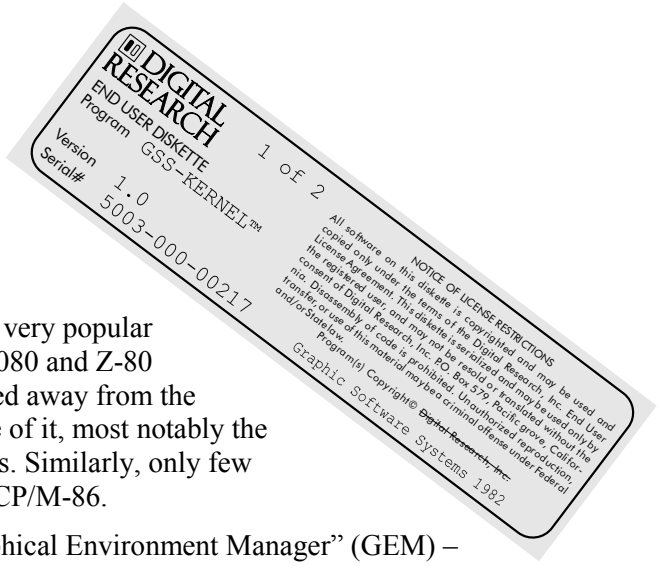
GSX-80 under CP/M 2.2



Martin Hepperle, March 2020

The “Graphical System Extension” (GSX) system was never very popular in the world of CP/M 2.2. Digital Research supported it on 8080 and Z-80 based systems only for a relatively short time and then it faded away from the public view. Only a few computer systems actually made use of it, most notably the Amstrad/Schneider computers, albeit already with CP/M-Plus. Similarly, only few software developers adopted the second implementation for CP/M-86.

However, GSX formed an essential component for the “Graphical Environment Manager” (GEM) – its “Virtual Device Interface” (VDI) is essentially an enhanced GSX. While less successful on the PC platform, GEM and its VDI were widely spread through the implementation for the Atari ST family.

Background

Most early microcomputers used terminals for input and output. In most cases these were connected with a serial interface. Typically, these terminals could display text and some of the more sophisticated terminals were able to show graphics. Later, when memory became cheaper, the terminal capabilities were built into the computers themselves. Usually a video board carried the required memory to buffer the screen content in text and graphics modes.

Each manufacturer developed his own hardware configuration and software had to be adapted to the hardware. Various sets of control sequences existed even for simple text terminals. Thus, the software was closely linked to the available hardware. When it came to graphics, the situation was even more diffracted. A bit mapped display needed a completely different software driver than a vector-based display. Similarly, raster printers and pen plotters required specific driver software.

In the early years, these drivers were built directly into the application software. Often, specific installer programs were used to patch control codes into the application programs. Later, the device specific parts were moved to driver libraries and each program came with a large set of drivers for displays, printers and plotters. Often, the main program would fit on a single diskette and the driver libraries would spread over 10 or more diskettes. Of course, each manufacturer developed his own in-house scheme.

In the mainframe world similar problems existed. The “Graphical Kernel System” (GKS) was an attempt to standardize the application programming interface with a strong focus on graphics, but also supporting text-based user interfaces through cursor positioning and character attributes.

The application programmer would write his application only for a single high-level interface and an intermediate software layer, the GKS Kernel, would translate these positioning and drawing commands through another interface to the low-level device drivers. These would interface directly to the final input and output hardware.

Thus, application programs would be portable between systems. The user would obtain drivers for his system components from the hardware vendor and could run the portable programs without the need for adaptation. Replacing a display or printer device would simply mean replacing the appropriate driver.

After many years of development (starting in 1976), GKS became an international standard in 1985 (ISO and DIN). While its first implementations were designed for high level languages like FORTRAN or Pascal, it was not limited to these languages.

A Graphical Kernel System for Microcomputers

The complete GKS standard was rather large and thus not very useful for small microcomputers of the era, which typically came with 64 KB of RAM or even less. Therefore, Digital Research and Graphics Software Systems Inc. developed a simplified system, which fit into this limited memory space while leaving room for the application programs. In 1982, Digital Research offered the GSX-80 system in form of the software products GSS-KERNEL, GSS-PLOT (a subroutine library) and GSS-4010 (a Tektronix terminal emulation).

By the end of 1983, a cooperation between Digital Research and TeleVideo lead to the software product "DR Draw" – an adaptation of TeleVideo's "TeleDraw" program.

In the same year, Digital Research published GSX-86 for CP/M-86 on Intel-based computers available.

Not many CP/M-80 systems relied on GSX-80 as a graphics interface and supplied drivers. In 1984, TeleVideo offered the portable TPC-1 which was advertised with GSX-80. Two GSX screen drivers were available for the BBC Microcomputer with a Z-80 second CPU extension and CP/M 2.2.

Using an advanced CP/M-Plus as an operating system the Schneider/Amstrad PCW "Joyce" systems were offered with GSX drivers.

Using GSX under CP/M 2.2

GSX is loaded together with your application program. The core system GDOS acts as an interface layer between your application program and a device-dependent driver. The application program communicates with the GDOS through a specific BDOS call which must be supplied with a function code and corresponding arrays of input and output parameters.

DRI provided a program GENGRAF.COM, which prepends a loader for the GDOS (contained in GSX.SYS) to your COM program. This loader brings the GDOS into memory, installs a BDOS interface, reserves memory for the first driver found in ASSIGN.SYS, loads it, moves your application back to address 100H where it belongs and finally starts it. This loader adds 380_H bytes to your program.

After GENGRAF has been applied once to your program, it is not needed for running your program. Only GSX.SYS must be present on the application disk.

Whenever you re-compile your COM program, you must apply GENGRAF again. If your program is not relocatable but compiled for a fixed end address (like Turbo Pascal COM programs, which by default stretch to the upper end of the TPA), you have to make sure that the end address is lowered to make room for the GSX system, including the largest driver you want to support. This may need some trial and error (the GSX.SYS file is about 4 Kbytes and a device driver adds between 8 and 12 Kbytes to this). The GSX system may need up to 18 KB including the largest Epson printer driver. If you only use display drivers, the memory needs are more like 12 KB. GSX is only loaded temporarily – memory is released when your program terminates.

If you forget to attach GSX with the GENGRAF program, all BDOS calls to GSX will return with an error code and the return-arrays will contain undefined values. In addition, the call to OPEN WORKSTATION will <u>not</u> display the GSX copyright banner shown in Figure 1.

```
-------------------------------------------------
GSX-80 1.1  01 Oct 83    Serial No 5000-1232-654321
Copyright (C) 1983
Digital Research, Inc.          All Rights Reserved
-------------------------------------------------
```

Figure 1: A copyright banner is shown when the loader is executed and has loaded the first driver declared in ASSIGN.SYS. An error message is displayed if the driver is not found.

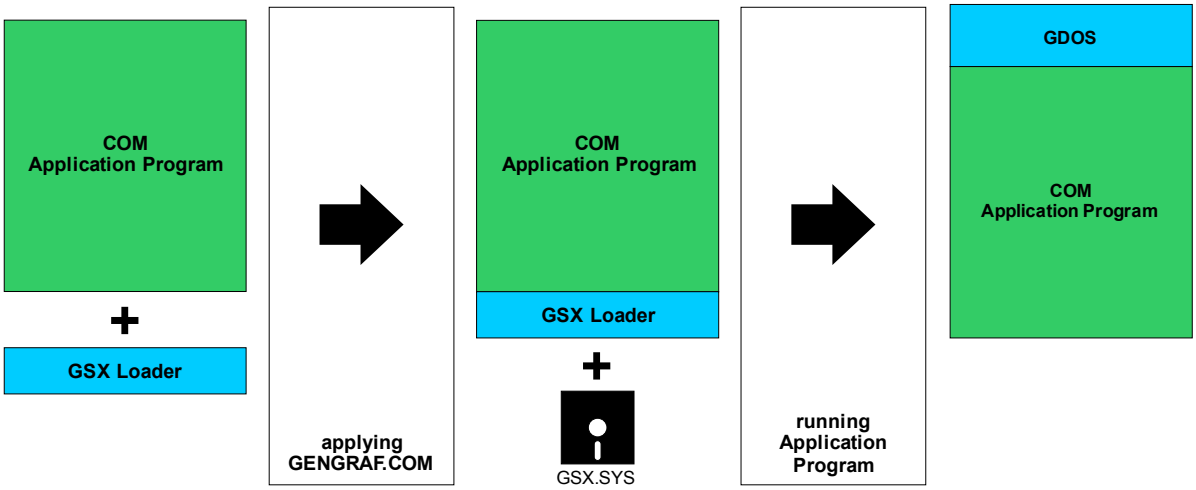


Figure 2: GENGRAF.COM attaches the loader for GSX.SYS to your program.

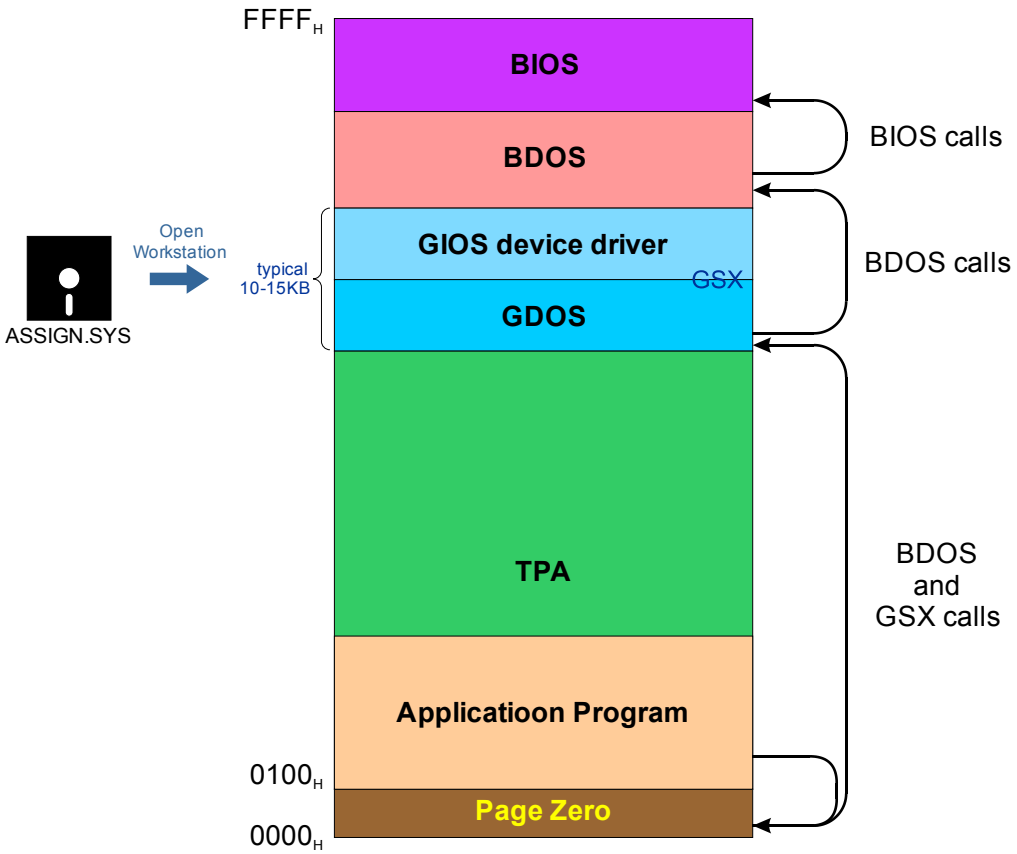


Figure 3: Similar to the BDOS and BIOS, the GDOS and GIOS are loaded above the Transient Program Area (TPA). BDOS and GSX function calls are redirected through the GDOS to filter out the calls to GSX.

Device Drivers

In a text file named ASSIGN.SYS, an ID number is assigned to each driver. This file contains a table with pairs of ID number and associated driver file name. It must contain at least one driver and each driver must have a unique ID. Digital Research's documentation for GSX describes the syntax.

GSX maintains a buffer area to load and swap drivers as needed. The first driver in the ASSIGN.SYS file defines the size of this buffer area. When a different driver is requested (e.g. after previewing a graph on screen before printing it), the previous driver is replaced by the new driver.

```
10 @:DDHP7470
01 @:DDHP2627
30 @:DDMF
```

Figure 4: Example ASSIGN.SYS file with the large plotter driver listed first, a terminal driver and a Metafile driver. The "@" character indicates that the driver shall be loaded from the current disk drive – it could also be replaced by a drive letter like "A".

The number of available GSX-80 device drivers for CP/M 2.2 was not too bad - Table 1 lists all the drivers, which I could find as well as the ones I wrote myself.

File Name	Size	Device	Author
DDHP7470.PRL	11 KB	HP 7470 plotter	Graphic Software Systems
DDHP7220.PRL	10 KB	HP 7220 plotter	Graphic Software Systems
DDHI3M.PRL	9 KB	Houston Instruments Hiplot DMP-3/4-443 plotter	Graphic Software Systems
DDHI7M.PRL	10 KB	Houston Instruments Hiplot DMP-6/7 plotter	Graphic Software Systems
DDSHINWA.PRL	12 KB	Seikosha matrix printer	Digital Research Inc.
DDMX80.PRL	12 KB	Epson MX-80 matrix printer with Graftrax Plus	Digital Research Inc.
DDFXHR8.PRL	15 KB	Epson 8 pin hi-res matrix printer	Digital Research Inc.
DDFXLR7.PRL	12 KB	Epson 7 pin lo-res matrix printer	Digital Research Inc.
DDFXLR8.PRL	12 KB	Epson 8 pin lo-res matrix printer	Digital Research Inc.
DDANADXM.PRL	12 KB	Anadex DP-9501 and DP-9001A matrix printer	Digital Research Inc.
DDCITOLR.PRL	12 KB	C.Itoh 8510A Low Resolution matrix printer	Digital Research Inc.
DDCNTXM.PRL	12 KB	Centronics 351, 352, and 353 matrix printer	Digital Research Inc.
DDDS180.PRL	12 KB	Datasouth DS180 matrix printer	Digital Research Inc.
DDLA50.PRL	12 KB	DEC LA50 matrix printer	Digital Research Inc.
DDLA100.PRL	12 KB	DEC LA100 matrix printer	Digital Research Inc.
DDOKI84.PRL	12 KB	OKIDATA Microline 84 step 2 matrix printer	Digital Research Inc.
DDVRET.PRL	11 KB	VT-100 with Digital Engineering Retro-Graphics[1]	Graphic Software Systems
DDGEN2.PRL	12 KB	Digital Engineering Retro-Graphics[1] Gen. II	Graphic Software Systems
DDHP2627.PRL	8 KB	HP 2627 graphics terminal	Martin Hepperle
DDHP2648.PRL	8 KB	HP 2648 graphics terminal	Martin Hepperle
DDHPGL.PRL	12 KB	HP-GL metafile	Martin Hepperle[2]
DDMF.PRL	7 KB	GEM metafile	Martin Hepperle
DDMFA.PRL	8 KB	MFA computer system, special video card	Martin Hepperle
DDPS.PRL	9 KB	Postscript metafile	Martin Hepperle
DDPX8.PRL	7 KB	Epson PX-8	Martin Hepperle
DDXTEK.PRL	8 KB	Tektronix graphics terminal	Udo Munk

Table 1: These drivers have been examined for this document.

From today's perspective, the variety of the existing drivers was rather limited. Only few terminal drivers and no metafile drivers exist. Therefore, I started to write some for my own hardware needs. A very helpful starting point was the DDXTEK driver, written by Udo Munk, which served as a template.

Each of my own drivers consists of a small assembler module and a major FORTRAN module. By minimizing the usage of FORTRAN runtime library routines and limiting the implementation to basic functions, the final drivers have an acceptable size. Of course, writing drivers completely in assembler would produce smaller code, but development would have been much more time consuming for me.

[1] "Retro-Graphics" were add-on boards produced by Digital Engineering. They were available for the Lear Siegler ADM-3A/3A+ and the Digital Equipment VT-100 terminals. These boards provided Tektronix 4010 graphics emulation with a screen buffer RAM of 128 KB.

[2] A modified variant of the driver developed by Graphic Software Systems.

Driver Properties via Turbo Pascal

The following simple Turbo Pascal 3.01 program queries and displays the capabilities of a device as defined by its driver. It is necessary to reserve some memory above the program, which can be done with the End option in the Compile menu of Turbo Pascal. Setting the end address to $A000_H$ leaves enough space for GSX and the largest drivers on my 62 KB CP/M system.

The following table of device properties has been produced by running this program on the GSX-80 drivers for CP/M 2.2.

```
Program GSX;

(* Purpose: query characteristics of a GSX device *)
(* Compile: set end address to $A000              *)
(* Add GSX: GENGRAF GSX                           *)
(* Creator: Martin Hepperle, 2020                 *)
(* ------------------------------------ *)
Var
 contrl  : Array[1..30] of Integer;
 intin   : Array[1..200] of Integer;
 ptsin   : Array[1..200] of Integer;
 intout  : Array[1..200] of Integer;
 ptsout  : Array[1..200] of Integer;
 pblock  : Array[1..5] of Integer;
 w,h     : Real;
(* ------------------------------------ *)
Procedure GSX_Init;
(* set up pointer array *)
Begin
 pblock[1] := Addr(contrl[1]);
 pblock[2] := Addr(intin[1]);
 pblock[3] := Addr(ptsin[1]);
 pblock[4] := Addr(intout[1]);
 pblock[5] := Addr(ptsout[1]);
End;
(* ------------------------------------ *)
Procedure GSX_OpenWS ( DriverID : Integer );
(* open workstation *)
Begin
 contrl[1] := 1;
 contrl[2] := 0;
 contrl[4] := 10;
 intin[1]  := DriverID;
 intin[2]  := 1;
 intin[3]  := 1;
 intin[4]  := 1;
 intin[5]  := 1;
 intin[6]  := 1;
 intin[7]  := 1;
 intin[8]  := 1;
 intin[9]  := 1;
 intin[10] := 1;

 Bdos(115,Addr(pblock[1]));

 WriteLn('x-resolution                ',(intout[1]+1));
 WriteLn('y-resolution                ',(intout[2]+1));
 WriteLn('pixel width in micrometer   ',intout[4]);
 WriteLn('pixel height in micrometer  ',intout[5]);
 w := 1.0*(intout[1]+1)*intout[4];
 h := 1.0*(intout[2]+1)*intout[5];
 WriteLn('0=precise scale supported   ',intout[3]);
 WriteLn('number of characters sizes  ',intout[6]);
 WriteLn('number of line types        ',intout[7]);
 WriteLn('number of line widths       ',intout[8]);
 WriteLn('number of marker types      ',intout[9]);
 WriteLn('number of marker sizes      ',intout[10]);
 WriteLn('number of fonts             ',intout[11]);
 WriteLn('number of patterns          ',intout[12]);
 WriteLn('number of hatch styles      ',intout[13]);
 WriteLn('number of colors            ',intout[14]);
```

```
 WriteLn('number of GDPs supported     ',intout[15]);
 WriteLn('1=color capability           ',intout[36]);
 WriteLn('1=text rotation capability   ',intout[37]);
 WriteLn('1=area fill capability       ',intout[38]);
 WriteLn('1=read cell array capability ',intout[39]);
 WriteLn('number of colors             ',intout[40]);
 WriteLn('number of locator devices    ',intout[41]);
 WriteLn('number of valuator devices   ',intout[42]);
 WriteLn('number of choice devices     ',intout[43]);
 WriteLn('number of string devices     ',intout[44]);
 WriteLn('workstation type             ',intout[45]);
 WriteLn('minimum character size       ',ptsout[2]);
 WriteLn('maximum character size       ',ptsout[4]);
 WriteLn('minimum line width           ',ptsout[5]);
 WriteLn('maximum line width           ',ptsout[7]);
 WriteLn('minimum marker size          ',ptsout[10]);
 WriteLn('maximum marker size          ',ptsout[12]);
End;
(* ---------------------------------- *)
Procedure GSX_CloseWS;
(* close workstation *)
Begin
 contrl[1] := 2;
 contrl[2] := 0;
 Bdos(115,Addr(pblock[1]));
End;
(* ---------------------------------- *)
Begin
 GSX_Init;
 GSX_OpenWS(1);
 GSX_CloseWS;
End.
(* ---------------------------------- *)
```

Figure 5: A simple program to list the characteristics of a GSX-80 driver.

Properties of GSX Drivers for CP/M 2.2

Property \ Driver	PS	MF	HP2627	HP2648	XTEK	PX8	MFA	VRET	GEN2	HP7470	HP7220	HI3M	HI7M
x-resolution	23171	23171	512	720	1024	480	480	1024	640	10300	15200	1881	2800
y- resolution	32767	32767	390	360	768	64	432	780	420	7560	10000	1401	1970
pixel width in micrometers	9	9	420	352	198	480	252	198	381	25	25	125	127
pixel height in micrometers	9	9	420	352	195	480	252	195	381	25	25	125	127
0=precise scale supported	0	0	1	1	1	1	1	1	1	0	0	0	0
number of characters sizes	0	0	4	4	4	1	1	4	0	0	0	5	5
number of line types	6	6	9	9	5	3	1	5	8	7	7	9	9
number of line widths	1	1	1	1	1	1	1	1	1	1	1	1	1
number of marker types	6	5	5	5	5	5	5	5	5	5	5	6	6
number of marker sizes	1	1	4	4	1	4	4	1	1	0	0	5	5
number of fonts	1	1	1	1	1	1	1	1	1	5	5	1	1
number of patterns	0	0	0	0	0	0	0	0	120	0	0	0	0
number of hatch styles	0	0	0	0	0	0	0	0	0	0	0	0	0
number of predefined colors	2	256	8	2	2	2	2	2	7	2	8	6	8
number of GDPs supported	0	0	0	0	0	0	0	0	4	0	0	0	1
color capability	0	1	1	0	0	0	0	0	0	1	1	1	1
text rotation capability	1	1	1	1	0	0	0	0	1	1	1	1	1
area fill capability	1	1	0	0	0	0	0	0	1	0	0	0	0
read cell array capability	0	0	0	0	0	0	0	0	0	0	0	0	0
number of colors	2	256	8	2	2	2	2	2	0	0	0	0	0
number of locator devices	0	0	1	1	1	0	0	1	1	1	1	0	0
number of valuator devices	0	0	0	0	0	0	0	0	0	0	0	0	0
number of choice devices	0	0	0	0	0	0	0	0	0	0	0	0	0
number of string devices	0	0	1	1	1	1	1	1	1	0	0	0	0
workstation type	4	4	2	2	2	2	2	2	2	2	2	2	2
minimum character size	327	327	588	637	427	4096	531	588	390	260	197	327	233
maximum character size	16385	16385	2353	2549	640	4096	1062	2479	25356	16380	16381	5239	3726
minimum line width	45	71	64	46	32	68	68	32	51	3	2	17	12
maximum line width	11585	7071	64	46	32	68	68	32	51	3	2	17	12
minimum marker size	327	327	672	728	341	1024	607	504	780	347	262	327	133
maximum marker size	8192	16385	2689	2913	341	8192	2427	504	780	16380	16381	5239	2129

Table 2: Properties of metafile, display and plotter drivers.

Property \ Driver	ANADXM	CITOLR	CNTXM	DS180	LA50	LA100	OKI84	MX80	SHINWA	FXHR8	FXLR7	FXLR8
x- resolution	576	1088	528	600	1152	1056	824	456	640	960	480	480
y- resolution	680	680	670	672	680	672	672	456	672	1368	672	672
pixel width in micrometers	338	186	385	338	176	192	246	353	317	212	423	423
pixel height in micrometers	352	352	352	352	352	352	352	296	352	176	352	352
0=precise scale supported	0	0	0	0	0	0	0	0	0	0	0	0
number of characters sizes	12	12	12	12	12	12	12	12	12	12	12	12
number of line types	6	6	6	6	6	6	6	6	6	6	6	6
number of line widths	1	1	1	1	1	1	1	1	1	1	1	1
number of marker types	5	5	5	5	5	5	5	7	5	5	5	5
number of marker sizes	12	12	12	12	12	12	12	12	12	12	12	12
number of fonts	1	1	1	1	1	1	1	1	1	1	1	1
number of patterns	6	6	6	6	6	6	6	6	6	6	6	6
number of hatch styles	6	6	6	6	6	6	6	0	6	6	6	6
number of predefined colors	2	2	2	2	2	2	2	2	2	2	2	2
number of GDPs supported	1	1	1	1	1	1	1	1	1	1	1	1
color capability	0	0	0	0	0	0	0	0	0	0	0	0
text rotation capability	1	1	1	1	1	1	1	1	1	1	1	1
area fill capability	1	1	1	1	1	1	1	0	1	1	1	1
read cell array capability	0	0	0	0	0	0	0	0	0	0	0	0
number of colors	2	2	2	2	2	2	2	2	2	2	2	2
number of locator devices	0	0	0	0	0	0	0	0	0	0	0	0
number of valuator devices	0	0	0	0	0	0	0	0	0	0	0	0
number of choice devices	0	0	0	0	0	0	0	0	0	0	0	0
number of string devices	0	0	0	0	0	0	0	0	0	0	0	0
workstation type	0	0	0	0	0	0	0	0	0	0	0	0
minimum character size	386	386	390	390	386	390	390	575	390	192	390	390
maximum character size	4626	4626	4681	4681	4626	4681	4681	6899	4681	2300	4681	4681
minimum line width	57	30	62	55	28	31	40	72	51	34	68	68
maximum line width	57	30	62	55	28	31	40	72	51	34	68	68
minimum marker size	386	386	390	390	386	390	390	575	390	192	390	390
maximum marker size	4626	4626	4681	4681	4626	4681	4681	6899	4681	2300	4681	4681

Table 3: Properties of matrix printer drivers.

A Simple CBASIC Demonstration Program

The following CBASIC-80 program is simply named G.BAS. It displays polylines, markers, and text. It can be compiled, linked and finally GSX can be attached with the following command sequence:

```
CB80 G
LK80 G
GENGRAF G
```

```
REM DEMONSTRATION PROGRAM FOR
REM CBASIC GRAPHICS EXTENSIONS
REM
REM Martin Hepperle, 2020
REM

%INCLUDE GRAPHCOM.BAS

DIM MX(10),MY(10)

REM Test Drivers 1, 2
FOR G = 1 TO 2
 PRINT "Testing driver ";G

 GRAPHIC OPEN G

 ASK DEVICE X.DIM,Y.DIM
 PRINT "Aspect ratio is ";X.DIM;" x ";Y.DIM

 SET BOUNDS Y.DIM,X.DIM

 SET BEAM "ON"

 REM outer frame
 PLOT (0,1),(1,1),(1,0),(0,0)

 SET CHARACTER HEIGHT 0.02
 GRAPHIC PRINT AT (0.05,0.95): "GSX Driver Test"

 SET CHARACTER HEIGHT 0.05
 GRAPHIC PRINT AT (0.05,0.85): "5 %"
 PLOT (0,0.85),(1,0.85)
 PLOT (0,0.90),(1,0.90)

 SET CHARACTER HEIGHT 0.1
 GRAPHIC PRINT AT (0.05,0.5): "10 %"
 PLOT (0,0.5),(1,0.5)
 PLOT (0,0.6),(1,0.6)

 MX(0)=0.25 : MY(0)=0.15
 MX(1)=0.50 : MY(1)=0.45
 MX(2)=0.75 : MY(2)=0.35

 SET CHARACTER HEIGHT 0.025
 SET JUSTIFY 0.0,0.5

 FOR T=1 TO 6
  SET LINE STYLE T
  MAT PLOT 2: MX, MY

  SET MARKER HEIGHT 0.03
  SET MARKER TYPE T
  SET LINE STYLE 1
  MAT MARKER 2: MX,MY
  SET JUSTIFY 0.0,0.5
  GRAPHIC PRINT AT (0.8,MY(2)): STR$(T)
```

```
  SET JUSTIFY 1.0,0.5
  SET TEXT ANGLE 51.5/57.5
  GRAPHIC PRINT AT (0.2,MY(0)): STR$(T)
  SET TEXT ANGLE 0

  MY(0) = MY(0) + 0.1
  MY(1) = MY(1) + 0.1
  MY(2) = MY(2) + 0.1

 NEXT T

 PRINT "Done."

 GRAPHIC CLOSE
NEXT G

END
```

Table 4: This simple test program G.BAS uses an ASSIGN.SYS file having devices with IDs 1 and 2.

Another CBASIC program which was used for testing was DEMOGRAF which came on the CBASIC-80 compiler diskette.

The following sections present some details for some selected drivers.

HP 7470 Plotter (DDHP7470.PRL)

Digital Research provided this driver together with the GSX-80 system. Unfortunately, the driver is hardwired to the PUN:/RDR: devices (usually a serial port). The serial interface of the plotter is programmed to use the ENQ/ACK handshaking. There is no option to configure it for output to a file (see next section).

If no plotter is attached to the serial port (PUN:/RDR:), input and output can be redirected to the console and some handshaking input must be provided to avoid hanging the program due to the missing response from the plotter.

```
STAT RDR:=TTY:
STAT PUN:=TTY:
```

The call to OPEN WORKSTATION outputs configuration commands for the serial interface.

```
[ESC].(
```

This first sequence activates the plotter in case it is in a Y-cable eavesdrop configuration.

```
[ESC].I80;5;6:
```

The second sequence sets the block size to 80, the characters for ENQ to 5 and for ACK to 6.

```
IN;[ENQ]
```

This following sequence initializes the plotter and sends an ENQ character. It then expects an ACK character from the plotter[3]. Pressing the Return key satisfies the handshake requirement of the driver too. Obviously, the driver does not test the reply from the plotter – it continues when just something is sent back. The next sequence sets up default line type and character properties.

[3] The ENQ/ACK handshaking is a standard protocol for many HP devices and terminals. It injects ENQ characters at regular intervals into the output stream. It then expects an ACK character to continue. This is very similar to the XON/XOFF protocol, only using different characters.

```
LT;CS0;SS;SI0.15,0.15;
```

This block of initialization commands is followed by the actual plotting commands.

A final call to CLOSE WORKSTATON homes the pen and expects an ACK to continue. Again, pressing the Return key resumes operation. It then stops the plotter operation in an eavesdrop configuration.

```
PU;PA0,0;[ENQ]
 [ESC].)
```

HP-GL Graphics Language (DDHPGL.PRL)

This driver was developed by extending the DDHP7470.PRL driver supplied by Digital Research. For this purpose, the original driver was disassembled and modified. The "new" driver writes the HP-GL command stream to a file named HPGL-1.PLT. To properly flush the file, output must be finished with a call to CLOSE WORKSTATION. Multiple image files can be created within a single program run. In this case, the digit is incremented each time a new file is created by an OPEN WORKSTATION call. You cannot use GSX input or locator functions when using this driver.

Tektronix 4014 Graphics Terminal (DDXTEK.PRL)

This display driver was written by Udo Munk in a mix of assembler and FORTRAN. Tektronix output is supported by the Unix Xterm program and by Teraterm under Windows. I used his sources as a template for writing my own drivers.

Matrix Printer Drivers (e.g. DDFXHR8.PRL)

These drivers use the "banding" technique to print the graphics using a small memory buffer. They create a temporary file and process it many times to create the printout. Unfortunately markers are created from bitmapped low-resolution data and look somewhat crude. Similarly, text is rendered with a built-in bitmap font defined by a matrix of only 8×8 pixels. Objects made from lines are printed out fine, though.

MFA Terminal Card (DDMFA.PRL)

This driver was written by myself. It supports a special graphics card which was developed in 2019 for the older German MFA "Microprocessor for Education". This modular 19-inch rack mountable system is based on the Intel 8085 CPU and capable of running CP/M 2.2.

Epson PX-8 (DDPX8.PRL)

Another driver written by myself. It provides the basic graphics functions for the LCD screen of the Epson PX-8. Due to the high width-to-height ratio of the screen, it is not so practical to run applications developed for other devices. Nevertheless, the system independency can be demonstrated even with such an unusual aspect ratio. The driver uses the PX-8's escape sequences and can position hardware text only on the relatively coarse row/column grid. Because you can configure the PX-8 with a RAM and User-BIOS of "arbitrary" size, the end address of Turbo-Pascal programs has to be adjusted accordingly. The $A000_H$ setting mentioned above works on my PX-8 with a RAM disk of 9 KB and 512 Bytes of User-BIOS.

HP 2627A Color Graphics Terminal (DDHP2627.PRL)

This driver was written by myself. It supports marker sizes and text rotation, as well as more line styles and 8 predefined colors.

Note that these old terminals had a small I/O buffer and this driver uses the ENQ/ACK protocol to control the data flow. This means that it sends out an ENQ character at regular intervals and expects an ACK character back from the terminal when it is ready to process more input.

HP 2648A Monochrome Graphics Terminal (DDHP2648.PRL)

Again, a driver written by myself. Like the driver for the 2627A, it supports marker sizes and text rotation as well as more line styles. It also uses the ENQ/ACK protocol to control the data flow.

The following sequence of pictures shows some of the output screens produced by the DEMOGRAF program and the DDHP2648 driver on a real HP 2648A terminal.

Figure 6: Text Size. The HP 2648A uses a scalable, pixel based font.

Figure 7: Locator. The graphics cursor can be controlled by the keyboard.

Figure 8: Text Alignment.

Figure 9: Marker Size.

Figure 10: Marker Type.

Figure 11: Polyline.

Figure 12: Line Styles. The HP 2648A provides 9 line styles.

Figure 13: Text Rotation. The HP 2648A is able to rotate text in steps of 90 degrees.

Figure 14: Viewport.

Figure 15: Window.

Figure 16: Thank you for watching.

Figure 17: The text output of DEMOGRAF can be found in the Alpha plane.

Postscript Metafile (DDPS.PRL)

This driver was written by myself. It writes a stream of Postscript pages to a file. An initial OPEN WORKSTATION command creates a new file “GSX.PS”. Any existing file is overwritten. Each subsequent CLEAR WORKSTATION command ejects the current page and starts a new page. Finally, a CLOSE WORKSTATION command is required to properly terminate and close the file.

The output file contains a minimum set of DSC comments to allow embedding as an Encapsulated Postscript file into documents, paging in Ghostscript or importing into graphics programs like Corel Draw.

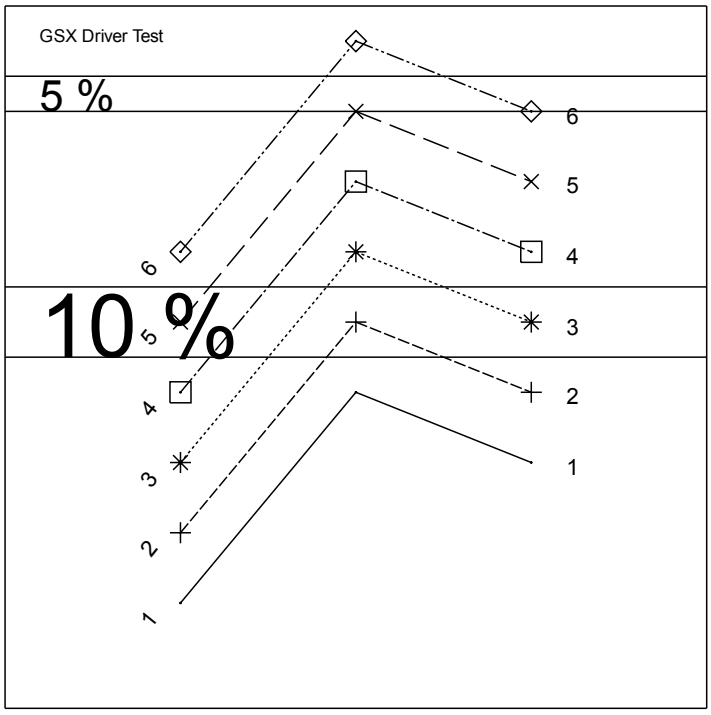


Figure 18: Output of G.BAS generated with DDPS.PRL and imported into Corel Draw.

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 596 834
%%Creator: GSX-80, Martin Hepperle
%%Pages: (atend)
%%BeginProlog

...

%%EndProlog
%%Page: 1 1

...

%%Page: 2 2

...

%%Trailer
%%Pages: 2
%%EOF
```

Figure 19: Structure of the generated Postscript file.

GEM Metafile (DDMF.PRL)

This driver was written by myself. It writes a stream of GSX records to a file. An initial OPEN WORKSTATION command creates a new file "GSX.GEM". Any existing file with this name is overwritten. A final CLOSE WORKSTATION command is required to properly terminate and close the file. It is advisable to send only a single graph to the metafile, otherwise the pages will be overlaid.

The resulting metafile can be imported e.g. into Corel Draw on a modern Windows PC.

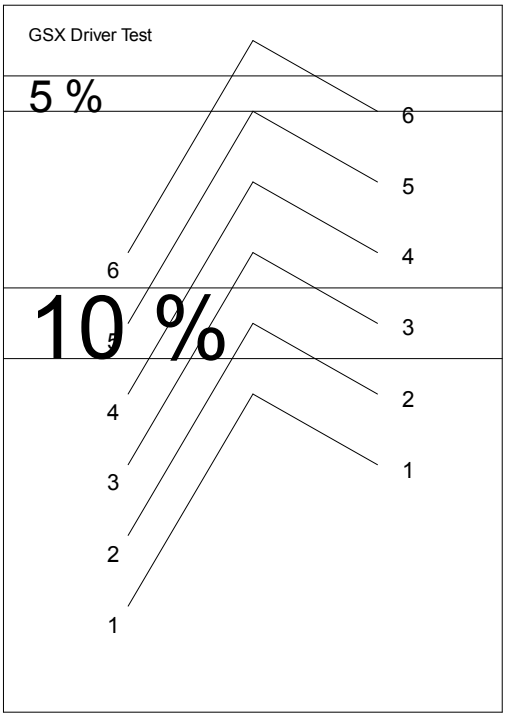


Figure 20: Output of G.BAS generated with DDMF.PRL and imported into Corel Draw. Better than nothing, but some attributes are obviously lost in translation – e.g. markers, line styles

and text orientation. However, they are contained in the metafile, so I blame this fault to Corel Draw.

Note on some GEM Metafiles

For debugging, I used some of the clipart files which came with the DR Draw program. I noticed that circles were always drawn twice. I discovered that the DR Draw files actually contained these circles twice: the first instance was a using the GDP primitive “Circle”, while the immediately following instance drew the same circle with the GDP primitive “Arc” with start and end angles of 0° respectively 360°.

Using GSX with FORTAN – A Tale of Two Libraries

If you don't like BASIC or Turbo Pascal, you can also use FORTRAN to control GSX output devices. Please use Microsoft FORTRAN Version 3.44 – all previous versions (including 3.4) have severe bugs, especially when working with arrays and subroutine parameters (the version of the compiler can be read from a listing generated with the /L option).

Low Level Interface GSXLIB

For interfacing directly to GSX, I have written a wrapper library which uses a small assembler routine to call the actual GSX functions. More FORTRAN routines can be added to the library for supporting more GSX functions.

I applied the following naming scheme loosely to the function names:

Character	Purpose
1	G = GSX or Graphics
2-3	Object to act on: WK = workstation, LN = line, MK = marker, TX = text
4-6	Action to perform: SET = set attributes or properties, PLT = plot, OUT = output

In order to reduce programming effort, some of the library subroutines set several properties for one type of object (e.g. lines, markers) in a single call. These properties stay in effect until they are changed.

All x and y coordinates are in the GSX system's NDC space, i.e. they range from 0 to 32767.

The GSXLIB library contains the following subroutines

Module	Subroutine	Parameters	Purpose
GSXWK	GWKOPN (ID)	INTEGER*2 ID	open workstation with ID
GSXWK	GWKCLO	none	close workstation
GSXWK	GWKCLR	none	clear workstation
GSXWK	GWKINQ (ISIZRC,SIZMM)	INTEGER*2 ISIZRC(2) REAL*4 SIZMM(2)	inquire device dimensions in pixels and millimeters
GSXLN	GLNSET (IWIDTH,ISTYLE,ICOLOR)	INTEGER*2 IWIDTH, ISTYLE, ICOLOR	set properties of lines
GSXLN	GLNPLT (IX0,IY0,IX1,IY1)	INTEGER*2 IX0, IY0, IX1, IY1	plot a line
GSXMK	GMKSET (ISIZE,IMARK,ICOLOR)	INTEGER*2 ISIZE, IMARK, ICOLOR	set properties of markers
GSXMK	GMKPLT (IX,IY)	INTEGER*2 IX, IY	plot a marker
GSXTX	GTXOUT (IX,IY,CTEXT,ILEN)	INTEGER*2 IX, IY, ILEN INTEGER*1 CTEXT(100)	output a text string
GSXTX	GTXSET(ISIZE, IDEGZ, ICOLOR)	INTEGER*2 ISIZE, IANGLE, ICOLOR	set properties of text, angle in 1/10 degrees

Additionally, the library uses an internal common block named "GSX$" for passing parameters through the assembler routine to the GSX kernel.

Building the Library

To rebuild the library, you can use these commands:

```
m80 =gsxcal
f80 =gsxwk
f80 =gsxln
f80 =gsxmk
f80 =gsxtx
lib80 gsxlib=gsxwk,gsxln,gwsxmk,gsxtx,gsxcal,/E
```

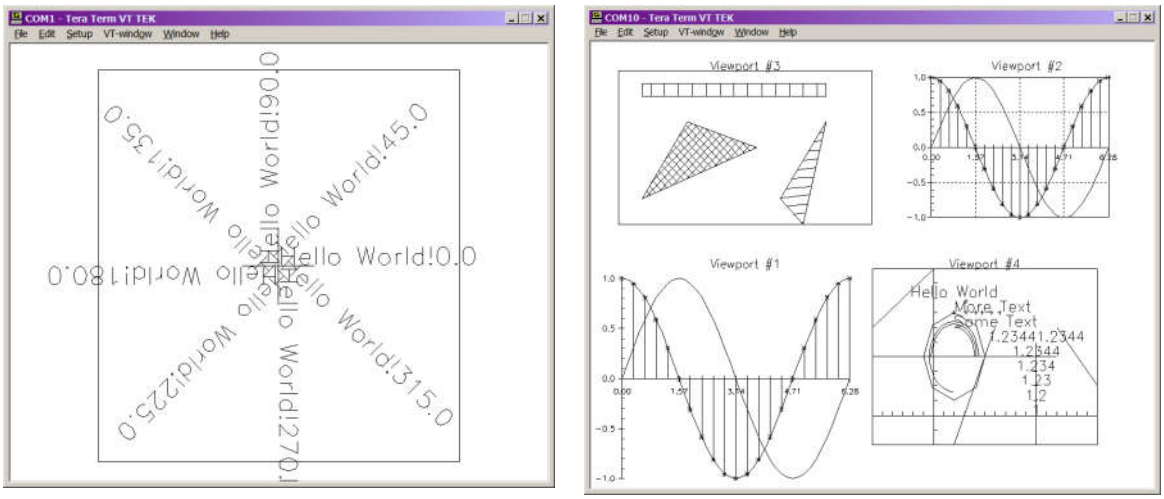
Note that the order of the modules is important. The routine GSXCAL has to be added last to the library, otherwise calls to GSXCAL in the GSXWK, GSXLN, GSXMK and GSXTX modules cannot be resolved and the library would have to be searched twice in all L80 calls.

To compile and link your program GSXPROG.FOR you can use the following command sequence:

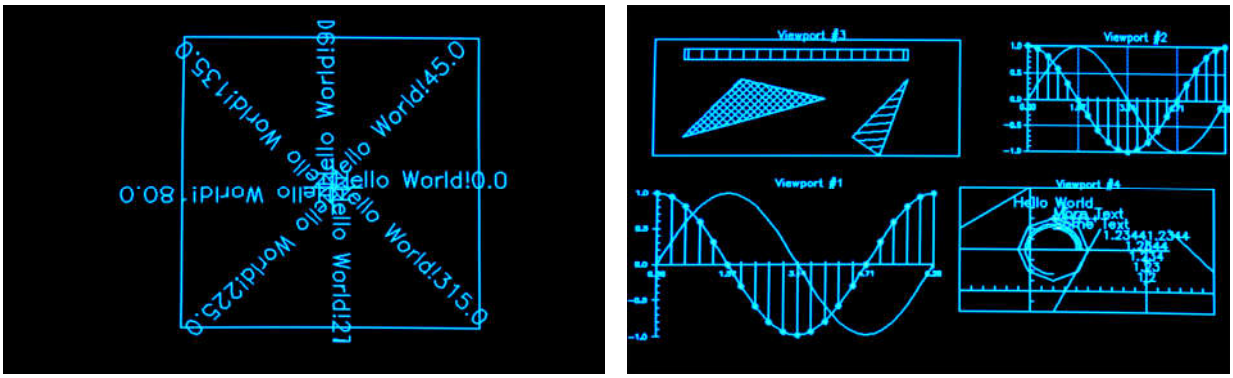
```
f80 =gsxprog
l80 gsxprog/N,gsxprog,gsxlib/S,forlib/S,/E
gengraf gsxprog
```

High-Level Interface GLIB

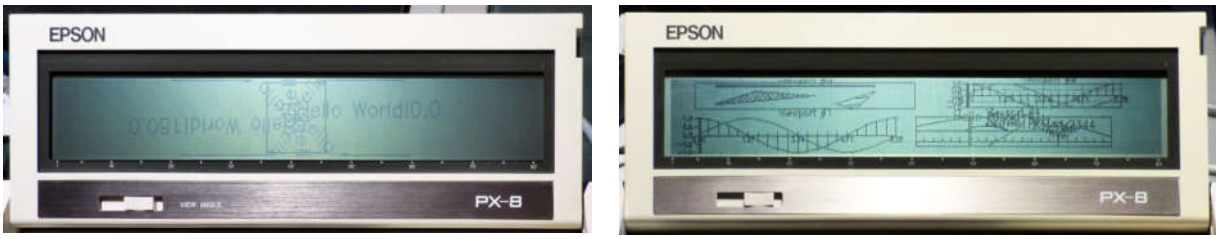
Because the GSXLIB core library is very basic, I have written another higher level library, called GLIB. This library makes use of the low-level library but provides more practical routines for setting up viewports and windows, drawing axes and grids as well as providing vector text output. Usually, you would use this library when writing real world programs. The routines provide a core for generating technical graphs.



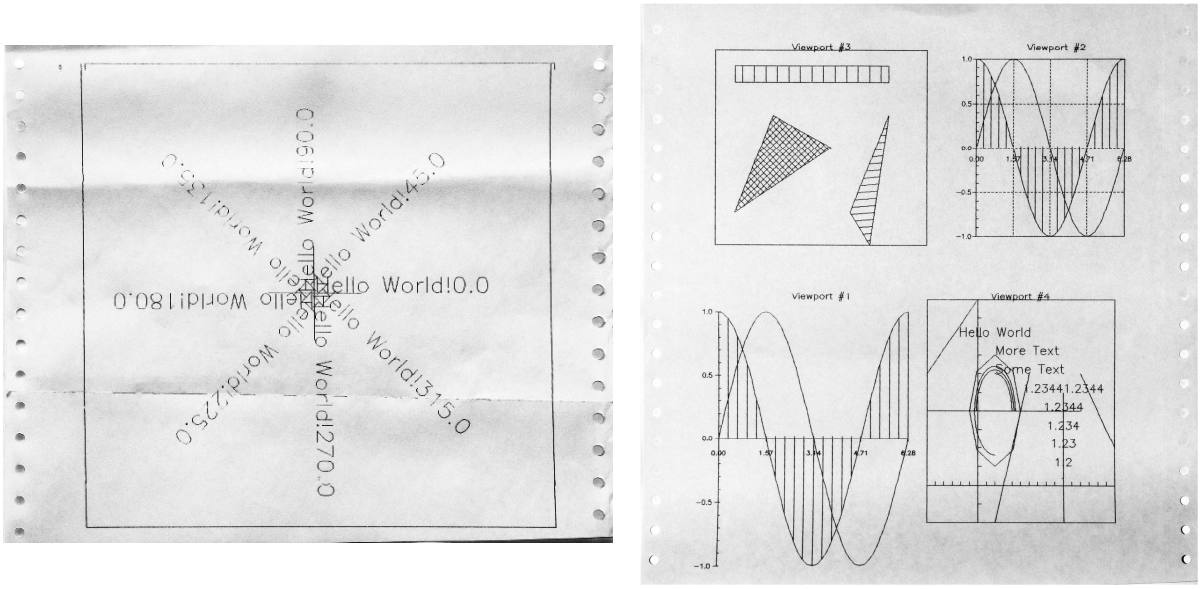
a) Teraterm Tektronix Window (DDXTEK.PRL).



b) HP 2648 Terminal (DDHP2648.PRL).



c) Epson PX-8 Laptop (DDPX8.PRL.



d) Epson MX-80 Printer DDFXHR8.PRL.

Figure 21: Examples of the output generated by the two demonstration programs GDEMO1 and GDEMO2.

Usage

The GLIB library uses a scaling system based on the concept of a viewport and a window. Think of drawing graphs on a sheet of paper. The paper corresponds to the full area of the output device.

The Viewport

The viewport defines the region of the paper to use for drawing. It is defined relative to the full size of the paper. You can also think of this as a percentage of the paper space. A viewport defined by an x-range of [0…1] and a y-range of [0.5…1.0] would cover the upper half of the output device. The default viewport covers the horizontal x = [0…1] and the vertical range y = [0…1], i.e. the full page.

The Window

The window defines your user coordinate (UC) system for the viewport. Most drawing routines use this system. If you want to plot days on the horizontal axes and hours on the vertical axis, you could define a window of x = [1…365] and y = [0…24]. This UC system is mapped into the viewport.

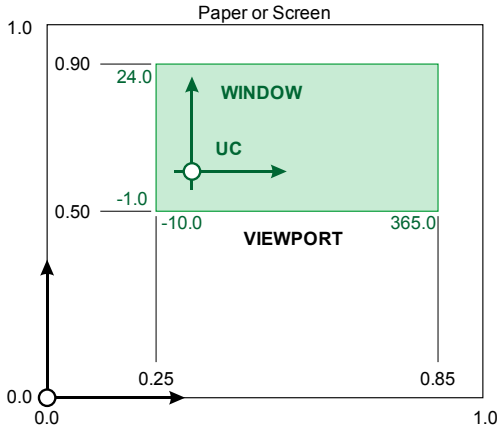


Figure 22: Setup of a Window for plotting hours versus days inside a Viewport [0.25…0.85, 0.5…0.9]. For demonstration, some extra margin has been added inside the Window, so that the origin of the green UC system is slightly offset.

Setting up your own UC system allows you to call GMOVE and GDRAW with day and hour numbers for the X and Y parameters.

After having plotted into this viewport, you could define another viewport on the same page, e.g. with an x-range of [0…1] and a y-range of [0.0…0.5]. Setting up the same window scaling and calling your plotting routine again would now produce the same plot as before, but in the lower half of the paper.

A typical GLIB program consists of this sequence (see `GDEMO1.FOR` and `GDEMO2.FOR`):

```
GINIT
        One or more blocks of:
        GVIEW
        GWIND
        … plotting routines …
GCLOSE
```

`GINIT` and `GCLOSE` are mandatory.

If not noted otherwise, all routines in the GLIB use the UC system. Only a few parameters for which a fixed size seems to be more desirable are given in millimeters (for example tick lengths and text heights). Depending on output device and device driver, these dimensions may not be accurate. Similarly, angles may be distorted if the output device has non-square pixels or the driver reports incorrect dimensions.

Also note that FORTRAN passes parameters by address (reference). By design, some routines update their input parameters, most notably the string output functions. This allows for easy writing a sequence of strings. Keep this in mind, when you reuse these parameters in the calling program.

The names of all routines in the GLIB library start with a 'G'. The names of internal common blocks start with "G$" to avoid collision with common blocks in your program.

The library contains the following subroutines (in alphabetical order):

Subroutine	Module	Parameters	
GARC(X,Y,R,PHI0,PHI1,DPHI) Draw a polygonal arc.	GARC	REAL*4 X,Y REAL*4 R REAL*4 PHI0 REAL*4 PHI1 REAL*4 DPHI	center point radius start angle (deg) end angle (deg) angular step (deg)
GCHAR(X,Y,ICHR,SIZE,ANGLE) Output a single character using a built-in vector font. X and Y are updated to the baseline starting point of the next character.	GCHAR	REAL*4 X,Y INTEGER*2 ICHAR REAL*4 SIZE REAL*4 ANGLE	starting point character code (32...126) character size in millimeters baseline angle (deg), 0=right, 90=up
GCLEAR Clear the output device screen.	GCLEAR	none	
GCLIP(LCLIP) Switch clipping to viewport on/off.	GCLIP	LOGICAL*1 LCLIP	clip flag, .TRUE. enable clipping .FALSE. no clipping
GCLOSE Close the output device. Must be called after all GLIB routines.	GCLOSE	none	
GDRAW(X,Y) Draw a line to the given point.	GDRAW	REAL*4 X,Y	point
GHATCH(X,Y,N,ANGLE,DIST) Hatch the interior of a convex polygon with straight lines at an angle.	GHATCH	REAL*4 X() REAL*4 Y() INTEGER*2 N REAL*4 ANGLE REAL*4 DIST	ordinates of corner points coordinates of corner points number of points in X and Y angle to x-axis in degrees distance between lines in millimeters
GINIT(ID) Select and open the device. Must be called before any other GLIB routine. Sets up defaults: - full-page viewport - a UC system [0.0...1.0] in x and y.	GINIT	INTEGER*2 ID (ASSIGN.SYS)	the GSX device ID
GMARK(X,Y) Plot a marker of the current type at the given point.	GMARK	REAL*4 X,Y	point
GMOVE(X,Y) Move pen to the given point.	GMOVE	REAL*4 X,Y	point
GNUM(X,Y,SIZE,ANGLE,VAL,NDIG,NALIGN) Output a REAL number. The X, Y point is updated to the baseline starting point of the next character. Alignment: NX, NY	GNUM	REAL*4 X,Y REAL*4 SIZE REAL*4 ANGLE REAL*4 VAL INTEGER*2 NDIG	starting point of text, updated to end point character size in millimeters angle of baseline (deg), 0=right, 90=up number to write number of digits behind

NY: 2 +-----+ 1 |A B C| 0 +-----+ NX: 0 1 2		the decimal point INTEGER*2 NALIGN 2-digits NX NY: alignment
GPOLY(X,Y,N) Draw a closed polygon through the points.	GPOLY	REAL*4 X() REAL*4 Y() INTEGER*2 N ordinates of corner points coordinates of corner points number of points in X and Y
GRECT(X0,Y0,X1,Y1) Draw a rectangle by 2 corner points.	GRECT	REAL*4 X0 REAL*4 Y0 REAL*4 X1 REAL*4 Y1 ordinate of corner point 1 coordinate of corner point 1 ordinate of corner point 2 coordinate of corner point 2
GSFONT(IFONT) Load a new font from disk. Font #1 has Latin characters, font #2 has Greek characters.	GCHAR	INTEGER*2 IFONT font number [1...2]
GSIZE(DEVXMM,DEVYMM) Inquire device size.	GINIT	REAL*4 DEVXMM REAL*4 DEVYMM width in millimeters height in millimeters
GSMARK(SIZE,NSTYLE,NCOLOR) Set size, style and color for subsequent markers.	GSMARK	REAL*4 SIZE INTEGER*2 NSTYLE INTEGER*2 NCOLOR marker size in millimeters marker style, 1=dot pen color index, 1=first
GSPEN(WIDTH,NSTYLE,NCOLOR) Set pen width, style and color for subsequent lines.	GSPEN	REAL*4 WIDTH INTEGER*2 NSTYLE INTEGER*2 NCOLOR pen width in millimeters pen style, 1=solid pen color index, 1=first
GTEXT(X,Y,T,NLEN,SIZE,ANGLE,NALIGN) Draw a text string. X and Y are updated to the baseline starting point of the next character. Alignment: NX, NY NY: 2 +-----+ 1 |A B C| 0 +-----+ NX: 0 1 2	GTEXT	REAL*4 X, Y INTEGER*1 T(LEN) INTEGER*2 NLEN REAL*4 SIZE REAL*4 ANGLE INTEGER*2 NALIGN position of start point character string ASCII codes in [32...126] number of characters size in millimeters baseline angle in degrees, 0=right, 90=up 2-digits NX NY: alignment
GVIEW(XLO,XHI,YLO,YHI) Define the viewport position.	GVIEW	REAL*4 XLO REAL*4 XHI REAL*4 YLO REAL*4 YHI left edge in unit right edge in unit bottom edge in unit system top edge in unit system
GWIND(XLO,XHI,YLO,YHI) Define UC system scaling for the current viewport.	GWIND	REAL*4 XLO REAL*4 XHI REAL*4 YLO REAL*4 YHI left edge of UC space right edge of UC space bottom edge of UC space top edge of UC space
GXALAB(X0,X1,YPOS,XSTEP,SIZE,ANGLE,NDIG) Label a horizontal axis.	GAXLAB	REAL*4 X0 REAL*4 X1 REAL*4 YPOS REAL*4 XSTEP REAL*4 SIZE REAL*4 ANGLE INTEGER*2 NDIG ordinate of start point ordinate of end point position of axis line distance between ticks text size in millimeters text angle (deg) number of digits behind the decimal point
GXAXIS(X0,X1,YPOS,XSTEP,YTICK) Draw a horizontal axis.	GAXIS	REAL*4 X0 REAL*4 X1 REAL*4 YPOS REAL*4 XSTEP REAL*4 YTICK start point end point position of axis distance of tick marks tick length in millimeters
GXRAST(X0,X1,XSTEP,Y0,Y1) Draw a grid of vertical lines.	GRAST	REAL*4 X0 REAL*4 X1 REAL*4 XSTEP REAL*4 Y0 REAL*4 Y1 ordinate of start point ordinate of end point distance between lines start point of each line end point of each line
GYALAB(Y0,Y1,XPOS,YSTEP,SIZE,ANGLE,NDIG) Label a vertical axis.	GAXLAB	REAL*4 Y0 REAL*4 Y1 REAL*4 XPOS REAL*4 YSTEP REAL*4 SIZE REAL*4 ANGLE INTEGER*2 NDIG start point end point position of axis distance of ticks text size in millimeters text angle in degrees number of digits behind the decimal point

GYAXIS(Y0,Y1,XPOS,YSTEP,XTICK) Draw a vertical axis.	GAXIS	REAL*4 Y0 start point REAL*4 Y1 end point REAL*4 XPOS position of axis REAL*4 YSTEP distance of tick marks REAL*4 XTICK tick length in millimeters
GYRAST(Y0,Y1,YSTEP,X0,X1) Draw a grid of horizontal lines.	GRAST	REAL*4 Y0 coordinate of start point REAL*4 Y1 coordinate of end point REAL*4 YSTEP distance between lines REAL*4 X0 start point of each line REAL*4 X1 end point of each line
Routines for internal usage	**Module**	**Parameters**
GAUTO(MIN,MAX,MAXDIV,LOW,HIGH,STEP) Determine "nice" bounds and step size for a given range without too many subdivisions. Useful for defining and labeling axes.	GAXIS	REAL*4 MIN start point of range REAL*4 MAX end point of range INTEGER*2 MAXDIV maximum number of divisions REAL*4 LOW lower bound (output) REAL*4 HIGH upper bound (output) REAL*4 STEP interval (output)
GDRAWC(X,Y) Draw a line to the given point. Clips to viewport. Used internally by GDRAW routine.	GDRAWC	REAL*4 X,Y point
GMOVEC(X,Y) Move pen to the given point. Clips to viewport. Used internally by GMOVE routine.	GMOVEC	REAL*4 X,Y point
GIDRAW(IX,IY) Draw a line to the given point in normalized device units (NDC). Used internally by GLIB routines.	GIDRAW	INTEGER*2 IX,IY point in NDC system
GIMOVE(IX,IY) Move the pen to the given point in normalized device units (NDC). Used internally by GLIB routines.	GIMOVE	INTEGER*2 IX,IY point in NDC system
GNUMST(VAL,NDIG,FMT,NLEN) Convert a REAL number to a string. Used internally by GLIB routines.	GNUM	REAL*4 VAL number to convert INTEGER*2 NDIG digits after decimal point INTEGER*1 FMT(20) a buffer for result INTEGER*2 NLEN length in FMT(1:LEN)
GCHSIZ(CCHR,SIZE,DX,DY) Determine the dimension of a character. Used internally by GLIB routines.	GCHSIZ	INTEGER*1 CCHR character code (32…126) REAL*4 SIZE height in millimeters REAL*4 DX width of character in UC REAL*4 DY height of character in UC
GTXSIZ(T,NLEN,SIZE,DX,DY) Determine the dimension of a string. Used internally by GLIB routines.	GTXSIZ	INTEGER*1 T(LEN) text string INTEGER*2 NLEN length of text REAL*4 SIZE height in millimeters REAL*4 DX width of string in UC REAL*4 DY height of string in UC
GRAST(X0,Y0,X1,Y1,XSTEP,YSTEP,NSTEPS) Draw a grid of parallel lines. Used internally by GLIB routines.	GRAST	REAL*4 X0 ordinate of start point REAL*4 Y0 coordinate of start point REAL*4 X1 ordinate of end point REAL*4 Y1 coordinate of end point REAL*4 XSTEP horizontal distance to shift REAL*4 YSTEP vertical distance to shift INTEGER*2 NSTEPS number of steps to perform

Building the Library

To compile the modules for the library you can use the `glib.sub` file with SUBMIT or manually issue these commands:

```
m80 =gcons
f80 =ginit
f80 =gclose
f80 =gclear
```

```
f80 =grect
f80 =garc
f80 =gwind
f80 =gview
f80 =gaxis
f80 =gaxlab
f80 =gspen
f80 =gsmark
f80 =gmark
f80 =grast
f80 =gnum
f80 =gtext
f80 =gchsiz
f80 =gchar
f80 =gmovec
f80 =gdrawc
f80 =gclip
f80 =gmove
f80 =gdraw
f80 =gimove
f80 =gidraw
f80 =gtrafo
f80 =gpoly
f80 =ghatch
```

The library must be built in this order to avoid unresolved references:

```
lib80
glib=ginit,gclose,gclear,gcons,grect,garc,gwind,gview,gaxis,gaxlab
grast,gnum,gtext,gchar,gmove,gdraw,gmovec,gdrawc,gclip,gimove,gidraw
gchsiz,gspen,gsmark,gmark,gtrafo,gpoly,ghatch
/E
```

Finally, the demonstration programs can be generated by these commands:

```
f80 =gdemo1
l80 gdemo1/N,gdemo1,glib/S,gsxlib/S,forlib/S,/E
gengraf gdemo1
```

Shown for `gdemo1`, the corresponding commands are used for `gdemo2` and `gdemo3`.

Internal Transformations

As a note to myself, I list the most common transformations (using internal variables from GLIB.FI)

from \ to	NDC	UC	mm
NDC		UC.x = XW0 + (NDC.x - DX - XV0)/SX UC.y = YW0 + (NDC.y - DY - YV0)/SY difference (e.g. for lengths or height) ΔUC.x = ΔNDC.x/SX ΔUC.y = ΔNDC.y/SY	mm.x = NDC.x * DEVXMM * SX / DEVSIX mm.y = NDC.y * DEVYMM *SY / DEVSIY
UC	NDC.x = DX + XV0 + (UC.x-XW0)*SX NDC.y = DY + YV0 + (UC.y-YW0)*SY difference (e.g. for lengths or height) ΔNDC.x = ΔUC.x*SX ΔNDC.y = ΔUC.y*SY		mm.x = ((UC.x - XW0)*SX + DY + YV0)* DEVXMM/ DEVSIX mm.y = ((UC.y - YW0)*SY + DY + YV0)* DEVYMM/ DEVSIY difference (e.g. for lengths or height) Δmm.x = ΔUC.x*SX* DEVXMM/ DEVSIX Δmm.y = ΔUC.y*SY* DEVYMM/ DEVSIY
mm	NDC.x = DEVSIX * mm.x / (DEVXMM*SX) NDC.y = DEVSIY * mm.y / (DEVYMM*SY)	UC.x = XW0 + (DEVSIX * mm.x / DEVXMM - DX - XV0)/SX UC.y = YW0 + (DEVSIY * mm.y / DEVYMM - DY - YV0)/SY difference (e.g. for lengths or height) ΔUC.x = DEVSIX * dΔmm.x / (DEVXMM*SX) ΔUC.y = DEVSIY * Δmm.y / (DEVYMM*SY)	
Frame		UC.x = XW0 + (DEVSIX * frame.x - DX - XV0)/SX UC.y = YW0 + (DEVSIY * frame.y - DY - YV0)/SY with mm to frame (frame.x = mm.x / DEVXMM) (frame.y = mm.y / DEVYMM)	

GEMVIEW

This is a small utility program to read and output GEM metafiles. You can use the driver DDMF to create a GEM metafile or use one created e.g. by DR Draw or other programs (DR Draw for CP/M-80 or CP/M-86 – DR Draw for MS-DOS uses a different format with extension PIX). With GEMVIEW, you can later send this file to a printer or translate it into plotter commands.

You start GEMVIEW with a command line like:

```
GEMVIEW -d:2 -r -o:2 c:file.gem
```

or, producing the same result:

```
GEMVIEW -c
-d:2 -r -o:2 c:file.gem
```

Parameter	Description	Default
`file`	the name of the metafile to read; without extension, "GEM" is assumed.	GSX.GEM
`-c ↵ cmdline`	read all parameters from a second command line to work around a known Turbo Pascal 3.01 for CP/M-80 command line length bug. Prompts with a '>'.	none
`-d:`*n*	n=1: set Debug level *1*, outputs one line for each record read. n=2: set Debug level *2*, outputs each record with its arguments.	0
`-o:`*id*	outputs to the device with the given *id*, as defined in ASSIGN.SYS.	1
`-r:`*n*	*n=1*: redirects PUN: and, LST: to the console and to a file GSX.LOG. *n=2*: redirects PUN:, LST: and, CON: to a file GSX.LOG. Any read to RDR: is satisfied with an ACK character.	0
`-s:`*n*	scale horizontal x-coordinates by integer *n*.	1
`-t:`*n*	scale vertical y-coordinates by integer *n*.	1
`-x:`*n*	shift horizontal x-coordinates by integer *n*.	0
`-y:`*n*	shift vertical y-coordinates by integer *n*.	0

Table 5: Available optional parameters.

Notes:

- GEMVIEW first tries to open the file with the given name. If this file does not exist, it appends 'GEM' to the name and tries again before failing.
- GEMVIEW is written in Turbo Pascal 3.01A. Unfortunately, Turbo Pascal for CP/M-80 has a bug with command line handling – it garbles the command line after 32 characters. If you want to specify many options, your command line may exceed this limit. In this case, you can shorten the name of your GEM metafile and omit the extension "GEM" to minimize the length of the command line.
  For command options up to 127 characters, use the –c option to enter the parameter list in a second command line. Wait until the GSX header has been displayed and the prompt '>' appears.
- GEMVIEW does not handle all GSX record types. Metafiles generated with later versions of GEM may contain many more record types than specified by the initial specification for GSX.
- GEMVIEW uses the header information to shift the coordinate values so that they fall into the positive integer range. Some metafiles contain incorrect size information in their header. As a result, some points may fall outside of the valid range [0…32767]. This typically results in partially reflected long lines across the image. In this case, you can try using the –x and –y options

to add integer offsets to the values in the metafile. Similarly, you can use the –s and –t options to scale the values e.g. by 2, as long as the integer range is not exceeded.

- The option –r:1 allows for testing drivers which communicate with RDR:, PUN: or LST: (for example, the plotter driver DDHP7470). With the option –r:2 CON: is captured too, so that no output is written to the screen (except for the initial GSX message).
  By TYPEing the GSX.LOG file, you can replay the commands to the terminal.
  Alternatively, you can capture the output using a terminal program like TeraTerm.

Figure 23: Digital Research advertisement for GSX software products.

The Digital VT100 Is Now A Graphical VT100.

That's right. What many professionals consider to be the best alphanumerics terminal around, the VT100* is better. A whole lot better.

The reason? Retro-Graphics.™ A breakthrough that transforms the VT100 into a high-performance graphics display terminal. With multiple character sizes. Dot-dash lines. Point plotting and vector drawing. Selective erase for quick, easy updating of the graphics display. In short, complete emulation of Tektronix® 4010 Series terminals. Which means complete compatibility with most existing graphics software, including Tektronix Plot 10™ and ISSCO's DISSPLA® and TELLAGRAF®.

Retro-Graphics VT100 graphics are all displayed on the 12" (diagonal) green-toned screen at 640 x 480 resolution. And refresh raster scan technology results in a bright, easy-to-read display even in high ambient light environments.

Retro-Graphics. Added Value From Digital Engineering.

Not Digital Equipment, mind you. Digital Engineering. A somewhat smaller but very bright group who are pioneers in retro-fit graphics. Adding value is nothing new for them. Just ask any of the thousands of Lear Siegler ADM-3A and 3A+ owners whose Dumb Terminals® became graphics powerhouses thanks to Retro-Graphics.

Of course, a great idea like Retro-Graphics always starts with a good one. And the VT100–featuring 96 upper/lower case ASCII characters, up to 132 characters per line, numeric and function keypad, detachable keyboard and a wide variety of screen customizing features–is a very good idea, indeed.

Together, they become one exceptional idea called the Retro-Graphics VT100. The perfect graphics addition for business, scientific and engineering design applications, regardless of whether you want to maintain DEC product continuity or are just looking for the highest quality graphics terminal at a cost hundreds less than the competition.

A Good Idea Is Where You Find It. And When.

Where you'll find the Retro-Graphics VT100, up and running, is at Info '80, New York Coliseum, New York City, October 6-9. Look for it at National Computer Communications' Booth #2132. Beginning in November, the Retro-Graphics VT100 will be sold through NCC as well as by selected other Digital Engineering distributors.

For more information on how and where to order, call or write Digital Engineering directly. As for when to order, we suggest immediately.

DIGITAL ENGINEERING
630 Bercut Drive
Sacramento, CA 95814
(916) 920-5800
TWX 910 367-2009

NATIONAL COMPUTER
Communications Corp.
260 West Avenue
Stamford, CT 06902
(203) 357-0004

*VT100™ is a trademark of Digital Equipment Corporation. Retro-Graphics™ is a trademark of Digital Engineering. Inc. Tektronix® and Plot 10™ are trademarks of Tektronix, Inc. DISSPLA® and TELLAGRAF® are registered trademarks of ISSCO.

Figure 24: Digital Engineering advertisement for the Retro-Graphics board for the VT-100.

VT100 GRAPHICS IN 4.4 MINUTES!

From Selanar Of Course!

SIMPLY ADD A CARD
Adding the Graphics 100 PC card to your existing VT100 or VT103 gives you the most versatile CRT terminal on the market today. No other component or CRT changes are required.

OR BUY OUR GRAPHICS TERMINAL
The Selanar GT100 combines the DEC VT100 and the Graphics 100 into a very competitively priced terminal, designed to fit your application.

NEED TEKTRONIX 4010 OUTPUT?
Our GT100 has all the capabilities of the VT100 plus our new Tektronix 4010 emulation mode. Create excellent graphics displays with packages like **PLOT 10, DISSPLA & TELAGRAF**, or any other package with 4010 output mode.

HOW ABOUT SOFTWARE?
Selanar supplies • Calcomp type Fortran subroutines for: RT-11, RSX, VMS. • Light pen support software • Hardcopy support software.

CALL US!
Let us show you how to get the most from your computer terminals.

(408) 727-2811

ISI SELANAR

ISI SELANAR INTERNATIONAL 2403 De La Cruz Blvd. Santa Clara, CA., 95050
EUROPEAN HEADQUARTERS: ISI COMPUTER Alte Landstrasse 5 D 8012 Ottobrunn Telefon (089) 60 60 71-72 Telex 5216290

Figure 25: ISI Selenar advertisement for their Graphics board for the VT-100.

References

[1] Langhorst, Fred E., Clarkson, Thomas B. III: “Realizing Graphics Standards for Microcomputers”, p. 256-268, BYTE Magazine, February 1983.
[2] “GSX Graphics Extension, Programmers Guide”, Digital Research Inc., 2nd ed., September 1983.
[3] “CBASIC Compiler Language, Graphics Guide”, Digital Research Inc., 1st ed., May 1983.
[4] Munk, Udo: CB80 and GSX software files, downloadable under https://www.autometer.de/unix4fun/z80pack/index.html [retrieved February 2020].
[5] Kotulla, Martin: “GSX ohne Geheimnisse”, Teil 1, p. 80-84, c’t 12, 1986; Teil 2, p. 116-123, c’t 1, 1987; Teil 3, p. 98-105, c’t 2, 1987; Teil 4, p. 124-127, c’t 3, 1987.
[6] Licher, Eckhard, von Massenbach, Thomas: “CP/M 2 lernt dazu”, Teil 1, p.124-135, c’t 1, 1987; Teil 2, p.78-85, c’t 2, 1987.
[7] Russ, Dave: “GSX – The Graphics Interface”, 80-Bus News, V3N5, September-October 1984.
[8] “FORTRAN-80 User’s Manual”, Microsoft, 1979.
[9] “FORTRAN-80 Reference Manual”, Microsoft, 1979.
[10] “Utility Software Package Reference Manual”, Microsoft, 1981.

Notes:

- Fred Langhorst, the author of the BYTE article, was DRI’s product manager for GSX.
- In 1988, the address of the co-developers of GSX and products like GSS-KERNEL was Graphic Software Systems, Inc., 25 117 Southwest Parkway, Wilsonville, OR 97070-9600, USA They also provided implementations of GKS for the PDP-I 1 under RT-11, RSX-11M and Unix.
- The driver DDGDC.PRL shows a banner, but then hangs when loading into a generic CP/M 2.2 system. It probably tries to initialize (nonexistent) hardware with the 7220 graphics chip. The company “miro” later also produced graphics cards for IBM-PC compatibles and Apples, focusing on the professional market.

```
GSX-80/GIOS 1.3  for miroGDC              31-AUG-83
Copyright (C) 1983
miro Datensysteme GmbH          All Rights Reserved
-------------------------------------------------
```

- During disassembly of some drivers supplied by DRI (DDMX80.PRL and DDHP7470.PRL) I noticed that these came with an appended copyright notice block of 256 bytes. This block is not displayed and serves no functional purpose. It is simply appended to the driver file.
- Thanks go to Emmanuel Roche for proofreading this document and providing comments and historic insight. As this is a living document, I am responsible for any remaining and newly introduced errors.