CPCL iOS SDK Reference Guide

Item	Content
Language	Swift 2.2
iOS Version	iOS 8.0 and above
framework	PrinterCommandSwift.framework

Update

Content	Date
Coding CPCL Command SDK	2016.4.21
Establishing CPCL SDK Reference Guide	2016.8.22

CPCL iOS SDK Reference Guide

- Update
- 1 Label Formatting Commands
  - 1.1 Label Sessions
  - 1.2 Barcode
    - 1.2.0 Barcode 1D
    - 1.2.1 Barocde Aztec
    - 1.2.2 Data Matrix
    - 1.2.3 DataBar(RSS) and CompositeBarcodes
    - 1.2.4 Maxicode
    - 1.2.5 PDF417
    - 1.2.6 QR Code
  - 1.3 Barcode Text
  - 1.4 Bat-Indicator
  - 1.5 Box
  - 1.6 CompressedGraphics
  - 1.7 Concat
  - 1.8 Count
  - 1.9 End/Print
  - 1.10 ExpandedGraphics
  - 1.11 FontGroup
  - 1.12 Image
  - 1.13 Centimeters, Dots, Inches, Millimeters
  - 1.14 Inverse Line
  - 1.15 Left, Center, Right

- 1.16 Line
- 1.17 Move
- 1.18 Multi-Line
- 1.19 Page Width
- 1.20 Pattern
- 1.21 PCX
- 1.22 PCXMAG
- 1.23 Persist
- 1.24 Rotate
- 1.25 Scale Text
- 1.26 Scale To Fit
- 1.27 Set Bold
- 1.28 Set Mag
- 1.29 Set Spacing
- 1.30 Temp Move
- 1.31 Text

2 Line Print Commands
- 2.1 Left Margin
- 2.2 LF equals CRLF (Line Print)
- 2.3 Orient (Line Print)
- 2.4 Line Print Position Adjust
- 2.5 Set LF
- 2.6 Set LP
- 2.7 Set LP Buffer
- 2.8 Set LP Timeout
- 2.9 Set Position
- 2.10 Line Feed
- 2.11 Carriage Return
- 2.12 Line Print Graphics

3 Font Commands
- 3.1 File Header
- 3.2 Char Set And Country

4 Media Management Commands
- 4.1 Auto Cal
- 4.2 Auto Pace
- 4.3 Bar Sense
- 4.4 Contrast
- 4.5 Feed
- 4.6 Form
- 4.7 Gap Sense
- 4.8 Journal
- 4.9 Label
- 4.10 Multi
- 4.11 No Pace
- 4.12 Out Of Paper
- 4.13 Pace

- 4.14 Paper Jam
- 4.16 PostFeed/PreFeed
- 4.17 Present AT
- 4.18 Reverse
- 4.19 Set Feed Length and Skip (Set FF)
- 4.20 Set TOF
- 4.21 Speed
- 4.22 Tone
- 4.23 Turn
- 4.24 Form Feed

5 Status Enquiry Commands
- 5.1 Name
- 5.2 Version
- 5.3 Printer Status
- 5.4 Extended Printer Status
- 5.5 Get Version Information

6 Utility and Diagnostic Commands
- 6.1 Abort
- 6.2 Baud
- 6.3 Beep
- 6.4 Capture
- 6.5 Check Sum/Check Sum Vertical
- 6.6 Char Count
- 6.7 Delay Actions
- 6.8 Display
- 6.9 Dump
- 6.10 Dump Image
- 6.11 Get Date
- 6.12 Get Time
- 6.13 Get Var
- 6.14 Line Terminator
- 6.15 Max Label Height
- 6.16 On Feed
- 6.17 On Low Battery
- 6.18 Re-Run
- 6.19 Set Date
- 6.29 Set Time
- 6.30 Set Version
- 6.31 Set Var and DO
- 6.32 Timeout
- 6.33 Wait
- 6.34 Label Session Position
- 6.35 Sound Printer Bell
- 6.36 Backspace
- 6.37 Get or Set CCL Key
- 6.38 Send Two-Key Report to Host

6.39 Send User Label Count
6.40 Acknowledge Reset
6.41 Shut Down Printer
6.42 Print Two-Key Report
7 Magnetic Card Reading Commands
7.1 MCR
7.2 MCR-CAN
7.3 MCR-QUERY
8 File Commands
8.1 Define and Use Format Sessions
8.2 Delete
8.3 Dir
8.4 End
8.5 File
8.6 Type

1 Label Formatting Commands

1.1 Label Sessions

- **Function**

> A label that contains information to be printed begins with the ! character, followed by a series of ASCIInumbers which represent information about the label that follows.
>
> In order for this session to be detected, the first character of the Offset parameter must be a digit.

- **Swift (iOS)**

```
public func cpclLabel(offset offset: Int, hRes: Int, vRes: Int, height: Int, quantity: Int)
```

- **Java (Android)**

```
```

- **C (Windows)**

```
```

- **Parameter**

Field Name	Description	Type	Valid Range
Offset	The number of units to offset all fields from the left side of the label horizontally.	Units Number	0 to 65535
Horizontal Res.	The horizontal resolution of this label, expressed in dots per inch.	Number (in dpi)	100 or 200
Vertical Res.	The vertical resolution of this label, expressed in dots per inch.	Number (in dpi)	100 or 200
Height	The height of the label in units.	Unit Number	0 to 65535
Quantity	The number of copies of the label to print.	Number (in quantity)	0 to 1024

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclCenter()
cpclText(rotate: 0, font: 4, fontSize: 0, x: 10, y: 0, text: "This is a ! command.")
cpclPrint()
```

1.2 Barcode

1.2.0 Barcode 1D

- **Function**

> The barcode command has several syntaxes based on the type of barcode being printed. The mostcommon syntax is the 1D barcode format.

- **Swift (iOS)**

```
public func cpclBarcode(type type: String, width: Int, ratio: Int, height: Int, x: Int, y: Int, barcode: String)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Type	Space Terminated String	The type of barcode to print.	See table below.
Width	5 Digit Unit Number	The width of a narrow bar.	0 to 65535 units
Ratio	5 Digit Number	The ratio of wide to narrow bars.	0 to 4, 20 to 30.
Height	5 Digit Unit Number	The height of the barcode.	0 to 65535 units
X	5 Digit Unit Number	The X position where the barcode begins	0 to 65535 units
Y	5 Digit Unit Number	The Y position where the barcode begins	0 to 65535 units
Data	Terminated String	The data to be encoded into a barcode.	Up to 8191 bytes of Alpha Numeric Data.

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclBarcode(type: CPCLBarcodeType.Code128.rawValue, width: 1, ratio: 1,
height: 50, x: 150, y: 10, barcode: "horizontal")
cpclText(rotate: 0, font: 4, fontSize: 0, x: 180, y: 60, text: "HORIZ.")
cpclBarcodeVertical(type: CPCLBarcodeType.Code128.rawValue, width: 1,
ratio: 1, height: 50, x: 10, y: 200, barcode: "vertical")
cpclText(rotate: 90, font: 4, fontSize: 0, x: 60, y: 180, text: "vertical")
cpclPrint()
```

1.2.1 Barocde Aztec

- **Function**

  - The `BARCODE AZTEC` command is used to print Aztec Code barcodes in the CPCL language.
  - `VBARCODE AZTEC` is identical to BARCODE, except it is oriented vertically.
  - Note that this command has optional parameters XD, EC, F, ME, M and ID. In order to use these parameters, the optional parameter must be followed by a space, then a number which complies with the table below.

- **Swift (iOS)**

```
public func cpclBarcodeAztec(xPos xPos: Int, yPos: Int, width: Int, erc:
Int, flags: Int, menu: Int, multi: Int, idField: String)
```

- **Parameter**

Parameter	Type	Description	Valid Range
X	5 Digit Unit Number	The X position where the barcode begins	0 to 65535 units
Y	5 Digit Unit Number	The Y position where the barcode begins	0 to 65535 units
Width	5 Digit Number	The dot-width of a single element in the code	1 to 36
ERL	5 Digit Number	The error recovery level or size	See below
Flags	5 Digit Number	Is the barcode using flag escapes?	0 or 1
Menu	5 Digit Number	Is the barcode a menu?	0 or 1
Multi	5 Digit Number	Is the barcode a structured append part?	0 or 1
ID	Or Space Terminated String	The structured append ID field	Up to 25 ASCII Characters
Data	Terminated String	The data to be encoded in the barcode	See below

- **Sample Code (Swift)**

1.2.2 Data Matrix

- **Function**

> - The `BARCODE DATAMATRIX` command is used to print Data Matrix barcodes in the CPCL language.
> - The command has a number of optional parameters, any number of which may be specified. If a parameter is specified more than once, the last value specified is used.

- **Swift (iOS)**

```
public func cpclDataMatrix(xPos xPos: Int, yPos: Int, scale: Int, ecc: Int,
columns: Int, rows: Int, format: Int, escapeChar: Int)
public func cpclAddBarcodeDataMatrixData(barcodeDataMatrix: String)
public func cpclEndDataMatrix()
```

- **Parameter**

Parameter	Type	Description	Valid Range
X	5 Digit Unit Number	The X position where the barcode begins	0 to 65535 units
Y	5 Digit Unit Number	The Y position where the barcode begins	0 to 65535 units
Scale	5 Digit Number	The scale of the barcode.	0 to 65535
ECC	5 Digit Number	Specifies the level of error correction.	See Below.
Columns	5 Digit Number	Specifies the number of columns to use.	See Below.
Rows	5 Digit Number	Specifies the number of rows to use.	See Below.
Format	5 Digit Number	Specifies the data format when ECC is not 200.	1 to 6
EscapeChar	Space Terminated String	Specifies the escape character to use.	See Below.
Data	Raw String	Data for barcode	See Below.

- **Sample Code (Swift)**

1.2.3 DataBar(RSS) and CompositeBarcodes

- **Function**

> - This command is used to print GS1 Databar (also known reduced space symbology) barcodes in CPCL, as well as Composite barcodes. There are 12 supported sub-types of GS1 Databar\Composite barcodes.
> - All barcode types supported by this command can consist of both a 1D and 2D component. If the barcode contains a 2D component, the output is referred to as a composite barcode. The 2D portion is optional, but the 1D portion is required.

> - If the parameters of the barcode are incorrect, in place of the barcode, a text message indicating the problem encountered while trying to form the barcode will be printed.

- **Swift (iOS)**

```
public func cpclBarcodeRSS(xPos xPos: Int, yPos: Int, scale: Int, lHeight: Int, sHeight: Int, segs: Int, type: Int, barcodeRSS: String)
```

- **Parameter**

Parameter	Type	Description	Valid Range
X	5 Digit Unit Number	The X position where the barcode begins	0 to 65535
Y	5 Digit Unit Number	The Y position where the barcode begins	0 to 65535
Scale	5 Digit Unit Number	The X and Y scaling factor of the barcode.	0 to 65535
LHeight	5 Digit Unit Number	Height of the linear part of the barcode	0 to 65535
SHeight	5 Digit Unit Number	Height of the separator between barcode parts	0 to 65535
Segs	5 Digit Unit Number	The maximum number of segments per row	0 to 65535
Type	5 Digit Number	The type of RSS barcode to print.	1 to 12
Data	Terminated String	1D and 2D Data to Print	8191 alphanumeric characters

- **Sample Code (Swift)**

1.2.4 Maxicode

- **Function**

> - The `BARCODE MAXICODE` command is used to print MaxiCode barcodes in the CPCL language. Only type-2 MaxiCode barcodes (which contain formatted data with a structured carrier message a d numeric postal code) can be used in CPCL.
> - `VBARCODE MAXICODE` is identical to `BARCODE MAXI`. MaxiCode barcodes do not

print in vertical orentation.
    - The barcode consists of fields that are represented with a field name followed by data. This field and data line can be repeated as many times as necessary to complete the barcode.
    - MaxiCode barcodes are always of a fixed size, there are no options to adjust its size.

- **Swift (iOS)**

```
public func cpclBarcodeMaxicode(xPos xPos: Int, yPos: Int, field: Int,
barcodeMaxicode: String)
```

- **Parameter**

Parameter	Type	Description	Valid Range
X	5 Digit Unit Number	The X position where the barcode begins	0 to 65535 units
Y	5 Digit Unit Number	The Y position where the barcode begins	0 to 65535 units
Field	Space Terminated String	The ratio of wide to narrow bars.	1 or 2
Data	Terminated String	Unit-width of the barcode in dots	0 to 65535 dots

- **Sample Code (Swift)**

1.2.5 PDF417

- **Function**

    - The `BARCODE PDF-417` command is used to print PDF417 barcodes in the CPCL language.
    - The command has a number of optional parameters, any number of which may be specified. If a parameter is specified more than once, the last value specified is used.
    - If a PDF-417 barcode's parameter's cause an error that would cause it not to print, a detailed error message will be displayed instead.
    - There is no option to use structured append PDF-417 barcodes in CPCL.

- **Swift (iOS)**

```
public func cpclBarcodePDF417(xPos xPos: Int, yPos: Int, XDot: Int, YDot:
Int, columes: Int, rows: Int, ecc: Int, binaryModel: Int)
public func cpclBarcodePDF417Vertical(xPos xPos: Int, yPos: Int, XDot: Int,
YDot: Int, columes: Int, rows: Int, ecc: Int, binaryModel: Int)
public func cpclBarcodePDF417Data(pdf417Data: String)
public func cpclBarcodePDF417End()
```

- **Parameter**

Parameter	Type	Description	Valid Range
X	5 Digit Unit Number	The X position where the barcode begins	0 to 65535 units
Y	5 Digit Unit Number	The Y position where the barcode begins	0 to 65535 units
XDot	5 Digit Number	The X size of a single element in dots	0 to 65535
YDot	5 Digit Number	The Y size of a single element in dots	0 to 65535
Columns	5 Digit Number	Specifies the number of columns to use.	1 to 30
Rows	5 Digit Number	Specifies the number of rows to use.	0 to 90
ECC	5 Digit Number	Specifies the error recovery level.	1 to 8
BinaryMode	Single Digit	Force binary compaction mode	0 or 1
Data	Raw String	The data to be encoded in the barcode.	See below.

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclCenter()
cpclBarcodePDF417(xPos: 10, yPos: 20, XDot: 3, YDot: 6, columes: 3, rows:
3, ecc: 2, binaryModel: 1)
cpclBarcodePDF417Data("pdf-417 data")
cpclBarcodePDF417End()
cpclText(rotate: 0, font: 1, fontSize: 0, x: 10, y: 120, text: "pdf data")
cpclText(rotate: 0, font: 1, fontSize: 0, x: 10, y: 160, text:
"abcde123456")
cpclPrint()
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 400, quantity: 1)
cpclBarcodePDF417Vertical(xPos: 30, yPos: 400, XDot: 3, YDot: 6, columes:
3, rows: 3, ecc: 3, binaryModel: 1)
cpclBarcodePDF417Data("pdf data vertical")
cpclBarcodePDF417End()
cpclText(rotate: 90, font: 1, fontSize: 0, x: 120, y: 400, text: "pdf
data")
cpclText(rotate: 90, font: 1, fontSize: 0, x: 170, y: 400, text:
"abcde123456")
cpclPrint()
```

1.2.6 QR Code

- **Function**

  > - The BARCODE QR command is used to print QR Code barcodes in the CPCL language.
  > - VBARCODE QR is identical to BARCODE, except it is oriented vertically.
  > - Note that this command has optional parameters M, U. In order to use these parameters, the M or U character must be followed by a space, then a number which complies with the table below.

- **Swift (iOS)**

```
public func cpclBarcodeQRcode(xPos xPos: Int, yPos: Int, model: Int,
unitWidth: Int)
public func cpclBarcodeQRcodeVertical(xPos xPos: Int, yPos: Int, model:
Int, unitWidth: Int, config: Int, barcodeQRcode: String)
public func cpclBarcodeQRcodeData(barcodeQRcode: String, config: String)
public func cpclBarcodeQRcodeEnd()
```

- **Parameter**

Parameter	Type	Description	Valid Range
X	5 Digit Unit Number	The X position where the barcode begins	0 to 65535 units
Y	5 Digit Unit Number	The Y position where the barcode begins	0 to 65535 units
Model	5 Digit Number	The ratio of wide to narrow bars.	1 or 2
Width	5 Digit Number	Unit-width of the barcode in dots	0 to 65535 dots
Config	Raw String	Configuration options for barcode	See Below.
Data	Raw String	Data for barcode	See Below.

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 500, quantity: 1)
cpclBarcodeQRcode(xPos: 50, yPos: 50, model: 2, unitWidth: 5)
cpclBarcodeQRcodeData("AQR code ABC123,N9876", config: "MM")
cpclBarcodeQRcodeEnd()
cpclPrint()
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclBarcodeQRcode(xPos: 50, yPos: 50, model: 2, unitWidth: 10)
cpclBarcodeQRcodeData("AQR code ABC123,N9876", config: "MM")
cpclBarcodeQRcodeEnd()
cpclPrint()
```

1.3 Barcode Text

- **Function**

> `BARCODE-TEXT` is used to specify if a human-readable text representation of barcode data should be printed below 1D barcodes. This text is applied for both line-print and label-based barcodes.

- **Swift (iOS)**

```
// print annotation text line
public func cpclBarcodeText(font font: Int, fontSize: Int, offset: Int)
public func cpclBarcodeText(trueTypeFont trueTypeFont: Int, xScale: Int,
yScale: Int, offset: Int)
// close annotation text line
public func cpclBarcodeTextOff()
```

- **Parameter**
  - **Pre-scaled Font Syntax**

Parameter	Type	Description	Range
FontNameOrNumber	Space-Terminated String	A font name or number to create the representation	See Below.
FontSize	3 Digit Number	The size of the font.	0 to 999
Offset	3 Digit Unit Number	How far in units the text is from the barcode	0 to 999

  - **Alternate Syntax – TTF Font Syntax**

Parameter	Type	Description	Range
TrueTypeFontName	Space-Terminated String	The filename of the TTF font with extension	See Below.
XScale	3 Digit Number	The X size of the font, in dots.	0 to 999
YScale	3 Digit Number	The Y size of the font, in dots	0 to 999
Offset	3 Digit Unit Number	How far in units the text is from the barcode	0 to 999

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 400, quantity: 1)
cpclCenter()
cpclBarcodeText(font: 4, fontSize: 0, offset: 10)
cpclBarcode(type: CPCLBarcodeType.Code128.rawValue, width: 1, ratio: 1,
height: 50, x: 0, y: 20, barcode: "barcode data")
cpclBarcodeVertical(type: "128", width: 1, ratio: 1, height: 50, x: 0, y:
390, barcode: "barcode data")
cpclBarcodeTextOff()
cpclPrint()
```

1.4 Bat-Indicator

- **Function**

> `BAT-INDICATOR` is used to print a graphical representation of the current charge state of the battery on a label.The `BAT-INDICATOR` command takes a series of optional parameters to specify its size and configuration.

- **Swift (iOS)**

```
public func cpclBatIndicator(xPos xPos: String, yPos: String)
```

- **Parameter**

Parameter	Type	Description	Valid Range
X	5 Digit Unit Number	The X origin of the graphic.	0 to 65535
Y	5 Digit Unit Number	The Y origin of the graphic.	0 to 65535

- **Sample Code**

1.5 Box

- **Function**

> The `BOX` command is used to draw a box.By default, `BOX` draws a box in solid black, but the pattern used to fill the box can be changed with the `PATTERN` command.The `BOX` command can be used with the justify commands `CENTER`, `LEFT` and `RIGHT` to align the box.

- **Swift (iOS)**

```
public func cpclBox(xPos xPos: Int, yPos: Int, xEnd: Int, yEnd: Int,
thickness: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
X	5 Digit Unit Number	The X origin of the box.	0 to 65535
Y	5 Digit Unit Number	The Y origin of the box.	0 to 65535
EndX	5 Digit Unit Number	The X coordinate where the box ends.	0 to 65535
EndY	5 Digit Unit Number	The Y coordinate where the box ends.	0 to 65535
Thickness	5 Digit Unit Number	The thickness of the lines in the box.	0 to 65535

- **Sample Code**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclBox(xPos: 0, yPos: 0, xEnd: 200, yEnd: 200, thickness: 1)
cpclPrint()
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclBox(xPos: 0, yPos: 0, xEnd: 200, yEnd: 200, thickness: 2)
cpclPrint()
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclBox(xPos: 0, yPos: 0, xEnd: 200, yEnd: 200, thickness: 3)
cpclPrint()
```

1.6 CompressedGraphics

- **Function**

> The `COMPRESSED-GRAPHICS` command is used to print raw binary bitmap data to the label. `COMPRESSED-GRAPHICS` is the same except it is oriented vertically. The data itself is not compressed per say, it is just more efficiently represented than in the `EXPANDED-GRAPHICS` command. The binary data is not compressed in this command.

- **Swift (iOS)**

```
public func cpclCompressedGraphics(imageWidth imageWidth: Int, imageHeight:
Int, x: Int, y: Int, bitmapData: NSData)
public func cpclCompressedGraphicsVertical(byteWidth byteWidth: Int,
height: Int, x: Int, y: Int, bitmapData: NSData)
```

- **Parameter**

Parameter	Type	Description	Valid Range
ByteWidth	3 Digit Number	The byte width of the image being transmitted.	0 to 999
Height	5 Digit Unit Number	The height of the data to follow in units.	0 to 65535
X	5 Digit Unit Number	The X origin of the graphic.	0 to 65535
Y	5 Digit Unit Number	The Y origin of the graphic.	0 to 65535
Data	Raw String	The data that makes up the bitmap to be printed.	See below.

- **Sample Code**

1.7 Concat

- **Function**

> The CONCAT command is used to concatenate multiple fonts and sizes of text on to a single line, and to align their top-lines in a specific way. The command starts with a CONCAT command, followed by one or more sub-commands. Each sub-command adds more text onto the concatenation from left to right.

- **Swift (iOS)**

```
// Concat Begin (Master Syntax)
public func cpclConcatStart(xPos xPos: Int, yPos: Int)
public func cpclConcatVerticalStart(xPos xPos: Int, yPos: Int)
// FontNameOrNumber
public func cpclConcatText(font font: Int, fontSize: Int, offset: Int,
text: String)
// Sub-Command – Scale-Text
public func cpclConcatScaleText(scaledFont scaledFont: Int, xScale: Int,
yScale: Int, offset: Int, text: String)
public func cpclConcatScaleTextVertical(scaledFont scaledFont: Int, xScale:
Int, yScale: Int, offset: Int, text: String)
// Sub-command – Font-Group
public func cpclConcatText(fontGroup fontGroup: Int, offset: Int, text:
String)
// Concat End
public func cpclConcatEnd()
```

- **Parameter**
  - **Master Syntax**

Parameter	Type	Description	Valid Range
X	5 Digit Unit Number	The X origin of the text string.	0 to 65535
Y	5 Digit Unit Number	The Y origin of the text string.	0 to 65535

  - **FontNameOrNumber**

Parameter	Type	Description	Valid Range
FontNameOrNumber	Space-Terminated String	A font name or number to create the text.	See Below.
FontSize	5 Digit Number	The size of the pre-scaled font.	0 to 65535
Offset	5 Digit Unit Number	How far from Y is the top of this text?	0 to 65535
Data	CR-LF Terminated String	The text data to be concatenated.	Up to 2024 characters1

- **Sub-Command – Scale-Text**

Parameter	Type	Description	Valid Range
ScaledFontName	Space-Terminated String	A scaled font used to create the text.	See Below.
XPoints	5 Digit Number	The X size of the scaled font, in points.	0 to 65535
YPoints	5 Digit Number	The Y size of the scaled font, in points.	0 to 65535
Offset	5 Digit Unit Number	How far from Y is the top of this text?	0 to 65535
Data	CR-LF Terminated String	The text data to be concatenated.	Up to 2024 characters1

- **Sub-command – Font-Group**

Parameter	Type	Description	Valid Range
FontGroupNumber	Space-Terminated String	The number of the font-group to use	0 to 10
Offset	5 Digit Unit Number	How far from Y is the top of this text?	0 to 65535
Data	CR-LF Terminated String	The text data to be concatenated.	Up to 2024 characters1

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclText(rotate: 0, font: 0, fontSize: 2, x: 0, y: 0, text: "concat
command")
cpclConcatStart(xPos: 0, yPos: 90)
cpclConcatText(font: 4, fontSize: 0, offset: 10, text: "$")
cpclConcatText(font: 4, fontSize: 1, offset: 20, text: "12")
cpclConcatText(font: 1, fontSize: 2, offset: 30, text: "34")
cpclConcatText(font: 8, fontSize: 0, offset: 60, text: "Hello")
cpclConcatText(font: 9, fontSize: 1, offset: 90, text: "World")
cpclConcatEnd()
cpclPrint()
```

1.8 Count

- **Function**

  - The `COUNT` command is used to increment or decrement a field on a label when the labels are printed in a batch, that is with a quantity greater than one on in the label session definition.
  - COUNT fields are not cleared at the start of each new label session. If you try to use COUNT on an unsupported field, or use COUNT without a field, and you previously adjusted a field successfully, the old field definition will be used for the second label and on in the batch (adjusts don't take effect until the second label)
  - No more than 30 COUNT commands can appear in a single label session. Any beyond this amount will have no effect.

- **Swift (iOS)**

```
// 计数，加减条码文本数据段数值
public func cpclCount(adjust: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Adjust	Terminated String	The amount to adjust the field.	Up to 20 ASCII digits, signed

- **Sample Code**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 3)
cpclAnnotation("Print 3 labels")
cpclCenter()
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 50, text: "testing 001")
cpclCount(1) // 计数，加减条码文本数据段数值
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 100, text: "barcode
value is 123456789")
cpclCount(-10)
cpclBarcode(type: CPCLBarcodeType.Code128.rawValue, width: 1, ratio: 1,
height: 50, x: 0, y: 150, barcode: "{A123456789")
cpclCount(-10)
cpclPrint()
```

1.9 End/Print

- **Function**

> - `PRINT`, and its alias `END` is used to terminate a CPCL label session, and create the resulting print out.
> - Every CPCL label session must be terminated with a PRINT command.
> - Every example for this section uses the PRINT command to terminate labels.

- **Swift (iOS)**

```
// 功能完全一样，End 是 Print 的别名
public func cpclPrint()
public func cpclEnd()
```

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclCenter()
cpclText(rotate: 0, font: 4, fontSize: 0, x: 10, y: 0, text: "This is a ! command.")
cpclPrint()
```

1.10 ExpandedGraphics

- **Function**

  - The `EXPANDED-GRAPHICS` command is used to print ASCII encoded bitmap data to the label.

  `EXPANDED-GRAPHICS` is the same except it is oriented vertically.

  - Unless the communication method does not permit the use of binary data, graphical data is more efficiently represented by using the `COMPRESSED-GRAPHICS` command.

- **Swift (iOS)**

```
public func cpclExpandedGraphics(byteWidth byteWidth: Int, height: Int, xPos: Int, yPos: Int, bitmapData: NSData)
public func cpclExpandedGraphicsVertical(byteWidth byteWidth: Int, height: Int, xPos: Int, yPos: Int, bitmapData: NSData)
```

- **Parameter**

Parameter	Type	Description	Valid Range
ByteWidth	3 Digit Number	The byte width of the image being transmitted.	0 to 999
Height	5 Digit Number	The height of the data to follow in units.	0 to 65535
X	5 Digit Number	The X origin of the graphic.	0 to 65535
Y	5 Digit Number	The Y origin of the graphic.	0 to 65535
Data	Raw String	The data that makes up the bitmap to be printed.	See below.

- **Sample Code**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
let data =
cpclEncodeText("F0F0F0F0F0F0F0F00F0F0F0F0F0F0F0FF0F0F0F0F0F0F0F00F0F0F0F0F0
F0F0F")
cpclExpandedGraphics(byteWidth: 2, height: 16, xPos: 45, yPos: 45,
bitmapData: data)
cpclPrint()
```

1.11 FontGroup

- **Function**

  - `FONT-GROUP` is used to define groups of pre-scaled fonts which can be used to for automatic font size calculation when used with various text commands in CPCL.
  - Up to a total of 10 font groups can be defined, numbering from 0 to 9. Each `FONT-GROUP` command defines a single font group, which an in turn contain up to 10 pairs of font name or number and size values. These values are repeated as many times as desired in the FONT-GROUP command before being CR-LF terminated. At least one pair must be specified to define the group, though two or more is recommended.
  - When a text command using a font group is used, the printer will automatically select a font from the font group which allows the text to fit in the area from the X and Y coordinates specified in the text command to the edge of the label area as defined by the `PAGE-WIDTH` command.

> - In order to make the text fit, the printer can not only select any of the fonts, but can change the spacing of the font, decreasing the space between characters to attempt to make it fit. If, even after decreasing the spacing to a minimal amount, the text will not fit, the text command is aborted and no text is printed all for that text command.
> - Font groups are supported in the `TEXT` and `CONCAT` commands in CPCL. Font groups do work with the MULTILINE command, however only one member of the font group is selected for the entire `MULTI-LINE` set, based on the font size needed for the widest line of text in the MULTI-LINE group.
> - Once defined in a label or utilities session, a font group remains defined until power is cycled. By default at power on, no font groups are defined.

- **Swift (iOS)**

```
public func cpclFontGroup(group: Int, font: Int, size: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Group	5 Digit Number	Specifies the group number to define	0 to 9
FontNameOrNumber	Space-Terminated String	A font name or number to create the representation	See Below.
FontSize	5 Digit Number	The size of the font.	0 to 65535

- **Sample Code**

1.12 Image

- **Function**

> - `IMAGE` is used to change the drawing method of pre-scaled text commands and `LINE` commands.
> - By default, image data drawn to a label ORed with existing label data, that is if there is already a black dot at the location being drawn to, the dot will continue to appear black.
> - By using the IMAGE command, you can change this behavior to XOR new drawing with existing label data, which means that if there is an existing black dot at the location being drawn to, the dot will be erased and replaced with a white dot.

> - IMAGE is persistent between labels, with the most recent setting in either a label or utilities session taking precedence. It can be changed as many times as needed per label.

- **Swift (iOS)**

```
public func cpclImageMode(mode: String)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Mode	CR-LF Terminated String	Specifies the drawing mode	OR or XOR

- **Sample Code**

1.13 Centimeters, Dots, Inches, Millimeters

- **Function**

> - The various IN- commands change the system of measurement for all parameters in CPCL which are identified as unit numbers to centimeters. At the beginning of each new label, the system is reset to dots (the equivalent of the IN-DOTS command).
> - If any of these commands are the very first one in a label session, the session definition's height parameter is re-evaluated in the new system of measurement. Otherwise the command only affects subsequent fields in the label session.
> - You can change the system of measurement as many times as necessary per label.

- **Swift (iOS)**

```
public func cpclInCentimeters()
public func cpclInDots()
public func cpclInChes()
public func cpclInMillimeters()
```

- **Parameter**

Requested Unit	Conversion Factor to Dots
Dots (Default)	1
Centimeters	80
Millimeters	8
Inches	203.2

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclInChes()
cpclOnFeed_Feed()
cpclPrint()

cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 240, quantity: 1)
cpclInDots()
cpclOnFeed_Feed()
cpclPrint()

cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 4, quantity: 1)
cpclInCentimeters()
cpclOnFeed_Feed()
cpclPrint()
```

1.14 Inverse Line

- **Function**

  - The `INVERSE-LINE` command is used to draw a line which inverts the label area over which it is drawn.
  - LINE has two modes of operation based on whether or not one side of the line is flat.
  - If the line is straight (X and EndX are the same or Y and EndY are the same), you can align the line using the `CENTER`, `LEFT` and `RIGHT` commands.
  - If the line is diagonal, it cannot be aligned or filled.

- **Swift (iOS)**

```
public func cpclInverseLine(xPos xPos: Int, yPos: Int, xEnd: Int, yEnd:
Int, thickness: Int)
public func cpclReverseLine(xPos xPos: Int, yPos: Int, xEnd: Int, yEnd:
Int, thickness: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
X	5 Digit Unit Number	The X origin of the line.	0 to 65535
Y	5 Digit Unit Number	The Y origin of the line.	0 to 65535
EndX	5 Digit Unit Number	The X coordinate where the line ends.	0 to 65535
EndY	5 Digit Unit Number	The Y coordinate where the line ends.	0 to 65535
Thickness	5 Digit Unit Number	The thickness of the line.	0 to 65535

- **Sample Code**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclCenter()
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 45, text: "save")
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 95, text: "more")
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 145, text: "more")
cpclInverseLine(xPos: 0, yPos: 45, xEnd: 145, yEnd: 45, thickness: 45)
cpclInverseLine(xPos: 0, yPos: 95, xEnd: 145, yEnd: 95, thickness: 45)
cpclInverseLine(xPos: 0, yPos: 145, xEnd: 145, yEnd: 145, thickness: 45)
cpclPrint()
```

1.15 Left, Center, Right

- **Function**

  - The LEFT command is used to change the justification of supported fields. It is part of a series of justification commands which includes `LEFT`, `CENTER` and `RIGHT`.
  - LEFT is the default justification for labels. Each time a new label session is started, it is set to LEFT alignment (specifically LEFT 0). Once used, LEFT is persistent within the label session.

- **Swift (iOS)**

```
public func cpclLeft(range: Int)
public func cpclLeft()

public func cpclCenterWithRange(range: Int)
public func cpclCenter()

public func cpclRight(range: Int)
public func cpclRight()
```

- **Parameter**

Parameter	Type	Description	Valid Range
Range	5 Digit Unit Number	Sets `FONT-GROUP` field width.	0 to 65535

- **Mark**

  The following commands support justification：

BARCODE (all 1D types)	BARCODE MAXICODE	BOX
CONCAT	INVERSE-LINE, REVERSE-LINE	LINE
PCX, PCX90, PCX180, PCX270	PCXMAG	SCALE-TEXT
SCALE-TO-FIT	TEXT, TEXT90, TEXT180, TEXT270	COMPRESSED-GRAPHICS, EXPANDED- GRAPHICS

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 500, quantity: 1)
cpclCenter()
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 0, text: "center")
cpclLeft()
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 50, text: "left")
cpclRight()
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 100, text: "right")
cpclCenterWithRange(200)
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 150, text: "center 200")
cpclRight(200)
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 200, text: "right 200")
cpclLeft()
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 250, text: "left 200")
cpclPrint()
```

1.16 Line

- **Function**

  > - The `LINE` command is used to draw a line.
  > - LINE has two modes of operation based on whether or not one side of the line is flat.
  > - If the line is straight (X and EndX are the same or Y and EndY are the same), you can align the line using the CENTER, LEFT and RIGHT commands. You can also use the `PATTERN` command to fill the line with a pattern. The PATTERN command can be used to create white lines.
  > - If the line is diagonal, it cannot be aligned or filled.

- **Swift (iOS)**

```
public func cpclLine(xPos xPos: Int, yPos: Int, xEnd: Int, yEnd: Int,
thickness: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
X	5 Digit Unit Number	The X origin of the line.	0 to 65535
Y	5 Digit Unit Number	The Y origin of the line.	0 to 65535
EndX	5 Digit Unit Number	The X coordinate where the line ends.	0 to 65535
EndY	5 Digit Unit Number	The Y coordinate where the line ends.	0 to 65535
Thickness	5 Digit Unit Number	The thickness of the line.	0 to 65535

- **Sample Code**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclLine(xPos: 0, yPos: 0, xEnd: 200, yEnd: 0, thickness: 1)
cpclLine(xPos: 0, yPos: 0, xEnd: 200, yEnd: 200, thickness: 2)
cpclLine(xPos: 0, yPos: 0, xEnd: 0, yEnd: 200, thickness: 3)
cpclPrint()
```

1.17 Move

- **Function**

  - The `MOVE` command moves the origin of a label. The effect is most pronounced when using a media that synchronizes to marks. It does affect continuous mode labels, but the effects may be difficult to see, particularly with the MoveUp parameter.
  - The values specified by MOVE are added to those specified by `TEMP-MOVE` to determine the final position. Note that TEMP-MOVE and MOVE are both negative aware. This means that if a MOVE 20 20 was in effect, and a TEMP-MOVE -20 -20 was also in effect, the net location used would be 0,0.
  - The MOVE command is available in both label and utilities sessions and is persistent between labels, but is reset when the printer is power cycled.

- **Swift (iOS)**

```
public func cpclMove(right right: String, up: String)
```

- **Parameter**

Parameter	Type	Description	Valid Range
MoveRight	5 Digit Unit Number	How far to move the label to the right.	0 to 65535
MoveUp	5 Digit Unit Number	How far to move the label up.	0 to 65535

- **Sample Code**

1.18 Multi-Line

- **Function**

  - The `MULTILINE` command is used to print a number of lines of text using the same font without having to manually specify the spacing or positioning of each line.
  - The basic format of the command is the MULTILINE command followed by LineHeight and then a fully formed text command without its final parameter which would specify the data to print, followed by a CR and LF.
  - After that, as many Data lines as necessary can be specified without limit. Each field will be printed as if it was an individual text command, with the vertical position increasing by LineHeight for each line.

- **Swift (iOS)**

```
public func cpclMultiLineStart(lineHeight lineHeight: Int)
...
public func cpclInsertTextLine(textLine: String)
...
public func cpclMultiLineEnd()
```

- **Parameter**

Parameter	Type	Description	Valid Range
LineHeight	5 Digit Unit Number	Spacing between each line in units.	0 to 65535
TextCommand	Command	A command specifying formatting for the text.	See Below.
Data	CR-LF Terminated String	A text line to print using the formatting.	See Below.

- **Mark**

  the following text commands are supported for use with MULTILINE.

  > - TEXT, TEXT90, TEXT180, TEXT270, VTEXT, T, T90,T180,T270, VT
  > - SCALE-TO-FIT, VSCALE-TO-FIT, STF, VSTF
  > - SCALE-TEXT, VSCALE-TEXT, ST, VST

- **Sample Code**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 228, quantity: 1)
cpclMultiLineStart(lineHeight: 40)
cpclText(rotate: 0, fontGroup: 1, x: 0, y: 0, text: "10")
cpclInsertTextLine("line 1")
cpclInsertTextLine("line 2")
cpclInsertTextLine("line 3")
cpclInsertTextLine("line 4")
cpclMultiLineEnd()
cpclPrint()
```

1.19 Page Width

- **Function**

  > - The PAGE-WIDTH command is used to specify the width a label session. The height of the session is defined in the session header.
  > - Some printers have built-in sensors to detect the width of the currently installed media. If this sensor is installed and enabled, setting a PAGE-WIDTH of 0 will use the detected media width for the label.
  > - The PAGE-WIDTH command must either be used in a utilities session, or before any command which creates output on the label is used. If used in a label session, it should be the first command after the session line (or the second if you are using a

> units command such as IN-CENTIMERES to modify the session line).
> - It is not recommended to change PAGE-WIDTH once you have started drawing to the label, as any fields drawn so far will become corrupted.

- **Swift (iOS)**

```
public func cpclPageWidth(width: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Width	5 Digit Unit Number	Spacing between each line in units.	0 to 65535

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclPageWidth(100)
cpclCenter()
cpclText(rotate: 0, font: 4, fontSize: 0, x: 10, y: 0, text: "This is a !
command")
cpclPrint()

cpclUtilitySession()
cpclPageWidth(80)
cpclPrint()
cpclInsertTextLine("this text is printed with label money with set to 80
dots.")

cpclUtilitySession()
cpclPageWidth(150)
cpclPrint()
cpclInsertTextLine("this text is printed with label money with set to 150
dots.")

cpclUtilitySession()
cpclPageWidth(300)
cpclPrint()
cpclInsertTextLine("this text is printed with label money with set to 300
dots.")

cpclUtilitySession()
cpclPageWidth(400)
cpclPrint()
cpclInsertTextLine("this text is printed with label money with set to 400
dots.")
```

1.20 Pattern

- **Function**

  - The `PATTERN` command is used to change the fill patterns of the `SCALE-TEXT` and `SCALE-TO-FIT` (when using CSF fonts only), some LINE commands (must be horizontal or vertical lines), and all BOX commands.
  - By default and at the start of each label session, PATTERN is set to 100, which is solid black. PATTERN is persistent within the same label session.
  - Lines or boxes drawn with PATTERN values besides 100 start and end one dot higher than their pixel position indicates. This behavior does not occur with SCALE-TEXT.

- **Swift (iOS)**

```
public func cpclPatternNumber(patternNumber: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Pattern	5 Digit Number	The pattern number to use	0 to 106

- **Sample Code**

1.21 PCX

- **Function**

  - The `PCX` command is used to print a ZSoft PCX file.
  - The PCX file must be a 2-color, 1-plane RLE-encoded PCX file. If it is not, command is aborted and the binary data of the PCX flows into the label data, often causing the printer to enter an unpredictable state.
  - PCX supports the justification commands LEFT, RIGHT and CENTER, but if the PCX is too wide to fit on the current label (as determined with `PAGE-WIDTH`), no image will be printed at all. This does not occur if the image is too tall – it will be properly clipped in this case.
  - The origins of the rotated versions of the PCX commands (`PCX90`, `PCX180` and `PCX270`) do not use the same math for their origins as TEXT and other commands do. See the example for more details.

- **Swift (iOS)**

```
public func cpclPCX(rotation rotation: Int, xPos: Int, yPos: Int,
bitmapData: NSData)
```

- **Parameter**

Parameter	Type	Description	Valid Range
X	5 Digit Unit Number	The X origin of the PCX.	0 to 65535
Y	5 Digit Unit Number	The Y origin of the PCX.	0 to 65535
Data	Raw String	Binary data containing the PCX file	See below.

- **Sample Code**

1.22 PCXMAG

- **Function**

  - The `PCXMAG` command is used to print a ZSoft PCX file that has been magnified vertically and horizontally.
  - The requirements for the PCX file itself are identical to the `PCX` command. The only difference between this command and PCX is the addition of the magnification parameters.
  - PCXMAG supports the justification commands LEFT, RIGHT and CENTER. Unlike with PCX, if the image flows off the right side of the page, it does not wrap, the remainder of the image data is disposed of.
  - If the unscaled image is too wide to fit on the page (as defined by page width), no image will be printed.
  - The use of alignment commands (RIGHT and CENTER) with PCXMAG is not supported.

- **Swift (iOS)**

```
public func cpclPCXMAG(xPos xPos: Int, yPos: Int, XMag: Int, YMag: Int,
bitmapData: NSData)
```

- **Parameter**

Parameter	Type	Description	Valid Range
X	5 Digit Unit Number	The X origin of the PCX.	0 to 65535
Y	5 Digit Unit Number	The Y origin of the PCX.	0 to 65535
XMag	5 Digit Unit Number	The X magnification factor of the PCX	0 to 65535
YMag	5 Digit Unit Number	The Y magnification factor of the PCX	0 to 65535
Data	Raw String	Binary data containing the PCX file	See below.

- **Sample Code**

1.23 Persist

- **Function**

  - PERSIST is used to determine if label memory is erased at the end of each label session. By default, when a label session ends, the label memory is cleared. This option can be changed so that the image of the last label is retained and merged with any subsequent label commands.
  - The setting is preserved between label sessions, but is always set to OFF at startup.

- **Swift (iOS)**

```
public func cpclPersist(option: String)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Option	Terminated String	When set to ON, label images persist	ON or OFF

- **Sample Code**

1.24 Rotate

- **Function**

  - The `ROTATE` command is used to specify the rotation of a scalable or TrueType font within the printer.
  - ROTATE can also be used with any of these commands when they are part of a `MULTI-LINE` session, but not within the MULTI-LINE session itself.
  - For all of these commands, if the rotated text extends off the top or right edge of the label, the text will be truncated, and the final character on the line may not be fully formed. In the left and bottom direction, a fully formed partial character will be printed instead.
  - `SCALE-TO-FIT` and `CONCAT` does not support use of the ROTATE command with TrueType fonts.

- **Swift (iOS)**

```
public func cpclRotate(degrees: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Degrees	5 Digit Number	The number of degrees to rotate counter-clockwise	0 to 65535

- **Mark**

  The following commands support rotation

  - CONCAT, VCONCAT
  - SCALE-TEXT, VSCALE-TEXT
  - SCALE-TO-FIT, VSCALE-TO-FIT

- **Sample Code**

1.25 Scale Text

- **Function**

  - The `SCALE-TEXT` command is used print scaled text in CPCL from either a scaled or TrueType font.
  - `VSCALE-TEXT` has the same functionality, but orients the text rotated 90 degrees.
  - When the text generated by SCALE-TEXT flows off the edge of the page, it does not wrap. On the top edge it is truncated to the nearest character that will fully fit (when using VSCALE-TEXT). On the right edge, it is truncated to at the last pixel of the page width, including any partial characters.

- **Swift (iOS)**

```
public func cpclScaleText(scaledFont: String, xScale: Int, yScale: Int, x: Int, y: Int, text:String)
public func cpclScaleTextVertical(scaledFont: String, xScale: Int, yScale: Int, x: Int, y: Int, text:String)
```

- **Parameter**

Parameter	Type	Description	Valid Range
ScaledFontName	Space-Terminated String	A scaled font used to create the text.	See Below.
XPoints	5 Digit Number	The X size of the scaled font, in points.	0 to 65535
YPoints	5 Digit Number	The Y size of the scaled font, in points.	0 to 65535
X	5 Digit Unit Number	The X origin of the scaled text in units.	0 to 65535
Y	5 Digit Unit Number	The Y origin of the scaled text in units.	0 to 65535
Data	CR-LF Terminated String	The text data to be printed.	Up to 8191 characters.

- **Sample Code**

1.26 Scale To Fit

- **Function**

  - The `SCALE-TO-FIT` command is used print scaled text which is to fit within a particular bounding box in CPCL. The command is similar in syntax to the `SCALE-TEXT` command, but the difference is that the sizing parameters XScale and YScale are now replaced with Width and Height. The font's vertical and horizontal size will be selected to fit within the box specified.
  - `SCALE-TO-FIT` can be used with TrueType or scaled fonts.
  - `VSCALE-TO-FIT` has the same functionality, but orients the text rotated 90 degrees.
  - Because `SCALE-TO-FIT` can only contain a single line of text, the height of the output text is always Height.

- **Swift (iOS)**

```
public func cpclScaleToFit(scaleFont: String, width: Int, height: Int, x: Int, y: Int, text: String)
```

- **Parameter**

Parameter	Type	Description	Valid Range
ScaledFontName	Space-Terminated String	A scaled font used to create the text.	See Below.
Width	5 Digit Unit Number	The width of the box to contain the text.	0 to 65535
Height	5 Digit Unit Number	The height of the box to contain the text.	0 to 65535
X	5 Digit Unit Number	The X origin of the scaled text in units.	0 to 65535
Y	5 Digit Unit Number	The Y origin of the scaled text in units.	0 to 65535
Data	CR-LF Terminated String	The text data to be printed.	Up to 8191 characters.

- **Sample Code**

1.27 Set Bold

- **Function**

> - `SET BOLD` is used to add a faux bolding effect to pre-scaled fonts in CPCL. It accomplishes this by redrawing the text one or more times, shifting one pixel to the right side of the page each time. The spacing of the characters does not change as a result of the bolding.
> - The number of shifts and writes is specified by the Boldness parameter.
> - The command affects any pre-scaled text drawing in CPCL, no matter what command it comes from.

- **Swift (iOS)**

```
public func cpclSetBold(boldness: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Boldness	3-digit Unit Number	Sets the boldness of the text.	0 to 999

- **Sample Code (Swift)**

```
cpclLineMode()
cpclSetBold(5)
cpclInsertTextLine("this text is in bold 5")

cpclLineMode()
cpclSetBold(0)
cpclInsertTextLine("this text is in bold 0")
```

1.28 Set Mag

- **Function**

  - `SETMAG` is used to set the output scaling of pre-scaled fonts, overriding the sizing behaviors defined in the font files themselves. When a non-zero value for SETMAG is used, that value replaces that element in the sizing of all pre-scaled fonts, regardless of what command is used to draw them.
  - SETMAG's value is preserved between labels, and also shared with line print mode.

- **Swift (iOS)**

```
public func cpclSetMag(width width: Int, height: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Width	3-digit Number	Sets the width multiplier of the font.	0 to 16
Height	3-digit Number	Sets the height multiplier of the font.	0 to 127

- **Sample Code**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 500, quantity: 1)
cpclText(rotate: 0, font: 0, fontSize: 0, x: 0, y: 0, text: "Text")
cpclText(rotate: 0, font: 0, fontSize: 1, x: 0, y: 30, text: "T90")
cpclText(rotate: 0, font: 0, fontSize: 2, x: 0, y: 90, text: "T180")
cpclText(rotate: 0, font: 0, fontSize: 3, x: 0, y: 120, text: "T270")

cpclSetMag(width: 2, height: 2)
cpclText(rotate: 0, font: 0, fontSize: 0, x: 0, y: 180, text: "text")
cpclText(rotate:
0, font: 0, fontSize: 1, x: 0, y: 250, text: "T90")
cpclText(rotate: 0, font: 0, fontSize: 2, x: 0, y: 330, text: "T180")
cpclText(rotate: 0, font: 0, fontSize: 3, x: 0, y: 400, text: "T270")
cpclPrint()
```

1.29 Set Spacing

- **Function**

  - `SETSP` is used to set the horizontal spacing between characters. The command adjusts the spacing of both pre-scaled and scaled fonts. The command can only increase the spacing between characters, it cannot decrease it.
  - If the spacing is large enough, this may cause the text to flow off the edge of the page. In this case, what happens to the extra text is a function of the underlying command used to print the text (i.e. wraps with TEXT, is truncated with `SCALE-TEXT`).
  - SETSP is not supported for TrueType fonts.
  - SETSP is reset at the start of each label to 0 for pre-scaled fonts. It is not reset for scaled fonts.

- **Swift (iOS)**

```
public func cpclSetSpacing(spacing: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Spacing	3-digit Unit Number	Sets the spacing between characters of a font	See Below.

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 300, quantity: 1)
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 10, text: "normal
spacing")
cpclSetSpacing(1)
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 50, text: "normal
spacing 1")
cpclSetSpacing(2)
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 80, text: "normal
spacing 2")
cpclSetSpacing(3)
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 110, text: "normal
spacing 3")
cpclSetSpacing(4)
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 140, text: "normal
spacing 4")
cpclSetSpacing(10)
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 170, text: "normal
spacing 10")
cpclPrint()
```

1.30 Temp Move

- **Function**

  - The `TEMP-MOVE` command moves the origin of the next or current label session. The effect is most pronounced when using a media that synchronizes to marks. It does affect continuous mode labels, but the effects may be difficult to see, particularly with the MoveUp parameter.
  - The TEMP-MOVE command is available in both label and utilities sessions, and only applies to the next or current label session. After the session ends, TEMP-MOVE is reset to zero.
  - The values specified by TEMP-MOVE are added to those specified by `MOVE` to determine the final position. Note that TEMP-MOVE and MOVE are both negative aware. This means that if a MOVE 20 20 was in effect, and a TEMP-MOVE -20 -20 was also in effect, the net location used would be 0,0.
  - Syntax and functionality wise, TEMP-MOVE is identical to MOVE.

- **Swift (iOS)**

```
public func cpclTempMove(right: Int, up: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
MoveRight	5 Digit Unit Number	How far to move the label to the right.	0 to 65535
MoveUp	5 Digit Unit Number	How far to move the label up.	0 to 65535

- **Sample Code**

1.31 Text

- **Function**

  - The `TEXT` command is used to print text in CPCL. The command can be used with both pre-scaled and TrueType fonts, but not with scalable fonts (use `SCALE-TEXT` for these instead). The TEXT command can also be used to print using font groups.
  - There are three syntaxes to the command. The first is for pre-scaled fonts, the second for TrueType fonts, and the third is for font group.
  - The alignment commands CENTER, LEFT and RIGHT are supported for all forms of the TEXT command.

- **Swift (iOS)**

```
// Pre-scaled Font Syntax
public func cpclText(rotate rotate: Int, font: Int, fontSize: Int, x: Int,
y: Int, text: String)
// Alternate Syntax -TTF Font Syntax
public func cpclText(rotate rotate: Int, trueTypeFont: Int, xScale: Int,
yScale: Int, x: Int, y: Int, text: String)
// Alternate Syntax -Font-Group
public func cpclText(rotate rotate: Int, fontGroup: Int, x: Int, y: Int,
text: String)
```

- **Parameter**

  - **Pre-scaled Font Syntax**

Parameter	Type	Description	Valid Range
FontNameOrNumber	Space-Terminated String	A font name or number to create the representation	See Below.
FontSize	5 Digit Number	The size of the font.	0 to 65535
X	5 Digit Unit Number	The X origin of the text in units.	0 to 65535
Y	5 Digit Unit Number	The Y origin of the text in units.	0 to 65535
Data	CR-LF Terminated String	The text data to be printed.	Up to 8191 characters

- **Alternate Syntax – TTF Font Syntax**

Parameter	Type	Description	Valid Range
TrueTypeFontName	Space-Terminated String	The filename of the TTF font with extension	See Below.
XScale	5 Digit Number	The X size of the font, in dots.	10 to 1450
YScale	5 Digit Number	The Y size of the font, in dots	10 to 1450
X	5 Digit Unit Number	The X origin of the text in units.	0 to 65535
Y	5 Digit Unit Number	The Y origin of the text in units.	0 to 65535
Data	CR-LF Terminated String	The text data to be printed.	Up to 8191 characters

- **Alternate Syntax – Font-Group**

Parameter	Type	Description	Valid Range
FontGroupNumber	Space-Terminated String	The number of the font-group to use	0 to 10
X	5 Digit Unit Number	The X origin of the text in units.	0 to 65535
Y	5 Digit Unit Number	The Y origin of the text in units.	0 to 65535
Data	CR-LF Terminated String	The text data to be printed.	Up to 8191 characters

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 200, quantity: 1)
cpclText(rotate: 0, font: 4, fontSize: 0, x: 200, y: 100, text: "Text")
cpclText(rotate: 90, font: 4, fontSize: 0, x: 200, y: 100, text: "T90")
cpclText(rotate: 180, font: 4, fontSize: 0, x: 200, y: 100, text: "T180")
cpclText(rotate: 270, font: 4, fontSize: 0, x: 200, y: 100, text: "T270")
cpclPrint()
```

2 Line Print Commands

2.1 Left Margin

- **Function**

> The `MARGIN` command is used to offset line print printing by a fixed amount. This command affects all data printed via line print text or any utilities command which results in printout.

- **Swift (iOS)**

```
public func cpclLineMargin(offset: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Offset	3 Digit Unit Number	Specifics the distance from the left edge in units	0 to 999

- **Sample Code**

2.2 LF equals CRLF (Line Print)

- **Function**

  - the `LP-LF-EQUALS-CRLF` command is used to tell the printer that when printing line print text, the LF character is equivalent to the two characters CR and LF.
  - This command only applies to line print text; it has no effect on any other part of the printer's operation.
  - The value is persistent until reboot or until changed. The default value is OFF.

- **Swift (iOS)**

```
public func cpclLinePrintSetLF(option: String)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Option	Space terminated string	Specifies the enable or disable of the function	ON or OFF

- **Sample Code**

2.3 Orient (Line Print)

- **Function**

  - The `LP-ORIENT` command is used to specify if line print operates in standard mode, or in rotated mode.
  - The implementation of LP-ORIENT is such that the data received is only reversed and rotated, and as such, to create readable text, significant modification of the format of the text is required before transmission to the printer to make it readable. See the example for an indication on how to send the data.There is only one rotation supported which is 270 degrees.
  - The only other supported option is 0, which is the power on default. The value specified is retained until it is changed or the printer's power is cycled.

> - Only line print text is affected by the LP-ORIENT command. When using LP-ORIENT 270, the LMARGIN and `SETLF` commands have no effect; the `MARGIN` is fixed at 0, and SETLF is forced to 10.
> - When using LP-ORIENT 270, the height of the page is determined by the number of lines transmitted.
> - LP-ORIENT 270 is not supported for TrueType fonts.

- **Swift (iOS)**

```
public func cpclLinePrintOrient(option: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Option	Space terminated string	Specifies the amount of rotation	0 or 270

- **Sample Code**

2.4 Line Print Position Adjust

- **Function**

> - RX, RY, and RXY are used in line print mode to specify that an element should be positioned relative to the line print text element printed. The command can be used at any time to adjust the line print cursor position, even in the middle of a string of line print characters.
> - This command only impacts line print text and the g command. It has no effect on labels or utilities-based line print commands.

- **Swift (iOS)**

```
public func cpclLinePrintAdjust(xPos xPos: Int)
public func cpclLinePrintAdjust(yPos yPos: Int)
public func cpclLinePrintAdjust(xPos xPos: Int, yPos: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
XValue	5 Digit Number	Specifies the value to use for the X position	-32767 to 32767
YValue	5 Digit Number	Specifies the value to use for the Y position	-32767 to 32767

- **Sample Code**

2.5 Set LF

- **Function**

  - `SETLF` is used in line print mode to set how much media is fed by the printer when the LF character is received by the printer.
  - At power on, this value is set to 10, which puts a space of 10 pixels between each row of line print printed characters. Note that this space is an addition to any space defined by the font itself, which is specified by the `SETLP` command on page 176. Any value set is persistent until reboot, except if the AUTOCAL command is run, which sets it to 0.
  - SETLF can perform an important function regarding the ability to print `PCX` images and barcodes in line print mode. When a barcode or PCX image is printed in line print, without any modifications, the maximum height of the output will be the value specified by SETLF (10 by default) plus the height value specified for SETLP (24 by default), or 34 pixels. Adjusting the SETLF value up allows a larger maximum height to be printed. See page 175 for an introduction to line print for more information.

- **Swift (iOS)**

```
public func cpclSetLF(height: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Height	5 digit Unit Number	Specifies the height fed when an LF is received	0 to 32767

- **Sample Code (Swift)**

```
cpclLineMode()
cpclSetLF(0)
cpclInsertTextLine("height of each line 0")
cpclInsertTextLine("text line")
cpclInsertTextLine("text line")
cpclInsertTextLine("text line")

cpclLineMode()
cpclSetLF(60)
cpclInsertTextLine("height of each line 60")
cpclInsertTextLine("text line")
cpclInsertTextLine("text line")
cpclInsertTextLine("text line")

cpclLineMode()
cpclSetLF(80)
cpclInsertTextLine("height of each line 80")
cpclInsertTextLine("text line")
cpclInsertTextLine("text line")
cpclInsertTextLine("text line")

cpclLineMode()
cpclSetLF(160)
cpclInsertTextLine("height of each line 160")
cpclInsertTextLine("text line")
cpclInsertTextLine("text line")
cpclInsertTextLine("text line")

cpclLineMode()
cpclSetLP(font: 4, fontSize: 0, lineSpacing: 80)
cpclSetLF(40)
cpclPrint()
cpclInsertTextLine("height of each line 40")
cpclInsertTextLine("text line")
cpclInsertTextLine("text line")
cpclInsertTextLine("text line")
```

2.6 Set LP

- **Function**

  > - `SETLP` is used to set the name or number and size of the font which is to be used for printing line print text. The font can either be a pre-scaled font or a TrueType

font.
    - The default values for SETLP use the pre-scaled font syntax, and specify font number 7, size 0, and a line spacing of 24.

- **Swift (iOS)**

```
// Pre-Scaled Font Syntax
public func cpclSetLP(font font: Int, fontSize: Int, lineSpacing: Int)
// Alternate Syntax - TTF Font Syntax
public func cpclSetLP(trueTypeFont font: String, xScale: Int, yScale: Int,
lineSpacing: Int)
```

- **Parameter**
    - **Pre-scaled Font Syntax**

Parameter	Type	Description	Valid Range
FontNameOrNumber	Space-Terminated String	A font name or number to create the representation	See Below.
FontSize	3 Digit Number	The size of the font.	0 to 65535
LineSpacing	5 Digit Unit Number	The amount of space to place between lines in units.	0 to 65535

    - **Alternate Syntax – TTF Font Syntax**

Parameter	Type	Description	Valid Range
TrueTypeFontName	Space-Terminated String	The filename of the TTF font with extension	See Below.
XScale	5 Digit Unit Number	The X size of the font, in units.	10 to 1450 dots
YScale	5 Digit Unit Number	The Y size of the font, in units.	10 to 1450 dots
LineSpacing	5 Digit Unit Number	The amount of space to place between lines in units.	0 to 65535

- **Sample Code (Swift)**

```
cpclLineMode()
cpclSetLP(font: 4, fontSize: 2, lineSpacing: 46)
cpclInsertTextLine("font 4 size 2 vertical spacing 46")

cpclLineMode()
cpclSetLP(font: 1, fontSize: 2, lineSpacing: 46)
cpclInsertTextLine("font 1 size 2 vertical spacing 46")

cpclLineMode()
cpclSetLP(font: 1, fontSize: 0, lineSpacing: 46)
cpclInsertTextLine("font 1 size 0 vertical spacing 46")

cpclLineMode()
cpclSetLP(font: 1, fontSize: 2, lineSpacing: 92)
cpclInsertTextLine("font 1 size 2 vertical spacing 92")

cpclLineMode()
cpclSetLP(font: 4, fontSize: 0, lineSpacing: 40)
cpclInsertTextLine("font 4 size 0 vertical spacing 40")
```

2.7 Set LP Buffer

- **Function**

- `SETLP-BUFFER` is used to set the height of the line print buffer. This buffer represents the maximum area that can be written to in a single line print transaction.
- Typically the size of this buffer does not require adjustment because Link-OS CPCL will seamlessly join line print buffers while printing normal text or line print graphics.
- There are a small number of cases for which this command is needed however, such as printing a `PCX` or Barcode longer than the default SETLP-BUFFER size.
- The default buffer at power on is 2400 dots. Any value set will remain active until it is changed or until reboot.

- **Swift (iOS)**

```
public func cpclSetLPBuffer(height: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Height	5 digit Unit Number	Specifies the size of the line print buffer in units.	0 to 2400 dots

- **Sample Code**

2.8 Set LP Timeout

- **Function**

- `SETLP-TIMEOUT` sets how long the printer waits for additional line print text data before forcing termination and printing of un-terminated data.
- Normally lines in line print text are terminated with a CR, an LF or a CR and LF character. If data arrives which is not terminated with a CR or LF, this timeout specifies how long the printer will wait for more data before terminating the line and printing the data.
- Each time a character destined for line print is received, the timer is reset. In addition, the timer is reset when a line print barcode or `PCX` command is processed.
- The default value at power on is 4 (representing 500ms). The value is persistent until power cycle or changed with this command. Setting this command to a value of 0 disables the line print timeout, and characters will be held forever until a terminator is received.

- **Swift (iOS)**

```
public func cpclSetLPTimeout(timeout: Int)
```

- **Parameter**

Parameter	Type	Description	Range
Timeout	5-digit Number	Specifies line print timeout in 1/8th of a second increments	0 to 99999

- **Sample Code**

2.9 Set Position

- **Function**

  - `X`, `Y`, and `XY` are used in line print mode to specify that an element should be positioned. The command can be used at any time to adjust the line print cursor position, even in the middle of a string of line print characters.
  - This command only impacts line print text and the g command. It has no effect on labels or utilities-based line print commands.

- **Swift (iOS)**

```
public func cpclSetPosition(xPos xPos: Int, yPos: Int)
public func cpclSetPosition(xPos xPos: Int)
public func cpclSetPosition(yPos yPos: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
XValue	5 Digit Number	Specifies the value to use for the X position	-32767 to 32767
YValue	5 Digit Number	Specifies the value to use for the Y position	-32767 to 32767

- **Sample Code (Swift)**

```
cpclLineMode()
cpclSetPosition(xPos: 0, yPos: 0)
cpclInsertTextLine("XY")

cpclLineMode()
cpclSetPosition(xPos: 50, yPos: 50)
cpclInsertTextLine("XY")

cpclLineMode()
cpclSetPosition(xPos: 150, yPos: 150)
cpclInsertTextLine("XY")

cpclLineMode()
cpclSetPosition(xPos: 200, yPos: 200)
cpclInsertTextLine("XY")
```

2.10 Line Feed

- **Function**

  > - is a line print character which is used to advance the cursor along the Y axis while not resetting the X axis.
  > - The height advanced in the Y direction is specified by the SETLF command. By default, the value is 10,

- **Swift (iOS)**

```
public func cpclLineFeed()
```

- **Sample Code**

2.11 Carriage Return

- **Function**

  > is a line print character which is used to advance the cursor along the Y axis and also reset the X axis to 0.

- **Swift (iOS)**

```
public func cpclCarriageReturn()
```

- **Sample Code**

2.12 Line Print Graphics

- **Function**

  > - The g command is used to print graphics in line print mode.
  > - Each g command is used to print a single pixel line of graphic data. The width of the line is specified in the first two bytes following the g command as a 16-bit integer. All current printers with CPCL only require a single byte to represent maximum width, though both bytes must be specified.
  > - The design intent of the g command is that the SETLP and SETLF commands first be used to configure the total line height to 1 pixel, since each g command only prints one line at a time. g by itself will not affect any print out; the or commands must be used at the end of the line to terminate it. Generally this involves setting the LF height to 1 with the SETLF command, and setting the font height to 0 with the SETLP command. See the example below for more information.
  > - The g command positions its output at the current line print cursor location. The LMARGIN, RX, RY, RXY, X, Y and XY commands can all be used to control the position of the line print cursor. The LMARGIN command is often used to avoid having to specify large blocks of blank space at the start of lines. The RY command can be used to easily skip areas of blank paper without having to send large amounts of NUL data.

- **Swift (iOS)**

```
public func cpclLinePrintGraphics(byteWidth byteWidth: Int, bitmapData:
NSData)
```

- **Parameter**

Parameter	Type	Description	Valid Range
ByteWidth	2 Byte Binary Number	The byte width of the data for this dot line	0 to 104 in practice
Data	Raw String	The data that makes up the bitmap to be printed.	See below.

- **Sample Code**

3 Font Commands

3.1 File Header

- **Function**

> - CPF and ECPF fonts both begin with a human readable file header, followed by binary data. The header contains identification information and the size table mentioned above. All CPF fonts begin with the text CISBF. This identifies the file as a font.
> - At startup, all files on the printer are checked for this header, and if it is found, the printer attempts to load them as CPF fonts.
> - The items beginning with a dash may appear in any order in the font file (although the `–END-FONT-INF` must be at the end)

- **Swift (iOS)**

```
public func cpclFileHeader(description description: Int, fontNumber: Int,
sizeOut: Int)
public func cpclFileHeaderDefineScalingFactor(heightMult heightMult:Int,
widthMult: Int, pageOffset: Int)
public func cpclEndOfHeaderDesignation()
```

- **Parameter**

Field Name	Description	Type	Valid Range
Description	Description of the font file.	Terminated String	Up to 30 characters
FontNumber	The font number, optional.	Terminated String	8 to 63
SizeCount	The number of entries in the sizes table.	Terminated String	1 to 255
HeightMult	The height scaling factor of this size.	Space Terminated String	0 to 65535
WidthMult	The width scaling factor of this size.	Space or CRLF Terminated	0 to 65535
PageOffset	The address location of the bitmaps	4 binary bytes (uint32)	All values valid.

- **Sample Code**

3.2 Char Set And Country

- **Function**

> - This command is used to set the encoding for CPF, CSF and TTF fonts in CPCL. For more information about the various internal supported options, see the previous section.

> - Besides the list of internally supported encodings, custom encodings from ZPL can also be used in CPCL by using the filename in place of the Name option above. These files are referred to by the filename, without their extension, which must be .DAT. See the ZPL documentation for more information on custom character mappings.

- **Swift (iOS)**

```
public func cpclCountry(country: String)
```

- **Parameter**

Name	Encoding Type
BIG5	BIG5 Encoding. TTF maps to UTF points, multi byte.
CHINA	EUC-CN Encoding, TTF maps to UTF points, multi byte.
CP850	USA with substitutions, single byte. Box and graphics chars not supported.
CP874	Font defines encoding for CPF, TTF maps to UTF points. , single byte.
FRANCE	USA with 7-bit substitutions, single byte.
GERMANY	USA with 7-bit substitutions, single byte.
ITALY	USA with 7-bit substitutions, single byte.
JAPAN	GBK Encoding, TTF maps to UTF points, multi byte.
JAPAN-S	GBK Encoding. TTF maps to UTF points, multi byte.
KOREA	GBK Encoding, TTF maps to UTF points, multi byte.
LATIN9	USA with substitutions, single byte.
NORWAY	USA with 7-bit substitutions, single byte.
SPAIN	USA with 7-bit substitutions, single byte.
SWEDEN	USA with 7-bit substitutions, single byte.
THAI	Thai multi-byte encoding. Superseded by CP874.
UK	USA with 7-bit substitutions, single byte.
USA	Font defines encoding, single byte.
VIETNAM	Font defines ECPF encoding, TTF maps to UTF points, multi byte.

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 250, quantity: 1)
cpclCenter()
cpclCountry("CHINA")
cpclText(rotate: 0, font: 5, fontSize: 0, x: 10, y: 10, text: "chinese
traditional sample")
cpclSetMag(width: 1, height: 1)
cpclText(rotate: 0, font: 9, fontSize: 0, x: 10, y: 50, text: "Print")
cpclText(rotate: 0, font: 8, fontSize: 1, x: 10, y: 100, text: "Print")
cpclText(rotate: 0, font: 8, fontSize: 3, x: 10, y: 150, text: "Print")
cpclPrint()
```

4 Media Management Commands

4.1 Auto Cal

- **Function**

  - The `AUTODIAL` command is used to calibrate the detection thresholds of the printer based on the currently selected media type.
  - By default, the printer is preset to calibration thresholds of 70 for mark\bar media and 50 for gap media. These values are selected to be compatible with a wide range of medias, however if the media is unusual performance may be improved through this calibration process. The current values are viewable on the two-key report . The command uses two labels worth of media to perform the calibration.

- **Swift (iOS)**

```
public func cpclAutoCal()
```

- **Sample Code**

4.2 Auto Pace

- **Function**

  - For printers which have the peeler accessory, the `AUTO-PACE` activates the use of the peeler sensor, and will cause the printer to print one label, and then wait for its removal before printing the next label, which can be part of a batch, or can be a single quantity label. The batch is defined by the quantity parameter which is part of the label session definition. See page 18 for more information on the label session.
  - No other printing operations will occur while the printer is waiting for the label to be

> taken, including line print or prints from other languages.
> - On some models, the peeler sensor must be physically activated before it can be used, and not all models have the peeler accessory. See your user manual for more information. On printers which do not have the peeler accessory, AUTO-PACE acts the same as the PACE command.

- **Swift (iOS)**

```
public func cpclAutoPace(delay: Int)
public func cpclAutoPace()
```

- **Parameter**

Parameter	Type	Description	Valid Range
Delay	5 Digit Unit Number	How long to delay in 0.125 second increments	0 to 65535

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclAnnotation("tell printer to print a label")
cpclAnnotation("after each 'feed' key press")
cpclAnnotation("until all 3 labels are printed")
cpclAutoPace()
cpclAnnotation("center the text")
cpclCenter()

cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 10, text: "printer 3
labels")
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 90, text: "using PACE")

cpclPrint()
```

4.3 Bar Sense

- **Function**

> - The `BAR-SENSE` command is used to configure the system to use the reflective sensor for detecting media and synchronization marks. This is the recommended setting for media with marks, and for continuous media. This command is the opposite of `GAP-SENSE`.

- BAR-SENSE takes an optional parameter which is used to set the detection threshold.
- Once set, either in a label or utilities session, it is persistent for all subsequent labels until power off. The setting (as well as the threshold) can be made permanent using the zpl.save SGD. See the examples for more information.
- The setting is shared with the ZPL setting specified by ^MN, and is equivalent to ^MNM.

- **Swift (iOS)**

```
public func cpclBarSense(threshold: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Threshold	5 Digit Unit Number	Media detection threshold for bar media	0 to 255

- **Sample Code**

4.4 Contrast

- **Function**

  - The `CONTAST` command is a legacy form of print darkness adjustment which gives a coarse adjustment to the darkness of printed labels. Setting CONTRAST temporarily sets the print.contrast SGD to the value provided.
  - The `TONE` command provides significantly more granularity for configuring the darkness of printed labels.
  - TONE and CONTRAST are unique settings in CPCL, and are not interchangeable.
  - The value set for TONE overrides CONTRAST if the value of TONE is not zero. This includes if the SGD print.tone is set to a value other than zero, even during if no CPCL TONE command has been issued.
  - In order for CONTRAST to have any effect, TONE must be zero.
  - The default value for CONTRAST is 0, but at power on the current CONTRAST value is set to the value of the print.contrast SGD.

- **Swift (iOS)**

```
public func cpclContrast(value: Int)
```

- **Parameter**

CONTRAST value	TONE Value	~SD Value
0	0	10
1	100	20
2	200	30
3	200	30

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclContrast(0)
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 0, text: "hello world")
cpclContrast(1)
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 0, text: "hello world")
cpclContrast(2)
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 0, text: "hello world")
cpclContrast(3)
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 0, text: "hello world")
cpclPrint()
```

4.5 Feed

- **Function**

  - The FEED command is used to move media the specified amount as soon as the command is processed.
  - FEED can be used with both negative and positive numbers. Note that there is no mechanism that prevents you from feeding an excessively large negative value causing the printer to lose control of the media.
  - The FEED operation will ignore all gap or marks on the paper but will terminate if the printer detects an out of media condition or other error.

- **Swift (iOS)**

```
public func cpclFeed(amount: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Amount	5 Digit Unit Number	How much to feed in units.	-4000 to 4000 dots

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 150, quantity: 1)

cpclSpeed(5)
cpclText(rotate: 0, font: 5, fontSize: 0, x: 0, y: 20, text: "print at
spped 5")
cpclPrint()

cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclSpeed(4)
cpclText(rotate: 0, font: 5, fontSize: 0, x: 0, y: 20, text: "print at
spped 4")
cpclPrint()

cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclSpeed(3)
cpclText(rotate: 0, font: 5, fontSize: 0, x: 0, y: 20, text: "print at
spped 3")
cpclPrint()

cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclSpeed(2)
cpclText(rotate: 0, font: 5, fontSize: 0, x: 0, y: 20, text: "print at
spped 2")
cpclPrint()
```

4.6 Form

- **Function**

  - The `FORM` command, when used in a label session, signals that after the label session is close with `PRINT` or `END`, and is done printing, the printer should attempt to synchronize to a mark or gap on the media after the label is printed, taking into account all adjustments (the TOF value from `SET-TOF` and label skip from `SETFF`).
  - The command may appear anywhere in the label session but always applies at the end. The command is not persistent and must appear in each label session in order

to take effect.
    - The printer will search for the distance specified by the `SETFF` command (or by the media.feed_length SGD) for the mark before giving up. No error occurs if the printer cannot find the mark.
    - This differs from the command when used in a utilities session, which is covered in the next section.

- **Swift (iOS)**

```
public func cpclForm()
```

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclCenter()
cpclText(rotate: 0, font: 4, fontSize: 0, x: 10, y: 0, text: "This is a
FORM command.")
cpclForm()
cpclPrint()
```

4.7 Gap Sense

- **Function**

    - The GAP-SENSE command is used to configure the system to use the transmissive sensor for detecting media and synchronization marks. This is the recommended setting for media with gaps or notches. This command is the opposite of BAR-SENSE.
    - GAP-SENSE takes an optional parameter which is used to set the detection threshold.
    - Once set, either in a label or utilities session, it is persistent for all subsequent labels until power off. The setting (as well as the threshold) can be made permanent using the zpl.save SGD. See the examples for more information.

- **Swift (iOS)**

```
public func cpclGapSense(threshold: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Threshold	5 Digit Unit Number	Media detection threshold for gap media	0 to 255

- **Sample Code**

4.8 Journal

- **Function**

> The `JOURNAL` command (and the associated LABEL command) determine which of the two modes of operation are used when a mark or gap is encountered while a label is being printed after a label session is completed.

- **Swift (iOS)**

```
public func cpclJournal()
```

- **Sample Code**

4.9 Label

- **Function**

> The `LABEL` command (and the associated `JOURNAL` command) determine which of the two modes of operation are used when a mark or gap is encountered while a label is being printed after a label session is completed.

- **Swift (iOS)**

```
public func cpclLabel()
```

- **Sample Code**

4.10 Multi

- **Function**

> - `MUTLI` is used to set the number of labels which print horizontally across the page. The horizontal labels are exact duplicates across the page; it is not possible to specify unique content for each horizontal label.
> - When MULTI is active with a value of 2 or 3, the maximum print width is divided by the factor specified, and this becomes the new maximum value for the `PAGE-WIDTH` command.

- Similar to `PAGE-WIDTH`, MULTI can affect the size of the drawing canvas when activated. For that reason, it is recommended to place the `MULTI` command as the first item in the label session, before the PAGE- WIDTH or any drawing commands. Value of 2 and 3 are not supported on Epoch printers.
- If used in a utilities session, the value specified applies only to line print, and is persistent. If used in a label session, the setting only applies to the current label being printed, and is not persistent.
- The `MULTI` command has no effect in line print if the text being printed is more than one line long.

- **Swift (iOS)**

```
public func cpclMulti(quantity: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Quantity	5 Digit Unit Number	How many labels to print across the page	1 to 3

- **Sample Code**

4.11 No Pace

- **Function**

- `NO-PACE` terminates the use of `PACE` or `AUTO-PACE` and disables any pause operation that occurs between prints. `NO-PACE` is the default state for the printer at power on.
- The command is always persistent when used. If used in a label session, it takes effect on the label that will be printed when the session ends and any subsequent sessions.

- **Swift (iOS)**

```
public func cpclNoPace()
```

- **Sample Code**

4.12 Out Of Paper

- **Function**

> - `ON-OUT-OF-PAPER` is used to specify what happens when the printer runs out of paper, or encounters an unexpected mark or gap while printing a label in LABEL mode.
> - When the printer is used in LABEL mode, the printed area of the label should not encounter a mark or gap. Instead, the FORM command should be used after the label content is printed to synchronize to the end of the form. The label area defined in the label session should be 32 dots less than the actual size of the label to ensure proper functionality. If the printer detects a mark or gap while printing the label (not while using the `FORM` command), this command specifies what happens.
> - In addition, the command provides two separate configuration items regarding if the configuration applies to line print mode, and also specify an optional file to run when the end of form situation occurs unexpectedly.
> - The default values for this command is PURGE 1, and `PURGE-LP OFF`, and RUN mode is disabled.
> - When RUN mode is activated, PURGE and WAIT are disabled, as is the Retries parameter. The retries is forced to 1 when RUN mode is being used. Newly received labels are handled as if the printer is in WAIT mode.

- **Swift (iOS)**

```
public func cpclOneOutOfPaperMode(mode: String, retries: Int, state:
String, filename: String, unused: Int)
```

- **Parameter**
  - **Standard Syntax**

Parameter	Type	Description	Valid Range
Mode	Space-Terminated String	A font name or number to create the representation	PURGE or WAIT
Retries	5 Digit Number	The number of attempts to make before aborting.	0 to 65535

  - **PURGE-LP Syntax**

Parameter	Type	Description	Valid Range
State	Space-Terminated String	Specifies the state for line print purging.	ON or OFF

- **RUN Syntax**

Parameter	Type	Description	Valid Range
Filename	Space-Terminated String	The filename to run when an error occurs.	Any valid filename.
Unused	5 Digit Unit Number	An unused parameter that must be specified.	0 to 65535

- **Sample Code**

4.13 Pace

- **Function**

  > - `PACE` is used with label sessions with a print quantity greater than one to pause between each label. If such a label session is initiated, the printer will print one label, then stop, and wait for the feed key to be pressed before printing the next label. This behavior continues until all labels in the batch are printed. After the final label is printed, normal operation is resumed.
  > - No other printing operations will occur while the printer is printing, including line print or prints from other languages.
  > - The command disables `AUTO-PACE` if it is in effect.The command is always persistent when used. If used in a label session, it takes effect on the label that will be printed when the session ends and any subsequent sessions.

- **Swift (iOS)**

```
public func cpclPace()
```

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclAnnotation("tell printer to print a label")
cpclAnnotation("after each 'feed' key press")
cpclAnnotation("until all 3 labels are printed")
cpclPace()
cpclAnnotation("center the text")
cpclCenter()

cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 10, text: "printer 3
labels")
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 90, text: "using PACE")
cpclForm()

cpclPrint()
```

4.14 Paper Jam

- **Function**

  - The `PAPER-JAM` command specifies how a paper-jam condition is detected, and what happens when it occurs.
  - The function can use either the media sensor and check to see if a mark or gap occurs to find if there has been a paper jam, or can use the peeler sensor to detect if no label is present after a label session is sent for printing.
  - When using any mode besides `NONE`, if a paper jam is detected, an error will occur requiring the user to open the head, remove the jam, and then close the head before printing will continue.
  - The command can also return an optional message to the host when a jam occurs, see the `ALERT` parameter.
  - The command is persistent once set either in a label or in a utilities session.

- **Swift (iOS)**

```
public func cpclPaperjamMode(mode: String, unused: String, message: String)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Mode	Space Terminated String	Sensor to use for detecting jam condition	See Below
Unused	5 Digit Number	Required unused parameter for some modes.	See Below
Message	Quoted String	Message to be sent to host when using ALERT.	Up to 100 characters.

- **mode**

Mode	Description	Requires Unused
NONE	Disable all paper-jam detection (default)	No
PRESENTATION	Use peeler sensor to detect paper jams.	No
INDEX	Use mark or gap sensor to detect paper jams.	No
BAR	Use mark or gap sensor to detect paper jams.	Yes
GAP	Use mark or gap sensor to detect paper jams.	Yes

- **Sample Code**

4.16 PostFeed/PreFeed

- **Function**

> - The `POSTFEED/PREFEED` command is used to perform an additional media movement before a label or line print section is printed.
> - `POSTFEED/PREFEED` can be used with both negative and positive numbers. Note that there is no mechanism that prevents you from feeding an excessively large negative value causing the printer to lose control of the media.
> - Using the `POSTFEED/PREFEED` command in a utilities session will cause it to be persistent until power off, and will also cause the setting to affect line print. If used in a label session, it will only apply to the label session in which it is used.
> - The `POSTFEED/PREFEED` operation will ignore all gap or marks on the paper but will terminate if the printer detects an out of media or other error condition.

- **Swift (iOS)**

```
public func cpclPostFeed(amount: Int)
public func cpclPreFeed(amount: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Amount	5 Digit Unit Number	How much to feed in units.	-4000 to 4000 dots

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclPreFeed(40)
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 20, text: "prefeed
example")
cpclPrint()
```

4.17 Present AT

- **Function**

  - When enabled, the `PRESENT-AT` command causes the printer to feed an additional amount once a label or line print section has been printed, and then when the next feed operation of any kind occurs, the motion is undone via an equal reverse feed. This allows the print to be pushed out to be at the correct tear off position, but still have the next print be registered to the start of the print operation.
  - `PRESENT-AT` can be used in either a utilities session or a label session. The effect is persistent until power cycle and in both cases takes effect as soon as the next print operation completes.
  - To disable `PRESENT-AT`, set Amount and Delay to 0. These are also the values used at power on.If PRESENT-AT was being used, and was then disabled, the printer will still properly reverse the last PRESENT-AT operation to ensure proper registration.Typically,
  - PRESENT-AT values are small, usually up to about 40. Using longer values may cause registration issues as the media drifts left to right as it is reversed.

- **Swift (iOS)**

```
public func cpclPresentAtWithAmount(amount: Int, delay: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Amount	5 Digit Unit Number	How much to feed in units.	0 to 20000 dots
Delay	5 Digit Number	How long to delay before feeding in 1/8th of seconds	0 to 240

- **Sample Code**

4.18 Reverse

- **Function**

  - The `REVERSE` command, when used in a label session, is used to perform an additional media movement before a label is printed.
  - Its function is identical to that of `PREFEED` when used with a negative value, although reverse only supports positive values.
  - Note that there is no mechanism that prevents you from reversing an excessively large value causing the printer to lose control of the media.
  - The `REVERSE` operation will ignore all gap or marks on the paper but will terminate if the printer detects an out of media or other error condition.
  - Once set, the setting is persistent until power cycle.
  - The label session version of `REVERSE` also affects line print, but there is no way to change the setting except in a label session due to the different way REVERSE operates in utilities sessions.

- **Swift (iOS)**

```
public func cpclReverse(amount: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Amount	5 Digit Unit Number	How much to feed in units.	0 to 4000 dots

- **Sample Code**

4.19 Set Feed Length and Skip (Set FF)

- **Function**

- `SETFF` is used to configure the length of a feed operation, and the advance amount after a mark or gap ends before the next label begins.
- The values set with this command are persistent until power off regardless of whether set from a label or utilities session. They can be made permanent by using the zpl.save SGD. See the examples below.

- **Swift (iOS)**

```
public func cpclSetFeed(length length: Int, skip: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
FeedLength	5 Digit Unit Number	How long is a form feed operation in units?	0 to 20000 dots
FeedSkip	5 Digit Unit Number	How long to move after a mark or gap ends?	5 to 50 dots

- **Sample Code**

4.20 Set TOF

- **Function**

  - The `SET-TO` F command is used to adjust where the sensing of a mark or gap occurs relative to the print line.
  - With the default value of 0, the printer positions the trailing edge of the mark or gap at the print line, so that when the next dot row is printed, it will be placed just after the mark or gap. All space up to this point is part of the previous label, and similarly, the same space at the end of the label is included in this label's space.
  - Adjusting this value to be a positive number moves the resting position of the end of the mark or gap to be closer to the source roll of media (inside the printer body). Adjusting the value to be a negative number moves the resting position of the end of the mark or gap to be further from the print line outside the printer. The feed skip value specified by `SETF` F is always applied after this positioning operation.If used in a utilities session, the setting takes effect on the FORM operation.
  - If used in a label session, the setting takes effect on that label session. In both cases, the setting is persistent until power cycle. The setting can be made permanent using the zpl.save SGD.

- **Swift (iOS)**

```
public func cpclSetTOF(amount: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Amount	5 Digit Unit Number	How much to feed in units.	-400 to 400 dots

- **Sample Code**

4.21 Speed

- **Function**

  - The `SPEED` command is used to set the maximum speed at which printout occurs.When used in a label session, the `SPEED` command takes effect on the label.
  - When used as a utilities command, it takes effect immediately for all subsequent printouts.
  - The default speed is 3. Note that print speed in CPCL is not in units, but rather is a scale from 0 to 5. Higher values indicate faster printing, and lower values indicate slower printing. In addition, print speed is not absolute in CPCL. The printer may print slower than the value selected based on factors such as print head temperature or battery levels.
  - This command sets the SGD media.speed, but does not set it to the number specified by Value, instead the number specified is converted to inches per second (which is what media.speed is represented in) and set. The actual value media.speed is set to varies from printer model to printer model based on the capabilities of the printer.
  - Using the `SPEED` command causes the speed to be set permanently, persisting not only though label and utilities sessions, but through reboots as well.

- **Swift (iOS)**

```
public func cpclSpeed(value: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Value	1 Digit Number	Maximum speed to print in an arbitrary scale	0 to 5

- **Sample Code**

4.22 Tone

- **Function**

  > - The `TONE` command is used to set the darkness of the printout from the printer. It provides the same kind of adjustment ability as the `CONTRAST` command, but with more fine control.
  > - The range for `TONE` is -100 to 200, and the default is 0. Higher values are more dark and lower values are less dark. The TONE and `CONTRAST` command both take precedence over `SPEED`, and the printout will be slowed as necessary to reach the desired darkness value.
  > - When the value is set in a label or utilities session, the value becomes permanent, similar to `SPEED`.
  > - This command affects the print.tone SGD, but due to print.tone's interaction with the print.tone_format SGD, the print.tone value may be in a different format than that of the `TONE` command. If the print.tone_format SGD is set to CPCL, the value of the print.tone SGD exactly mirrors that of the TONE command. If the print.tone_format SGD is set to ZPL, the value specified by the TONE command will be mapped to a 0.0 to 30.0 value system used by ZPL's ~SD command. The converted darkness is identical to the original TONE value.
  > - The `CONTRAST` command, which predated the TONE command, provides more rough control over the darkness level of the printout. When the TONE command is set to a non-zero value, the `CONTRAST` command (and the print.contrast SGD) are ignored. When TONE is set to zero, if `CONTRAST` is non-zero, it will be used. Note that unlike `TONE`, `CONTRAST` does not save its value permanently when set.

- **Swift (iOS)**

```
public func cpclTone(value: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Value	5 Digit Number	The relative darkness of the printout.	-100 to 200

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclTone(-99)
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 0, text: "hello world")
cpclPrint()

cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclTone(0)
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 0, text: "hello world")
cpclPrint()

cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclTone(100)
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 0, text: "hello world")
cpclPrint()

cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclTone(200)
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 0, text: "hello world")
cpclPrint()
```

4.23 Turn

- **Function**

  > - `TURN` changes the orientation of the printed label or of line print text, specifying either 0 degree rotation (top out first), or 180 degree rotation (bottom out first). When using this feature with line print mode, each line print buffer is individually rotated, which may cause some unexpected issues.
  > - Once set via a label or utilities session, the command is persistent until power cycle. The value specified in a utilities or in a label session applies to both session types.
  > - Only the image portion of the label is rotated with this command; any media alignment commands function as expected.

- **Swift (iOS)**

```
public func cpclTurn(degrees: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Degrees	5 Digit Number	The orientation of the label.	0 or 180

- **Sample Code**

4.24 Form Feed

- **Function**

  > - is used to simulate pressing of the form feed key on the printer when the printer is not in a session. This causes the printer to immediately attempt to synchronize to a mark or gap on the media, taking into account all adjustments (the TOF value from SET-TOF and label skip from SETFF).
  > - The printer will search for the distance specified by the SETFF command (or by the media.feed_length SGD) for the mark before giving up. No error occurs if the printer cannot find the mark.
  > - The command is very similar to the utilities FORM command but has an important difference. The FF command will cause the printer to execute the feed action as defined with the ON-FEED command. If this command is set to IGNORE, FF will have no function. If set to REPRINT, FF will reprint the last printed label session. By default, the printer is set to ON-FEED FEED, so the command will cause a feed operation.
  > - Note that the FF.BAT file activated by physically pressing the feed key is not connected to the FF command, the bat file will not run when FF is received by the printer.

- **Swift (iOS)**

```
public func cpclFormFeed()
```

- **Sample Code**

5 Status Enquiry Commands

5.1 Name

- **Function**

  > `NAME` returns the null-terminated name of the application currently running on the printer. In Link-OS printers, this is the same as the appl.name SGD.

- **Swift (iOS)**

```
public func cpclName()
```

- **Sample Code**

5.2 Version

- **Function**

  > `VERSION` returns a four-byte null-terminated representation of the version number of the version currently running on the printer. In Link-OS printers, this is the same as the appl.version SGD.

- **Swift (iOS)**

```
public func cpclVersion()
```

- **Sample Code (Swift)**

```
cpclVersion()
cpclPrint()
```

5.3 Printer Status

- **Function**

  > - h is an escape command which is used to determine printer status.
  > - This command is an escape command, and is not valid within sessions. It can only be used when the printer is not currently in a session.
  > - The command returns a single byte, which is a bit field indicating various status attributes. To get the status of a particular field, mask off the unused bits.

- **Swift (iOS)**

```
public func cpclPrinterStatus()
```

- **Sample Code**

5.4 Extended Printer Status

- **Function**

  > - i is an escape command which is used to determine additional printer status.
  > - This command is an escape command, and is not valid within sessions. It can only be used when the printer is not currently in a session.
  > - The command returns a single byte, which is a bit field indicating various status attributes. To get the status of a particular field, mask off the unused bits.

- **Swift (iOS)**

```
public func cpclExtendedPrinterStatus()
```

- **Sample Code**

5.5 Get Version Information

- **Function**

  - v is an escape command which is used to obtain a string with printer information.
  - This command is an escape command, and is not valid within sessions. It can only be used when the printer is not currently in a session.
  - The response of the command is a null terminated string of variable length which contains the product name, firmware version, compile date, CRC, and the device's serial number.

- **Swift (iOS)**

```
public func cpclGetVersionInfomation()
```

- **Sample Code**

6 Utility and Diagnostic Commands

6.1 Abort

- **Function**

  `ABORT` terminates a label session in progress without performing any printing. Any label data received so far as part of this session is lost, except if `PERSIST` is enabled

- **Swift (iOS)**

```
public func cpclAbort()
```

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclPageWidth(240)
cpclBox(xPos: 0, yPos: 0, xEnd: 200, yEnd: 200, thickness: 10)
cpclBox(xPos: 50, yPos: 50, xEnd: 220, yEnd: 220, thickness: 10)
cpclAbort() // 终止打印两个矩形
cpclPrint()
```

6.2 Baud

- **Function**

  > - `BAUD` sets the current baud rate of the serial port. When executed, it permanently sets the comm.baud SGD. When the command is issued, it takes effect immediately and is persistent through reboots.
  > - The command has no effect on printers which do not have serial ports. The default value of `BAUD` depends on the particular printer.

- **Swift (iOS)**

```
public func cpclBaud(value: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Value	5 Digit Number	The baud rate of the serial port.	See below.

  - value

    > Specifies the baud rate to set the serial communications port to. Valid values are 115200, 57600, 38400, 19200, 4800, 2400, and 1200. Not all printers support all baud rates. If the baud rate isn't supported on the current printer or is invalid, the BAUD command has no effect and the baud rate remains unchanged.

- **Sample Code**

6.3 Beep

- **Function**

  > `BEEP` causes the printer to produce an audible beep. The length of the beep is specified in 1/8th of a second increments. While the beep is active other tasks such as printing may be paused.

- **Swift (iOS)**

```
public func cpclBeep(duration: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Duration	5 Digit Number	The length of the beep in 1/8ths of a second	0 to 65535

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 210, quantity: 1)
cpclBeep(32)
cpclPrint()
```

6.4 Capture

- **Function**

  - `CAPTURE` provides a tool that can be used to capture data that the printer receives to a file on the printer, and can also optionally be used to put the printer into a dump mode which prints the characters received rather than interpreting them.
  - The `CAPTURE` command directly sets the input.capture SGD until reboot.

- **Swift (iOS)**

```
public func cpclCapture(mode: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Mode	CR-LF Terminated String	The capture mode to enable	PRINT, RUN, OFF

- **Sample Code**

6.5 Check Sum/Check Sum Vertical

- **Function**

  - `CHECKSUM` returns a four-byte, null-terminated, pre-calculated checksum of the firmware image on the printer.
  - In earlier printers, this command calculated the checksum of the firmware and returned it. Because it takes an extended amount of time to calculate the checksum, this command returns a fixed checksum for compatibility, and the new `CHECKSUM`

command is used to verify the checksum.
    - `CHECKSUM` is used to validate that the checksum returned by the CHECKSUM command matches the actual checksum of the firmware.
    - The `CHECKSUM` function returns a pre-calculated checksum in order to provide a quick response to the command for backward compatibility. To validate that the firmware actually matches that checksum the `CHECKSUM` command is used.
    - The `CHECKSUM` command can take many seconds to execute as it calculates the checksum. Once complete, it will return either two or three characters, either the word yes or no with no termination. During the time of calculation, the printer is unavailable to receive other commands.
    - If the command returns yes, the checksum provided by `CHECKSUM` has been validated. If the command returns no, the checksum calculated did not match that provided by CHECKSUM.

- **Swift (iOS)**

```
public func cpclCheckSum()
public func cpclCheckSumVertical()
```

- **Sample Code**

6.6 Char Count

- **Function**

    - `CHAR-COUNT` returns a null terminated string indicating the number of characters received since the last time the CHAR-COUNT command was issued. The value is always set to 0 at startup. When the value is read out, it is reset to 0.
    - The size of the `CHAR-COUNT` command is always included in the number of characters counted, because the value is returned after the command is parsed.
    - `CHAR-COUNT` counts the received characters for all data received by the printer's parsers on all interfaces, not just data which is received by the CPCL parser.

- **Swift (iOS)**

```
public func cpclCharCount()
```

- **Sample Code**

6.7 Delay Actions

- **Function**

    DELAYED-ACTIONS is used to execute a file that is stored on the printer after a delay.

- **Swift (iOS)**

```
public func cpclDelayedActionsWithFileName(filename: String, delay: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
FileName	Space-Terminated String	Filename to execute after delay	A valid filename
Delay	5-Digit Number	Time to delay in 8th of a second increments	0 to 65535

- **Sample Code**

6.8 Display

- **Function**

> - `DISPLAY` is used to display raw text on the display of the printer. The command is parsed but has no effect on printers which do not have displays.
> - The command has two options. TEXT (and its alias T) are used to place text on the display, and the LCD- DBG INFO option is used to obtain information about the display.

- **Swift (iOS)**

```
public func cpclDisplay(message: String)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Message	Terminated String	Text to place on the screen.	Any Text.

- **Sample Code**

6.9 Dump

- **Function**

> `DUMP` is an alias for CAPTURE PRINT in Link-OS printers. See that command on page 258 for more information.

- **Swift (iOS)**

```
public func cpclDump()
```

- **Sample Code**

6.10 Dump Image

- **Function**

> - DUMP-IMAGE dumps the label memory of a label session currently being rendered.
> - The command can either dump label memory in bytes or in bits. If the BYTES option is specified, the label data is returned in hexadecimal bytes, otherwise it is returned in binary.
> - This command can only be used in label sessions, and the label data is dumped at the time the command is parsed. Note that the dumping can take a rather long time, and during that time no other system operations can occur.

- **Swift (iOS)**

```
public func cpclDumpImageWithBitsWithLineCount(lineCount: Int, start: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
LineCount	5-Digit Number	The number of dot lines to dump	0 to 65535
Start	5-Digit Number	The starting dot line to dump	0 to 65535

- **Sample Code**

6.11 Get Date

- **Function**

> - `GET-DATE` returns the current date as set on the printer's real-time clock to the host. While all printers have a real-time clock, not all of them are battery backed up to preserve the time when the printer is powered off. See your printer's documentation for more information.
> - The date is formatted in the form `mm-dd-yyyy`, followed by a NUL character.
> - The date can be set either via the rtc.date SGD or the `SET-DATE` CPCL command.

- **Swift (iOS)**

```
public func cpclGetDate()
```

- **Sample Code**

6.12 Get Time

- **Function**

  - `GET-TIME` returns the current time as set on the printer's real-time clock to the host. While all printers have a real-time clock, not all of them are battery backed up to preserve the time when the printer is powered off. See your printer's documentation for more information.
  - The time is formatted in the form `hh:mm:ss`, followed by a NUL character. The time is always in 24-hour format.
  - The time can be set either via the rtc.time SGD or the `SET-TIME` CPCL command.

- **Swift (iOS)**

```
public func cpclGetTime()
```

- **Sample Code**

6.13 Get Var

- **Function**

  - `GETVAR` is the primary method of retrieving the current setting of a configuration setting from the printer. These configuration settings are called SGD settings, short for Set-Get-Do, which outlines the three commands that can be used to interact with the settings, `SETVAR`, `GETVAR` and `DO`.
  - These three commands are always available on every Link-OS printer, regardless of current language selected. Even Link-OS Printers which do not support the CPCL language support these CPCL commands.
  - The `GETVAR` command can be in lower or upper case, but must be of a single case (it cannot be mixed). Generally CPCL commands must be in upper case, but this one is supported in both cases for backwards compatibility.
  - When used to access settings, the `GETVAR` command returns the current value of the setting. Some settings are preserved between power cycles, and some settings can also have temporary values that will be reset to the saved value with the printer is reset.
  - The `GETVAR` command cannot be used to determine properties about the setting, only its current operating value. See the documentation of each SGD for more information.
  - The response to `GETVAR` is a quote-bound string specifying the value of the setting

requested. If the setting is not available or not valid on this printer, the response will be a double-quote bound question mark.
    - Requesting some settings which are not supported my result in an empty pair of double-quotes.
    - SGD settings are organized into branches, which group settings of similar type. The available setting branches vary depending on the model of printer.

- **Swift (iOS)**

```
public func cpclGetVarWithSettingName(settingName: String)
```

- **Sample Code**

6.14 Line Terminator

- **Function**

    - `LT` is used to change the way CPCL lines are terminated. At power on, all lines in CPCL must be terminated with . All examples and text within this document refer to this default behavior. It is possible, however, to change the way lines are terminated using this command.
    - Regardless of what mode is selected, lines can always be LF terminated, and any CR that exists before the `LF` will be consumed. Lines can also always be NUL terminated.
    - When an invalid command is received by CPCL, the parser always consumes the data up to the next `CR- LF`, regardless of the setting of `LT`.
    - The various parameter types in CPCL act differently with regard to how they handle excess characters besides the specific ones specified, or other details on how they are terminated.
    - For example, while a numeric parameter will consume any excess CRs before an LF, a file parameter such as in `GETVAR` will not. It is also possible to terminate commands with spaces, and in some cases arbitrary characters. The rules vary based on the parameter type that is the last one in the command, or in some cases, the command itself.
    - For this reason, it is best to select the proper terminator and use it as specified to ensure correct behavior.

- **Swift (iOS)**

```
public func cpclLineTerminator(mode: String)
```

- **Parameter**

Mode	Description
LF	Lines are terminated with LF. A single CR before the LF is always ignored.
CR-LF	Same as LF.
CR	Lines are terminated with CR or LF. An LF character after a CR will not be processed and will fall out to the next parser.
CR-X-LF	Lines are terminated with , but zero or more characters (including NULs) may appear between the CR and the LF. These characters between the CR and LF will be consumed and ignored.

- **Sample Code**

6.15 Max Label Height

- **Function**

  - `MAX-LABEL-HEIGHT` is a command provided for backwards compatibility. In previous CPCL products, it returned the maximum number of dot lines a label session could be due to memory limitations.
  - In Link-OS, this value is always 65535, which is the maximum label height supported in CPCL. The value is null terminated.

- **Swift (iOS)**

```
public func cpclMaxLabelHeight()
```

- **Sample Code**

6.16 On Feed

- **Function**

  - ON-FEED specifies what occurs when the user presses the feed button.
  - By default, and each time the printer is powered on, the feed button on the printer causes the printer to feed to the next mark or gap, or in the case of continuous media the page length set by SETFF. See page 232 for more information on that command. This command allows to you change the function of the feed button to either have it reprint the last label, or ignore the key press.

- **Swift (iOS)**

```
public func cpclOnFeed_Feed()
public func cpclOnFeed_Reprint()
public func cpclOnFeed_Ignore()
```

- **Parameter**

Value	Description
FEED	Paper is fed to mark or gap as described above. This is the default behavior.
REPRINT	The last label printed is printed again. If no label has been printed since power on, no motion occurs.
IGNORE	The key press is ignored, and no motion occurs.

- **Sample Code**

6.17 On Low Battery

- **Function**

  - `ON-LOW-BATTERY` specifies an optional alarm and message which is presented to the user when the battery of the printer is low.
  - The three options in the command, `ALERT`, `ALARM` and `NONE` are all optional, any number or none of them may be present, though if none are, the command is parsed but has no effect.
  - By default, both the audio and text alert are disabled (which is equivalent to the NONE option).
  - The notifications only take effect on the point of transition to battery low. If the battery is already low when the commands are sent, they will not have any effect until the battery transitions from normal to low again.

- **Swift (iOS)**

```
public func cpclOnLowBatteryWithAlertText(altertText: String, alarmLength:
Int)
public func cpclOnLowBateryWithNone()
```

- **Parameter**

Parameter	Type	Description	Valid Range
AlertText	Quoted String Parameter	Specifies the text to be displayed when the battery is low on the display	0 to 100 characters
AlarmLength	5-Digit Numeric Parameter	Specifies length of audible alarm in 1/8th of a second increments	0 to 65535

- **Sample Code (Swift)**

```
cpclUtilitySession()
cpclOnLowBatteryWithAlertText("Low Battery Alert!", alarmLength: 40)
```

6.18 Re-Run

- **Function**

> `RE-RUN` is a special purpose command that can only be used within formats which are stored on the printer. The command instructs the printer to run the format file again once it has completed execution.

- **Swift (iOS)**

```
public func cpclReRun()
```

- **Sample Code**

6.19 Set Date

- **Function**

> - `SET-DATE` sets the current date on the printer's real-time clock. While all printers have a real-time clock, not all of them are battery backed up to preserve the time when the printer is powered off. See your printer's documentation for more information.
> - The date is formatted in the form `mm-dd-yyyy`.
> - The date can be read via the rtc.date SGD, or via the `GET-DATE` CPCL command.

- **Swift (iOS)**

```
public func cpclSetDate(date: String)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Date	CR-LF Terminated String	The date to set the real-time clock to.	See below.

- **Sample Code**

6.29 Set Time

- **Function**

> `SET-TIME` sets the current time on the printer's real-time clock. While all printers have a real-time clock, not all of them are battery backed up to preserve the time when the printer is powered off. See your printer's documentation for more information.
>
> The time is formatted in the form `hh:mm:ss`.
>
> The time can be read via the rtc.time SGD, or via the `GET-TIME` CPCL command.

- **Swift (iOS)**

```
public func cpclSetTime(time: String)
```

- **Sample Code**

6.30 Set Version

- **Function**

> - `SET-VERSION` is used to set the response of the `VERSION` command (as well as the appl.version SGD). This function is provided so that applications that rely on a particular reply for the VERSION command can continue to function with newer versions of software.
> - The change of value is temporary, and cannot be stored permanently. Upon reboot, it will always reset to its default value which

- **Swift (iOS)**

```
public func cpclSetVersion(version: String)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Version	Terminated String	The version to set appl.version to	Up to 20 characters

- **Sample Code**

6.31 Set Var and DO

- **Function**

> - `SETVAR` is the primary method of setting the value of a printer configuration setting. These configuration settings are called SGD settings, short for Set-Get-Do, which outlines the three commands that can be used to interact with the settings, `SETVAR`, `GETVAR` and `DO`.
> - These three commands (SETVAR, GETVAR and DO) are always available on every Link-OS printer, regardless of current language selected. Even Link-OS Printers which do not support the CPCL language support these CPCL commands.
> - The `SETVAR` command, and its alias `DO`, can be in lower or upper case, but must be of a single case (it cannot be mixed). Generally CPCL commands must be in upper case, but this one is supported in both cases for backwards compatibility.
> - Some settings are not actually settings, but triggers for actions. These are commonly called “DO” commands. The device.reset SGD is one such example of a DO. Setting the device.reset SGD to any value, including an empty string, causes the printer to reboot. It is the setting which determines if an action is taken or if a setting is changed. Either setting command may be used to activate the function.
> - When `SETVAR` is used to set a setting which is persistent between power cycles, SETVAR will set the setting in such a way that it is saved. This statement may seem obvious, but some commands which set settings set them in such a way that they will be restored to the last saved value on power cycle.
> - Not all settings persist between power cycles; see the documentation of each SGD for more information.

- **Swift (iOS)**

```
public func cpclSetVarAndDoWithSettingName(settingName: String, value:
String)
```

- **Parameter**
  - SettingName

    > Specifies the name of a setting to which is to be set. This string should be bound by double quotation marks.

  - Value

> Specifies the value to set the setting to. This string should be bound by double quotation marks.

- **Sample Code**

6.32 Timeout

- **Function**

  - `TIMEOUT` is used to set the amount of time the printer will sit idle before it powers down. This function generally only applies to printers powered by batteries.
  - Idle is considered by the printer to be a state without printouts, user interaction with the keypad, or changes in printer status (such as head open to close).
  - Setting the `TIMEOUT` to zero disables the power down on idle functionality.
  - This function explicitly sets power.inactivity_timeout, but the value set by TIMEOUT only persists until reboot (or power down). Setting the SGD explicitly will make the setting permanent. See the SGD documentation for more information on the function.

- **Swift (iOS)**

```
public func cpclTimeout(value: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Value	5 Digit Number	The length of the timeout in 1/8ths of a second	0 to 65535.

- **Sample Code (Swift)**

```
cpclLineMode()
cpclTimeout(48)
cpclPrint()
```

6.33 Wait

- **Function**

  - `WAIT` is used to add a delay in a label session after a label is printed.
  - Only one `WAIT` command can be used per label session. The final WAIT command

> in the label is the one which is executed. If the label is reprinted, or if it is part of a batch, the wait will occur each time it is printed.

- **Swift (iOS)**

```
public func cpclWait(duration: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Duration	5-Digit Number	Length to delay in 1/8ths of a second	0 to 65535

- **Sample Code (Swift)**

```
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 150, quantity: 5)

cpclWait(80)
cpclText(rotate: 0, font: 5, fontSize: 0, x: 0, y: 60, text: "Delay 10
seconds")
cpclForm()

cpclPrint()
```

6.34 Label Session Position

- **Function**

> - X, Y, and XY are used to specify the value of X and Y parameters in label sessions, eliminating the need to specify them in label formatting commands.
> - The X, Y and XY commands have a different effect if used in Line Print mode. See those commands on page 181 for more information.
> - These commands affect many functions in CPCL. Anywhere the parameter names [X] or [Y] are used in this document, this function will replace them if the commands are used in label sessions.
> - X permits you to set and omit the X parameter, Y permits you to set and omit the Y parameter, and XY does both. Each can be individually activated and deactivated.
> - When the label session ends, the X, Y and XY commands are deactivated.

- **Swift (iOS)**

```
public func cpclSetLabelPosition(xPos xPos: Int, yPos: Int)
public func cpclSetLabelPosition(xPos xPos: Int)
public func cpclSetLabelPosition(yPos yPos: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
XValue	5 Digit Number	Specifies the value to use for the X position	-1 to 65535
YValue	5 Digit Number	Specifies the value to use for the Y position	-1 to 65535

- **Sample Code**

6.35 Sound Printer Bell

- **Function**

> is an escape command which sounds the printer's bell. It has no parameters. The character is ASCII character 7.

- **Swift (iOS)**

```
public func cpclSoundPrinterBell()
```

- **Sample Code**

6.36 Backspace

- **Function**

> - is an escape character which is used to back up the text cursor one position along the X axis. The amount backed up is determined by the last character printed. If no character has been printed, no motion occurs.
> - This command applies to line print mode, as well as any label command in the family of TEXT commands .
> - The BS command us not supported when using TrueType fonts with ROTATE values besides 0.

- **Swift (iOS)**

```
public func cpclBackspace()
```

- **Sample Code**

6.37 Get or Set CCL Key

- **Function**

> The CCL Key is the character used to start sessions. By default, and at power on, the Key is set to the exclamation point character .

- **Swift (iOS)**

```
public func cpclGetOrSetCCLKey(key: Int)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Key	Single Raw Byte	Specifies the CCL Key.	0 to 255

- **Sample Code**

6.38 Send Two-Key Report to Host

- **Function**

> - I is used to instruct the printer to transmit the two-key report to the host, rather than printing it. The V command is used to print the two-key report.
> - The two key report returned via this command has the same fields and information as the printed report, although it omits any barcodes that may be printed on the report.

- **Swift (iOS)**

```
public func cpclSendTwoKeyReportToHost()
```

- **Sample Code**

6.39 Send User Label Count

- **Function**

> - JRU is used to retrieve the user label count, which is incremented any time a label is printed, regardless of control language the label originated from.
> - The value is returned as a 16-bit number (2 bytes), which represent the same value as contained in the SGD odometer.user_label_count.

- **Swift (iOS)**

```
public func cpclSendUserLabelCount()
```

- **Sample Code**

6.40 Acknowledge Reset

- **Function**

  - N is used to acknowledge the fact that the printer has been reset. The reset status is part of the
  - h command, representing bit 4. At power on, that bit initially is a value of 1. After issuing the
  - N command, the value of that bit changes to 0 and remains so until the printer is reset.

- **Swift (iOS)**

```
public func cpclAcknowledgeReset()
```

- **Sample Code**

6.41 Shut Down Printer

- **Function**

  - p is used to shut down the printer. This feature only applies to printers which can shut themselves down, which includes all battery-based Link-OS printers.
  - As soon as the command is received, the printer will power off.
  - This function is equivalent to setting the power.shutdown SGD to anything.

- **Swift (iOS)**

```
public func cpclShutDownPrinter()
```

- **Sample Code**

6.42 Print Two-Key Report

- **Function**

  V is used to instruct the printer print a copy of the two-key report the two-key report.

- **Swift (iOS)**

```
public func cpclPrintTwoKeyReport()
```

- **Sample Code**

7 Magnetic Card Reading Commands

7.1 MCR

- **Function**

  - The MCR command is used to configure all aspects of the magnetic card reading system in CPCL.
  - The Timeout parameter and at least one track must be specified, but all other parameters are optional.
  - The MCR command is very versatile and can be used in a number of ways. Be sure to see the examples section for some examples of the various ways the command can be used.
  - At the start of each MCR command, all options in the MCR system are reset to the values listed as defaults below, for both options and parameters.
  - On printers without magnetic card readers, this command is parsed but ignored.

- **Swift (iOS)**

```
public func cpclMCRWithTimeout(timeout: Int,delimiter: String, errorPrefix: String, prefix: String, postfix: String)
```

- **Parameter**

Parameter	Type	Description	Valid Range
Timeout	5 Digit Number	The timeout of the MCR command	0 to 65535
Delimiter	Space Terminated String	The delimiter for the track number designators	Any 2 characters
ErrorPrefix	Space Terminated String	Text placed before an error message	Up to 12 characters
Prefix	Space Terminated String	Unit-width of the barcode in dots	Up to 12 characters
PostFix	Space Terminated String	Configuration options for barcode	Up to 12 characters

- **Sample Code**

7.2 MCR-CAN

- **Function**

  - MCR-CAN is used to abort an active MCR command. The command can be used to abort sessions with the MUTLIPLE option in the MCR command, and also will abort an MCR session with a timeout which has not yet expired.
  - Any reads attempted once MCR-CAN has been issued will be ignored. If a read has occurred which has not yet been retrieved via the MCR-QUERY command, that data will still be available via the MCR-QUERY command and is not cleared.
  - If no MCR session is active, the MCR-CAN command has no effect.
  - See the MCR command in the previous session for more information on the MCR-CAN command.

- **Swift (iOS)**

```
public func cpclMCRCan()
```

- **Sample Code**

7.3 MCR-QUERY

- **Function**

> - `MCR-QUERY` is used to obtain the result of a magnetic card reader swipe when the MCR configuration command contains the QUERY option.
> - By default, the MCR command will return scanned card data directly to the host as soon as the card is scanned. If this behavior is not desirable, the QUERY option can be added to the MCR command to indicate that the `MCR-QUERY` command is used to obtain the data on the card once it is scanned, rather than sending it as soon as it is available.
> - The `MCR-QUERY` command will only return card data one time. As soon as it returns the card data, the card data is cleared from the printer's memory, so issuing the command more than one time per card read will not work.
> - If you issue the `MCR-QUERY` command and there is no read data present, no data will be returned by the printer.
> - In the case that the MULTIPLE option was specified in the MCR command along with QUERY, it is possible that if the user scans multiple cards in the interval between reads, reads can be lost. `MCR- QUERY` will only return the data for the most recent scan in this case.

- **Swift (iOS)**

```
public func cpclMCRQuery()
```

- **Sample Code**

8 File Commands

8.1 Define and Use Format Sessions

- **Function**

> `DEFINE-FORMAT`, or it alias DF is used to create formats for use with `USE-FORMAT`, or it's alias UF, or for general use within the printer. The format of the sessions is as follows.

- **Swift (iOS)**

```
public func cpclDefineFormat(filename: String)
public func cpclUseFormat(filename: String)
```

- **Parameter**

Field Name	Description	Type	Valid Range
Filename	The name of a filename to create on the file system.	Terminated String	38 alpha-numeric characters plus drive letter and period for extension

- **Sample Code (Swift)**

```
cpclDefineFormat("SHELF.FMT")
cpclLabel(offset: 0, hRes: 200, vRes: 200, height: 250, quantity: 1)
cpclCenter()
cpclText(rotate: 0, font: 4, fontSize: 3, x: 0, y: 15, text: "\\")
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 75, text: "\\")
cpclBarcode(type: CPCLBarcodeType.UPCA.rawValue, width: 1, ratio: 1,
height: 40, x: 0, y: 135, barcode: "\\")
cpclText(rotate: 0, font: 4, fontSize: 0, x: 0, y: 195, text: "\\")
cpclForm()
cpclPrint()

cpclUseFormat("SHELF.FMT")
cpclInsertTextLine("$22.99")
cpclInsertTextLine("sweatshirt")
cpclInsertTextLine("40123456789")
```

8.2 Delete

- **Function**

> `DELETE` is used to remove a file from disk. The file to be deleted can either be on the E drive or the R drive. All files on these drives can be deleted.

- **Swift (iOS)**

```
public func cpclFileDelete(filename: String)
```

- **Parameter**

Parameter	Type	Description	Valid Range
FileName	CR-LF Terminated String	The name of the file to delete	See below

- **Sample Code**

8.3 Dir

- **Function**

> `DIR` is used to retrieve a listing of the files on the E drive and R drive of the printer. The files are returned in a list format with the file name on the left side and the file size on the right side. Files on the R drive are represented by the presence of (r) after their file name. The files are displayed in the order they were loaded on to their respective drive. The R drive files always appear last on the list.

- **Swift (iOS)**

```
public func cpclFileDir()
```

- **Sample Code**

8.4 End

- **Function**

> - The `END` command, when used in a `DEFINE-FILE` or `DF` session instructs the printer to terminate the session and close the file being defined.
> - This command is an alternative to the PRINT command, which closes file after writing the PRINT command to the file.
> - This command is quite commonly used when defining batch files for the printer, as they typically contain only configuration commands, and not any material to be printed.

- **Swift (iOS)**

```
public func cpclFileEnd()
```

- **Sample Code**

8.5 File

- **Function**

> - The `FILE` command has two functions. The first is the ability to rename existing files, similar to the file.rename SGD. The second function is the ability to calculate a CRC16 on an existing file on the printer.
> - Either or both of the options may be specified in any order, though the new file name must follow the RENAME command.

- **Swift (iOS)**

```
public func cpclRenameFile(oldFilename oldFilename: String, newFilename:
String)
```

- **Parameter**

Parameter	Description	Valid Range
FileName	The name of the file to operate on	up to 38 characters with a 5 character extension after the dot
NewFileName	The new name of the file if using the RENAME option	up to 38 characters with a 5 character extension after the dot

- **Sample Code**

8.6 Type

- **Function**

  - The type command is used instruct the printer to transmit the contents of a file to the host. The file will be transmitted on the same port on which the command was received.
  - Not all files can be typed. The same rules apply to this command as do to the file.type SGD. The following file extensions cannot be typed. An attempt to type a file with any of these extensions will result in no data being returned.

- **Swift (iOS)**

```
public func cpclFileType(filename: String)
```

- **Parameter**

Parameter	Description	Valid Range
filename	The name of the file to operate on	up to 38 characters with a 5 character extension after the dot

- **Sample Code**