nexp10&gt;</td>
<td>2D human readables On/Off 0 = No human readable 2D field. 1 = Print human readable 2D field.</td>
</tr>
</tbody>
</table>

1/ There are restrictions in the standard for the minimum size for each RSS bar code, even if it is possible to print an RSS bar code in any height. The height should relate to the magnification. RSS Stacked differs, because the bar code rows do not have the same height. BARHEIGHT or BARSET<nexp> specifies the height of the lower row and height of the upper row is automatically calculated from the height of the lower row.

For RSS-14E, the width is depending on input and the minimum height is 33X, where X is the width of the most narrow element as specified by BARMAG or BARSET<nexp>.
The RSS-14 Expanded can be used for intelligent encoding of the input data. They can be created with different encoding methods and compressed data fields. To understand how to create intelligent bar codes with RSS-14E, see Chapter 7 “Symbol Requirements for RSS Expanded” in the AIM specification: International Symbology Specification Reduced Space Symbology (RSS), AIM Inc. ITS/99-0012, Version 1.0 1999-10-29.
### RSS-14 Expanded Stacked (Composite)

#### Separate Instructions

| BARTYPE: | "RSS14ES" |
| BARRATIO: | Not applicable. |
| BARMAG: | Specifies the x-dimension width in dots of the most narrow element for both the linear bar code and the 2D code. Default: 2. |
| BARHEIGHT: | Specifies the height of the linear bar code\. Default: 100. |
| BARFONT ON/OFF: | No restriction. |

#### BARSET Instruction

**BARSET**

<table>
<thead>
<tr>
<th>BARSET</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>[#&lt;ncon&gt;]</td>
<td>Optionally specifies the start parameter in the syntax</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;sexp&gt;</th>
<th>Bar code type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;RSS14ES&quot;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;nexp&gt;,&lt;nexp1&gt;,&lt;nexp2&gt;,&lt;nexp3&gt;,&lt;nexp4&gt;,&lt;nexp5&gt;,&lt;nexp6&gt;,&lt;nexp7&gt;,&lt;nexp8&gt;,&lt;nexp9&gt;,&lt;nexp10&gt;</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Placeholder</td>
<td>Value insignificant.</td>
<td></td>
</tr>
<tr>
<td>Placeholder</td>
<td>Value insignificant.</td>
<td></td>
</tr>
<tr>
<td>Barmag/Enlargement</td>
<td>See BARMAG.</td>
<td></td>
</tr>
<tr>
<td>Barheight</td>
<td>See BARHEIGHT.</td>
<td></td>
</tr>
<tr>
<td>Segments per row</td>
<td>2-22 (default 2). Multiples of 2 only.</td>
<td></td>
</tr>
<tr>
<td>Placeholder</td>
<td>Value insignificant.</td>
<td></td>
</tr>
<tr>
<td>Height of separator row</td>
<td>The height is the same for the pattern row between the 2D bar code and the linear bar code as for the pattern rows between the linear rows. If too low a height is entered, the height will be changed to the smallest legal height for the selected magnification. Minimum 1X, maximum 2X.</td>
<td></td>
</tr>
<tr>
<td>Height of each row in 2D bar code in dots</td>
<td>Default: 0 (= 3 × BARMAG)</td>
<td></td>
</tr>
<tr>
<td>Separator character between data to linear bar code and 2D bar code</td>
<td>ASCII 2-255 dec. Default: ASCII 124 dec. ASCII 48-57 dec. not allowed.</td>
<td></td>
</tr>
<tr>
<td>2D human readable On/Off</td>
<td>0 = No human readable 2D field. 1 = Print human readable 2D field.</td>
<td></td>
</tr>
</tbody>
</table>

#### Input Data

<table>
<thead>
<tr>
<th>No. of characters:</th>
<th>Max. 71 numeric or 41 alphanumerical characters</th>
</tr>
</thead>
</table>

\(^1\). There are restrictions in the standard for the minimum size for each RSS bar code, even if it is possible to print an RSS bar code in any height. The height should relate to the magnification. RSS Stacked differs, because the bar code rows do not have the same height. BARHEIGHT or BARSET<sexp> specifies the height of the lower row and height of the upper row is automatically calculated from the height of the lower row.

For RSS-14ES, the width is depending on input and the minimum height is 34X per row + 3*1X per separator, where X is the width of the most narrow element as specified by BARMAG or BARSET<sexp>.
Remarks

The RSS-14 Expanded Stacked can be used for intelligent encoding of the input data. They can be created with different encoding methods and compressed data fields. To understand how to create intelligent bar codes with RSS-14 ES, see Chapter 7 “Symbol Requirements for RSS Expanded” in the AIM specification: International Symbology Specification Reduced Space Symbology (RSS), AIM Inc. ITS/99-0012, Version 1.0 1999-10-29.

Example

Example of RSS14 Expanded Stacked for a variable weight item (0.001 kilogram increments) and with recommended minimum height selected:

Start parameter: #4
Most narrow element width in dots: 2
Height in dots: 68
Segments per row (nexp₅): 2
Character exclusion (nexp₆): 1
Separator pattern row height (nexp₇): 4
Height of rows in 2D bar code (nexp₈): 4
Separator character (nexp₉): 124
2D human readables (nexp₁₀): 1

Data: 01900123456789003103001750|9876543210

Explanation of data: 01 9001234567890 0 3103 001750
Application Identifier (AI): 01
AI 01 item ID: 9001234567890
(In this method, the first digit must be 9.)
Digit: 0
(The value is insignificant but a digit must be entered as a place holder.)
Application Identifier (AI): 3103
AI 3103 variable weight element string: 001750
2D data string: 9876543210

BARTYPE "RSS14ES"
BARSET #4,2,68,2,1,4,4,124,1
PRBAR "01900123456789003103001750|9876543210"
Setup Bar Codes

Intermec Fingerprint v8.xx-compatible EasyCoder printers can optionally be fitted with an EasySet bar code wand or a scanner (see the printer’s User’s Guide). By reading a special bar code containing encoded data for one or several setup parameters, the printer’s setup can easily be changed, even by a person without any knowledge of Intermec Fingerprint, the Direct Protocol, or their supporting software.

You could print such bar codes with your printer and paste them on a board in the vicinity of the printer. When, for example, the operator needs to switch to another type of media, he or she will only have to pick up the EasySet wand or scanner and read the appropriate bar code.

The only bar code that can be used for this purpose is a Code 128 containing the function character FNC3 (ASCII 130 dec). If the FNC character is missing, the printer will regard the bar code as containing ordinary ASCII input to the “wand:” device.

Please refer to the EasySet Bar Code Wand Setup manual for syntax and parameter descriptions.
This chapter lists the supported formats for RFID tags. These include tables for the input parameters needed to write tags according to the EPCGlobal tag standards. Other non-EPC formats available are listed as well.
Chapter 6—RFID Tag Formats

RFID Tags

There are many RFID media and tag manufacturers, not all necessarily making intercompatible equipment. The information in this chapter applies to supported tags and media. Other media may not function as described below, or not at all. Contact your local support organization for information on supported media.

EPCGlobal Tag Format Specifications

EPC (Electronic Product Code) tag formats have been derived from EAN.UCC standards. The purpose has been to be able to program RFID tags with the same information from EAN.UCC encoding schemes normally printed on bar codes. See later in this chapter an example where a GTIN (Global Trade Item Number) normally printed as an EAN128 bar code can be programmed to an RFID tag as an SGTIN-96.

The tables on tag formats listed below contain only the information on the input necessary to program the tags with Fingerprint firmware. More information is in fact contained by the tags, such as "header" and "partition". The header is uniquely defined by the tag format, and is therefore not included in the input. The "partition" does not need to be entered either, as it is defined by the length of other fields, for example "company prefix" and "item reference", and is automatically calculated by Fingerprint firmware. It is therefore very important to enter the correct number of digits in each field, even if they may contain leading zeroes.

All values entered for EPCGlobal formats must be integers entered as strings. The exception being USDOD formats, which do not strictly conform to EPCGlobal norms.

More information on EPCGlobal and USDOD tag formats can be obtained from EPCGlobal at www.epcglobalinc.org and from the US Department of Defense at www.dodrfid.org. The following list conforms to revision 1.27 of the EPC Tag Data Standards Version 1.1. The Gen 2 standard does not mention the use of 64-bit tags, although they are supported by Fingerprint.

**SGTIN-64**

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NUMBER OF DIGITS</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter:</td>
<td>Not applicable</td>
<td>0-7</td>
</tr>
<tr>
<td>Company Prefix Index:</td>
<td>Not applicable</td>
<td>0-16 383</td>
</tr>
<tr>
<td>Item Reference:</td>
<td>Not applicable</td>
<td>0-1 048 575</td>
</tr>
<tr>
<td>Serial Number:</td>
<td>Not applicable</td>
<td>0-33 554 431</td>
</tr>
<tr>
<td>URI Representation</td>
<td>&quot;urn:epc:tag:sgtin-64:Filter.CompPreIndex.ItemReference.SerialNumber&quot;</td>
<td></td>
</tr>
</tbody>
</table>
### SGTIN-96

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NUMBER OF DIGITS</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter:</td>
<td>Not applicable</td>
<td>0-7</td>
</tr>
<tr>
<td>Company Prefix:</td>
<td>6-12(^1)</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Item Reference:</td>
<td>7-1(^1)</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Serial Number:</td>
<td>Not applicable</td>
<td>0-274 877 906 943</td>
</tr>
</tbody>
</table>

URI Representation:  

Special Note: \(^1\)Number of digits in Company Prefix and Item Reference must total 13.

### SCCC-64

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NUMBER OF DIGITS</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter:</td>
<td>Not applicable</td>
<td>0-7</td>
</tr>
<tr>
<td>Company Prefix Index:</td>
<td>Not applicable</td>
<td>0-16 383</td>
</tr>
<tr>
<td>Serial Reference:</td>
<td>Not applicable</td>
<td>0-99 999 999 999</td>
</tr>
</tbody>
</table>

URI Representation:  
"urn:epc:tag:sscc-64:Filter.CompPreIndex.SerialReference"

### SCCC-96

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NUMBER OF DIGITS</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter:</td>
<td>Not applicable</td>
<td>0-7</td>
</tr>
<tr>
<td>Company Prefix:</td>
<td>6-12(^2)</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Serial Reference:</td>
<td>11-5(^2)</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

URI Representation:  

Special Note: \(^2\)Number of digits in Company Prefix and Serial Reference must total 17.

### SGLN-64

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NUMBER OF DIGITS</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter:</td>
<td>Not applicable</td>
<td>0-7</td>
</tr>
<tr>
<td>Company Prefix Index:</td>
<td>Not applicable</td>
<td>0-16 383</td>
</tr>
<tr>
<td>Location Reference:</td>
<td>Not applicable</td>
<td>0-999 999</td>
</tr>
<tr>
<td>Serial Number(^3):</td>
<td>Not applicable</td>
<td>0-524 287</td>
</tr>
</tbody>
</table>

URI Representation:  
"urn:epc:tag:sgln-64:Filter.CompPreIndex.LocationReference.SerialNumber"

Special Note: \(^3\)Serial Number can be left out when writing to a tag. (EAN.UCC specifications do not yet allow the use of Serial Number in SGLN tags)
## SGLN-96

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NUMBER OF DIGITS</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter:</td>
<td>Not applicable</td>
<td>0-7</td>
</tr>
<tr>
<td>Company Prefix:</td>
<td>6-12&lt;sup&gt;4&lt;/sup&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Location Reference:</td>
<td>6-0&lt;sup&gt;5&lt;/sup&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Serial Number:&lt;sup&gt;6&lt;/sup&gt;</td>
<td>Not applicable</td>
<td>0-2 199 023 255 551</td>
</tr>
</tbody>
</table>


Special Notes:
- <sup>4</sup> Number of digits in Company Prefix and Location Reference must total 12.
- <sup>5</sup> An empty Location Reference must be entered as an empty string: "".
- <sup>6</sup> Serial Number can be left out when writing to a tag. (EAN.UCC specifications do not yet allow the use of serial number in SGLN tags)

## GRAI-64

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NUMBER OF DIGITS</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter:</td>
<td>Not applicable</td>
<td>0-7</td>
</tr>
<tr>
<td>Company Prefix Index:</td>
<td>Not applicable</td>
<td>0-16 383</td>
</tr>
<tr>
<td>Asset Type:</td>
<td>Not applicable</td>
<td>0-999 999</td>
</tr>
<tr>
<td>Serial Number:</td>
<td>Not applicable</td>
<td>0-524 287</td>
</tr>
</tbody>
</table>

URI Representation: `urn:epc:tag:grai-64:Filter.CompPreIndex.AssetType.SerialNumber`

## GRAI-96

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NUMBER OF DIGITS</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter:</td>
<td>Not applicable</td>
<td>0-7</td>
</tr>
<tr>
<td>Company Prefix:</td>
<td>6-12&lt;sup&gt;7&lt;/sup&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Asset Type:</td>
<td>6-0&lt;sup&gt;7,8&lt;/sup&gt;</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Serial Number:</td>
<td>Not applicable</td>
<td>0-274 877 906 943</td>
</tr>
</tbody>
</table>


Special Note:
- <sup>7</sup> Number of digits in Company Prefix and Asset Type must total 12.
- <sup>8</sup> An empty Asset Type must be entered as an empty string: "".

## GIAI-64

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NUMBER OF DIGITS</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter:</td>
<td>Not applicable</td>
<td>0-7</td>
</tr>
<tr>
<td>Company Prefix Index:</td>
<td>Not applicable</td>
<td>0-16 383</td>
</tr>
<tr>
<td>Individual Asset Reference:</td>
<td>Not applicable</td>
<td>0-549 755 813 887</td>
</tr>
</tbody>
</table>

### GIAI-96

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NUMBER OF DIGITS</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter:</td>
<td>Not applicable</td>
<td>0-7</td>
</tr>
<tr>
<td>Company Prefix:</td>
<td>6-12</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Individual Asset Reference:</td>
<td>18-12</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>


Special Note: Number of digits in Company Prefix and asset reference must total 24.

### GID-96

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NUMBER OF DIGITS</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Manager Number:</td>
<td>Not applicable</td>
<td>0-268 435 455</td>
</tr>
<tr>
<td>Object Class:</td>
<td>Not applicable</td>
<td>0-16 777 215</td>
</tr>
<tr>
<td>Serial Number:</td>
<td>Not applicable</td>
<td>0-68 719 476 735</td>
</tr>
</tbody>
</table>

URI Representation: "urn:epc:tag:gid-96:GeneralManagerNumber.ObjectClass.SerialNumber"

### USDOD-64

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NUMBER OF DIGITS</th>
<th>RANGE/ALLOWED VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter:</td>
<td>Not applicable</td>
<td>0-3</td>
</tr>
<tr>
<td>Government Managed Identifier:</td>
<td>Exactly 5 characters</td>
<td>0-9, SPACE, and A-Z (not I or O)</td>
</tr>
<tr>
<td>Serial Number:</td>
<td>Not applicable</td>
<td>0-16 777 215</td>
</tr>
</tbody>
</table>


### USDOD-96

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NUMBER OF DIGITS</th>
<th>RANGE/ALLOWED VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter:</td>
<td>Not applicable</td>
<td>0-15</td>
</tr>
<tr>
<td>Government Managed Identifier:</td>
<td>Exactly 5 characters</td>
<td>0-9, SPACE, and A-Z (not I or O)</td>
</tr>
<tr>
<td>Serial Number:</td>
<td>Not applicable</td>
<td>0-68 719 476 735</td>
</tr>
</tbody>
</table>

Chapter 6—RFID Tag Formats

Uniform Resource Identifier (URI)

A standardized method to write to a tag is provided by the EPC standard to simplify encoding of tags. The URI (Uniform Resource Identifier) representations for EPC tag formats contain just the fields necessary to distinguish objects from each other, and are included in the tables above. In order to use the URIs, the tag format "EPC-URN" must be previously specified by the "TAGFORMAT" command. EPC is a namespace in the Uniform Resource Name (URN) encoding scheme.

Example

```
10  TAGFIELD "@EPC"
20  TAGFORMAT "EPC-URN"
30  TAGWRITE
    "urn:epc:tag:sscc-96:0.12345678.987654321"
```

Other EPC Tag Input Methods

While it is recommended to use the syntax described in the previous sections to write an EPC tag, it is possible to manually write the full length of the 64-bit or 96-bit chip in hexadecimal format. For this purpose, the tag formats "EPC-HEX64" and "EPC-HEX96" are available.

```
<table>
<thead>
<tr>
<th>TAGFORMAT</th>
<th>NUMBER OF BYTES</th>
<th>ALLOWED VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;EPC-HEX64&quot;</td>
<td>8</td>
<td>0-9 and A-F</td>
</tr>
<tr>
<td>&quot;EPC-HEX96&quot;</td>
<td>12</td>
<td>0-9 and A-F</td>
</tr>
</tbody>
</table>
```

Examples

```
10  TAGFIELD "@ID"
20  TAGFORMAT "EPC-HEX64"
30  TAGWRITE "1122334455667788"

10  TAGFIELD "@ID"
20  TAGFORMAT "EPC-HEX96"
30  TAGWRITE "11223344556677889900AABB"
```

EPC Tag Writing Example

The following example, also available from the EPCGlobal standard document, is provided to facilitate users who currently print bar codes with EAN.UCC encoding schemes and wish to include that information in RFID tags as well. We use a Serialized Global Trade Item (SGTIN) formed by the GTIN 10614141007346 and serial number 2. The GTIN is in fact formed by an indicator, company prefix and item reference.

GTIN Indicator: 1
Company Prefix: 0614141
Item reference: 00734
Check digit: 6
Serial Number: 2

This would normally be encoded as an EAN128 bar code:
Chapter 6—RFID Tag Formats

The Application Identifiers (AI) are dropped, as this information is included in the header. The check digit is dropped. The GTIN indicator is repositioned at the leftmost position of the item reference, to form a new item reference. A filter value of 3 is chosen for this example (signifying Standard Trade Item Group, though filter definitions are non-normative at this time). This gives us the necessary values to program an SGTIN-96 tag.

Filter: 3
Company Prefix : 0614141
Item reference: 100734
Serial Number: 2

We could use the following Fingerprint programs to program an SGTIN-96 tag with the above information. The first example assumes a Gen 2 tag, and the second an ISO 18000-6B chip (see Tag Memory Allocation section later):

Examples

```
10 FILTER$ = "3"
20 PREFIX$ = "0614141"
30 ITEM$ = "100734"
40 SERIAL$ = "2"
50 TAGFIELD "@EPC"
60 TAGFORMAT "SGTIN-96"
70 TAGWRITE FILTER$, PREFIX$, ITEM$, SERIAL$

10 TAGFIELD "@DATA",10,12
20 TAGFORMAT "EPC-URN"
30 TAGWRITE "urn:epc:tag:sgtin-96:3.
   0614141.100734.2"
```

Tag Memory Allocation

Different tag standards exist currently, several of which can be programmed with EPC tag data. Tag standards currently supported by Fingerprint firmware, are EPC UCode 1.19, ISO 18000-6B, EPC Class 1 version 1, and EPC Class 1 Generation 2. These tags have different memory structures, and thus may need different input parameters for the TAGFIELD and TAGFORMAT statement. The tag memory structure for these supported tags are shown in the following tables.
## UCODE EPC 1.19 Tag Memory Allocation

<table>
<thead>
<tr>
<th>Byte</th>
<th>@ID</th>
<th>@DATA</th>
<th>@ALL</th>
<th>Read/Write</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>Read only</td>
<td>EPCGlobal ID (Hex EF04)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td></td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td></td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td></td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>8</td>
<td></td>
<td>Read/Write</td>
<td>EPC Data</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>5</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>6</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>16</td>
<td></td>
<td>Read/Write</td>
<td>EPC Data</td>
</tr>
<tr>
<td>17</td>
<td>9</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>10</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>11</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>12</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>13</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>14</td>
<td>22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>15</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>8</td>
<td>24</td>
<td></td>
<td>Read/Write</td>
<td>EPC Data</td>
</tr>
<tr>
<td>25</td>
<td>9</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>10</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>31</td>
<td>47</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Remarks

When writing EPC data to the @ID segment of a Ucode EPC 1.19 tag, Fingerprint ignores input parameters (and other defaults) for the TAGFIELD command, and automatically starts writing at byte 2 of the @ID segment. When writing to the @DATA segment, valid start byte and field length parameters must be entered in the TAGFIELD instruction.

**Note:** No manual mapping of the EPC data to the UCode EPC 1.19 memory structure is necessary, as this is done automatically by Fingerprint firmware.
## UCODE HSL (ISO 18000-6B) Tag Memory Allocation

<table>
<thead>
<tr>
<th>Byte</th>
<th>@ID</th>
<th>@DATA</th>
<th>@ALL</th>
<th>Read/Write</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>Read/only</td>
<td>Unique ID</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
<td>Read/only</td>
<td>Unique ID</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td>2</td>
<td>Read/only</td>
<td>Unique ID</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td>3</td>
<td>Read/only</td>
<td>Unique ID</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
<td>4</td>
<td>Read/only</td>
<td>Unique ID</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td></td>
<td>5</td>
<td>Read/only</td>
<td>Unique ID</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td></td>
<td>6</td>
<td>Read/only</td>
<td>Unique ID</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td></td>
<td>7</td>
<td>Read/only</td>
<td>Unique ID</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>0</td>
<td>8</td>
<td>Read/only</td>
<td>Tag Manufacturer</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>1</td>
<td>9</td>
<td>Read/only</td>
<td>Tag Hardware Type</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>2</td>
<td>10</td>
<td>Read/only</td>
<td>Tag Hardware Type</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>3</td>
<td>11</td>
<td>Read/only</td>
<td>Tag Hardware Type</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>4</td>
<td>12</td>
<td>Read/only</td>
<td>Tag Hardware Type</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>5</td>
<td>13</td>
<td>Read/only</td>
<td>Tag Hardware Type</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>6</td>
<td>14</td>
<td>Read/only</td>
<td>Tag Hardware Type</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>7</td>
<td>15</td>
<td>Read/only</td>
<td>Tag Hardware Type</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>8</td>
<td>16</td>
<td>Read/only</td>
<td>Tag Hardware Type</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>9</td>
<td>17</td>
<td>Read/only</td>
<td>Tag Hardware Type</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>10</td>
<td>18</td>
<td>Read/only</td>
<td>Tag Hardware Type</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>11</td>
<td>19</td>
<td>Read/only</td>
<td>User Data</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>12</td>
<td>20</td>
<td>Read/only</td>
<td>Definition and format of User Data determined by Tag Memory Layout</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
<td>Read/only</td>
<td>User Data</td>
</tr>
<tr>
<td>223</td>
<td></td>
<td>215</td>
<td>223</td>
<td>Read/only</td>
<td>User Data</td>
</tr>
</tbody>
</table>

1 The actual byte length of the tag may vary.

**Remarks**

When writing to an ISO 18000-6B tag, EPC data must be written to the @DATA field, and valid start byte and field length parameters must be entered in the TAGFIELD instruction.

**Example**

```
10  FILTER$ = "3"
20  PREFIX$ = "0614141"
30  ITEM$ = "100734"
40  SERIAL$ = "2"
50  TAGFIELD "@DATA",10,12
60  TAGFORMAT "SGTIN-96"
70  TAGWRITE FILTER$, PREFIX$, ITEM$, SERIAL$
```
Chapter 6—RFID Tag Formats

Class 1 (EPC Class 1 Version 1) Tag Memory Allocation

<table>
<thead>
<tr>
<th>Byte</th>
<th>Row</th>
<th>Pointer value (bit)</th>
<th>@ID</th>
<th>EPC 64</th>
<th>EPC 96</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>16</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>32</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>48</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>64</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>80</td>
<td>8</td>
<td>Kill Passcode - R/W</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td>9</td>
<td>Lock bits - /-</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>96</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>7</td>
<td>112</td>
<td>12</td>
<td>Kill Passcode - R/W</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td>Lock bits - /-</td>
<td></td>
</tr>
</tbody>
</table>

Remarks

Class 1 tags are used to store EPC data. EPC data is stored in the @ID segment, bytes 0-7 for 64 bit and bytes 1-11 for 96-bit tags. The values for the parameters <nexp2> and <nexp3> for the TAGFIELD command do not need to be entered when writing EPC data, as these are assigned automatically.

The Kill passcode can be addressed as byte 12 in EPC-96 (byte 8 in EPC-64). It must however be addressed independently with a TAGFIELD, "@ID", 12, 1 command.

The Class 1 standard defines a tag CRC (cyclic redundancy check) for tag ID bits, and if incorrect, the tag will not be detected by a TAGREAD operation.

Erasing all tag information is done by the following command:

```
TAGFIELD "@ID", 0, 0
TAGFORMAT "HEX"
TAGWRITE ""
```
### Gen 2 (EPC Class 1 Generation 2) Tag Memory Allocation

<table>
<thead>
<tr>
<th>FIELD</th>
<th>@RESERVED</th>
<th>@EPC</th>
<th>@TID</th>
<th>@USER</th>
<th>Content</th>
<th>Read/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESERVED</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td>Kill Password</td>
<td>R/W</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td>Access Password</td>
<td>R/W</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>7</td>
<td>8</td>
<td></td>
<td>Optional</td>
<td>R/W</td>
</tr>
<tr>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPC</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td>CRC-16</td>
<td>R/-</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td>Protocol Control Bits</td>
<td>R/W</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>7</td>
<td>8</td>
<td></td>
<td>EPC data</td>
<td>R/W</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>10</td>
<td>11</td>
<td></td>
<td>Optional data</td>
<td>R/W</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>13</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TID</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td>0xE2</td>
<td>R/-</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td>Tag mask designer identifier + vendor-defined tag model number</td>
<td>R/-</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>Optional</td>
<td>R/-</td>
</tr>
<tr>
<td>USER</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td>Optional User Data</td>
<td>R/W</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Remarks**
The @RESERVED field contains kill and access passwords. These are not always implemented, and the tag acts as if they were zero-valued passwords.
Chapter 6—RFID Tag Formats

The @EPC segment is specifically designed to hold EPC values, and is at least 96 bits long. When writing EPC information to the @EPC segment, values for the <nexp2> and <nexp3> values in the TAGFIELD command need not be defined.

The @USER bank allow for user-specific data storage. The organization of user memory for EPCGlobal applications is vendor-defined.

Non-standard Tag Formats

Fingerprint allows users to program RFID tags with other data than EPC tag data. The available tag formats are "NUM", "HEX", and "ASCII", representing numeric, hexadecimal, and ASCII data respectively. When reading tags, the data is assumed to be in the format last specified by the TAGFORMAT command. The last example below, exemplifies how the ASCII characters written as "RFID", will be read as "52464944" if the last format is "HEX".

Examples

10   TAGFIELD "@ID",2,4
20   TAGFORMAT "NUM"
30   TAGWRITE 1234

10   TAGFIELD "@DATA",10,12
20   TAGFORMAT "HEX"
30   TAGWRITE "1123345567889900AABB"

10   TAGFIELD "@DATA",10,4
20   TAGFORMAT "ASCII"
30   TAGWRITE "RFID"
40   TAGFORMAT "HEX"
50   TAGREAD A$
60   PRINT A$
RUN

yields

52464944
This chapter lists the possible error messages that can be returned to the host when an error occurs.
## Error table

<table>
<thead>
<tr>
<th>Code</th>
<th>Message/Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>1</td>
<td>Syntax error.</td>
</tr>
<tr>
<td>2</td>
<td>Unbalanced parenthesis.</td>
</tr>
<tr>
<td>3</td>
<td>Feature not implemented.</td>
</tr>
<tr>
<td>4</td>
<td>Evaluation syntax error.</td>
</tr>
<tr>
<td>5</td>
<td>Unrecognized token.</td>
</tr>
<tr>
<td>6</td>
<td>Tokenized line too long.</td>
</tr>
<tr>
<td>7</td>
<td>Evaluation stack overflow.</td>
</tr>
<tr>
<td>8</td>
<td>Error in exactab.</td>
</tr>
<tr>
<td>9</td>
<td>Undefined token.</td>
</tr>
<tr>
<td>10</td>
<td>Non-executing token.</td>
</tr>
<tr>
<td>11</td>
<td>Evaluation stack underflow.</td>
</tr>
<tr>
<td>12</td>
<td>Type mismatch.</td>
</tr>
<tr>
<td>13</td>
<td>Line not found.</td>
</tr>
<tr>
<td>14</td>
<td>Division with zero.</td>
</tr>
<tr>
<td>15</td>
<td>Font not found.</td>
</tr>
<tr>
<td>16</td>
<td>Bar code device not found.</td>
</tr>
<tr>
<td>17</td>
<td>Bar code type not implemented.</td>
</tr>
<tr>
<td>18</td>
<td>Disk full.</td>
</tr>
<tr>
<td>19</td>
<td>Error in file name.</td>
</tr>
<tr>
<td>20</td>
<td>Input line too long.</td>
</tr>
<tr>
<td>21</td>
<td>Error stack overflow.</td>
</tr>
<tr>
<td>22</td>
<td>RESUME without error.</td>
</tr>
<tr>
<td>23</td>
<td>Image not found.</td>
</tr>
<tr>
<td>24</td>
<td>Overflow in temporary string buffer.</td>
</tr>
<tr>
<td>25</td>
<td>Wrong number of parameters.</td>
</tr>
<tr>
<td>26</td>
<td>Parameter too large.</td>
</tr>
<tr>
<td>27</td>
<td>Parameter too small.</td>
</tr>
<tr>
<td>28</td>
<td>RETURN without GOSUB</td>
</tr>
<tr>
<td>29</td>
<td>Error in startup file.</td>
</tr>
<tr>
<td>30</td>
<td>Assign to a read-only variable.</td>
</tr>
<tr>
<td>31</td>
<td>Illegal file number.</td>
</tr>
<tr>
<td>32</td>
<td>File is already open.</td>
</tr>
<tr>
<td>33</td>
<td>Too many files open.</td>
</tr>
<tr>
<td>34</td>
<td>File is not open.</td>
</tr>
<tr>
<td>35</td>
<td>Cutter device not found.</td>
</tr>
<tr>
<td>36</td>
<td>User break.</td>
</tr>
<tr>
<td>37</td>
<td>Illegal line number.</td>
</tr>
<tr>
<td>38</td>
<td>Run statement in program.</td>
</tr>
<tr>
<td>39</td>
<td>Parameter out of range.</td>
</tr>
<tr>
<td>40</td>
<td>Illegal bar code ratio.</td>
</tr>
<tr>
<td>41</td>
<td>Memory overflow.</td>
</tr>
<tr>
<td>42</td>
<td>Memory overflow.</td>
</tr>
<tr>
<td>43</td>
<td>Error stack underflow.</td>
</tr>
<tr>
<td>44</td>
<td>File is write protected.</td>
</tr>
<tr>
<td>45</td>
<td>Unknown store option.</td>
</tr>
<tr>
<td>46</td>
<td>Store already in progress.</td>
</tr>
<tr>
<td>47</td>
<td>Unknown store protocol.</td>
</tr>
<tr>
<td>48</td>
<td>No store defined.</td>
</tr>
<tr>
<td>49</td>
<td>NEXT without FOR.</td>
</tr>
<tr>
<td>50</td>
<td>Bad store record header.</td>
</tr>
<tr>
<td>51</td>
<td>Bad store address.</td>
</tr>
<tr>
<td>52</td>
<td>Bad store record.</td>
</tr>
<tr>
<td>53</td>
<td>Bad store checksum.</td>
</tr>
<tr>
<td>54</td>
<td>Bad store record end.</td>
</tr>
<tr>
<td>55</td>
<td>Remove in ROM.</td>
</tr>
<tr>
<td>56</td>
<td>Illegal communication channel.</td>
</tr>
<tr>
<td>57</td>
<td>Subscript out of range.</td>
</tr>
<tr>
<td>58</td>
<td>Field overflow.</td>
</tr>
<tr>
<td>59</td>
<td>Bad record number.</td>
</tr>
<tr>
<td>60</td>
<td>Too many strings.</td>
</tr>
<tr>
<td>61</td>
<td>Error in setup file.</td>
</tr>
<tr>
<td>62</td>
<td>File is list protected.</td>
</tr>
<tr>
<td>63</td>
<td>ENTER function.</td>
</tr>
<tr>
<td>64</td>
<td>FOR without NEXT</td>
</tr>
<tr>
<td>65</td>
<td>Evaluation overflow.</td>
</tr>
<tr>
<td>66</td>
<td>Bad optimizing type.</td>
</tr>
<tr>
<td>67</td>
<td>Error from communication channel.</td>
</tr>
<tr>
<td>68</td>
<td>Unknown execution entity.</td>
</tr>
<tr>
<td>69</td>
<td>Not allowed in immediate mode.</td>
</tr>
<tr>
<td>70</td>
<td>Line label not found.</td>
</tr>
<tr>
<td>71</td>
<td>Line label already defined.</td>
</tr>
<tr>
<td>72</td>
<td>IF without ENDIF.</td>
</tr>
<tr>
<td>73</td>
<td>ENDIF without IF.</td>
</tr>
<tr>
<td>74</td>
<td>ELSE without ENDIF.</td>
</tr>
<tr>
<td>75</td>
<td>ELSE without IF.</td>
</tr>
<tr>
<td>76</td>
<td>WHILE without WEND.</td>
</tr>
<tr>
<td>77</td>
<td>WEND without WHILE</td>
</tr>
<tr>
<td>78</td>
<td>Not allowed in execution mode.</td>
</tr>
<tr>
<td>79</td>
<td>Not allowed in a layout.</td>
</tr>
<tr>
<td>80</td>
<td>Download timeout</td>
</tr>
<tr>
<td>81</td>
<td>Exit to system</td>
</tr>
<tr>
<td>82</td>
<td>Invalid cont environment</td>
</tr>
<tr>
<td>83</td>
<td>ETX Timeout</td>
</tr>
<tr>
<td>1001</td>
<td>Not implemented.</td>
</tr>
<tr>
<td>1002</td>
<td>Memory too small.</td>
</tr>
<tr>
<td>1003</td>
<td>Field out of label.</td>
</tr>
<tr>
<td>1004</td>
<td>Wrong font to chosen direction.</td>
</tr>
<tr>
<td>Code</td>
<td>Message/Explanation</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>1005</td>
<td>Out of paper.</td>
</tr>
<tr>
<td>1006</td>
<td>No field to print.</td>
</tr>
<tr>
<td>1007</td>
<td>Lss too high.</td>
</tr>
<tr>
<td>1008</td>
<td>Lss too low.</td>
</tr>
<tr>
<td>1009</td>
<td>Invalid parameter.</td>
</tr>
<tr>
<td>1010</td>
<td>Hardware error.</td>
</tr>
<tr>
<td>1011</td>
<td>I/O error.</td>
</tr>
<tr>
<td>1012</td>
<td>Too many files opened.</td>
</tr>
<tr>
<td>1013</td>
<td>Device not found.</td>
</tr>
<tr>
<td>1014</td>
<td>File not found.</td>
</tr>
<tr>
<td>1015</td>
<td>File is read-only.</td>
</tr>
<tr>
<td>1016</td>
<td>Illegal argument.</td>
</tr>
<tr>
<td>1017</td>
<td>Result too large.</td>
</tr>
<tr>
<td>1018</td>
<td>Bad file descriptor.</td>
</tr>
<tr>
<td>1019</td>
<td>Invalid font.</td>
</tr>
<tr>
<td>1020</td>
<td>Invalid image.</td>
</tr>
<tr>
<td>1021</td>
<td>Too large argument for MAG.</td>
</tr>
<tr>
<td>1022</td>
<td>Head lifted.</td>
</tr>
<tr>
<td>1023</td>
<td>Incomplete label.</td>
</tr>
<tr>
<td>1024</td>
<td>File too large.</td>
</tr>
<tr>
<td>1025</td>
<td>File does not exist.</td>
</tr>
<tr>
<td>1026</td>
<td>Label pending.</td>
</tr>
<tr>
<td>1027</td>
<td>Out of transfer ribbon.</td>
</tr>
<tr>
<td>1028</td>
<td>Paper type is not selected.</td>
</tr>
<tr>
<td>1029</td>
<td>Printhead voltage too high.</td>
</tr>
<tr>
<td>1030</td>
<td>Character is missing in chosen font.</td>
</tr>
<tr>
<td>1031</td>
<td>Next label not found.</td>
</tr>
<tr>
<td>1032</td>
<td>File name too long.</td>
</tr>
<tr>
<td>1033</td>
<td>Too many files are open.</td>
</tr>
<tr>
<td>1034</td>
<td>Not a directory.</td>
</tr>
<tr>
<td>1035</td>
<td>File pointer is not inside the file.</td>
</tr>
<tr>
<td>1036</td>
<td>Subscript out of range.</td>
</tr>
<tr>
<td>1037</td>
<td>No acknowledge received within specified time.</td>
</tr>
<tr>
<td>1038</td>
<td>Communication checksum error.</td>
</tr>
<tr>
<td>1039</td>
<td>Not mounted.</td>
</tr>
<tr>
<td>1040</td>
<td>Unknown file operating system.</td>
</tr>
<tr>
<td>1041</td>
<td>Error in fos structure.</td>
</tr>
<tr>
<td>1042</td>
<td>Internal error in mcs.</td>
</tr>
<tr>
<td>1043</td>
<td>Timer table full.</td>
</tr>
<tr>
<td>1044</td>
<td>Low battery in memory card.</td>
</tr>
<tr>
<td>1045</td>
<td>Media was removed.</td>
</tr>
<tr>
<td>1046</td>
<td>Memory checksum error.</td>
</tr>
<tr>
<td>1047</td>
<td>Interrupted system call.</td>
</tr>
</tbody>
</table>
### Chapter 7—Error Messages

<table>
<thead>
<tr>
<th>Code</th>
<th>Message/Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1213</td>
<td>Speedo error six.</td>
</tr>
<tr>
<td>1214</td>
<td>Squeeze or clip not supported.</td>
</tr>
<tr>
<td>1215</td>
<td>Character data not available.</td>
</tr>
<tr>
<td>1216</td>
<td>Unknown font.</td>
</tr>
<tr>
<td>1217</td>
<td>Font format is not supported.</td>
</tr>
<tr>
<td>1218</td>
<td>Correct mapping table is not found.</td>
</tr>
<tr>
<td>1219</td>
<td>Font is in the wrong direction.</td>
</tr>
<tr>
<td>1220</td>
<td>Error in external map table.</td>
</tr>
<tr>
<td>1221</td>
<td>Map table was not found.</td>
</tr>
<tr>
<td>1222</td>
<td>Double byte map table is missing.</td>
</tr>
<tr>
<td>1223</td>
<td>Single byte map table is missing.</td>
</tr>
<tr>
<td>1224</td>
<td>Character map function is missing.</td>
</tr>
<tr>
<td>1225</td>
<td>Double byte font is not selected.</td>
</tr>
<tr>
<td>1301</td>
<td>Index outside collection bounds.</td>
</tr>
<tr>
<td>1302</td>
<td>Collection could not be expanded.</td>
</tr>
<tr>
<td>1303</td>
<td>Parameter is not a collection.</td>
</tr>
<tr>
<td>1304</td>
<td>Item not a member of the collection.</td>
</tr>
<tr>
<td>1305</td>
<td>No compare function, or compare returns faulty</td>
</tr>
<tr>
<td>1306</td>
<td>Tried to insert a duplicate item.</td>
</tr>
<tr>
<td>1320</td>
<td>No RFID support installed.</td>
</tr>
<tr>
<td>1321</td>
<td>No tag found.</td>
</tr>
<tr>
<td>1322</td>
<td>Access outside tag memory.</td>
</tr>
<tr>
<td>1323</td>
<td>Tag format error.</td>
</tr>
<tr>
<td>1324</td>
<td>RFID inactive.</td>
</tr>
<tr>
<td>1325</td>
<td>Not supported tag type.</td>
</tr>
<tr>
<td>1601</td>
<td>Reference Font Not Found.</td>
</tr>
<tr>
<td>1602</td>
<td>Error in Wand-Device.</td>
</tr>
<tr>
<td>1603</td>
<td>Error in Slave Processor.</td>
</tr>
<tr>
<td>1604</td>
<td>Print Shift Error.</td>
</tr>
<tr>
<td>1605</td>
<td>No Hardware Lock.</td>
</tr>
<tr>
<td>1606</td>
<td>Testfeed not done.</td>
</tr>
<tr>
<td>1607</td>
<td>General Print Error.</td>
</tr>
<tr>
<td>1608</td>
<td>Access Denied.</td>
</tr>
<tr>
<td>1609</td>
<td>Specified Feed Length Exceeded.</td>
</tr>
<tr>
<td>1610</td>
<td>Illegal Character Map File.</td>
</tr>
<tr>
<td>1701</td>
<td>Cutter not back in position after cut.</td>
</tr>
<tr>
<td>1702</td>
<td>Cutter has not reached upper position: unsuccessful cut.</td>
</tr>
<tr>
<td>1703</td>
<td>Cutter not back in position after unsuccessful cut.</td>
</tr>
<tr>
<td>1704</td>
<td>Cutter open.</td>
</tr>
<tr>
<td>1710</td>
<td>Power supply Generic Error</td>
</tr>
<tr>
<td>1711</td>
<td>Power supply Pending.</td>
</tr>
<tr>
<td>1712</td>
<td>Power supply Status OK.</td>
</tr>
<tr>
<td>1713</td>
<td>Power supply Power Fail</td>
</tr>
<tr>
<td>1714</td>
<td>Power supply Over Volt V24</td>
</tr>
<tr>
<td>1715</td>
<td>Power supply Under Volt V24</td>
</tr>
<tr>
<td>1716</td>
<td>Power supply Over Volt VSTM</td>
</tr>
<tr>
<td>1717</td>
<td>Power supply Under Volt VSTM</td>
</tr>
<tr>
<td>1718</td>
<td>Power supply Over Temperature</td>
</tr>
<tr>
<td>1719</td>
<td>Power supply Error</td>
</tr>
<tr>
<td>1820</td>
<td>No route to host</td>
</tr>
<tr>
<td>1821</td>
<td>Disc quota exceeded</td>
</tr>
<tr>
<td>1833</td>
<td>Connection refused</td>
</tr>
</tbody>
</table>
This appendix illustrates the various keyboards of the EasyCoder printer models that can use Intermec Fingerprint v8.70 and shows the id. numbers, position numbers, and ASCII values for the individual keys and key combinations. Id numbers are used with the ON KEY GOSUB command. The keyboard may be remapped using the KEYBMAP$ command. An example program to map the alphanumeric keyboard of a PX-series printer is given.
Appendix A—Keyboards

EasyCoder PF2i/4i/4ci

Actual Keyboard Appearance

ID Numbers

Unshifted keys id. numbers

Shifted keys id. numbers
EasyCoder PF2i/4i/4ci, cont.

Position Numbers

ASCII Values

Unshifted keys ASCII values

Shifted keys ASCII values
**EasyCoder PM4i**

**Actual Keyboard Appearance**

![Actual keyboard appearance](image)

**ID Numbers**

- **Unshifted keys; id. numbers**
  - 17
  - 10 11 12 18
  - Shift 13 14 16
  - 7 8 9 15
  - 4 5 6 19
  - 1 2 3
  - 21 0 20

- **Shifted keys; id. numbers**
  - 117
  - 110 111 112 118
  - Shift 113 114 116
  - 107 108 109 115
  - 104 105 106 119
  - 101 102 103
  - 121 100 120
Appendix A—Keyboards

EasyCoder PM4i, cont.

Position Numbers

ASCII Values

Unshifted keys; ASCII values

Shifted keys; ASCII values
Appendix A—Keyboards

EasyCoder PX4i/6i

Actual Keyboard Appearance

![Actual keyboard appearance](image)

ID Numbers

Unshifted keys id. numbers

Shifted keys id. numbers
EasyCoder PX4i/6i, cont.

Position Numbers

ASCII Values

Unshifted keys ASCII values

Shifted keys ASCII values
Appendix A—Keyboards

EasyCoder PX4i/6i Alphanumeric keyboard

Actual Keyboard Appearance

Position Numbers

ID Numbers and ASCII Values

The EasyCoder PX4i/6i printers do not automatically map the keys of the alphanumeric keyboard to ASCII and ID values. The alphanumeric keyboard is simply a skin, placed on top of an existing keypad. It is necessary to map the keys to desired ASCII values. The appearance of the alphanumeric keyboard shall thus only be seen as an example. ID numbers cannot be explicitly mapped, but follow the corresponding ASCII values. ID values exist only for the 22 keys (and their respective shifted keys) which are commonly available on all the keyboards shown above. The alphanumeric keyboard must be mapped after every reboot, according to the user’s requirements. The KEYBMAP$ command (see Chapter 2) is used to assign ASCII values to each corresponding position number. The example program on the next page maps the keyboard to the values shown on the actual keyboard, according to the ANSI character set (which is used by most Terminal programs, such as Windows Hyperterminal).
Mapping the Keyboard

In order to map (or remap) the keyboard of the printer, you must identify the ASCII values for each character you want available. The KEYBMAP$ command used to map the keyboard (see Chapter 2) requires a string containing all 64 keyboard characters. You must map the complete keyboard for two (three in PX printers with alphanumeric keyboard) cases: no Shift key pressed, and Shift key pressed (plus Alt key pressed for PX printers). The example below maps the PX-printers keyboard to the actual alphanumeric keyboard appearance.

**Example**

```plaintext
10 REM PX Alphanumeric keyboard mapping.
20 REM Character Set = ANSI (NASC -2)
30 REM One keyboard column per program row
40 NASC -2

90 REM Unshifted keys
100 A$ = CHR$(128)+"aq" + STRING$(2,0)
110 A$ = A$ + "zsw" + STRING$(2,0)
120 A$ = A$ + "xde" + STRING$(2,0)
130 A$ = A$ + "cfr" + STRING$(2,0)
140 A$ = A$ + "vg" + STRING$(2,0)
150 A$ = A$ + "bhy" + CHR$(6)+CHR$(7)
160 A$ = A$ + "nju" + CHR$(29) + CHR$(1)
170 A$ = A$ + "mki" + CHR$(4) + CHR$(2)
180 A$ = A$ + ",lo" + CHR$(5) + CHR$(3)
190 A$ = A$ + "+p" + CHR$(28)+CHR$(30)
200 A$ = A$ + " " + CHR$(8) + "147"
210 A$ = A$ + CHR$(31)+"0258"
220 A$ = A$ + CHR$(13) + "369"

290 REM Shifted keys
300 B$ = CHR$(0)+"AQ" + STRING$(2,0)
310 B$ = B$ + "ZSW" + STRING$(2,0)
320 B$ = B$ + "XDE" + STRING$(2,0)
330 B$ = B$ + "CFR" + STRING$(2,0)
340 B$ = B$ + "VG" + STRING$(2,0)
350 B$ = B$ + "BHY" + CHR$(0)+CHR$(0)
360 B$ = B$ + "NBU" + CHR$(157) + CHR$(129)
370 B$ = B$ + "MKI" + CHR$(132) + CHR$(130)
380 B$ = B$ + ",LO" + CHR$(133) + CHR$(131)
390 B$ = B$ + "+P" + CHR$(156)+CHR$(158)
400 B$ = B$ + " " + CHR$(136) + "147"
410 B$ = B$ + CHR$(159)+"0258"
420 B$ = B$ + CHR$(141) + "369"
```
Appendix A—Keyboards

490 REM Alt keys
500 C$ = CHR$(128) + CHR$(167) + "!" + STRING$(2,0)
510 C$ = C$ + "<" + CHR$(64) + CHR$(34) +
        STRING$(2,0)
520 C$ = C$ + ">" + CHR$(126) + CHR$(128) +
        STRING$(2,0)
530 C$ = C$ + "[" + CHR$(163) + ")" +
        STRING$(2,0)
540 C$ = C$ + "]" + STRING$(2,0)
550 C$ = C$ + CHR$(94) + "_" + CHR$(6) + CHR$(7)
560 C$ = C$ + ";" + CHR$(29) + CHR$(1)
570 C$ = C$ + ":(?" + CHR$(4) + CHR$(2)
580 C$ = C$ + CHR$(246) + ":(" + CHR$(5) + CHR$(3)
590 C$ = C$ + CHR$(228) + CHR$(229) + ":=" +
        CHR$(28) + CHR$(30)
600 C$ = C$ + CHR$(248) + CHR$(8) + CHR$(252) +
        CHR$(233) + CHR$(231)
610 C$ = C$ + CHR$(31) + CHR$(230) + CHR$(223) +
        CHR$(232) + CHR$(241)
620 C$ = C$ + CHR$(13) + CHR$(250) + CHR$(224) +
        CHR$(181)

1000 KEYBMAP$(0)=A$
1010 KEYBMAP$(1)=B$
1020 KEYBMAP$(5)=C$
1100 END

EasyCoder PD41

The EasyCoder PD41 keyboard has only one button with Id. number 17
when pressed and 117 when released. This gives the corresponding ASCII
values of 16 and 144. The keyboard map string is only six characters long
(compared to 64 for PF/PM/PX printers), with the Print button having
position 1.
This appendix lists relevant topics, instructions and keywords and specifies on which page they are located in the manual.
Index

A
abbreviation
  constant 8
  expression 8
  variable 8
ABS instruction 9
absolute value 9
access mode
  random 189
  sequential 189
Account Secret 298
ACTLEN instruction 10
AddOn Codes 376
AIM International 344
align
  text in box 196–200
ALIGN instruction 11
AN. See ALIGN instruction
anchor point 11
ANSI 344
ANSI character set 173
array 70
  sorting 249
ASCII
  conversion 14, 34, 101
  tag format 416
ASC instruction 14
assign
  line from file to variable 154
  line to string variable 153
  value to variable 152

B
BARADJUST instruction 15
bar code
  AddOn 376
  Code 128 349
  Code 39 348
  Data Matrix 351
  EAN-13 353
  EAN-8 352
  EAN 128 354
enlargement. See bar code: magnification
  field 12
height 21, 25
Interleaved 2 of 5 356
interpretation 378
  enabling 19, 20
  font 17
  insertion point 18
  magnification 18
  vertical offset 18
width 18
list 343, 344
  magnification 24
  MaxiCode 357
  one-dimensional 347
  PDF417 361
  QR Code 363
  ratio 24
  RSS-14 365
two-dimensional 347
type 24
  UPC-A 374
  UPC-E 375
BARCODENAMES instruction 16
BARFONT instruction 17
BARHEIGHT instruction 21
BARMAG instruction 22
BARRATIO instruction 23
BARSET instruction 24
BARTYPE instruction 26
batch printing 192
BEEP instruction 27
BF. See BARFONT instruction
BH. See BARMAG instruction
bitmap image
  printing 201
bluelight. See Intermec Readiness Indicator
BM. See BARMAG instruction
box
  aligning text in 196–200
  creating 196
BR. See BARRATIO instruction
break
  branching to subroutine on 178
BREAK instruction 28
BREAK ON/OFF instruction 30
BT. See BARTYPE instruction
buffer
  dividing into fields 83
BUSY instruction 31

C
CGI
  request 184
channel. See communication channel
character
  retrieving from string 150
  search for in string 133
class 1 411
classification 411
CHDIR instruction 32
checksum
  calculate 53
CHECKSUM instruction 33
CHR$ instruction 14, 34
CLEANFEED instruction 35
CLEAR instruction 36
CLIP instruction 37
CLL instruction 38
CLOSE instruction 40
Code 128 349
Code 39 348
COMBUFF$ instruction 42
COM ERROR ON/OFF instruction 41
comments
    in program 223
communication channel
    centronics 31, 41, 42, 43, 45
debugging 63
error handling 41
interrupt 28, 47
listing 67
net1 42, 43, 45
selecting 239
uart 28, 30, 31, 41, 42, 43, 45
usb 41, 42, 43, 45
CompactFlash
    formatting 96
composite
    bar code 377
COMSET instruction 43
COMSET OFF instruction 45
COMSET ON instruction 46
COMSTAT instruction 47
CONT instruction 48
COPY instruction 49
COUNT& instruction 50
counter
    creating 50
CSUM instruction 53
CURDIRS instruction 52
CUT instruction 54
CUT ON/OFF instruction 55
cutter 54

D

data
    reading from device 131
    reading from file 131
Data Matrix 351
date
    formatting 56–59, 98, 171–172, 293–295
    returning 56
    setting 56
DATES instruction 56
DATEADD$ instruction 57
DATEDIFF instruction 59
day. See date
DBBREAK instruction 60
DBBREAK OFF instruction 61
DBEND instruction 62
DBSTDIO instruction 63
DBSTEP instruction 64
debugger 60
DELETE instruction 65
DELETEPFSVAR instruction 66
deleting
    files, directories 139
device
    closing 40, 75
communication. See communication channel
listing 67
open 189
reading string from 132
DEVICES instruction 67
DIM instruction 70
directory. See file system
Direct Protocol
    disabling 130
    enabling 130
DIR instruction 71
DIRNAMES instruction 74
DP. See Direct Protocol

E

EAN 376, 406
EAN-13 353
EAN-13 Composite 383
EAN-8 352
EAN-8 Composite 382
EAN.UCC 128 Composite 386
EAN 128 354
EAN International 344
EasySet 404
Electronic Product Code. See EPC
ELSE. See IF...THEN...(ELSE) instruction
END instruction 75
English character set 173
EOF instruction 76
EPC
    tag format 406
EPC-HEX64 410
EPC-HEX96 410
EPCGlobal 405
EPC UCODE 1.19 411
ERL instruction 77
ERRS instruction 79
ERR instruction 78
error
    explanation, retrieving 79
    finding in program 77
    handling, subroutine 181
    messages, defining 80
    number, returning 78
ERROR instructions 80
EXECUTE instructions 82
extracting
    part of string 168

F

faulty dot. See SET FAULTY DOT instruction
Appendix I—Index

FF. See FORMFEED instruction
field
bar code 12
dividing 83
partial 37
retrieving current 84
FIELD instruction 83
FIELDNO instruction 84
file
closing 40, 75
copying 49
deleting 139
end-of-file, check for 76
listing 87
loading 85
names, retrieving 86
open 189
reading record from 108
reading string from 131, 132
saving 235
storing. See file: loading
transfer 282, 283, 286
transferring 282
transfer status 285
FILE& LOAD instruction
instruction 85
file length 160
FILENAME$ instruction 86
FILES instruction 87
file system
current directory, change 32
current directory, return 52
description 4
directory, creating 170
directory, deleting 139
directory names, retrieving 74
FLOATCALC$ instruction 89
float numbers
calculating with 89
font
downloading 91
files 91
files, installing 121
in bar code interpretation 17
in human readable. See font: in bar code interpretation
name 17
name, retrieving 93, 94
selecting 90
single-byte 90
size 17
skew. See font: slant
slant 17, 91
tilt. See font: slant
TrueDoc 90, 92
TrueType 90, 92
FONTD instruction 92
FONT instruction 90
FONTNAME$ instruction 93

G
Gap Truncate Mode 148
Gen 2 270, 411, 415
German character set 173
GETASSOC$ instruction 109
GETASSOCNAME$ instruction 110
GET instruction 108
GETPFSVAR instruction 111
GIAI-64 408
GIAI-96 409
GID-96 409
GOSUB instruction 112
GOTO instruction 114
GRAI-64 408
GRAI-96 408
guard bar
in EAN/UPC bar codes 20

H
hardware
testing 106–107
HEAD instruction 115
human readable. See bar code: interpretation

I
IF. See IF...THEN...(ELSE) instruction
IF...THEN...(ELSE) instruction 117
II. See INVIMAGE instruction
image
bitmap, printing 201
printing 202
removing from memory 224
storing in memory 254
image buffer
clearing 38
saving 120
IMAGE BUFFER MIRROR instruction 119
IMAGE BUFFER SAVE instruction 120
IMAGE LOAD instruction 121
IMAGENAMES$ instruction 122
Appendix I—Index

images
names, listing 123
names, retrieving 122
IMAGES instruction 123
IMMEDIATE instructions 124
immediate mode 124–126
indicator
  controlling status 149
INKEY$ instruction 127
input
  receiving 128
INPUT# instruction 131
INPUS instruction 132
INPUT instruction 128
INPUT ON/OFF instruction 130
insertion point
  checking 213
INSTR instruction 133
Instruction
  ABS 9
  ACTLEN 10
  ASC 14
  BARADJUST 15
  BARCODENAMES 16
  BARFONT 17
  BARDHEIGHT 21
  BARMAG 22
  BARRATIO 23
  BARSET 24
  BARTYPE 26
  BEEP 27
  BREAK 28
  BREAK ON/OFF 30
  BUSY 31
  CHDIR 32
  CHECKSUM 33
  CHR$ 34
  CLEANFEED 35
  CLEAR 36
  CLIP 37
  CLEAR 38
  CLOSE 40
  COMBUF$ 42
  COM ERROR ON/OFF 41
  COMSET 43
  COMSET OFF 45
  COMSET ON 46
  COMSTAT 47
  CONT 48
  COPY 49
  COUNT 50
  CSUM 53
  CURDIR$ 52
  CUT 54
  CUT ON/OFF 55
  DATES 56
  DATEADD$ 57
  DATEDIFF 59
  DBBREAK 60
  DBBREAK OFF 61
  DBEND 62
  DBSTDIO 63
  DBSTEP 64
  DELETE 65
  DELETESPVAR 66
  DEVICES 67
  DIM 70
  DIR 71
  DIRNAMES$ 74
  END 75
  EOF 76
  ERL 77
  ERR 78
  ERRS 79
  ERROR 80
  EXECUTE 82
  FIELD 83
  FIELDNO 84
  FILE& LOAD 85
  FILENAME$ 86
  FILES 87
  FLOATCALC$ 89
  FONT 90
  FONTD 92
  FONTNAME$ 93
  FONT$ 94
  FOR...TO...NEXT 95
  FORMAT 96
  FORMAT$ 101
  FORMAT DATES$ 98
  FORMAT INPUT 99
  FORMAT TIMES$ 100
  FORMFEED 104
  FRI 105
  FUNCTION 106
  FUNCTION$ 107
  GET 108
  GETASSOC$ 109
  GETASSOCNAME$ 110
  GETPFSVAR 111
  GOSUB 112
  GOTO 114
  HEAD 115
  IF...THEN...(ELSE) 117
  IMAGE BUFFER MIRROR 119
  IMAGE BUFFER SAVE 120
  IMAGE LOAD 121
  IMAGENAMES$ 122
  IMAGES 123
  IMMEDIATE 124
  INKEY$ 127
  INPUT 128
  INPUT# 131
  INPUTS 132
  INPUT ON/OFF 130
  INSTR 133
Appendix I—Index

INVIMAGE 134
KEY BEEP 135
KEYBMAPS 137
KEY ON/OFF 136
KILL 139
LAYOUT 141
LAYOUT END 144
LAYOUT INPUT 145
LAYOUT RUN 146
LBLCOND 147
LED ON/OFF 149
LEFTS 150
LEN 151
LET 152
LINE INPUT 153
LINE INPUT# 154
LIST 155
LISTPFSVAR 157
LOAD 158
LOC 159
LOF 160
LSET 161
LTS& ON/OFF 162
MAG 163
MAKEASSOC 164
MAP 165
MERGE 167
MIBVAR& 168
MID$ 168
MKDIR 170
NAME DATES 171
NAME WEEKDAYS$ 172
NASC 173
NASCD 175
NEW 176
NORIMAGE 177
ON/OFF LINE 187
ON BREAK GOSUB 178
ON COMSET GOSUB 179
ON ERROR GOTO 181
ON GOSUB 182
ON GOTO 183
ON HTTP GOTO 184
ON KEY GOSUB 185
ON MIBVAR& GOSUB 188
OPEN 189
OPTIMIZE BATCH ON/OFF 192
PORTIN 193
PORTOUT ON/OFF 194
PRBAR 195
PRBOX 196
PRBUF 201
PRIMAGE 202
PRINT 203
PRINT# 205
PRINTFEED 206
PRINT KEY ON/OFF 204
PRINTONE 208
PRINTONE# 209
PRLINE 210
PRPOS 211
PRSTAT 213
PRTXT 215
PUT 217
RANDOM 218
RANDOMIZE 219
READY 220
REBOOT 221
REDIRECT OUT 222
REM 223
REMOVE IMAGE 224
RENDER ON/OFF 225
REPRINT ON/OFF 227
RESUME 228
RETURN 229
REWINDCONTROL 230
REWINDVOID 231
RIGHT$ 230
RSET 233
RUN 234
SAVE 235
SETASSOC 237
SET FAULTY DOT 15, 236
SETPFSVAR 238
SETSTDIO 239
SETUP 241
SETUP GET 244
SETUP KEY 245
SETUP WRITE 246
SGN 248
SORT 249
SOUND 250
SPACES$ 251
SPLIT 252
STOP 253
STORE IMAGE 254
STORE INPUT 255
STORE OFF 256
STR$ 257
STRINGS$ 258
SYSHEALTH 259
SYSHEALTHS 261
SYSVAR 262
TAGFIELD 270
TAGPROTECT 274
TAGREAD 275
TAGWRITE 276
TESTFEED 277
TICKS 278
TIMES$ 279
TIMEADDS 280
TIMEDIFF 281
TRANSFERS 287
TRANSFER NET 298
TRANSFERSET 288
TRANSFER STATUS 285
Appendix I—Index

TRANSFER ZMODEM 286
TRON/TROFF 289
VAL 290
VERBON/VERBOFF 291
VERSIONS 292
WEEKDAY 293
WEEKDAY$ 294
WEEKNUMBER 295
WHILE...WEND 296
XORMODE ON/OFF 297

Interleaved 2 of 5 356
Intermec Direct Protocol. See Direct Protocol
Intermec Ready-to-Work Indicator 81, 259
interrupt
  program. See program: execution, interrupt interruption
  communication. See communication channel: interrupt inverse
  print image 134
INVIMAGE instruction 134
IP. See INPUT instruction
IRI. See Intermec Ready-to-Work Indicator
ISO 18000-6B 411, 413
Italian character set 173

J

Japanese character set 173

K

key
  printer, enabling 136
KEY BEEP instruction 135
KEYBMAP$ instruction 137
KEY ON/OFF instruction 136
KILL instruction 139

L

label. See media
Label Taken Sensor
  enabling/disabling 162
layout
  creating 141–143
description, starting recording 145
description, stopping 144
  variable input 146
LAYOUT END instruction 144
LAYOUT INPUT instruction 145
LAYOUT instruction 141
LAYOUT RUN instruction 146
LBLCOND instruction 147
LED ON/OFF instruction 149
LEFT$ instruction 150
length, string 151
LEN instruction 151
LET instruction 152
LINE INPUT# instruction 154
LINE INPUT instruction 153

LIST instruction 155
LISTPFSVAR instruction 157
LOAD instruction 158
LOC instruction 159
LOF instruction 160
  loop
    in program 95, 296
LSET instruction 161
LTS& ON/OFF instruction 162

M

MAG instruction 163
magnifying
  bar code 22
  fonts and images 163
MAKEASSOC instruction 164
MAP instruction 165
mapping
  printer keys 137
MaxiCode 357
media
  feeding 104
  feed setup, overriding 147
  pulling 104
memory
  clearing 176
  formatting printer's 96
  free, number of bytes 105
MERGE instruction 167
MIBVAR& instruction 168
MicroPDF417 359
  bar code 359
MIDS instruction 168
mirror
  image buffer 119
MKDIR instruction 170
month. See date

N

NAME DATE$ instruction 171
NAME WEEKDAY$ instruction 172
NASCD instruction 175
NASC instruction 173
network
  opening connection 189
NEW instruction 176
NEXT. See FOR...TO...NEXT instruction
NI. See NORIMAGE instruction
NORIMAGE instruction 177
Norwegian character set 173
number
  formatting as string 101
  random, generating 218
numeric expression
  from string 290

O
Appendix I—Index

offset
  vertical 18
ON/OFF LINE instruction 187
ON BREAK GOSUB instruction 178
ON COMSET GOSUB instruction 179
ON ERROR GOTO instruction 181
ON GOSUB instruction 182
ON GOTO instruction 183
ON HTTP GOTO
  instruction 184
ON KEY GOSUB instruction 185
ON MIBVAR& GOSUB instruction 188
OPEN instruction 189
open network connection 189
OPTIMIZE BATCH ON/OFF instruction 192

P
PB. See PRBAR instruction
PCMAP 173
PDF417 361
PF. See PRINTFEED instruction
PL. See PRLINE instruction
PM. See PRIMAGE instruction
PORTIN instruction 193
PORTOUT ON/OFF instruction 194
Portuguese character set 173
power failure 157
  deleting variables at 66
PP. See PRPOS instruction
PRBAR instruction 195
PRBOX instruction 196
PRBUF instruction 201
PRIMAGE instruction 202
PRINT# instruction 205
print direction
  specifying 71–73
printer
  keys, enabling 136
  keys, mapping 137
PRINTFEED instruction 206
printhead. See thermal printhead
printing
  label 206
  text 215
  to channel 203
PRINT instruction 203
PRINT KEY ON/OFF instruction 204
PRINTONE# instruction 209
PRINTONE instruction 208
PRLINE instruction 210
program
  executing 234
  executing from within 82
  execution, ending 75
  execution, interrupt 28
  execution, resume 48
  execution, tracing 289
  listing 155
  stopping 253
PRPOS instruction 211
PRSTAT instruction 213
PRTXT instruction 215
PT. See PRTXT instruction
PUT instruction 217
PX. See PRBOX instruction

Q
QR Code 363

R
Random Access Mode 189
RANDOM instruction 218
RANDOMIZE instruction 219
random number
  generating 218
  generator, resetting 219
reading
  first character 127
READY instruction 220
REBOOT instruction 221
record, reading a 108
REDIRECT OUT instruction 222
REM instruction 223
REMOVE IMAGE instruction
  instruction 224
RENDER ON/OFF instruction 225
renumbering
  program lines 226
RENUM instruction 226
REPRINT ON/OFF instruction 227
restart. See PRTXT instruction
RESUME instruction 228
RETURN instruction 229
REWINDCONTROL instruction 230
REWINDVOID instruction 231
RFID tag 271, 406
  format 405
  memory allocation 411
  protecting 274
  reading 275
  writing 276, 410
RIGHT$ instruction 230
RSET instruction 233
RSS-14 365, 380
  composite 390–403
  Expanded 371
  Expanded Stacked 372
  Limited 370
  Stacked 367
  Stacked Omnidirectional 369
RUN instruction 234

S
SAVE instruction 235
scanner 404
search
  string 133
secret instruction 298
separator
  specifying for LAYOUT RUN 99
Sequential Access Mode 189
SETASSOC
  Instruction 237
SETASSOC instruction 237
SET FAULTY DOT instruction 15, 236
SETPFSVAR instruction 238
SETSTUDIO instruction 239
setup
  changing 241, 404
  retrieving 244
SETUP GET instruction 244
SETUP instruction 241
SETUP KEY instruction 245
setup mode
  entering 241
SETUP WRITE instruction 246
SGLN-64 407
SGLN-96 408
SGN instruction 248
SGTIN-64 406
SGTIN-96 407
Simple Network Management Protocol. See SNMP
SNMP 168, 188
SORT instruction 249
sound 135
SOUND instruction 27, 250
SPACES instruction 251
Spanish character set 173
SPLIT instruction 252
SSCC-64 407
SSCC-96 407
status
  checking printer 213
STOP instruction 253
STORE IMAGE instruction 254
STORE INPUT instruction 255
STORE OFF instruction 256
STR$ instruction 257
string
  format 101
  from numeric expression 257
  getting association value 109
  length, retrieving 151
  search in 133
  splitting into array 252
STRINGS instruction 258
string association
  traversing 110
subroutine
  branching to 112, 185
  returning from 229
Swedish character set 173
Syntax 8
SYSHEALTH$ instruction 261
SYSHEALTH instruction 259
system variables
  reading 262
  setting 262
SYSVAR instruction 262
T
tag
  EPC 406
TAGFIELD instruction 270
tag format
  ASCII 416
  EPC-HEX64 410
  EPC-HEX96 410
  GIAI-64 408
  GIAI-96 409
  GID-96 409
  GRAI-64 408
  GRAI-96 408
  HEX 416
  NUM 416
  SGLN-64 407
  SGLN-96 408
  SGTIN-64 406
  SGTIN-96 407
  SSCC-64 407
  SSCC-96 407
  USDOD-64 409
  USDOD-96 409
tag memory 411
TAGPROTECT instruction 274
TAGREAD instruction 275
tag structure 411
TAGWRITE instruction 276
TESTFEED instruction 277
THEN. See IF...THEN...(ELSE) instruction
  thermal printhead
    check 115
  Ticks instruction 278
time
  calculating difference 281
  formatting 100
  setting 279
  since power up 278
TIMES instruction 279
TIMEADDS$ instruction 280
TIMEDIFF instruction 281
TO. See FOR...TO...NEXT instruction
  TPH. See thermal printhead
  tracing
    program execution 289
TRANSFERS$ instruction 287
TRANSFER KERMIT
  Instruction 282
TRANSFER KERMIT instruction 282
TRANSFER NET instruction 298
TRANSFERSSET instruction 288
Appendix I—Index

TRANSFER STATUS instruction 285
TRANSFER ZMODEM instruction 286
TRON/TROFF instruction 289
TrueDoc. See font: TrueDoc
TrueType. See font: TrueType

U

uart
  communication channel 28, 30, 31
UCC 344, 406
Ucode EPC 1.19 411
UCODE HSL 413
Uniform Resource Identifier 410
UPC 376
UPC-A 374
UPC-A Composite 385
UPC-E 375
UPC-E Composite 384
URI. See Uniform Resource Identifier
USB. See communication channel: usb
USDOD-64 409
USDOD-96 409
UTF-8 173–175

V

VAL instruction 290
variable
  recovering 111
  system. See system variables
VERBON/VERBOFF instruction 291
verbosity
  setting 291
VERSION$ instruction 292
VOID label
  rewinding 231

W

wand 404
WEEKDAY$ instruction 294
WEEKDAY instruction 293
WEEKNUMBER instruction 295
WEND. See WHILE...WEND instruction
WHILE. See WHILE...WEND instruction
WHILE...WEND instruction 296

X

XORMODE ON/OFF instruction 297

Y

year. See date
Appendix I—Index