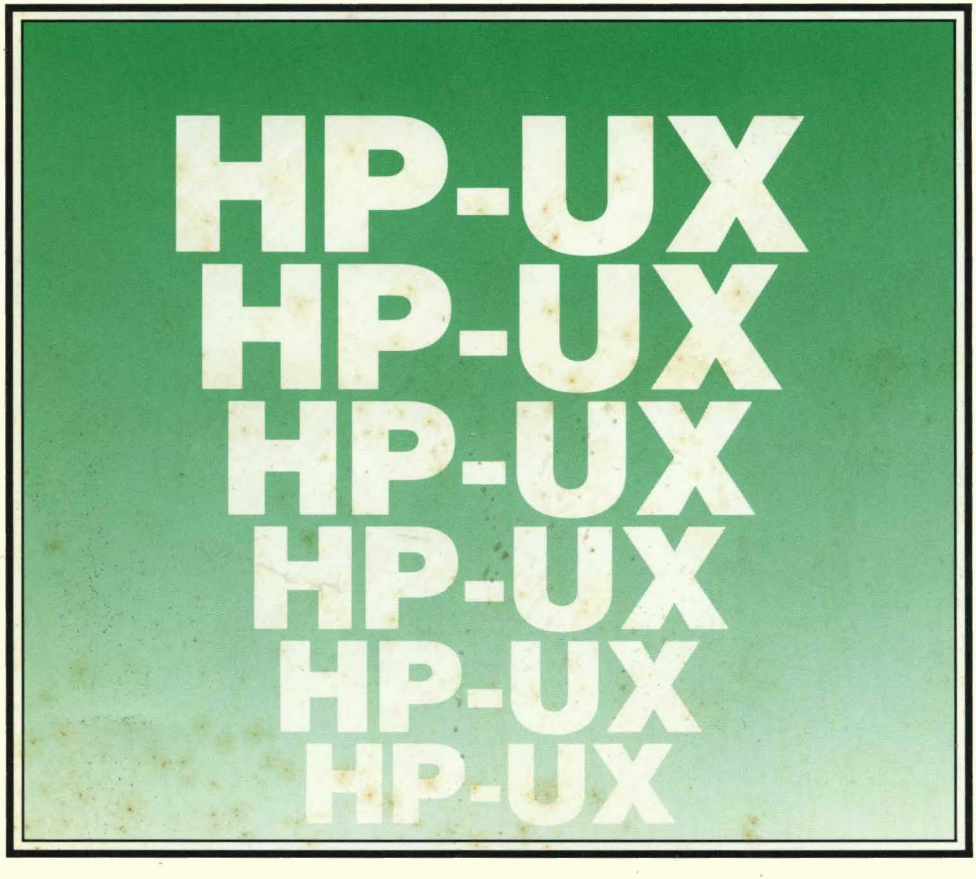
HEWLETT PACKARD

HP-UX Concepts and Tutorials

Vol. 6: Graphics



HP-UX Concepts and Tutorials
Vol. 6: Graphics

Manual Reorder No. 97089-90070

Hewlett-Packard Company
3404 East Harmony Road, Fort Collins, Colorado 80525

Printing History

New editions of this manual will incorporate all material updated since the previou edition. Update packages may be issued between editions and contain replacement an additional pages to be merged into the manual by the user. Each updated page will b indicated by a revision date at the bottom of the page. A vertical bar in the margi indicates the changes on each page. Note that pages which are rearranged due to change on a previous page are not considered revised.

The manual printing date and part number indicate its current edition. The printin date changes when a new edition is printed. (Minor corrections and updates which ar incorporated at reprint do not cause the date to change.) The manual part numbe changes when extensive technical changes are incorporated.

July 1984...First Edition – Part numbered 97089-90004 was 4 volumes and was shippe with HP-UX 4.0 on Series 500 Computers and with HP-UX 2.1, 2.2, 2.3, and 2.4 o Series 200 Computers. Each volume did not have an individual part number. Thi was obsoleted in April, 1985 and replaced with Manual Kit #97070-87903 whic includes:

Title	Manual P/N	Binder P/N
Vol. 1: Text Processing and Formatting	97089-90020	9282-1023
Vol. 2: Programming Environment	97089-90030	9282-1023
Vol. 3: Software Development Tools	97089-90040	9282-1023
Vol. 4: Shells and Miscellaneous Tools	97089-90050	9282-1023
Vol. 5: Data Communications	97089-90060	9282-1023
Vol. 6: Graphics	97089-90070	9282-1023

April 1985...Edition 1 – Volume 6: Graphics

Contents

The articles contained in *HP-UX Concepts and Tutorials* are provided to help you use the commands and utilities provided with HP-UX. The articles have several sources. Some were written at Hewlett-Packard specifically for HP computers. Others were written at Bell Laboratories or University of California at Berkeley and have been tailored for HP computers.

HP-UX Concepts and Tutorials has six volumes:

- Volume 1: Text Processing and Formatting
- Volume 2: Programming Environment
- Volume 3: Software Development Tools
- Volume 4: Shells and Miscellaneous Tools
- Volume 5: Data Communications
- Volume 6: Graphics

This is "Vol. 6: Graphics" and the article it includes is:

1. Starbase

Warranty Statement

Hewlett-Packard products are warranted against defects in materials and workmanship. For Hewlett-Packard computer system products sold in the U.S.A. and Canada, this warranty applies for ninety (90) days from the date of shipment.* Hewlett-Packard will, at its option, repair or replace equipment which proves to be defective during the warranty period. This warranty includes labor, parts, and surface travel costs, if any. Equipment returned to Hewlett-Packard for repair must be shipped freight prepaid. Repairs neccessitated by misuse of the equipment, or by hardware, software, or interfacing not provided by Hewlett-Packard are not covered by this warranty.

HP warrants that its software and firmware designated by HP for use with a CPU will execute its programming instructions when properly installed on that CPU. HP does not warrant that the operation of the CPU, software, or firmware will be uninterrupted or error free.

NO OTHER WARRANTY IS EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. HEWLETT-PACKARED SHALL NOT BE LIABLE FOR CONSEQUENTIAL DAMAGES.

HP 9000 Series 200

For the HP 9000 Series 200 family, the following special requirements apply. The Model 216 computer comes with a 90-day, Return-to-HP warranty during which time HP will repair your Model 216, however, the computer must be shipped to an HP Repair Center.

All other Series 200 computers come with a 90-Day On-Site warranty during which time HP will travel to your site and repair any defects. The following minimum configuration of equipment is necessary to run the appropriate HP diagnostic programs: 1) .5 Mbyte RAM; 2) HP-compatible 3.5″ or 5.25″ disc drive for loading system functional tests, or a system install device for HP-UX installations; 3) system console consisting of a keyboard and video display to allow interaction with the CPU and to report the results of the diagnostics.

To order or to obtain additional information on HP support services and service contracts, call the HP Support Services Telemarketing Center at (800) 835-4747 or your local HP Sales and Support office.

*For other countries, contact your local Sales and Support Office to determine warranty terms.

Table of Contents

Section 1: An Introduction to Starbase Graphics

Graphics with C 1
Welcome 1
Preparation 2
Device Drivers 3
A Shell Script 3
The Data to be Plotted 5
The Data in Clist Format 6
The Results Desired 7
A Template C Program 8
The Include File 8
The Gopen Function 9
The Gclose Function 10
A Review 11
Virtual Device Coordinates 12
Device Coordinates 12
World Coordinates 13
Setting the VDC Extent 14
Clipping 15
Isotropic (undistorted) Mapping 16
Anisotropic (distorted) Mapping 18
Viewports 19
Rectangles 20
Plotting the Data 22
Labels 24
Tic Marks 26
The Final Picture 27
The whole program 28
Graphics with Fortran77 31
Welcome 31
Preparation 32
Device Drivers 33
A Shell Script 33
The Data to be Plotted 35
The Data as a Clist 36
A Template Fortran77 Program 37
The Include File 38
The Gopen Function 38
The Gclose Function 39

A Review 40
Virtual Device Coordinates 40
Device Coordinates 41
World Coordinates 41
Setting the VDC Extent 42
Clipping 43
Isotropic (undistorted) Mapping 44
Anisotropic (distorted) Mapping 46
Viewports 47
Rectangles 48
Plotting the Data 50
Labels 52
Tic Marks 54
The Final Picture 55
The Whole Program 56
Graphics with Pascal 59
Welcome 59
Preparation 60
Device Drivers 60
A Shell Script 61
The Data to be Plotted 62
The Data as a Clist 63
The Results Desired 64
A Template Pascal Program 65
The Include Files 65
The Gopen Function 66
The Gclose Function 67
A Review 68
Virtual Device Coordinates 68
Device Coordinates 69
World Coordinates 69
Setting the VDC Extent 70
Clipping 71
Isotropic (undistorted) Mapping 72
Anisotropic (distorted) Mapping 74
Viewports 75
Rectangles 76
Plotting the Data 78
Labels 80
Tic Marks 82
The whole program 84

Section 2: Working with Text

Introduction 87
The Four Methods of Drawing Text 88
Text Attributes and Defaults 90
Template Programs 91
A Template for C Programs 92
A Template for Fortran77 Programs 92
A Template for Pascal Programs 93
Default Text 94
C Syntax 94
Fortran77 Syntax 94
Pascal Syntax 95
Larger Text 96
C Syntax 96
Fortran77 Syntax 96
Pascal Syntax 96
Larger Text Continued 98
C Syntax 98
Fortran77 Syntax 98
Pascal Syntax 99
Text Precision 100
Wide and Narrow Characters 101
C Syntax 101
Fortran77 Syntax 101
Pascal Syntax 102
Character Expansion Factor 103
C Syntax 103
Fortran77 Syntax 103
Pascal Syntax 104
Intra Character Space 106
C Syntax 106
Fortran77 Syntax 106
Pascal Syntax 107
Character Slant 108
C Syntax 108
Fortran77 Syntax 108
Pascal Syntax 109
Text Path 111
C Syntax 111
Fortran77 Syntax 111
Pascal Syntax 112
Text Line Path 113

C Syntax 113
Fortran77 Syntax 113
Pascal Syntax 114
Text Orientation 115
C Syntax 115
Fortran77 Syntax 116
Pascal Syntax 116
Text Line Space 118
Fortran77 Syntax 118
Pascal Syntax 119
Text Alignment 120
Text alignment - TA_LEFT 122
C Syntax 122
Fortran77 Syntax 122
Pascal Syntax 123
Text alignment - TA_CENTER 124
C Syntax 124
Fortran77 Syntax 124
Pascal Syntax 125
Text alignment - TA_RIGHT 126
C Syntax 126
Fortran77 Syntax 126
Pascal Syntax 127
Text alignment - TA_CONTINUOUS_HORIZONTAL #1 128
C Syntax 128
Fortran77 Syntax 128
Pascal Syntax 129
Text alignment - TA_CONTINUOUS_HORIZONTAL #2 130
C Syntax 130
Fortran77 Syntax 130
Pascal Syntax 131
Text alignment - TA_NORMAL_HORIZONTAL 132
C Syntax 132
Fortran77 Syntax 132
Pascal Syntax 133
Text alignment - TA_TOP 134
C Syntax 134
Fortran77 Syntax 134
Pascal Syntax 135
Text alignment - TA_CAP 136
C Syntax 136
Fortran77 Syntax 136

Pascal Syntax 137
Text alignment - TA_HALF 138
C Syntax 138
Fortran77 Syntax 138
Pascal Syntax 139
Text alignment - TA_BASE 140
C Syntax 140
Fortran77 Syntax 140
Pascal Syntax 141
Text alignment - TA_BOTTOM 142
C Syntax 142
Fortran77 Syntax 142
Pascal Syntax 143
Text alignment - TA_CONTINUOUS_VERTICAL Example #1 144
C Syntax 144
Fortran77 Syntax 144
Pascal Syntax 145
Text alignment - TA_CONTINUOUS_VERTICAL Example #2 146
C Syntax 146
Fortran77 Syntax 146
Pascal Syntax 147
Text alignment - TA_NORMAL_VERTICAL 148
C Syntax 148
Fortran77 Syntax 148
Pascal Syntax 149
Designate Character Set 150
Text Font Index 153
C Syntax 153
Fortran77 Syntax 153
Pascal Syntax 154
Text Color 155

Section 3: Starbase Color Graphics

Introduction 157
Color Generation 160
An Overview 160
The Frame Buffer 162
Color Categories 162
The Color Map 163
Color Planes 163
Color Map Entries 163
Color Generation Hardware 163

Selecting a Color 164
Shell Scripts 164
A Script for C Programs 165
A Script for Fortran77 Programs 165
A Script for Pascal Programs 166
Skeleton Programs 167
A Template Program for C 167
A Template Program for Fortran77 167
A Template Program for Pascal 168
Example Color Programs 168
A Color Map 168
Background Color 170
Polyline and Polymarker Color 171
The Color Cube 174

Section 4: Input
Introduction 177
Locator Devices 177
Choice Devices 177
Triggering 178
Input Methods 178
Sampling 178
Requests 178
Events 179
Tracking 179
Sampling 180
Example 1 181
Example 2 182

Modeling and Viewing
Introduction 185
Two-Dimensional Viewing 186
Example 1: 187
Example 2: 187
Using The Transformation Matrix Stack 188
Example 3: 189
Three-Dimensional Viewing 190
Multiple Active Devices 191

An Introduction to Starbase Graphics 1

Graphics with C

Welcome

This manual is designed to teach you how to use some fundamental procedures that are included in the Starbase Graphics Library on your HP-UX system. These procedures can be accessed from the C, Fortran77 and Pascal programming languages available for your system. This subsection will present a progressive example using the C Programming Language. The following subsections cover the same material using the Fortran77 Programming Language and the the Pascal programming language.

This manual was written with the assumption that you are familiar with your HP-UX Operating System, including the appropriate programming languages, compilers, linkers, editors, etc. If this is not the case, the information you need is available in the extensive documentation provided with your system.

This manual also assumes that you are familiar with basic computer graphics concepts. If this is not the case, there are many excellent books and courses available to provide this information.

The example programs used in this tutorial are included with your system in the `/usr/lib/starbase/demos` directory. You are encouraged to compile and run these programs as you read the manual. When you see how the programs work, modify them as you desire to see how each part contributes to the whole. Be aware that the example programs assume a system configuration which may differ from yours, and that the **gopen** procedure parameters and the device drivers linked to the program may need changing to match your system.

NOTE

Demonstration programs and routines in `/usr/lib/starbase/demos` are for the purpose of instruction only. They are not part of the HP-UX package, and as such, they are not covered by any warranty, expressed or implied. Hewlett Packard shall not be liable for incidental or consequential damages in connection with, or arising out of, the furnished, performance, or use of these routines.

Preparation

The directions concerning the execution of the example programs presented in this manual assumes your current directory is `/usr/lib/starbase/demos`. To make this directory your current working directory, use the **c**hange **d**irectory (`cd`) command as follows:

```
cd /usr/lib/starbase/demos
```

A discussion of the hierarchial directory structure used by the HP-UX Operating System is found in the *Systems Administrator* manual supplied with your system.

For further details on the `cd` command, use the `man` command. For example,

```
man cd
```

The `man` command prints the reference page for the specified command on the display.

The example programs presented in this manual are included with your operating system. They are located in the `/usr/lib/starbase/demos` directory. The examples are provided as source code. To read the code, use the HP-UX `more` or `cat` commands. To modify the code, use an appropriate editor, such as the `vi` editor.

To execute the example programs on other devices, they must be compiled and linked to the correct drivers. The shell script below will accomplish this for you.

Device Drivers

The Starbase device drivers are located in the directory `/usr/lib` and have file names appropriate to access by the "-l" option of the **c** compiler (cc) command.

Take a moment and list the device driver files in `/usr/lib` to see which device drivers are available with your system. All device driver file names begin with `libdd` prefix, followed by the device type. To list these files, type in:

```
ls /usr/lib/libdd*
```

As you can see, there are several drivers to choose from.

The HP-GL device driver is (`/usr/lib/libddhpgl.a`) and the HP 262x device driver is (`/usr/lib/libdd262x.a`). These are the device drivers included in the example programs.

For further details concerning the Starbase Device Drivers, read the appropriate sections in the *Starbase Device Drivers Library* manual.

A Shell Script

The following shell script was used to compile and execute the example programs discussed in this manual. The script works with both the Bourne Shell and the C Shell (the C shell calls the Bourne shell automatically).

The "-l" option was used to reduce typing. This compiler option prepends `/usr/lib/lib` to the filename and appends `.a` to the file name. Thus,

```
... -lddhpgl
```

is processed as if it were

```
... /usr/lib/libddhpgl.a
```

The HP-GL and HP 262x device drivers were linked in by the script. The `sb1` and `sb2` files are included to resolve all unresolved references generated by the Starbase programs.

```
PROG=$1
shift
cc -o $PROG $PROG.c $* -lddhpgl -ldd262x -lsb1 -lsb2
$PROG
```

This script is stored as an executable file under the file name `CC` in:

```
/usr/lib/starbase/demos.
```

To compile and execute the example program `example1c.c`, enter the following:

```
CC example1c
```

To execute the compiled program `example1c` again, type in:

```
example1c
```

If you want the same program executed on an HP-GL plotter, type in:

```
example1c /dev/hpgl hpgl
```

To execute the example with other devices, you must link the correct devices to the program.

To link the HP 98700 driver using this script, just type in:

```
CC example1c -ldd98700
```

To execute the example after being linked with the HP 98700 device driver, just type in:

```
example1c /dev/hp98700 hp98700
```

The Data to be Plotted

The following data is to be presented in graphical form. The example program to do this follows.

Table 1-1. Test Data

Test	Value	Test	Value	Test	Value	Test	Value
0	0.1610	1	0.1625	2	0.1625	3	0.1628
4	0.1636	5	0.1631	6	0.1627	7	0.1608
8	0.1610	9	0.1606	10	0.1607	11	0.1617
12	0.1614	13	0.1626	14	0.1634	15	0.1640
16	0.1656	17	0.1660	18	0.1644	19	0.1651
20	0.1635	21	0.1641	22	0.1628	23	0.1619
24	0.1630	25	0.1624	26	0.1627	27	0.1644
28	0.1644	29	0.1657	30	0.1660	31	0.1670
32	0.1672	33	0.1666	34	0.1658	35	0.1662
36	0.1646	37	0.1633	38	0.1634	39	0.1636
40	0.1645	41	0.1652	42	0.1656	43	0.1677
44	0.1689	45	0.1680	46	0.1696	47	0.1680
48	0.1674	49	0.1677	50	0.1669	51	0.1655
52	0.1665	53	0.1662	54	0.1667	55	0.1668
56	0.1681	57	0.1688	58	0.1687	59	0.1707
60	0.1716	61	0.1716	62	0.1694	63	0.1698
64	0.1683	65	0.1683	66	0.1671	67	0.1681
68	0.1683	69	0.1684	70	0.1681	71	0.1698
72	0.1705	73	0.1723	74	0.1730	75	0.1734
76	0.1714	77	0.1722	78	0.1716	79	0.1696
80	0.1702	81	0.1699	82	0.1684	83	0.1706
84	0.1696	85	0.1715	86	0.1730	87	0.1737
88	0.1739	89	0.1751	90	0.1732	91	0.1747
92	0.1729	93	0.1717	94	0.1710	95	0.1707
96	0.1706	97	0.1709	98	0.1713	99	0.1720

The Data in Clist Format

The first column shows the first few file entries as a clist without move/draw indicators. The second column shows the first few data entries as a clist with move(1)/draw(0) indicators.

Table 1-2. Data in Clist Format

Clist without Move/Draw Indicators	Clist with Move/Draw Indicators
0	0
0.1610	0.1610
1	0 (a move indicator)
0.1625	1
2	0.1625
0.1625	1 (a draw indicator)
3	2
0.1628	0.1625
4	1 (another draw indicator)
0.1636	3
5	0.1628
0.1631	1
6	4
0.1627	0.1636
7	1
0.1608	5
8	0.1631
0.1610	1
9	6
0.1606	0.1627
10	1
0.1607	7
11	0.1608
0.1617	1
12	8
0.1614	0.1610
13	1
0.1626	9
14	0.1606
0.1634	1
15	10
0.1640	0.1607
16	1
0.1656	11
17	0.1617
0.1660	1
18	12
0.1644	0.1614
19	1
0.1651	13

The Results Desired

The following picture shows the plot to be created by the example programs described in the subsection.

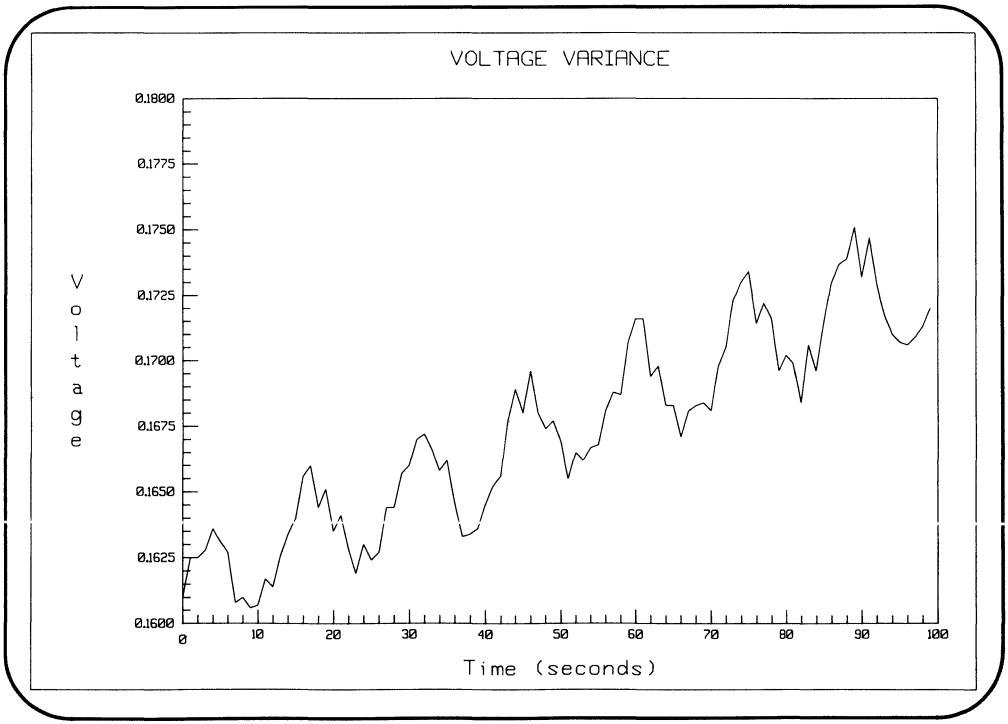


Figure 1-1. The Results Desired

A Template C Program

The following is a template program that will be filled with Starbase procedure and function calls to draw the example voltage plot. As listed, this program opens the specified device, clears the clipping area and then closes the device. The capability of executing the program on an HP262x Graphics Terminal or a specified device is included.

For more information on C programming, read *The C Programming Language* by Brian W. Kernighan and Dennis M. Ritchie. This book is included with your HP-UX Manual Set.

This program is stored as *example1c.c.*

```
#include <starbase.c.h>

main(argc,argv)
int argc; char *argv[];
{
     int fildes;

     if (argc > 2) fildes=gopen(argv[1],OUTDEV,argv[2],INIT);
     else fildes=gopen("/dev/tty",OUTDEV,"hp262x",INIT);
     if (fildes == -1) exit(-1);

     gclose(fildes);
}
```

The Include File

The file `starbase.c.h` contains the definitions and terms needed for any Starbase program written in C. This file is located in the `/usr/include` directory.

To see what definitions and terms are needed, use the `more` command. The syntax is:

```
more /usr/include/starbase.c.h
```

The Gopen Function

The function `gopen` opens the specified graphic device. This function requires four parameters:

1. **Device File Name** - Device files are created by the System Administrator. Device files are located in the `/dev` directory. In the following examples, the device file `tty` is used.
2. **Performance Mode** - This parameter may be one of the following:

- OUTDEV - device driver is to output information to the device.
- INDEV - device driver is to read information from the device.
- OUTINDEV - device driver can send and receive information to and from the device.

The following examples use OUTDEV since the example program is to be drawn on the output device's display.

3. **Character Representation of the Driver Type** - For the following example, this will be `hp262x`. This parameter identifies the driver to be used. The `hp262x` driver interrogates the device specified by the device file name and tests to see if the device is an HP 2623 Graphics Terminal or an HP 2627 Color Graphics Terminal, and acts accordingly.

   A possible character representation for an HP-GL plotter is `hpgl`.

4. **Operation Mode** - This parameter may be one of the following:

   0 - open the device, but do nothing else.
   INIT - open and initialize the device in a device dependent way.
   RESET - open and completely initialize the device.
   SPOOLED - open the device for spooled operation. Spooled output may be saved in a file for later display.
   THREE_D - open the device and set Starbase to 3-dimensional mode.

   These modes may be combined with the proper OR syntax. For C this is the vertical bar (|) as in (INIT|SPOOLED).

By doing an OR operation on INIT and SPOOLED, the graphic device can be opened, initialized and spooled to a file.

For example:

```
fildes = gopen("spoolfile",OUTDEV,"hpgl",INIT|SPOOLED);
```

If the opening is not successful, `gopen` returns a value of `-1`. If the opening is successful, a positive integer (file descriptor) is returned and the HP-GL device is initialized and ready for spooled output.

Further information can be obtained about this procedure by using the `man` command. Just type in:

```
man 3 gopen
```

The Gclose Function

To properly close a graphics device, use the `gclose` function. The `fildes` file descriptor identified by the `gopen` function is used to identify a particular device for all following Starbase procedures and functions, including `gclose`.

Further information can be obtained about this procedure by using the `man` command. Just type in:

```
man 3 gclose
```

A Review

As a review, the following table lists information about the files, functions and commands just discussed.

Table 1-3. A Review

Required Items	Learn More
The cd Command	/usr/man/man1/cd.1 (man cd)
The cc Compiler Command	/usr/man/man1/cc.1 (man cc)
Shell Scripts	HP-UX Concepts and Tutorials, volume 3.
Device Drivers	Starbase Device Drivers Library. /usr/lib/libdd* (ls /usr/lib/libdd...)
C Programming	The C Programming Language by Kernighan and Ritchie
Definitions and Terms needed for any Starbase program in C.	/usr/include/starbase.c.h (more /usr/include/starbase.c.h)
gopen	/usr/man/man3/gopen.3g (man 3 gopen)
gclose	/usr/man/man3/gclose.3g (man 3 gclose)

Virtual Device Coordinates

The default Cartesian coordinate system used by the Starbase procedures is called the Virtual Device Coordinate (VDC) system. This system can be considered a 2-dimensional subset of the Cartesian plane with default x- and y-axis values ranging from 0.0 to 1.0. In 3-dimensional graphics, this system can also be considered a subset of Cartesian 3-space with default x-, y- and z-axis values ranging from 0.0 to 1.0. Locations in VDC space are specified in VDC values as specified by the `vdc_extent` procedure.

The procedure `gopen` automatically creates a Virtual Device Coordinate to Device Coordinate (DC) transformation matrix for each opened graphic device. All VDC data is processed through this matrix before being set to the target device. Since each device has a unique transformation matrix, the same data will appear as expected on each device.

The range of values used for this system can be altered with the `vdc_extent` procedure.

Use the `push_vdc_matrix` procedure to force the current working matrix to be the VDC to DC transformation matrix.

Device Coordinates

Device coordinate Space is the Cartesian 2-dimensional space that is the device's output surface. The range and magnitude of x- and y-axis values is defined by each device.

Device Coordinate procedures bypass the VDC to DC transformation matrix. Such procedures begin with the characters *dc*, for "device coordinate". For example, `dcdraw` will draw a line on the specified device using the untransformed coordinate values specified.

To transform VDC values to DC values, use the `vdc_to_dc` procedure.

World Coordinates

World coordinates can have any value the user desires. This is handy when the data is to be used for purposes other than making a drawing.

Since world coordinates can be of any value, they must be transformed to device coordinates before being drawn on a graphic device. This transformation can be done in several ways. For example:

- A transformation matrix with the appropriate values to transform the World Coordinate data into Device Coordinate data may be supplied by the user.
- A transformation matrix with the appropriate values to transform the World Coordinate data into Virtual Device Coordinate data may be supplied by the user. This matrix could then be concatenated (matrix multiplied) with the Virtual Device Coordinate to Device Coordinate transformation matrix to make a World Coordinate to Device Coordinate Matrix.

For instance, in 2-dimensional mode, a matrix containing:

$$\begin{matrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \\ 0 & 0 \end{matrix}$$

will rotate world coordinates counter-clockwise by an angle of θ.

- **The Easiest Way** is to use the `vdc_extent` procedure to change the ranges of values used to define the Virtual Device Coordinate Space so that the world coordinate values are now contained in VDC space. Any time the `vdc_extent` procedure is used, a new Virtual Device Coordinate to Device Coordinate matrix is calculated that maps Virtual Device Coordinate Space into Device Coordinate Space.

Setting the VDC Extent

In the example problem, we want to plot voltage values against the associated measurement number. Our voltage data ranges from just over 0.1600 through just under 0.1800, while we have 100 measurements (from 1 to 100).

Logical values for the vdc extent would be -20 to 105 on the x-axis and 0.1575 to 0.1825 in the y-axis. This will allow us to plot the data in a rectangle of size 0-100 on the x-axis and 0.1600-0.1800 on the y-axis and still leave room for labels within the vdc extent and outside the plotting area. Since we are doing 2-dimensional graphics, the z-coordinates are both 0.0. The syntax to do this is:

```
vdc_extent(fildes,-20.0,0.1575,0.0,105.0,0.1825,0.0);
```

Detailed information on this procedure can be found using the HP-UX `man` command as follows:

```
man 3 vdc_extent
```

NOTE

The decision was made to follow ANSI parameter patterns in the Starbase procedures. These parameter patterns are NOT CONSISTENT. One procedure may call for x1,y1,z1,x2,y2,z2 while the next may require x1,x2,y1,y2,z1,z2. Hopefully this inconstancy will be removed from the ANSI standard and thus from the Starbase procedures.

Clipping

Clipping is the process of not allowing selected graphic data to be displayed. There are two clipping processes:

- Hard Clip - This is clipping done by the graphic output device that physically stops plotting outside the physical limits of the device's display area. For example, a plotter's pen can not draw past the physical edge of the plotter's platen.
- Soft Clip - This is clipping done by the graphics software to limit the visible portion of the plot. Soft clipping can be done to both 2-dimensional and 3-dimensional data.

Starbase provides two clipping boundaries:

- The Clip Rectangle - A rectangle can be specified with the `clip_rectangle` procedure that defines subsequent clipping boundaries.
- The VDC Extent - The rectangle defined by the `vdc_extent` procedure can also be used to define subsequent clipping boundaries.

  The `clip_indicator` procedure identifies which of the two boundaries to clip to.

The `clip_depth` procedure is used to define the front and back clipping planes for 3-dimensional graphics. Front and back clipping is enabled or disabled with the `depth_indicator` procedure.

Isotropic (undistorted) Mapping

Isotropic mapping is when one unit in the x-axis **exactly equals** one unit in the y-axis. This is sometimes referred to as *undistorted* plotting.

The `mapping_mode` procedure sets the mapping mode to either anisotropic or isotropic plotting mode. The default is isotropic.

The following example program draws an **isotropic** square around the default Virtual Device Coordinate space. This program is in file *example2c.c*.

```
#include <starbase.c.h>

main(argc,argv)
int argc; char *argv[];
{
     int fildes;

     if (argc > 2) fildes=gopen(argv[1],OUTDEV,argv[2],INIT);
     else fildes=gopen("/dev/tty",OUTDEV,"hp262x",INIT);
     if (fildes == -1) exit(-1);

     interior_style(fildes,INT_HOLLOW,TRUE);
     rectangle(fildes,0.0,0.0,1.0,1.0);

     gclose(fildes);
}
```

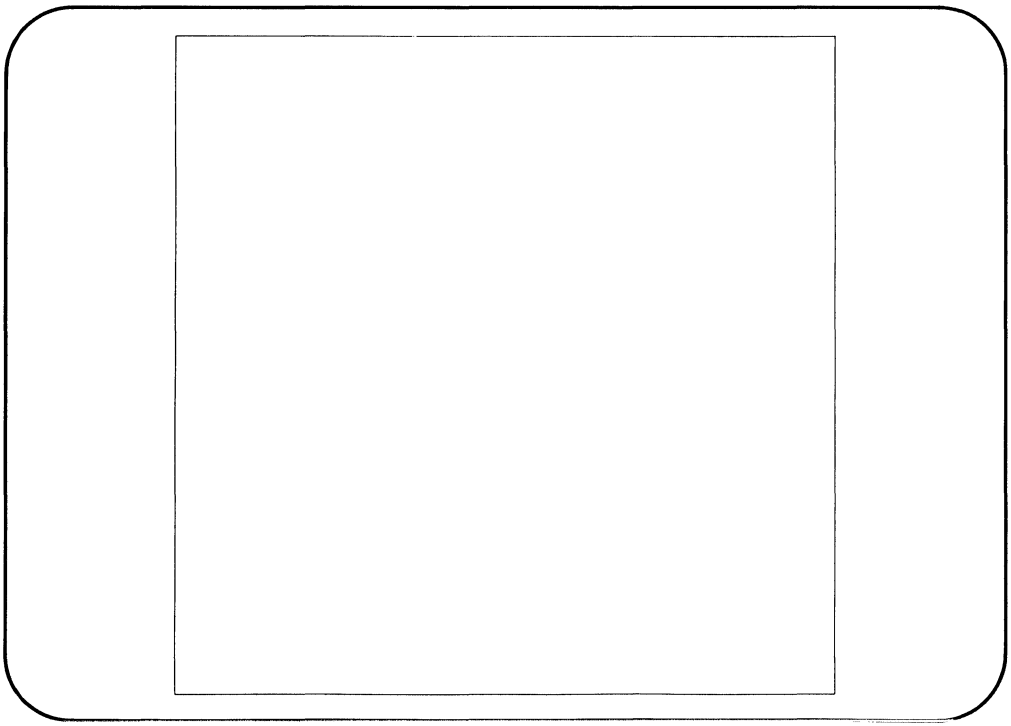


Figure 1-2. An Isotropic Frame of VDC Space

Anisotropic (distorted) Mapping

Anisotropic mapping where one unit in the X direction does not have to equal one unit in the Y direction. This is sometimes called *distorted* plotting.

The `mapping_mode` procedure sets the mapping mode to either anisotropic or isotropic plotting mode. The following single line was added to the preceeding isotropic program and the anisotropic results are shown in Figure 1-3. The rectangle still frames the VDC default space, but now the space fills the usable portion of the display. This program is in *example3c.c*.

```
mapping_mode(fildes,TRUE);
```

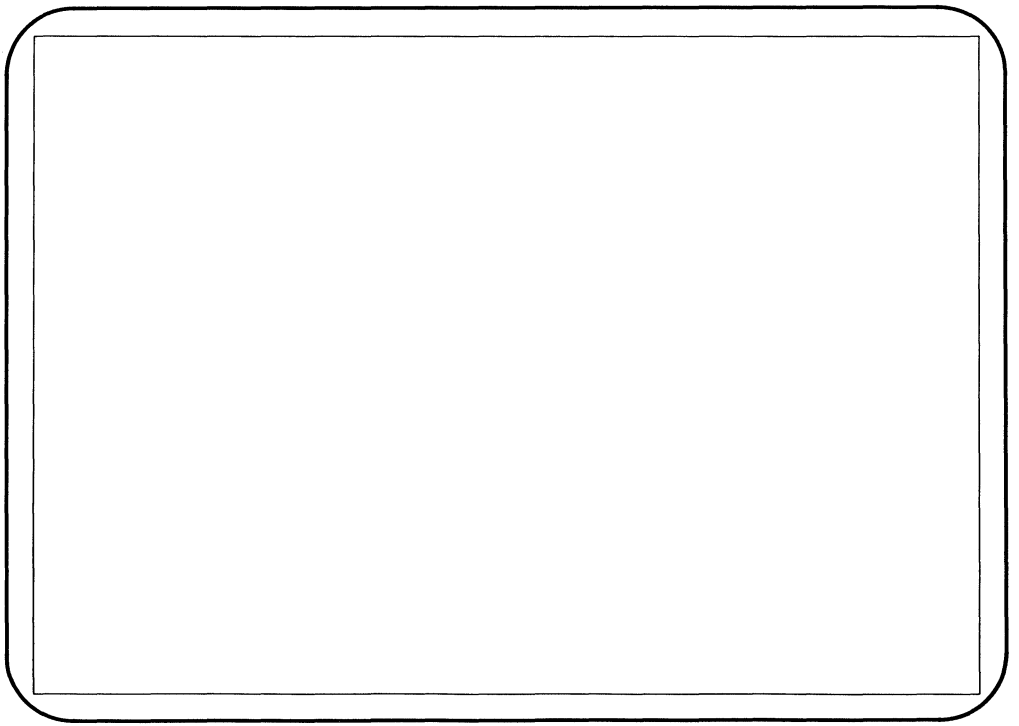


Figure 1-3. An Anisotropic Frame of VDC Space

Since the scale for the voltage measurements is not the same as the scale for the number of measurements, anisotropic plotting is appropriate.

Viewports

The viewport is the portion of the output display that is used to draw the picture. This can be all or only a part of the display.

The viewport boundaries is defined by several procedures.

- `set_p1_p2` - This procedure defines the physical region that contains the viewport.
- `mapping_mode` - This procedure defines the viewport as *isotropic* (distorted) or *anisotropic* (undistorted). If anisotropic, the viewport is the same as the rectangle defined by the `set_p1_p2` procedure. If the viewport is isotropic, the viewport is the largest rectangle with the same aspect ratio as the VDC extent that will fit in the rectangle defined by the `set_p1_p2` procedure and is positioned in that rectangle by the `viewport_justification` procedure.
- `viewport_justification` - This procedure defines the fraction of "white space" (the area of the `set_p1_p2` rectangle not within the viewport) to the left or bottom of the viewport. The default is .5 (50%), so the viewport is centered in the rectangle defined by `set_p1_p2`. This procedure only applies to isotropic viewports.

Rectangles

The `rectangle` procedure provides an easy way to draw rectangles. All you need to specify is the device needing a rectangle and the coordinates of two opposite corners of the rectangle. A default rectangle has a filled interior and no perimeter.

To see the extent of the Virtual Device Coordinate Space and to draw a frame around the location on the display, two rectangles are drawn. The syntax for these rectangles is:

```
rectangle(fildes,-20,0.1575,105.0,0.1825);
rectangle(fildes,0,0.1600,100.0,0.1800);
```

To make rectangles hollow, and add the perimeter, use the `interior_style` procedure. The following syntax will do the job.

```
interior_style(fildes,INT_HOLLOW,TRUE);
```

The following program segment is added to the program in order to draw the rectangles. The program is stored as *example4c.c*.

```
vdc_extent(fildes,-20.0,0.1575,0.0,105.0,0.1825,0.0);
clip_rectangle(fildes,-20.0,105.0,0.1575,0.1825);
mapping_mode(fildes,TRUE);

interior_style(fildes,INT_HOLLOW,TRUE);
rectangle(fildes,-20.0,0.1575,105.0,0.1825);
rectangle(fildes,0.0,0.1600,100.0,0.1800);
```

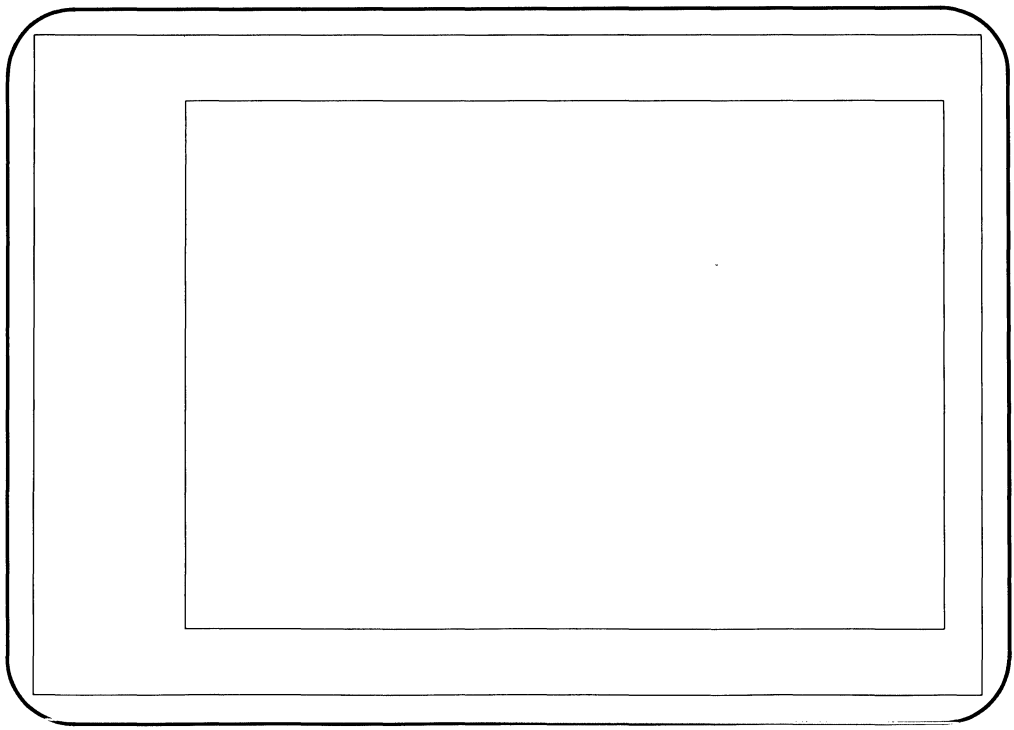


Figure 1-4. Two Rectangles

Plotting the Data

The following program will read the plot data from the file *data* and draw the results in Virtual Device Coordinate space. The main procedure is `polyline` which will move the graphics pen to the first (X,Y) data location and then draw to the remaining data locations. The data is read into a single-dimension array usable by `polyline`. The number of points to be drawn to is *i/2* and the `FALSE` flag indicates that there are no move-draw (0 for move, 1 for draw) indicators in the data.

You need to include <stdio.h> to access the file *data*. The syntax is:

```
#include <stdio.h>

        .
        .
        .

        fp = fopen("data","r");
        while(i<200 && fscanf(fp,"%f%f",&coords[i],&coords[i+1]) != EOF)
                i+=2;
        fclose(fp);

        polyline2d(fildes,coords,i/2,FALSE);
```

The results of this modification to the example program are shown in Figure 1-5. This drawing was made with *example5c.c*.

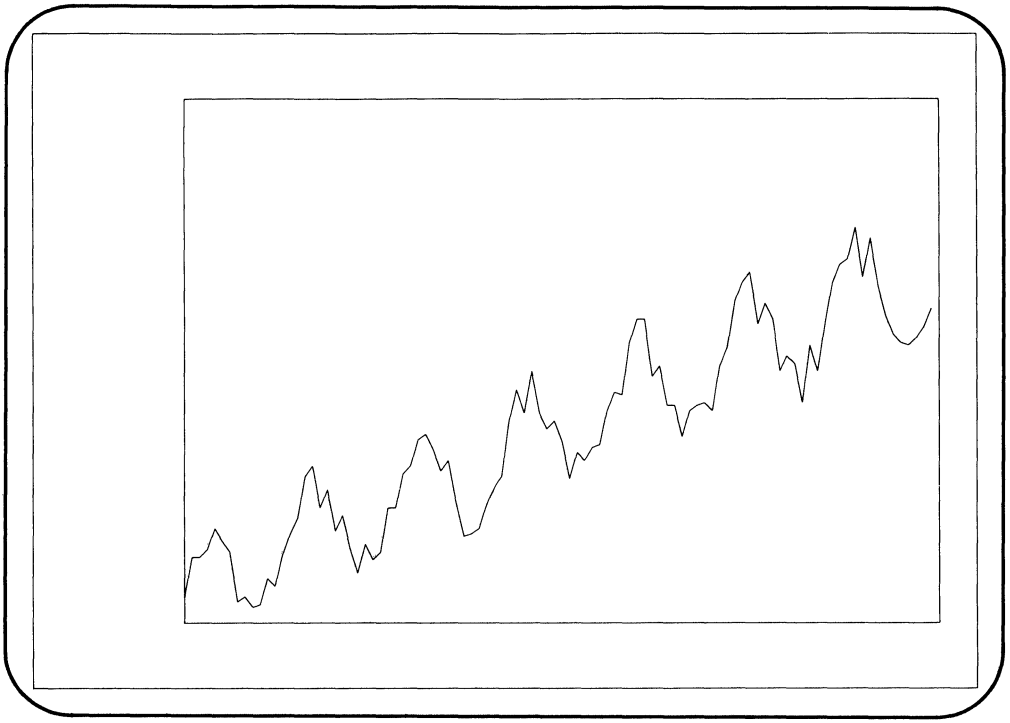


Figure 1-5. Plotted Data Only

Labels

Let us now label the drawing. We will use two fonts, different character paths, and different text alignments to show some of the capabilities of the system.

The syntax to do this is:

```
text_alignment(fildes,TA_CENTER,TA_TOP,0.0,0.0);
character_height(fildes,0.001);
text2d(fildes,50.0,0.1588,"Time (seconds)",VDC_TEXT,FALSE);

text_path(fildes,PATH_DOWN);
text_alignment(fildes,TA_CENTER,TA_HALF,0.0,0.0);
text2d(fildes,-14.0,0.1700,"Voltage",VDC_TEXT,FALSE);

text_font_index(fildes,2);
character_path(fildes,PATH_RIGHT);
text_alignment(fildes,TA_CENTER,TA_BOTTOM,0.0,0.0);
text2d(fildes,50.0,0.1810,"VOLTAGE VARIANCE",VDC_TEXT,FALSE);
```

Our example is now shown in Figure 1-6. This program is stored in *example6c.c*.

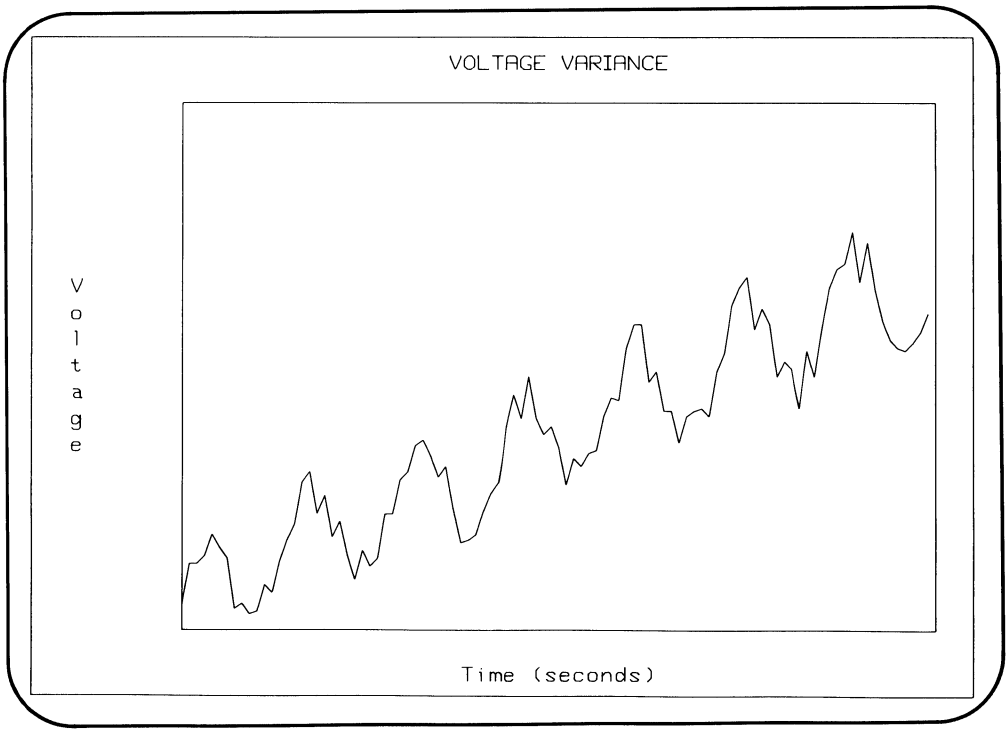


Figure 1-6. A Labeled Plot

Tic Marks

We now need to add the numerated tic marks. This is done with two loops. We use `sprintf` to change the tic mark numbers into strings to be drawn as text.

```
        text_alignment(fildes,TA_CENTER,TA_CONTINUOUS_VERTICAL,0.0,1.2);
        character_height(fildes,0.0006);

        x = 0.0;
        sprintf(number,"%d",(int)x);
        text2d(fildes,x,0.1598,number,VDC_TEXT,FALSE);
        for (i=10; i--;) {
                for (j=5; j--;) {
                        move2d(fildes,x,0.1600);
                        draw2d(fildes,x,0.1603);
                        x += 2.0;
                }
                move2d(fildes,x,0.1600);
                draw2d(fildes,x,0.1605);
                sprintf(number,"%d",(int)x);
                text2d(fildes,x,0.1600,number,VDC_TEXT,FALSE);
        }
        text_alignment(fildes,TA_CONTINUOUS_HORIZONTAL,TA_HALF,1.2,0.0);
        y = 0.1600;
        sprintf(number,"%5.4f",y);
        text2d(fildes,0.0,y,number,VDC_TEXT,FALSE);
        for (i=8; i--;) {
                for (j=5; j--;) {
                        move2d(fildes,0.0,y);
                        draw2d(fildes,1.0,y);
                        y += 0.0005;
                }
                move2d(fildes,0.0,y);
                draw2d(fildes,2.0,y);
                sprintf(number,"%5.4f",y);
                text2d(fildes,0.0,y,number,VDC_TEXT,FALSE);
        }
```

With the tic marks, we have the final picture.

The Final Picture

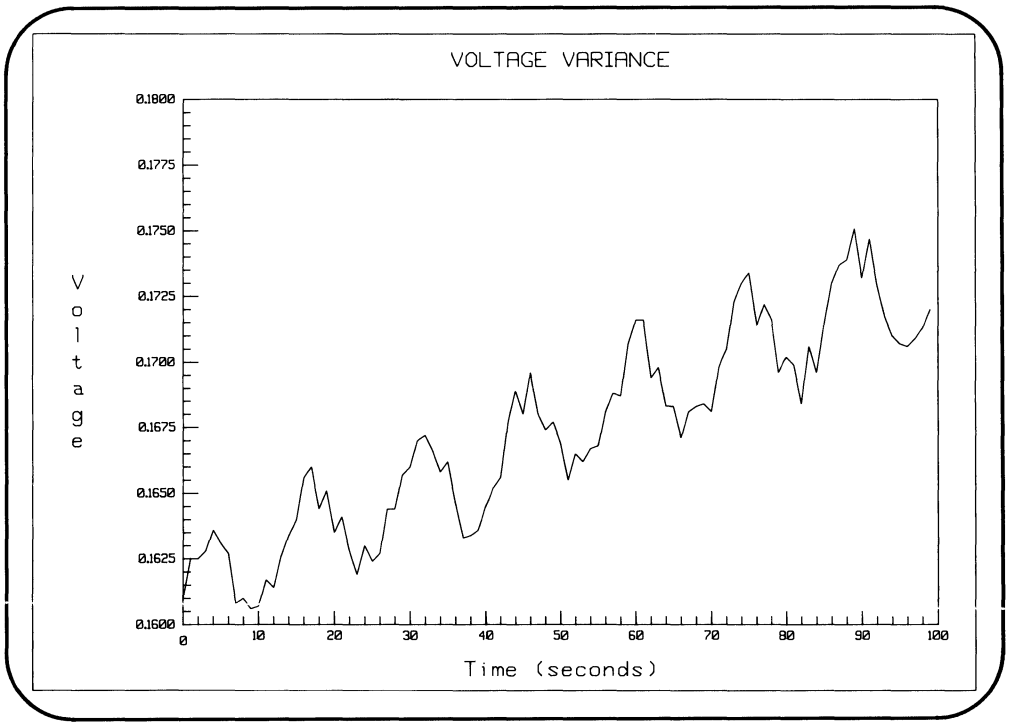


Figure 1-7. The Final Picture

The whole program

```
#include <stdio.h>
#include <starbase.c.h>

main(argc,argv)
int argc; char *argv[];
{
     int fildes;
     int i=0, j=0;
     float x, y;
     float coords[200];
     FILE *fp;
     char number[80];

     if (argc > 2) fildes=gopen(argv[1],OUTDEV,argv[2],INIT);
     else fildes=gopen("/dev/tty",OUTDEV,"hp262x",INIT);
     if (fildes == -1) exit(-1);

     vdc_extent(fildes,-20.0,0.1575,0.0,105.0,0.1825,0.0);
     clip_rectangle(fildes,-20.0,105.0,0.1575,0.1825);
     mapping_mode(fildes,TRUE);

     interior_style(fildes,INT_HOLLOW,TRUE);
     rectangle(fildes,-20.0,0.1575,105.0,0.1825);
     rectangle(fildes,0.0,0.1600,100.0,0.1800);

     text_alignment(fildes,TA_CENTER,TA_TOP,0.0,0.0);
     character_height(fildes,0.001);
     text2d(fildes,50.0,0.1588,"Time (seconds)",VDC_TEXT,FALSE);

     character_path(fildes,PATH_DOWN);
     text_alignment(fildes,TA_CENTER,TA_HALF,0.0,0.0);
     text2d(fildes,-14.0,0.1700,"Voltage",VDC_TEXT,FALSE);

     text_font_index(fildes,2);
     character_path(fildes,PATH_RIGHT);
     text_alignment(fildes,TA_CENTER,TA_BOTTOM,0.0,0.0);
     text2d(fildes,50.0,0.1810,"VOLTAGE VARIANCE",VDC_TEXT,FALSE);

     fp = fopen("data","r");
     while(i<200 && fscanf(fp,"%f%f",&coords[i],&coords[i+1]) != EOF)
             i+=2;
     fclose(fp);

     polyline2d(fildes,coords,i/2,FALSE);

     text_alignment(fildes,TA_CENTER,TA_CONTINUOUS_VERTICAL,0.0,1.2);
     character_height(fildes,0.0006);

     x = 0.0;
```

```
        sprintf(number,"%d",(int)x);
        text2d(fildes,x,0.1598,number,VDC_TEXT,FALSE);
        for (i=10; i--;) {
                  for (j=5; j--;) {
                        move2d(fildes,x,0.1600);
                        draw2d(fildes,x,0.1603);
                        x += 2.0;
                }
                move2d(fildes,x,0.1600);
                draw2d(fildes,x,0.1605);
                sprintf(number,"%d",(int)x);
                text2d(fildes,x,0.1600,number,VDC_TEXT,FALSE);
        }

        text_alignment(fildes,TA_CONTINUOUS_HORIZONTAL,TA_HALF,1.2,0.0);

        y = 0.1600;
        sprintf(number,"%5.4f",y);
        text2d(fildes,0.0,y,number,VDC_TEXT,FALSE);
        for (i=8; i--;) {
                for (j=5; j--;) {
                        move2d(fildes,0.0,y);
                        draw2d(fildes,1.0,y);
                        y += 0.0005;
                }
                move2d(fildes,0.0,y);
                draw2d(fildes,2.0,y);
                sprintf(number,"%5.4f",y);
                text2d(fildes,0.0,y,number,VDC_TEXT,FALSE);
        }

        gclose(fildes);
}
```

Notes

Graphics with Fortran77

Welcome

This manual is designed to teach you how to use some fundamental procedures that are included in the Starbase Graphics Library on your HP-UX system. These procedures can be accessed from the C, Fortran77 and Pascal programming languages available for your system. This subsection will present a progressive example using the Fortran77 Programming Language. The previous subsection covered the same material using the C Programming Language and the next subsection covers the same material using the Pascal programming language.

Starbase Graphics programs written in Fortran77 require two conventions for procedure passing.

1. All matrices passed to Starbase from Fortran77 must have the row and column indices reversed. Starbase assumes the order used by both Pascal and C for all matrices processed.
2. Whenever passing strings as character* variables, or constants, the end of the string must be marked with a char(0) {NULL}.

This manual was written with the assumption that you are familiar with your HP-UX Operating System, including the appropriate programming languages, compilers, linkers, editors, etc. If this is not the case, the information you need is available in the extensive documentation provided with your system. There are also many excellent books and courses avaliable to provide information concerning C, Fortran77 and Pascal.

This manual also assumes that you are familiar with basic computer graphics concepts. If this is not the case, there are many excellent books and courses available to provide this information.

The example programs used in this tutorial are included with your system in the `/usr/lib/starbase/demos` directory. You are encouraged to run these programs as you read the manual. When you see how the programs work, modify them as you desire to see how each part contributes to the whole. Be aware that the example programs assume a system configuration which may differ from yours, and that the **gopen** procedure parameters and the device drivers linked to your program may need changing to match your system.

NOTE

Demonstration programs and routines in `/usr/lib/starbase/demos` are for the purpose of instruction only. They are not part of the HP-UX package, and as such, they are not covered by any warranty, expressed or implied. Hewlett Packard shall not be liable for incidental or consequential damages in connection with, or arising out of, the furnished, performance, or use of these routines.

Preparation

The directions concerning the execution of the example programs presented in this manual assumes your current directory is `/usr/lib/starbase/demos`. To make this directory your current working directory, use the **c**hange **d**irectory (`cd`) command as follows:

```
cd /usr/lib/starbase/demos
```

A discussion of the hierarchial directory structure used by the HP-UX Operating System is found in the *Systems Administrator* manual supplied with your system.

For further details on the `cd` command, use the `man` command. For example,

```
man cd
```

The `man` command prints the reference page for the specified command on the display.

The example programs presented in this manual are included with your operating system. They are located in the `/usr/lib/starbase/demos` directory.

The examples are provided as source code. To read the code, use the HP-UX `more` or `cat` commands. To modify the code, use an appropriate editor, such as the *vi* editor.

To execute the example programs on other devices, they must be compiled and linked to the correct drivers. The shell script shown below will accomplish this for you.

Device Drivers

The Starbase device drivers are located in the directory `/usr/lib` and have file names appropriate to access by the "-l" option of the **Fortran77** compiler (`fc`) command.

Take a moment and list the device driver files in `/usr/lib` to see which device drivers are available with your system. All device driver file names begin with `libdd` prefix, followed by the device type. To list these files, type in:

```
ls /usr/lib/libdd*
```

As you can see, there are several drivers to choose from.

The HP-GL device driver is `/usr/lib/libddhpgl.a` (accessed with `-lddhpgl`) and the HP 262x device driver is `/usr/lib/libdd262x.a` (accessed with `-ldd262x`).

For further details concerning the Starbase Device Drivers, read the appropriate sections in the *Starbase Device Drivers Library* manual.

A Shell Script

The following shell script was used to compile and execute the example programs discussed in this manual. The script works with both the Bourne Shell and the C Shell (the C shell calls the Bourne shell automatically).

The "-l" option was used to reduce typing. This compiler option prepends `/usr/lib/lib` to the filename and appends `.a` to the file name. Thus,

```
... -lddhpgl
```

is processed as if it were

```
... /usr/lib/libddhpgl.a
```

The `hp262x` driver is included with the script. The `sb1` and `sb2` files are included to resolve all unresolved variables generated by the Starbase programs.

```
PROG=$1
shift
fc -o $PROG $PROG.f $* -ldd262x -lsb1 -lsb2
$PROG
```

This script is stored as an executable file under the file name `FC` in the `/usr/lib/starbase/demos` directory.

To compile and execute the example program **example1f.f**, enter the following:

```
FC example1f
```

To execute the compiled program *example1f* again, type in:

```
example1f
```

If you want the same program executed on an HP-GL plotter, you must edit the program and modify the **gopen** procedure call.

To execute the example with other devices, you must link the correct devices to the program.

For example, to link HP 98700 device driver to the example program, just type in:

```
FC example1f -ldd98700
```

The Data to be Plotted

The following data is to be presented in graphical form. The example program to do this follows.

Table 1-4. Test Data

Test	Value	Test	Value	Test	Value	Test	Value
0	0.1610	1	0.1625	2	0.1625	3	0.1628
4	0.1636	5	0.1631	6	0.1627	7	0.1608
8	0.1610	9	0.1606	10	0.1607	11	0.1617
12	0.1614	13	0.1626	14	0.1634	15	0.1640
16	0.1656	17	0.1660	18	0.1644	19	0.1651
20	0.1635	21	0.1641	22	0.1628	23	0.1619
24	0.1630	25	0.1624	26	0.1627	27	0.1644
28	0.1644	29	0.1657	30	0.1660	31	0.1670
32	0.1672	33	0.1666	34	0.1658	35	0.1662
36	0.1646	37	0.1633	38	0.1634	39	0.1636
40	0.1645	41	0.1652	42	0.1656	43	0.1677
44	0.1689	45	0.1680	46	0.1696	47	0.1680
48	0.1674	49	0.1677	50	0.1669	51	0.1655
52	0.1665	53	0.1662	54	0.1667	55	0.1668
56	0.1681	57	0.1688	58	0.1687	59	0.1707
60	0.1716	61	0.1716	62	0.1694	63	0.1698
64	0.1683	65	0.1683	66	0.1671	67	0.1681
68	0.1683	69	0.1684	70	0.1681	71	0.1698
72	0.1705	73	0.1723	74	0.1730	75	0.1734
76	0.1714	77	0.1722	78	0.1716	79	0.1696
80	0.1702	81	0.1699	82	0.1684	83	0.1706
84	0.1696	85	0.1715	86	0.1730	87	0.1737
88	0.1739	89	0.1751	90	0.1732	91	0.1747
92	0.1729	93	0.1717	94	0.1710	95	0.1707
96	0.1706	97	0.1709	98	0.1713	99	0.1720

The Data as a Clist

The first column shows the first few file entries as a clist without move/draw indicators. The second column shows the first few data entries as a clist with move(1)/draw(0) indicators.

Table 1-5. Test Data

Clist without Move/Draw Indicators	Clist with Move/Draw Indicators
0	0
0.1610	0.1610
1	0 (a move indicator)
0.1625	1
2	0.1625
0.1625	1 (a draw indicator)
3	2
0.1628	0.1625
4	1 (another draw indicator)
0.1636	3
5	0.1628
0.1631	1
6	4
0.1627	0.1636
7	1
0.1608	5
8	0.1631
0.1610	1
9	6
0.1606	0.1627
10	1
0.1607	7
11	0.1608
0.1617	1
12	8
0.1614	0.1610
13	1
0.1626	9
14	0.1606
0.1634	1
15	10
0.1640	0.1607
16	1
0.1656	11
17	0.1617
0.1660	1
18	12
0.1644	0.1614
19	1
0.1651	13

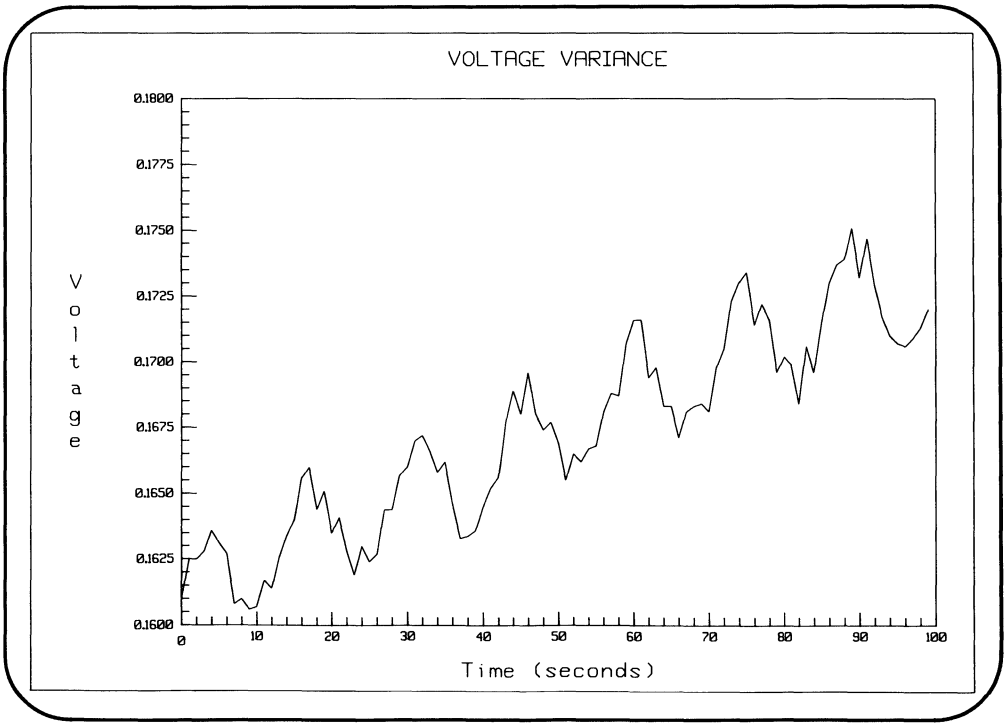


Figure 1-8. The Results Desired

A Template Fortran77 Program

The following is a template program that will be filled with Starbase procedure and function calls to draw the example voltage plot. As listed, this program opens the specified device, clears the clipping area and then closes the device.

This program is stored as *example1f.f.*

```
include '/usr/include/starbase.f1.h'
program chart
character NULL
parameter(NULL = char(0))
include '/usr/include/starbase.f2.h'

integer*4 fildes

fildes = gopen('/dev/tty'//NULL, OUTDEV, 'hp262x'//NULL, INIT)
if (fildes .eq. -1) stop

fildes = gclose(fildes)
end
```

The Include File

The files `starbase.f1.h` and `starbase.f2.h` contain the definitions and terms needed for any Starbase program written in Fortran77. These files are located in the `/usr/include` directory.

To see what definitions and terms are needed, use the `more` command. The syntax is:

```
more /usr/include/starbase.f1.h
```

and

```
more /usr/include/starbase.f2.h
```

The Gopen Function

The function `gopen` opens the specified graphic device. This function requires four parameters:

1. **Device File Name** - Device files are created by the System Administrator. Device files are located in the `/dev` directory. In the following examples, the device file `tty` is used.
2. **Performance Mode** - This parameter may be one of the following:

- OUTDEV - device driver is to output information to the device.
- INDEV - device driver is to read information from the device.
- OUTINDEV - device driver can send and receive information to and from the device.

The following examples use OUTDEV since the example program is to be drawn on the output device's display.

3. **Character Representation of the Driver Type** - For the following example, this will be `hp262x`. This parameter identifies the driver to be used. The `hp262x` driver interrogates the device specified by the device file name and tests to see if the device is an HP 2623 Graphics Terminal or an HP 2627 Color Graphics Terminal, and acts accordingly.

   A possible character representation for an HP-GL plotter is `hpgl`.

4. **Operation Mode** - This parameter may be one of the following:

   0 - open the device, but do nothing else.
   INIT - open and initialize the device in a device dependent way.
   RESET - open and completely initialize the device.
   SPOOLED - open the device for spooled operation. Spooled output may be saved in a file for later display.
   THREE_D - open the device and set Starbase to 3-dimensional mode.

   These modes may be combined with the proper ADD syntax. In Fortran77 this is the plus sign (+) as in (INIT+SPOOLED).

By doing an ADD operation on INIT and SPOOLED, the graphic device can be opened, initialized and spooled to a file.

For example:

```
fildes = gopen("spoolfile",OUTDEV,"hpgl",INIT+SPOOLED);
```

If the opening is not successful, `gopen` returns a value of `-1`. If the opening is successful, a positive integer (file descriptor) is returned and the HP-GL device is initialized and ready for spooled output.

Further information can be obtained about this procedure by using the `man` command. Just type in:

```
man 3 gopen
```

The Gclose Function

To properly close a graphics device, use the `gclose` function. The `fildes` file descriptor identified by the `gopen` function is used to identify a particular device for all following Starbase procedures and functions, including `gclose`.

Further information can be obtained about this procedure by using the `man` command. Just type in:

```
man 3 gclose
```

A Review

As a review, the following table lists information about the files, functions and commands just discussed.

Table 1-6. A Review

Required Items	Learn More
The cd Command	/usr/man/man1/cd.1 (man cd)
The fc Compiler Command	/usr/man/man1/fc.1 (man fc)
Shell Scripts	HP-UX Concepts and Tutorials, volume 3.
Device Drivers	Starbase Device Drivers Library. /usr/lib/libdd* (ls /usr/lib/libdd...)
C Programming	The C Programming Language by Kernighan and Ritchie
Definitions and Terms needed for any Starbase program written in Fortran77	/usr/include/starbase.f1.h /usr/include/starbase.f2.h (more /usr/include/starbase.f1.h) (more /usr/include/starbase.f2.h)
gopen	/usr/man/man3/gopen.3g (man 3 gopen)
gclose	/usr/man/man3/gclose.3g (man 3 gclose)

Virtual Device Coordinates

The default Cartesian coordinate system used by the Starbase procedures is called the Virtual Device Coordinate (VDC) system. This system can be considered a 2-dimensional subset of the Cartesian plane with default x- and y-axis values ranging from 0.0 to 1.0. In 3-dimensional graphics, this system can also be considered a subset of Cartesian 3-space with default x-, y- and z-axis values ranging from 0.0 to 1.0. Locations in VDC space are specified in VDC values as specified by the `vdc_extent` procedure.

The procedure `gopen` automatically creates a Virtual Device Coordinate to Device Coordinate (DC) transformation matrix for each opened graphic device. All VDC data is processed through this matrix before being set to the target device. Since each device has a unique transformation matrix, the same data will appear as expected on each device.

The range of values used for this system can be altered with the `vdc_extent` procedure.

Use the `push_vdc_matrix` procedure to force the current working matrix to be the VDC to DC transformation matrix.

Device Coordinates

Device coordinate Space is the Cartesian 2-dimensional space that is the device's output surface. The range and magnitude of x- and y-axis values is defined by each device.

Device Coordinate procedures bypass the VDC to DC transformation matrix. Such procedures begin with the characters *dc*, for "device coordinate". For example, `dcdraw` will draw a line on the specified device using the untransformed coordinate values specified.

To transform VDC values to DC values, use the `vdc_to_dc` procedure.

World Coordinates

World coordinates can have any value the user desires. This is handy when the data is to be used for purposes other than making a drawing.

Since world coordinates can be of any value, they must be transformed to device coordinates before being drawn on a graphic device. This transformation can be done in several ways. For example:

- A transformation matrix with the appropriate values to transform the World Coordinate data into Device Coordinate data may be supplied by the user.
- A transformation matrix with the appropriate values to transform the World Coordinate data into Virtual Device Coordinate data may be supplied by the user. This matrix could then be concatenated (matrix multiplied) with the Virtual Device Coordinate to Device Coordinate transformation matrix to make a World Coordinate to Device Coordinate Matrix.

For instance, in 2-dimensional mode, a matrix containing:

$$\begin{matrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \\ 0 & 0 \end{matrix}$$

will rotate world coordinates counter-clockwise by an angle of θ.

- **The Easiest Way** is to use the `vdc_extent` procedure to change the ranges of values used to define the Virtual Device Coordinate Space so that the world coordinate values are now contained in VDC space. Any time the `vdc_extent` procedure is used, a new Virtual Device Coordinate to Device Coordinate matrix is calculated that maps Virtual Device Coordinate Space into Device Coordinate Space.

Setting the VDC Extent

In the example problem, we want to plot voltage values against the associated measurement number. Our voltage data ranges from just over 0.1600 through just under 0.1800, while we have 100 measurements (from 1 to 100).

Logical values for the vdc extent would be -20 to 105 on the x-axis and 0.1575 to 0.1825 in the y-axis. This will allow us to plot the data in a rectangle of size 0-100 on the x-axis and 0.1600-0.1800 on the y-axis and still leave room for labels within the vdc extent and outside the plotting area. Since we are doing 2-dimensional graphics, the z-coordinates are both 0.0. The syntax to do this is:

```
call vdc_extent(fildes,-20.0,0.1575,0.0,105.0,0.1825,0.0);
```

Detailed information on this procedure can be found using the HP-UX `man` command as follows:

```
man 3 vdc_extent
```

NOTE

The decision was made to follow ANSI parameter patterns in the Starbase procedures. These parameter patterns are NOT CONSISTENT. One procedure may call for x1,y1,z1,x2,y2,z2 while the next may require x1,x2,y1,y2,z1,z2. Hopefully this inconstancy will be removed from the ANSI standard and thus from the Starbase procedures.

Clipping

Clipping is the process of not allowing selected graphic data to be displayed. There are two clipping processes:

- Hard Clip - This is clipping done by the graphic output device that physically stops plotting outside the physical limits of the device's display area. For example, a plotter's pen can not draw past the physical edge of the plotter's platen.
- Soft Clip - This is clipping done by the graphics software to limit the visible portion of the plot. Soft clipping can be done to both 2-dimensional and 3-dimensional data.

Starbase provides two clipping boundaries:

- The Clip Rectangle - A rectangle can be specified with the `clip_rectangle` procedure that defines subsequent clipping boundaries.
- The VDC Extent - The rectangle defined by the `vdc_extent` procedure can also be used to define subsequent clipping boundaries.

  The `clip_indicator` procedure identifies which of the two boundaries to clip to.

The `clip_depth` procedure is used to define the front and back clipping planes for 3-dimensional graphics. Front and back clipping is enabled or disabled with the `depth_indicator` procedure.

Isotropic (undistorted) Mapping

Isotropic mapping is when one unit in the x-axis **exactly equals** one unit in the y-axis. This is sometimes referred to as *undistorted* plotting.

The *mapping_mode* procedure sets the mapping mode to either anisotropic or isotropic plotting mode. The default is isotropic.

The following example program draws an **isotropic** square around the default Virtual Device Coordinate space. This program is in file *example2f.f*.

```
include '/usr/include/starbase.f1.h'
program chart
character NULL
parameter(NULL = char(0))
include '/usr/include/starbase.f2.h'

integer*4 fildes

fildes = gopen('/dev/tty'//NULL, OUTDEV, 'hp262x'//NULL, INIT)
if (fildes .eq. -1) stop

call interior_style(fildes,INT_HOLLOW,TRUE)
call rectangle(fildes,0.0,0.0,1.0,1.0)

fildes = gclose(fildes)
end
```

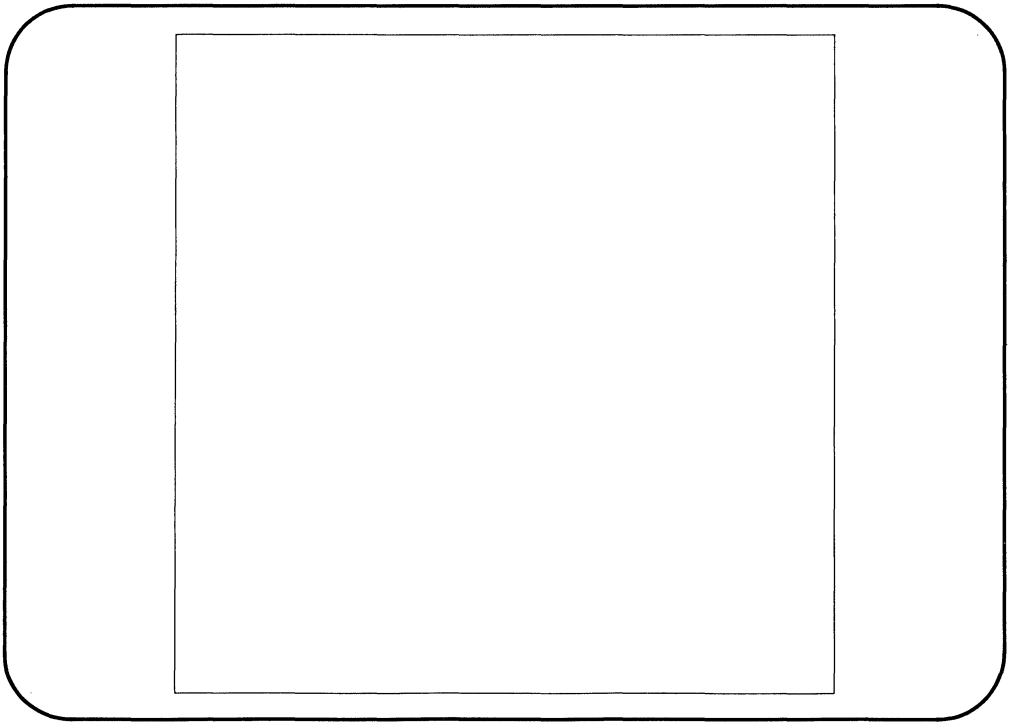


Figure 1-9. An Isotropic Frame of VDC Space

Anisotropic (distorted) Mapping

Anisotropic mapping where one unit in the X direction does not have to equal one unit in the Y direction. This is sometimes called *distorted* plotting.

The `mapping_mode` procedure sets the mapping mode to either anisotropic or isotropic plotting mode. The following single line was added to the preceeding isotropic program and the anisotropic results are shown Figure 1-10. The rectangle still frames the VDC default space, but now the space fills the usable portion of the display. This program is in *example3f.f*.

```
call mapping_mode(fildes,TRUE);
```

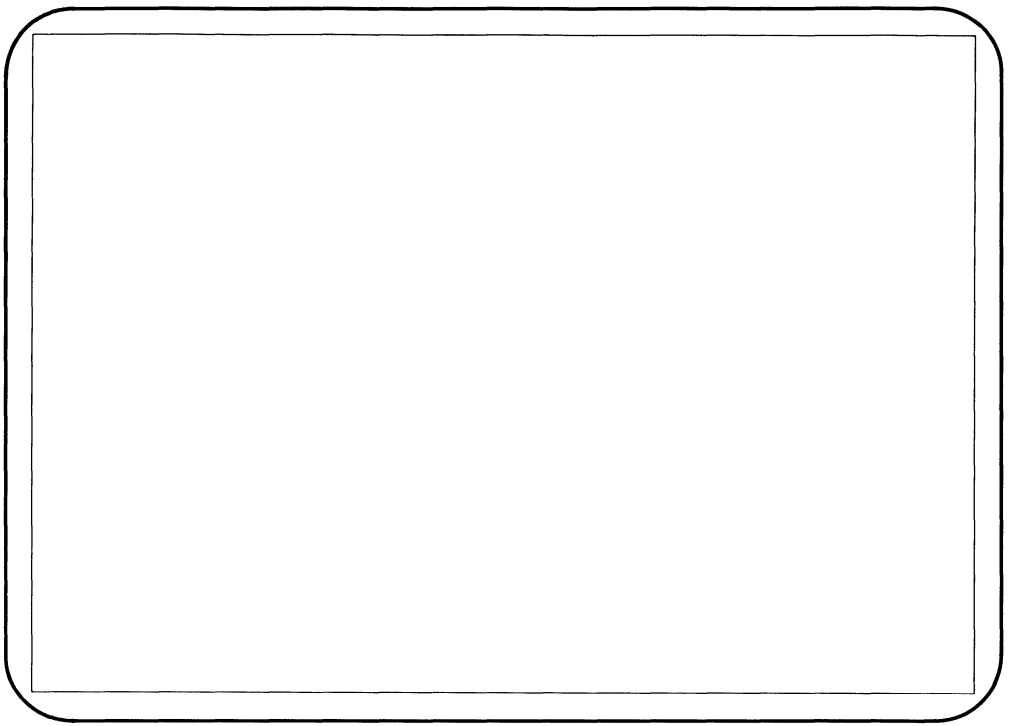


Figure 1-10. An Anisotropic Frame of VDC Space

Since the scale for the voltage measurements is not the same as the scale for the number of measurements, anisotropic plotting is appropriate.

Viewports

The viewport is the portion of the output display that is used to draw the picture. This can be all or only a part of the display.

The viewport boundaries is defined by several procedures.

- `set_p1_p2` - This procedure defines the physical region that contains the viewport.
- `mapping_mode` - This procedure defines the viewport as *isotropic* (distorted) or *anisotropic* (undistorted). If anisotropic, the viewport is the same as the rectangle defined by the `set_p1_p2` procedure. If the viewport is isotropic, the viewport is the largest rectangle with the same aspect ratio as the VDC extent that will fit in the rectangle defined by the `set_p1_p2` procedure and is positioned in that rectangle by the `viewport_justification` procedure.
- `viewport_justification` - This procedure defines the fraction of "white space" (the area of the `set_p1_p2` rectangle not within the viewport) to the left or bottom of the viewport. The default is .5 (50%), so the viewport is centered in the rectangle defined by `set_p1_p2`. This procedure only applies to isotropic viewports.

Rectangles

The `rectangle` procedure provides an easy way to draw rectangles. All you need to specify is the device needing a rectangle and the coordinates of two opposite corners of the rectangle. A default rectangle has a filled interior and no perimeter.

To see the extent of the Virtual Device Coordinate Space and to draw a frame around the location on the display, two rectangles are drawn. The syntax for these rectangles is:

```
call rectangle(fildes,-20,0.1575,105.0,0.1825);
call rectangle(fildes,0,0.1600,100.0,0.1800);
```

To make rectangles hollow, and add the perimeter, use the `interior_style` procedure. The following syntax will do the job.

```
call interior_style(fildes,INT_HOLLOW,TRUE);
```

The following program segment is added to the program in order to draw the rectangles. The program is stored as *example4f.f*.

```
include '/usr/include/starbase.f1.h'
program chart
character NULL
parameter(NULL = char(0))
include '/usr/include/starbase.f2.h'

integer*4 fildes, status

fildes = gopen('/dev/tty'//NULL, OUTDEV, 'hp262x'//NULL, INIT)
if (fildes .eq. -1) stop

call vdc_extent(fildes,-20.0,0.1575,0.0,105.0,0.1825,0.0)
call clip_rectangle(fildes,-20.0,105.0,0.1575,0.1825)
call mapping_mode(fildes,TRUE)

call interior_style(fildes,INT_HOLLOW,TRUE)
call rectangle(fildes,-20.0,0.1575,105.0,0.1825)
call rectangle(fildes,0.0,0.1600,100.0,0.1800)

status = gclose(fildes)
end
```

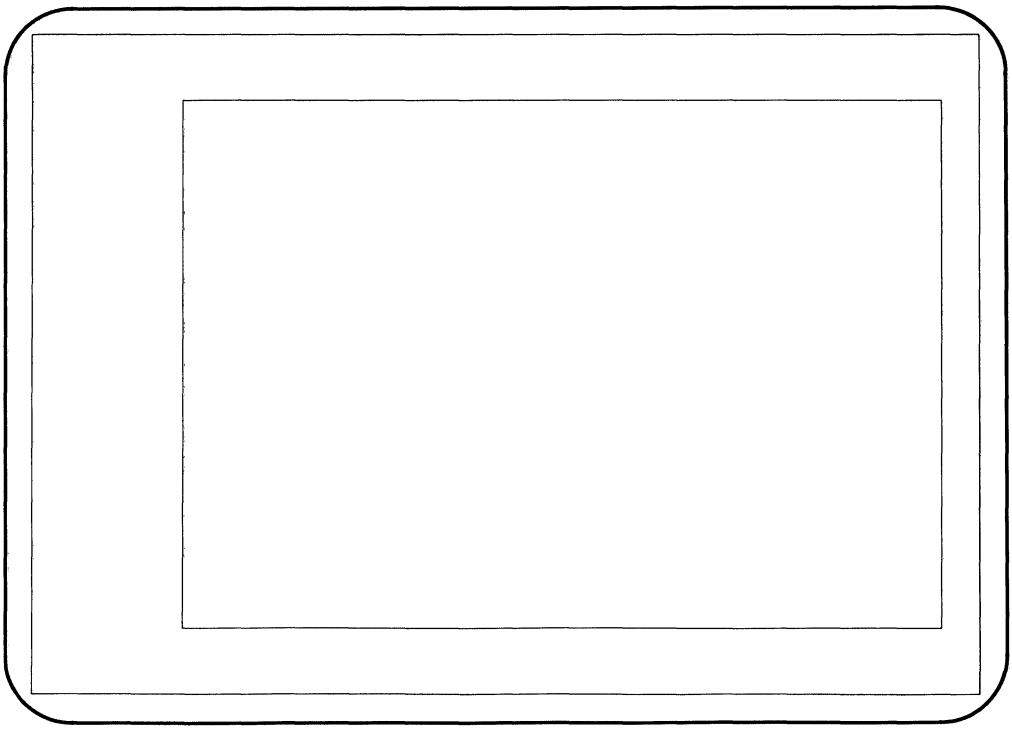


Figure 1-11. Two Rectangles

Plotting the Data

The following program will read the plot data from the file *data* and draw the results in Virtual Device Coordinate space. The main procedure is `polyline` which will move the graphics pen to the first (X,Y) data location and then draw to the remaining data locations. The data is read into a single-dimension array usable by `polyline`. The number of points to be drawn to is *i/2* and the `FALSE` flag indicates that there are no move-draw (0 for move, 1 for draw) indicators in the data.

The syntax is:

```
        include '/usr/include/starbase.f1.h'
        program chart
        character NULL
        parameter(NULL = char(0))
        include '/usr/include/starbase.f2.h'

        integer*4 fildes
        integer*4 i, j
        real x, y
        real coords(200)

        fildes = gopen('/dev/tty'//NULL, OUTDEV, 'hp262x'//NULL, INIT)
        if (fildes .eq. -1) stop

        call vdc_extent(fildes,-20.0,0.1575,0.0,105.0,0.1825,0.0)
        call clip_rectangle(fildes,-20.0,105.0,0.1575,0.1825)
        call mapping_mode(fildes,TRUE)

        call interior_style(fildes,INT_HOLLOW,TRUE)
        call rectangle(fildes,0.0,0.1575,105.0,0.1825)
        call rectangle(fildes,0.0,0.1600,100.0,0.1800)

        open (unit=9,file='data')
        do i=1,199,2
                read(9,*,end=20) coords(i),coords(i+1)
        end do
20      close(9)

        call polyline2d(fildes,coords,i/2,FALSE)

        fildes = gclose(fildes)
        end
```

The results of this modification to the example program are shown in Figure 1-12. This drawing was made with *example5f.f.*

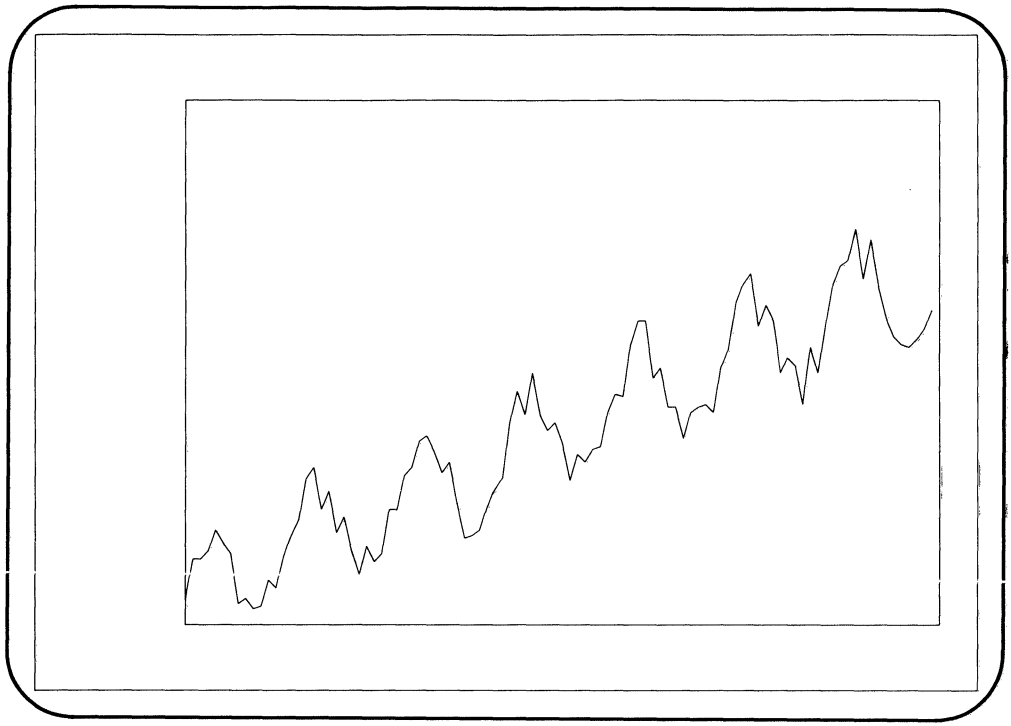


Figure 1-12. Plotted Data

Labels

Let us now label the drawing. We will use two fonts, different character paths, and different text alignments to show some of the capabilities of the system.

The syntax to do this is:

```
      include '/usr/include/starbase.f1.h'
      program chart
      character NULL
      parameter(NULL = char(0))
      include '/usr/include/starbase.f2.h'

      integer*4 fildes
      integer*4 i, j
      real x, y
      real coords(200)
      character*80 number

      fildes = gopen('/dev/tty'//NULL, OUTDEV, 'hp262x'//NULL, INIT)
      if (fildes .eq. -1) stop

      call vdc_extent(fildes,-20.0,0.1575,0.0,105.0,0.1825,0.0)
      call clip_rectangle(fildes,-20.0,105.0,0.1575,0.1825)
      call mapping_mode(fildes,TRUE)

      call interior_style(fildes,INT_HOLLOW,TRUE)
      call rectangle(fildes,-20.0,0.1575,105.0,0.1825)
      call rectangle(fildes,0.0,0.1600,100.0,0.1800)

      call text_alignment(fildes,TA_CENTER,TA_TOP,0.0,0.0)
      call character_height(fildes,0.001)
      call text2d(fildes,50.0,0.1588,'Time (seconds)'//NULL,VDC_TEXT,
     + FALSE)

      call character_path(fildes,PATH_DOWN)
      call text_alignment(fildes,TA_CENTER,TA_HALF,0.0,0.0)
      call text2d(fildes,-14.0,0.1700,'Voltage'//NULL,VDC_TEXT,FALSE)

      call text_font_index(fildes,2)
      call character_path(fildes,PATH_RIGHT)
      call text_alignment(fildes,TA_CENTER,TA_BOTTOM,0.0,0.0)
      call text2d(fildes,50.0,0.1810,'VOLTAGE VARIANCE'//NULL,VDC_TEXT,
     + FALSE)

      open (unit=9,file='data')
      do i=1,199,2
            read(9,*,end=20) coords(i),coords(i+1)
      end do
20    close(9)
```

```
call polyline2d(fildes,coords,i/2,FALSE)

fildes = gclose(fildes)
end
```

Our example is now shown Figure 1-13. This program is stored in *example6f.f*.

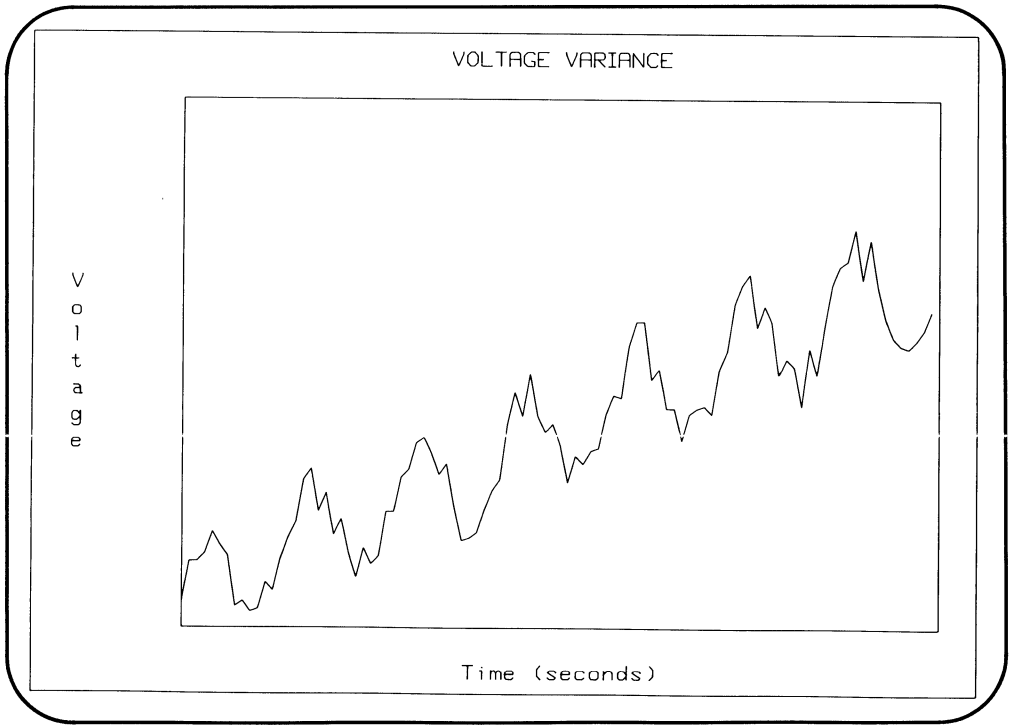


Figure 1-13. A Labeled Plot

Tic Marks

We now need to add the numerated tic marks. This is done with two loops. We use `write` to change the tic mark numbers into strings to be drawn as text.

```
      call text_alignment(fildes,TA_CENTER,
     + TA_CONTINUOUS_VERTICAL,0.0,1.2)
      call character_height(fildes,0.0006)

      x = 0.0
      write(unit=number,fmt='(I3)') int(x)
      call text2d(fildes,x,0.1598,number(1:3)//NULL,VDC_TEXT,FALSE)
      do i=1,10
              do j=1,5
                      call move2d(fildes,x,0.1600)
                      call draw2d(fildes,x,0.1603)
                      x = x + 2.0
              end do
              call move2d(fildes,x,0.1600)
              call draw2d(fildes,x,0.1605)
              write(unit=number,fmt='(I3)') int(x)
      call text2d(fildes,x,0.1600,number(1:3)//NULL,VDC_TEXT,FALSE)
      end do

      call text_alignment(fildes,
     + TA_CONTINUOUS_HORIZONTAL,TA_HALF,1.2,0.0)

      y = 0.1600
      write(unit=number,fmt='(f5.4)') y
      call text2d(fildes,0.0,y,number(1:5)//NULL,VDC_TEXT,FALSE)
      do i=1,8
              do j=1,5
                      call move2d(fildes,0.0,y)
                      call draw2d(fildes,1.0,y)
                      y = y + 0.0005
              end do
              call move2d(fildes,0.0,y)
              call draw2d(fildes,2.0,y)
              write(unit=number,fmt='(f5.4)') y
      call text2d(fildes,0.0,y,number(1:5)//NULL,VDC_TEXT,FALSE)
      end do
```

The Final Picture

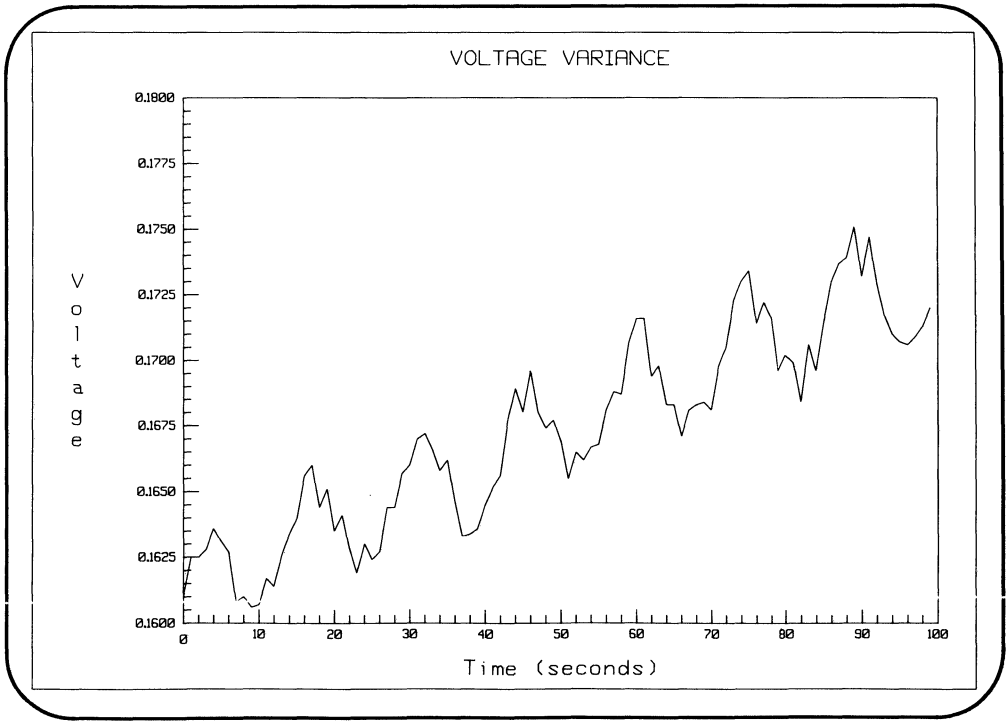


Figure 1-14. The Final Plot

The Whole Program

```
      include '/usr/include/starbase.f1.h'
      program chart
      character NULL
      parameter(NULL = char(0))
      include '/usr/include/starbase.f2.h'

      integer*4 fildes
      integer*4 i, j
      real x, y
      real coords(200)
      character*80 number

      fildes = gopen('/dev/tty'//NULL, OUTDEV, 'hp262x'//NULL, INIT)
      if (fildes .eq. -1) stop

      call vdc_extent(fildes,-20.0,0.1575,0.0,105.0,0.1825,0.0)
      call clip_rectangle(fildes,-20.0,105.0,0.1575,0.1825)
      call mapping_mode(fildes,TRUE)

      call interior_style(fildes,INT_HOLLOW,TRUE)
      call rectangle(fildes,-20.0,0.1575,105.0,0.1825)
      call rectangle(fildes,0.0,0.1600,100.0,0.1800)

      call text_alignment(fildes,TA_CENTER,TA_TOP,0.0,0.0)
      call character_height(fildes,0.001)
      call text2d(fildes,50.0,0.1588,'Time (seconds)'//NULL,VDC_TEXT,
     + FALSE)

      call character_path(fildes,PATH_DOWN)
      call text_alignment(fildes,TA_CENTER,TA_HALF,0.0,0.0)
      call text2d(fildes,-14.0,0.1700,'Voltage'//NULL,VDC_TEXT,FALSE)

      call text_font_index(fildes,2)
      call character_path(fildes,PATH_RIGHT)
      call text_alignment(fildes,TA_CENTER,TA_BOTTOM,0.0,0.0)
      call text2d(fildes,50.0,0.1810,'VOLTAGE VARIANCE'//NULL,VDC_TEXT,
     + FALSE)

      open (unit=9,file='data')
      do i=1,199,2
            read(9,*,end=20) coords(i),coords(i+1)
      end do
20    close(9)

      call polyline2d(fildes,coords,i/2,FALSE)

      call text_alignment(fildes,TA_CENTER,
     + TA_CONTINUOUS_VERTICAL,0.0,1.2)
      call character_height(fildes,0.0006)
```

```
        x = 0.0
        write(unit=number,fmt='(I3)') int(x)
        call text2d(fildes,x,0.1598,number(1:3)//NULL,VDC_TEXT,FALSE)
        do i=1,10
                do j=1,5
                        call move2d(fildes,x,0.1600)
                        call draw2d(fildes,x,0.1603)
                        x = x + 2.0
                end do
                call move2d(fildes,x,0.1600)
                call draw2d(fildes,x,0.1605)
                write(unit=number,fmt='(I3)') int(x)
call text2d(fildes,x,0.1600,number(1:3)//NULL,VDC_TEXT,FALSE)
        end do

        call text_alignment(fildes,
     + TA_CONTINUOUS_HORIZONTAL,TA_HALF,1.2,0.0)

        y = 0.1600
        write(unit=number,fmt='(f5.4)') y
        call text2d(fildes,0.0,y,number(1:5)//NULL,VDC_TEXT,FALSE)
        do i=1,8
                do j=1,5
                        call move2d(fildes,0.0,y)
                        call draw2d(fildes,1.0,y)
                        y = y + 0.0005
                end do
                call move2d(fildes,0.0,y)
                call draw2d(fildes,2.0,y)
                write(unit=number,fmt='(f5.4)') y
call text2d(fildes,0.0,y,number(1:5)//NULL,VDC_TEXT,FALSE)
        end do
        fildes = gclose(fildes)
        end
```

Notes

Graphics with Pascal

Welcome

This manual is designed to teach you how to use some fundamental procedures that are included in the Starbase Graphics Library on your HP-UX system. These procedures can be accessed from the C, Fortran77 and Pascal programming languages available for your system. This subsection will present a progressive example using the C Programming Language. The previous subsections covered the same material using the C and Fortran77 Programming Languages.

This manual was written with the assumption that you are familiar with your HP-UX Operating System, including the appropriate programming languages, compilers, linkers, editors, etc. If this is not the case, the information you need is available in the extensive documentation provided with your system. There are also many excellent books and courses avaliable to provide information concerning C, Fortran77 and Pascal.

This manual also assumes that you are familiar with basic computer graphics concepts. If this is not the case, there are many excellent books and courses available to provide this information.

The example programs used in this tutorial are included with your system in the `/usr/lib/starbase/demos` directory. You are encouraged to run these programs as you read the manual. When you see how the programs work, modify them as you desire to see how each part contributes to the whole. Be aware that the example programs assume a system configuration which may differ from yours, and that the **gopen** procedure parameters and device drivers linked to your program may need changing to match your system.

NOTE

Demonstration programs and routines in `/usr/lib/starbase/demos` are for the purpose of instruction only. They are not part of the HP-UX package, and as such, they are not covered by any warranty, expressed or implied. Hewlett Packard shall not be liable for incidental or consequential damages in connection with, or arising out of, the furnished, performance, or use of these routines.

Preparation

The directions concerning the execution of the example programs presented in this manual assumes your current directory is `/usr/lib/starbase/demos`. To make this directory your current working directory, use the **c**hange **d**irectory (`cd`) command as follows:

```
cd /usr/lib/starbase/demos
```

A discussion of the hierarchial directory structure used by the HP-UX Operating System is found in the *Systems Administrator* manual supplied with your system.

For further details on the `cd` command, use the `man` command. For example,

```
man cd
```

The `man` command prints the reference page for the specified command on the display.

The example programs presented in this manual are included with your operating system. They are located in the `/usr/lib/starbase/demos` directory.

The examples are provided as source code. To read the code, use the HP-UX `more` or `cat` commands. To modify the code, use an appropriate editor, such as the *vi* editor.

To execute the example programs on other devices, they must be compiled and linked to the correct drivers. The shell script shown below will accomplish this for you.

Device Drivers

The Starbase device drivers are located in the directory `/usr/lib` and have file names appropriate to access by the "-l" option of the Pascal compiler (`pc`) command.

Take a moment and list the device driver files in `/usr/lib` to see which device drivers are available with your system. All device driver file names begin with `libdd` prefix, followed by the device type. To list these files, type in:

```
ls /usr/lib/libdd*
```

As you can see, there are several drivers to choose from.

The HP-GL device driver is `/usr/lib/libddhpgl.a` (accessed with -lddhpgl) and the HP 262x device driver is `/usr/lib/libdd262x.a` (accessed with -ldd262x). These are the device drivers included in the example programs.

For further details concerning the Starbase Device Drivers, read the appropriate sections in the *Starbase Device Drivers Library* manual.

A Shell Script

The following shell script was used to compile and execute the example programs discussed in this manual. The script works with both the Bourne Shell and the C Shell (the C shell calls the Bourne shell automatically).

The "-l" option was used to reduce typing. This compiler option prepends `/usr/lib/lib` to the filename and appends `.a` to the file name. Thus,

```
... -lddhpgl
```

is processed as if it were

```
... /usr/lib/libddhpgl.a
```

The HP262x device driver is linked by the script. The `sb1` and `sb2` files are included to resolve all unresolved variables generated by the Starbase programs.

```
PROG=$1
shift
pc -o $PROG $PROG.p $* -ldd262x -lsb1 -lsb2
$PROG
```

This script is stored as an executable file under the file name `PC` in the `/usr/lib/starbase/demos` directory.

To compile and execute the example program `example1p.p`, enter the following:

```
PC example1p
```

To execute the compiled program *example1p* again, type in:

```
example1p
```

If you want the same program executed on an HP-GL plotter, the `gopen` procedure parameters must be modified.

To execute the example with other devices, you must link the correct devices to the program.

For example, to execute to link the HP 98700 device driver, just type in:

```
PC example1p -ldd98700
```

The Data to be Plotted

The following data is to be presented in graphical form. The example program to do this follows.

Table 1-7. Test Data

Test	Value	Test	Value	Test	Value	Test	Value
0	0.1610	1	0.1625	2	0.1625	3	0.1628
4	0.1636	5	0.1631	6	0.1627	7	0.1608
8	0.1610	9	0.1606	10	0.1607	11	0.1617
12	0.1614	13	0.1626	14	0.1634	15	0.1640
16	0.1656	17	0.1660	18	0.1644	19	0.1651
20	0.1635	21	0.1641	22	0.1628	23	0.1619
24	0.1630	25	0.1624	26	0.1627	27	0.1644
28	0.1644	29	0.1657	30	0.1660	31	0.1670
32	0.1672	33	0.1666	34	0.1658	35	0.1662
36	0.1646	37	0.1633	38	0.1634	39	0.1636
40	0.1645	41	0.1652	42	0.1656	43	0.1677
44	0.1689	45	0.1680	46	0.1696	47	0.1680
48	0.1674	49	0.1677	50	0.1669	51	0.1655
52	0.1665	53	0.1662	54	0.1667	55	0.1668
56	0.1681	57	0.1688	58	0.1687	59	0.1707
60	0.1716	61	0.1716	62	0.1694	63	0.1698
64	0.1683	65	0.1683	66	0.1671	67	0.1681
68	0.1683	69	0.1684	70	0.1681	71	0.1698
72	0.1705	73	0.1723	74	0.1730	75	0.1734
76	0.1714	77	0.1722	78	0.1716	79	0.1696
80	0.1702	81	0.1699	82	0.1684	83	0.1706
84	0.1696	85	0.1715	86	0.1730	87	0.1737
88	0.1739	89	0.1751	90	0.1732	91	0.1747
92	0.1729	93	0.1717	94	0.1710	95	0.1707
96	0.1706	97	0.1709	98	0.1713	99	0.1720

The Data as a Clist

The first column shows the first few file entries as a clist without move/draw indicators. The second column shows the first few data entries as a clist with move(1)/draw(0) indicators.

Table 1-8. Data in Clist Format

Clist without Move/Draw Indicators	Clist with Move/Draw Indicators
0	0
0.1610	0.1610
1	0 (a move indicator)
0.1625	1
2	0.1625
0.1625	1 (a draw indicator)
3	2
0.1628	0.1625
4	1 (another draw indicator)
0.1636	3
5	0.1628
0.1631	1
6	4
0.1627	0.1636
7	1
0.1608	5
8	0.1631
0.1610	1
9	6
0.1606	0.1627
10	1
0.1607	7
11	0.1608
0.1617	1
12	8
0.1614	0.1610
13	1
0.1626	9
14	0.1606
0.1634	1
15	10
0.1640	0.1607
16	1
0.1656	11
17	0.1617
0.1660	1
18	12
0.1644	0.1614
19	1
0.1651	13

The Results Desired

The following picture shows the plot to be created by the example programs described in the subsection.

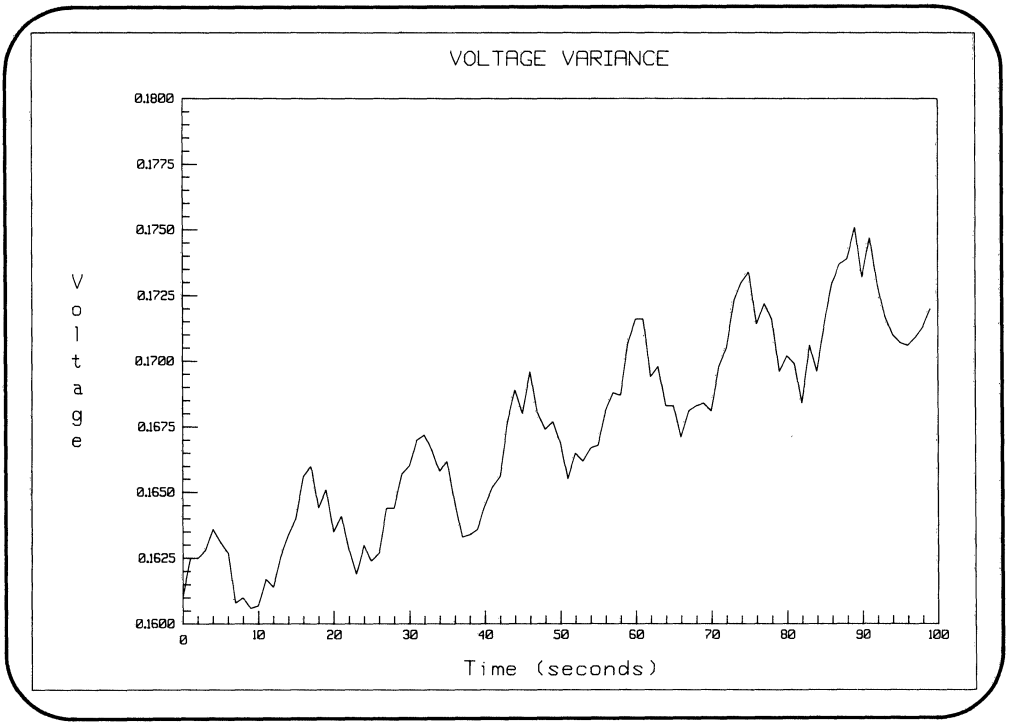


Figure 1-15. The Results Desired

A Template Pascal Program

The following is a template program that will be filled with Starbase procedure and function calls to draw the example voltage plot. As listed, this program opens the specified device, clears the clipping area and then closes the device. The capability of executing the program on an HP262x Graphics Terminal or a specified device is included.

This program is stored as *example1p.p.*

```
program main(input,output);

$include '/usr/include/starbase.p1.h'$

var
    fildes, status : integer;

$include '/usr/include/starbase.p2.h'$
procedure exit(result: integer); external;

begin {main}
   fildes:=gopen('/dev/tty',OUTDEV,'hp262x',INIT);
   if fildes = -1 then exit(-1);

   status := gclose(fildes);
end.
```

The Include Files

The files `starbase.p1.h` and `starbase.p2.h` contain the definitions and terms needed for any Starbase program written in Pascal. These files are located in the `/usr/include` directory.

To see what definitions and terms are needed, use the `more` command. The syntax is:

```
more /usr/include/starbase.p1.h
```

and

```
more /usr/include/starbase.p2.h
```

The Gopen Function

The function `gopen` opens the specified graphic device. This function requires four parameters:

1. **Device File Name** - Device files are created by the System Administrator. Device files are located in the `/dev` directory. In the following examples, the device file `tty` is used.
2. **Performance Mode** - This parameter may be one of the following:

- OUTDEV - device driver is to output information to the device.
- INDEV - device driver is to read information from the device.
- OUTINDEV - device driver can send and receive information to and from the device.

The following examples use OUTDEV since the example program is to be drawn on the output device's display.

3. **Character Representation of the Driver Type** - For the following example, this will be `hp262x`. This parameter identifies the driver to be used. The `hp262x` driver interrogates the device specified by the device file name and tests to see if the device is an HP 2623 Graphics Terminal or an HP 2627 Color Graphics Terminal, and acts accordingly.

   A possible character representation for an HP-GL plotter is `hpgl`.

4. **Operation Mode** - This parameter may be one of the following:

   0 - open the device, but do nothing else.
   INIT - open and initialize the device in a device dependent way.
   RESET - open and completely initialize the device.
   SPOOLED - open the device for spooled operation. Spooled output may be saved in a file for later display.
   THREE_D - open the device and set Starbase to 3-dimensional mode.

   These modes may be combined with the proper ADD syntax. In Pascal this is the plus sign (+) as in (INIT+SPOOLED).

By doing an ADD operation on INIT and SPOOLED, the graphic device can be opened, initialized and spooled to a file.

For example:

```
fildes = gopen('spoolfile',OUTDEV,'hpgl',INIT+SPOOLED);
```

If the opening is not successful, `gopen` returns a value of `-1`. If the opening is successful, a positive integer (file descriptor) is returned and the HP-GL device is initialized and ready for spooled output.

Further information can be obtained about this procedure by using the `man` command. Just type in:

```
man 3 gopen
```

The Gclose Function

To properly close a graphics device, use the `gclose` function. The `fildes` file descriptor identified by the `gopen` function is used to identify a particular device for all following Starbase procedures and functions, including `gclose`.

Further information can be obtained about this procedure by using the `man` command. Just type in:

```
man 3 gclose
```

A Review

As a review, the following table lists information about the files, functions and commands just discussed.

Table 1-9. A Review

Required Items	Learn More
The cd Command	/usr/man/man1/cd.1 (man cd)
The pc Compiler Command	/usr/man/man1/pc.1 (man pc)
Shell Scripts	HP-UX Concepts and Tutorials, volume 3.
Device Drivers	Starbase Device Drivers Library. /usr/lib/libdd* (ls /usr/lib/libdd...)
Definitions and Terms needed for any Starbase program written in Pascal.	/usr/include/starbase.p1.h /usr/include/starbase.p2.h (more /usr/include/starbase.p1.h) (more /usr/include/starbase.p2.h)
gopen	/usr/man/man3/gopen.3g (man 3 gopen)
gclose	/usr/man/man3/gclose.3g (man 3 gclose)

Virtual Device Coordinates

The default Cartesian coordinate system used by the Starbase procedures is called the Virtual Device Coordinate (VDC) system. This system can be considered a 2-dimensional subset of the Cartesian plane with default x- and y-axis values ranging from 0.0 to 1.0. In 3-dimensional graphics, this system can also be considered a subset of Cartesian 3-space with default x-, y- and z-axis values ranging from 0.0 to 1.0. Locations in VDC space are specified in VDC values as specified by the `vdc_extent` procedure.

The procedure `gopen` automatically creates a Virtual Device Coordinate to Device Coordinate (DC) transformation matrix for each opened graphic device. All VDC data is processed through this matrix before being set to the target device. Since each device has a unique transformation matrix, the same data will appear as expected on each device.

The range of values used for this system can be altered with the `vdc_extent` procedure.

Use the `push_vdc_matrix` procedure to force the current working matrix to be the VDC to DC transformation matrix.

Device Coordinates

Device coordinate Space is the Cartesian 2-dimensional space that is the device's output surface. The range and magnitude of x- and y-axis values is defined by each device.

Device Coordinate procedures bypass the VDC to DC transformation matrix. Such procedures begin with the characters *dc*, for "device coordinate". For example, `dcdraw` will draw a line on the specified device using the untransformed coordinate values specified.

To transform VDC values to DC values, use the `vdc_to_dc` procedure.

World Coordinates

World coordinates can have any value the user desires. This is handy when the data is to be used for purposes other than making a drawing.

Since world coordinates can be of any value, they must be transformed to device coordinates before being drawn on a graphic device. This transformation can be done in several ways. For example:

- A transformation matrix with the appropriate values to transform the World Coordinate data into Device Coordinate data may be supplied by the user.
- A transformation matrix with the appropriate values to transform the World Coordinate data into Virtual Device Coordinate data may be supplied by the user. This matrix could then be concatenated (matrix multiplied) with the Virtual Device Coordinate to Device Coordinate transformation matrix to make a World Coordinate to Device Coordinate Matrix.

For instance, in 2-dimensional mode, a matrix containing:

$$\begin{matrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \\ 0 & 0 \end{matrix}$$

will rotate world coordinates counter-clockwise by an angle of θ.

- **The Easiest Way** is to use the `vdc_extent` procedure to change the ranges of values used to define the Virtual Device Coordinate Space so that the world coordinate values are now contained in VDC space. Any time the `vdc_extent` procedure is used, a new Virtual Device Coordinate to Device Coordinate matrix is calculated that maps Virtual Device Coordinate Space into Device Coordinate Space.

Setting the VDC Extent

In the example problem, we want to plot voltage values against the associated measurement number. Our voltage data ranges from just over 0.1600 through just under 0.1800, while we have 100 measurements (from 1 to 100).

Logical values for the vdc extent would be -20 to 105 on the x-axis and 0.1575 to 0.1825 in the y-axis. This will allow us to plot the data in a rectangle of size 0-100 on the x-axis and 0.1600-0.1800 on the y-axis and still leave room for labels within the vdc extent and outside the plotting area. Since we are doing 2-dimensional graphics, the z-coordinates are both 0.0. The syntax to do this is:

```
vdc_extent(fildes,-20.0,0.1575,0.0,105.0,0.1825,0.0);
```

Detailed information on this procedure can be found using the HP-UX `man` command as follows:

```
man 3 vdc_extent
```

NOTE

The decision was made to follow ANSI parameter patterns in the Starbase procedures. These parameter patterns are NOT CONSISTENT. One procedure may call for x1,y1,z1,x2,y2,z2 while the next may require x1,x2,y1,y2,z1,z2. Hopefully this inconstancy will be removed from the ANSI standard and thus from the Starbase procedures.

Clipping

Clipping is the process of not allowing selected graphic data to be displayed. There are two clipping processes:

- Hard Clip - This is clipping done by the graphic output device that physically stops plotting outside the physical limits of the device's display area. For example, a plotter's pen can not draw past the physical edge of the plotter's platen.
- Soft Clip - This is clipping done by the graphics software to limit the visible portion of the plot. Soft clipping can be done to both 2-dimensional and 3-dimensional data.

Starbase provides two clipping boundaries:

- The Clip Rectangle - A rectangle can be specified with the `clip_rectangle` procedure that defines subsequent clipping boundaries.
- The VDC Extent - The rectangle defined by the `vdc_extent` procedure can also be used to define subsequent clipping boundaries.

  The `clip_indicator` procedure identifies which of the two boundaries to clip to.

The `clip_depth` procedure is used to define the front and back clipping planes for 3-dimensional graphics. Front and back clipping is enabled or disabled with the `depth_indicator` procedure.

Isotropic (undistorted) Mapping

Isotropic mapping is when one unit in the x-axis **exactly equals** one unit in the y-axis. This is sometimes referred to as *undistorted* plotting.

The *mapping_mode* procedure sets the mapping mode to either anisotropic or isotropic plotting mode. The default is isotropic.

The following example program draws an **isotropic** square around the default Virtual Device Coordinate space. This program is in file *example2p.p*.

```
program main(input,output);

$include '/usr/include/starbase.p1.h'$

var
     fildes: integer;

$include '/usr/include/starbase.p2.h'$
procedure exit(result: integer); external;

begin {main}
     fildes:=gopen('/dev/tty',OUTDEV,'hp262x',INIT);
     if fildes = -1 then exit(-1);

     interior_style(fildes,INT_HOLLOW,1);
     rectangle(fildes,0.0,0.0,1.0,1.0);

     fildes := gclose(fildes);
end.
```

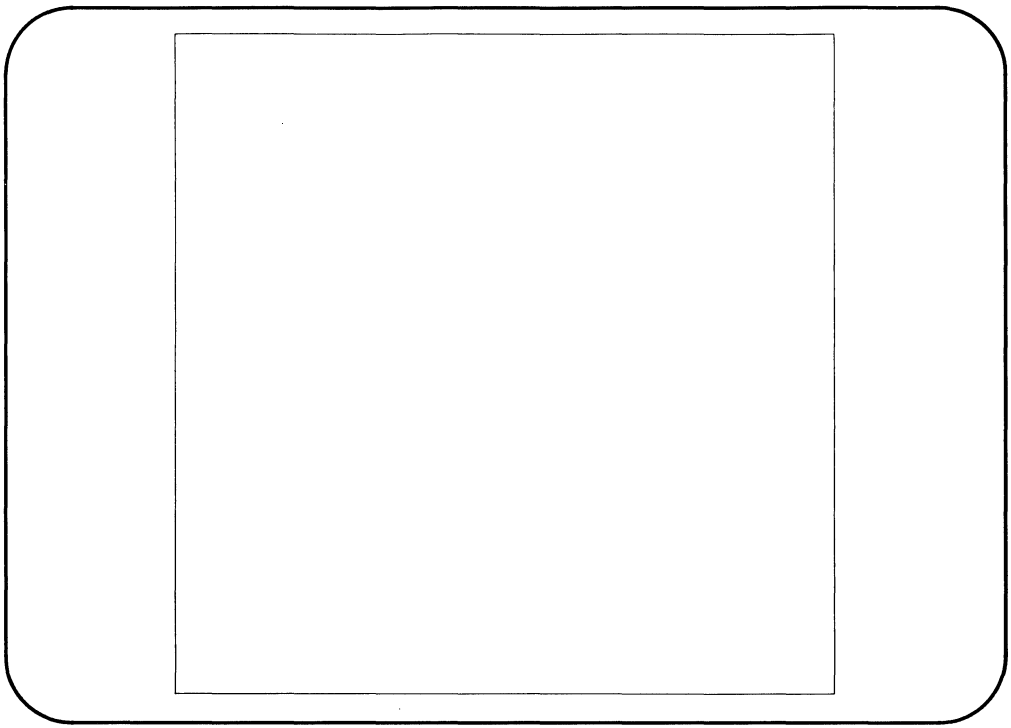


Figure 1-16. An Isotropic Frame of VDC Space

Anisotropic (distorted) Mapping

Anisotropic mapping where one unit in the X direction does not have to equal one unit in the Y direction. This is sometimes called *distorted* plotting.

The `mapping_mode` procedure sets the mapping mode to either anisotropic or isotropic plotting mode. The following single line was added to the preceeding isotropic program and the anisotropic results are shown in Figure 1-17. The rectangle still frames the VDC default space, but now the space fills the usable portion of the display. This program is in *example3p.p*.

```
mapping_mode(fildes,1);
```

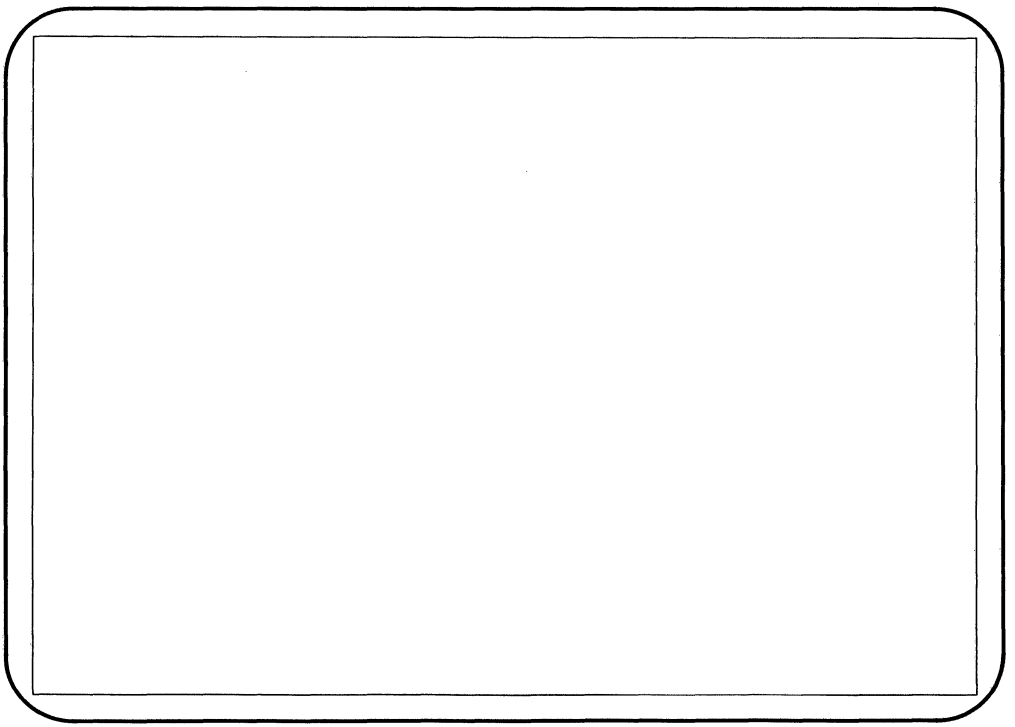


Figure 1-17. An Anisotropic Frame of VDC Space

Since the scale for the voltage measurements is not the same as the scale for the number of measurements, anisotropic plotting is appropriate.

Viewports

The viewport is the portion of the output display that is used to draw the picture. This can be all or only a part of the display.

The viewport boundaries is defined by several procedures.

- `set_p1_p2` - This procedure defines the physical region that contains the viewport.
- `mapping_mode` - This procedure defines the viewport as *isotropic* (distorted) or *anisotropic* (undistorted). If anisotropic, the viewport is the same as the rectangle defined by the `set_p1_p2` procedure. If the viewport is isotropic, the viewport is the largest rectangle with the same aspect ratio as the VDC extent that will fit in the rectangle defined by the `set_p1_p2` procedure and is positioned in that rectangle by the `viewport_justification` procedure.
- `viewport_justification` - This procedure defines the fraction of "white space" (the area of the `set_p1_p2` rectangle not within the viewport) to the left or bottom of the viewport. The default is .5 (50%), so the viewport is centered in the rectangle defined by `set_p1_p2`. This procedure only applies to isotropic viewports.

Rectangles

The `rectangle` procedure provides an easy way to draw rectangles. All you need to specify is the device needing a rectangle and the coordinates of two opposite corners of the rectangle. A default rectangle has a filled interior and no perimeter.

The syntax for drawing two rectangles, one to enclose the vdc extent and the other to enclose our plotting area is shown below.

```
rectangle(fildes,-20,0.1575,105.0,0.1825);
rectangle(fildes,0,0.1600,100.0,0.1800);
```

To make rectangles hollow, and add the perimeter, use the `interior_style` procedure. The following syntax will do the job.

```
interior_style(fildes,INT_HOLLOW,1);
```

The following program segment is added to the program in order to draw the rectangles. The program is stored as *example4p.p.*

```
program main(input,output);

$include '/usr/include/starbase.p1.h'$

const

var
     fildes : integer;

$include '/usr/include/starbase.p2.h'$
procedure exit(result: integer); external;

begin {main}
     fildes:=gopen('/dev/tty',OUTDEV,'hp262x',INIT);
     if fildes = -1 then exit(-1);

     vdc_extent(fildes,-20.0,0.1575,0.0,105.0,0.1825,0.0);
     clip_rectangle(fildes,-20.0,105.0,0.1575,0.1825);
     mapping_mode(fildes,1);

     interior_style(fildes,INT_HOLLOW,1);
     rectangle(fildes,-20.0,0.1575,105.0,0.1825);
     rectangle(fildes,0.0,0.1600,100.0,0.1800);

     fildes := gclose(fildes);
end.
```

To see the extent of the Virtual Device Coordinate Space and to draw a frame around the location on the display, two rectangles are drawn.

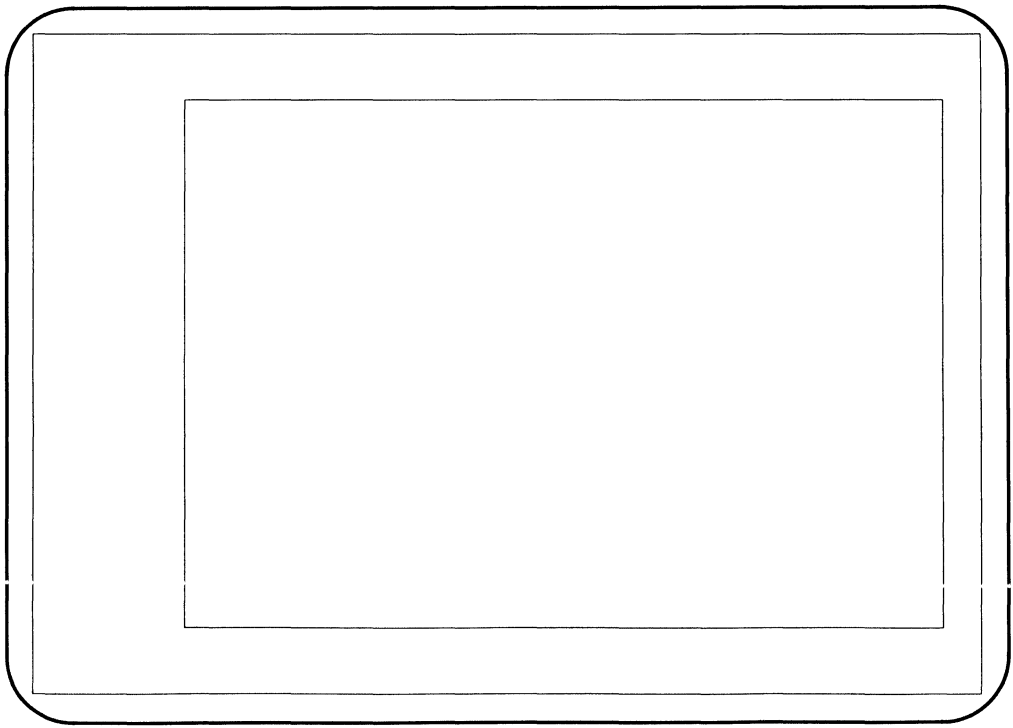


Figure 1-18. Two Rectangles

Plotting the Data

The following program will read the plot data from the file *data* and draw the results in Virtual Device Coordinate space. The main procedure is `polyline` which will move the graphics pen to the first (X,Y) data location and then draw to the remaining data locations. The data is read into a single-dimension array usable by `polyline`. The number of points to be drawn to is *i/2* and the `0` flag indicates that there are no move-draw (0 for move, 1 for draw) indicators in the data.

The syntax is:

```
program main(input,output);

$include '/usr/include/starbase.p1.h'$

const
     array_size = 200;

type
    array_type = array[0..array_size] of real;

var
    fildes, i : integer;
    coords: array_type;
    data: text;

$include '/usr/include/starbase.p2.h'$
procedure exit(result: integer); external;

begin {main}
     fildes:=gopen('/dev/tty',OUTDEV,'hp262x',INIT);
     if fildes = -1 then exit(-1);

     vdc_extent(fildes,-20.0,0.1575,0.0,105.0,0.1825,0.0);
     clip_rectangle(fildes,-20.0,105.0,0.1575,0.1825);
     mapping_mode(fildes,1);

     interior_style(fildes,INT_HOLLOW,1);
     rectangle(fildes,-20.0,0.1575,105.0,0.1825);
     rectangle(fildes,0.0,0.1600,100.0,0.1800);

     reset(data,'data');
     while(not eof(data)) do
     begin
         readln(data,coords[i],coords[i+1]);
         i := i+2;
     end;

     polyline2d(fildes,coords,i div 2,0);
```

```
    fildes := gclose(fildes);
end.
```

The results of this modification to the example program are shown in Figure 1-19. This drawing was made with *example5p.p.*

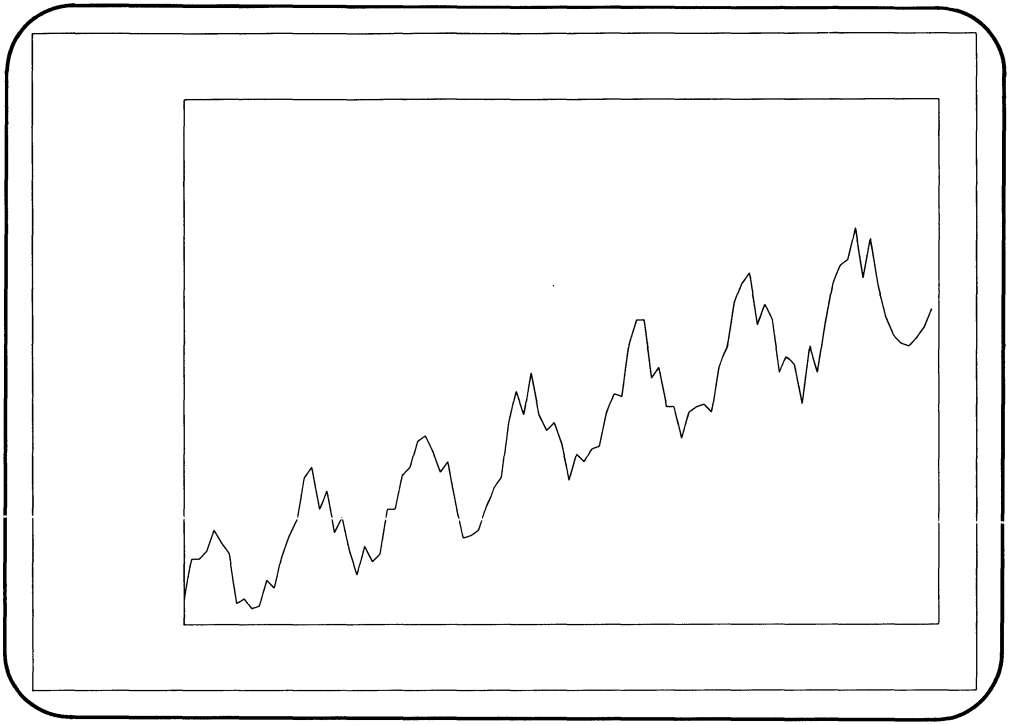


Figure 1-19. Plotted Data

Labels

Let us now label the drawing. We will use two fonts, different character paths, and different text alignments to show some of the capabilities of the system.

The syntax to do this is:

```
program main(input,output);

$include '/usr/include/starbase.p1.h'$

const
    array_size = 200;

type
    array_type = array[0..array_size] of real;

var
    fildes, i, j, length: integer;
    x, y: real;
    coords: array_type;
    data: text;
    number: string[80];

$include '/usr/include/starbase.p2.h'$
procedure exit(result: integer); external;

begin {main}
    fildes:=gopen('/dev/tty',OUTDEV,'hp262x',INIT);
    if fildes = -1 then exit(-1);

    vdc_extent(fildes,-20.0,0.1575,0.0,105.0,0.1825,0.0);
    clip_rectangle(fildes,-20.0,105.0,0.1575,0.1825);
    mapping_mode(fildes,1);

    interior_style(fildes,INT_HOLLOW,1);
    rectangle(fildes,-20.0,0.1575,105.0,0.1825);
    rectangle(fildes,0.0,0.1600,100.0,0.1800);

    text_alignment(fildes,TA_CENTER,TA_TOP,0.0,0.0);
    character_height(fildes,0.001);
    text2d(fildes,50.0,0.1588,'Time (seconds)',VDC_TEXT,0);

    character_path(fildes,PATH_DOWN);
    text_alignment(fildes,TA_CENTER,TA_HALF,0.0,0.0);
    text2d(fildes,-14.0,0.1700,'Voltage',VDC_TEXT,0);

    text_font_index(fildes,2);
    character_path(fildes,PATH_RIGHT);
    text_alignment(fildes,TA_CENTER,TA_BOTTOM,0.0,0.0);
```

```
    text2d(fildes,50.0,0.1810,'VOLTAGE VARIANCE',VDC_TEXT,0);

    reset(data,'data');
    while(not eof(data)) do
    begin
        readln(data,coords[i],coords[i+1]);
        i := i+2;
    end;

        polyline2d(fildes,coords,i div 2,0);

    fildes := gclose(fildes);
end.
```

Our example is now shown Figure 1-20. This program is stored in *example6p.p*.

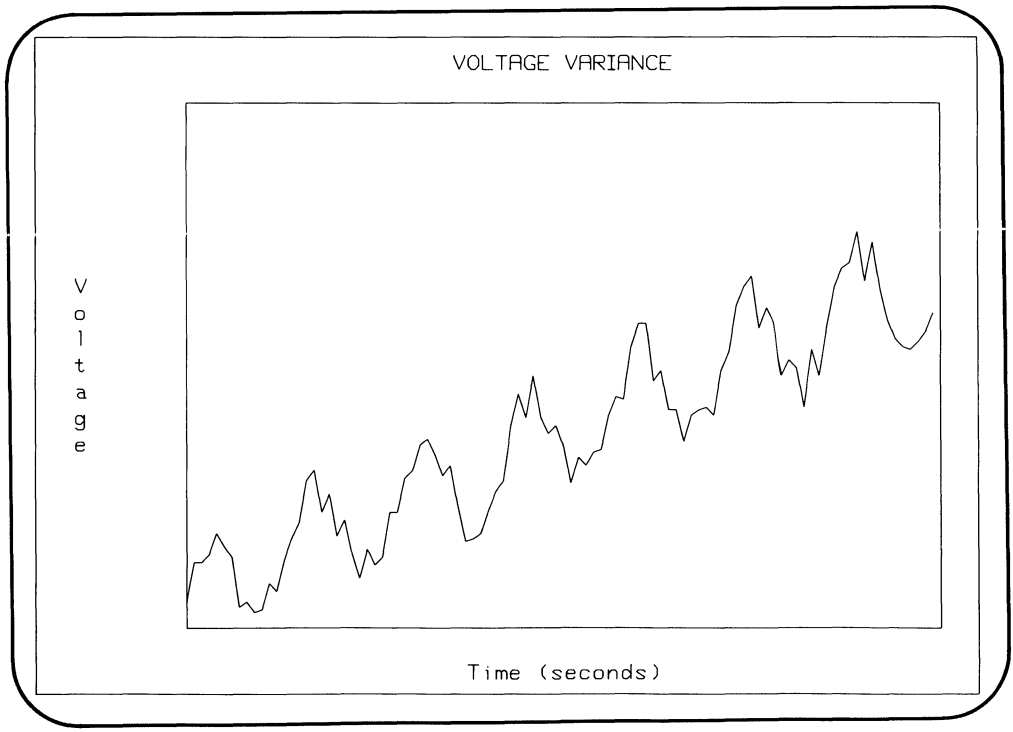


Figure 1-20. A Labeled Plot

Tic Marks

We now need to add the numerated tic marks. This is done with two loops. We use `strwrite` to change the tic mark numbers into strings to be drawn as text.

```
program main(input,output);

$include '/usr/include/starbase.p1.h'$

const
    array_size = 200;

type
     array_type = array[0..array_size] of real;

var
     fildes, i, j, length: integer;
     x, y: real;
     coords: array_type;
     data: text;
     number: string[80];

$include '/usr/include/starbase.p2.h'$
procedure exit(result: integer); external;

begin {main}
     .
     .
     .
     text_alignment(fildes,TA_CENTER,TA_CONTINUOUS_VERTICAL,0.0,1.2);
     character_height(fildes,0.0006);

     x := 0.0;
     strwrite(number,1,length,trunc(x):1);
     text2d(fildes,x,0.1598,number,VDC_TEXT,0);
     for i:=1 to 10 do
     begin
          for j:=1 to 5 do
          begin
             move2d(fildes,x,0.1600);
             draw2d(fildes,x,0.1603);
                  x := x + 2.0;
           end;
             move2d(fildes,x,0.1600);
             draw2d(fildes,x,0.1605);
             strwrite(number,1,length,trunc(x):1);
             text2d(fildes,x,0.1600,number,VDC_TEXT,0);
     end;

     text_alignment(fildes,TA_CONTINUOUS_HORIZONTAL,TA_HALF,1.2,0.0);
```

```
y := 0.1600;
strwrite(number,1,length,y:0:4);
text2d(fildes,0.0,y,number,VDC_TEXT,0);
for i:=1 to 8 do
begin
       for j:=1 to 5 do
       begin
            move2d(fildes,0.0,y);
            draw2d(fildes,1.0,y);
            y := y + 0.0005;
        end;
        move2d(fildes,0.0,y);
        draw2d(fildes,2.0,y);
        strwrite(number,1,length,y:0:4);
        ext2(fildes,0.0,y,number,VDC_TEXT,0);
end;
```

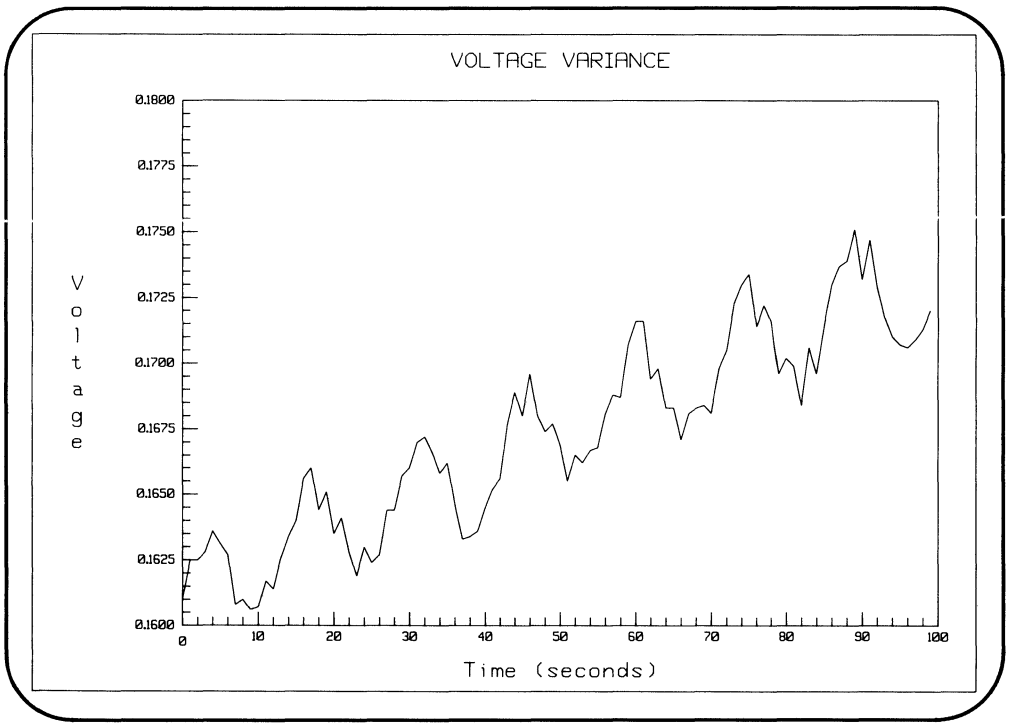


Figure 1-21. The Final Plot

The whole program

```
program main(input,output);

$include '/usr/include/starbase.p1.h'$

const
    array_size = 200;

type
    array_type = array[0..array_size] of real;

var
     fildes, i, j, length: integer;
     x, y: real;
     coords: array_type;
     data: text;
     number: string[80];

$include '/usr/include/starbase.p2.h'$
procedure exit(result: integer); external;

begin {main}
    fildes:=gopen('/dev/tty',OUTDEV,'hp262x',INIT);
    if fildes = -1 then exit(-1);

    vdc_extent(fildes,-20.0,0.1575,0.0,105.0,0.1825,0.0);
    clip_rectangle(fildes,-20.0,105.0,0.1575,0.1825);
    mapping_mode(fildes,1);

    interior_style(fildes,INT_HOLLOW,1);
    rectangle(fildes,-20.0,0.1575,105.0,0.1825);
    rectangle(fildes,0.0,0.1600,100.0,0.1800);

    text_alignment(fildes,TA_CENTER,TA_TOP,0.0,0.0);
    character_height(fildes,0.001);
    text2d(fildes,50.0,0.1588,'Time (seconds)',VDC_TEXT,0);

    character_path(fildes,PATH_DOWN);
    text_alignment(fildes,TA_CENTER,TA_HALF,0.0,0.0);
    text2d(fildes,-14.0,0.1700,'Voltage',VDC_TEXT,0);

    text_font_index(fildes,2);
    character_path(fildes,PATH_RIGHT);
    text_alignment(fildes,TA_CENTER,TA_BOTTOM,0.0,0.0);
    text2d(fildes,50.0,0.1810,'VOLTAGE VARIANCE',VDC_TEXT,0);

    reset(data,'data');
    while(not eof(data)) do
    begin
         readln(data,coords[i],coords[i+1]);
```

```
        i := i+2;
    end;

    polyline2d(fildes,coords,i div 2,0);

    text_alignment(fildes,TA_CENTER,TA_CONTINUOUS_VERTICAL,0.0,1.2);
    character_height(fildes,0.0006);

    x := 0.0;
    strwrite(number,1,length,trunc(x):1);
    text2d(fildes,x,0.1598,number,VDC_TEXT,0);
    for i:=1 to 10 do
    begin
        for j:=1 to 5 do
        begin
            move2d(fildes,x,0.1600);
            draw2d(fildes,x,0.1603);
            x := x + 2.0;
        end;
            move2d(fildes,x,0.1600);
            draw2d(fildes,x,0.1605);
            strwrite(number,1,length,trunc(x):1);
            text2d(fildes,x,0.1600,number,VDC_TEXT,0);
    end;

    text_alignment(fildes,TA_CONTINUOUS_HORIZONTAL,TA_HALF,1.2,0.0);

    y := 0.1600;
    strwrite(number,1,length,y:0:4);
    text2d(fildes,0.0,y,number,VDC_TEXT,0);
    for i:=1 to 8 do
    begin
        for j:=1 to 5 do
        begin
            move2d(fildes,0.0,y);
            draw2d(fildes,1.0,y);
            y := y + 0.0005;
        end;
        move2d(fildes,0.0,y);
        draw2d(fildes,2.0,y);
        strwrite(number,1,length,y:0:4);
        text2d(fildes,0.0,y,number,VDC_TEXT,0);
    end;

    fildes := gclose(fildes);
end.
```

Notes

Working with Text 2

Introduction

This section provides several examples of how text can be manipulated with Starbase. To be complete, the Text Alignment subsection gives numerous examples of how text can be aligned with respect to the coordinate position specified in the `text2d` or `text3d` procedures.

The Starbase Library Procedures discussed in this section include:

- Procedures to draw text:
  - text2d
  - text3d
  - append_text
  - dctext
- Procedures to specify character sizes:
  - character_expansion_factor
  - character_height
  - character_width
- Procedures to select different fonts and character sets:
  - designate_character_set
  - text_font_index
- Procedures to specify character locations relative to other characters:
  - character_slant
  - text_line_space
  - text_path
  - intra_character_space
  - text_line_path

- text_orientation
- text_alignment
- text_precision

- A Procedure to specify the method by which charcters and escape code sequences are interpreted:
  - character_switching_mode

The Four Methods of Drawing Text

- **append_text** - is used to add text to previously executed `text2d` or `text3d` procedures when the parameter `more` is specified as `TRUE` or not equal to `0`. This procedure also draws text at the current pen position.

  C Syntax:

  ```
  void append_text(fildes,string,xform,more);
  int fildes,xform,more;
  char *string;
  ```

  Fortran77 Syntax:

  ```
  subroutine append_text(fildes,string,xform,more)
  integer*4  fildes,xform,more
  character*(*) string
  ```

  Pascal Syntax:

  ```
  procedure append_text(fildes:integer;string:string255;xform,more:
  integer);
  ```

- **text2d** - draws text at the specified 2-dimensional Virtual Device Coordinate (x,y) position.

  The syntax for is procedure is:

  C Syntax:

  ```
  void text2d(fildes,x,y,string,xform,more);
  int fildes,xform,more;
  float x,y;
  char *string;
  ```

Fortran77 Syntax:

```
subroutine text2d(fildes,x,y,string,xform,more)
integer*4  fildes,xform,more
real x,y
character*(*) string
```

Pascal Syntax:

```
procedure text2d(fildes,x,y:integer;string:string255;xform,more:integer);
```

- **text3d** - draws text at the specified 3-dimensional Vitrual Device Coordinate (x,y,z) position.

  The syntax for is procedure is:

  C Syntax:

```
void text3d(fildes,x,y,z,string,xform,more);
int fildes,xform,more;
float x,y,z;
char *string;
```

  Fortran77 Syntax:

```
subroutine text3d(fildes,x,y,z,string,xform,more)
integer*4  fildes,xform,more
real x,y,z
character*(*) string
```

  Pascal Syntax:

```
procedure text3d(fildes,x,y,z:integer;string:string255;xform,more:integer);
```

- **dctext** - draws text at the specified device coordinate (dcx,dcy) position.

  C Syntax:

```
void dctext(fildes,dcx,dcy,string);
int fildes,dcx,dcy;
char *string;
```

  Fortran77 Syntax:

```
subroutine dctext(fildes,dcx,dcy,string)
integer*4  fildes,dcx,dcy
character*(*) string
```

  Pascal Syntax:

```
procedure dctext(fildes,dcx,dcy:integer;string:string255);
```

Text Attributes and Defaults

The following attributes are set using the values in `/usr/lib/starbase/defaults`.

- ISO 8-bit Character Mode.
- G0 and G2 point to the USASCII character set, font 1. This is the default active character set.
- G1 and G3 point to the HPROMAN character set, font 1.

When a graphics output device is opened with the `gopen` procedure, the following text defaults are in effect.

- Character_Path is PATH_RIGHT.
- Character_Slant is 0.
- Text_alignment is TA_NORMAL_HORIZONTAL and TA_NORMAL_VERTICAL.
- Text_color_index is 1.
- Text_line_path is PATH_DOWN.
- Text_precision is STROKE_TEXT.
- All other default values are machine-dependent.

Except for `dctext` which is not transformed, all text is transformed with one of three matrices described below. The transformation to be done is specified by the `xform` parameter of the text procedures.

- If `xform` = `0`, then the Font Transformation Matrix is pre-concatenated with the Virtual Device Coordinate (VDC) to Device Coordinate (DC) Transformation Matrix and used as the font to device coordinate transformation matrix.
- If `xform` = `1`, the Font Transformation Matrix is pre-concatenated to the transformation matrix on the top of the matrix stack and the resulting matrix is used as the font to device coordinate transformation matrix.
- If `xform` = `2`, the transformation matrix on top of the matrix stack is used as the font to device coordinate transformation matrix.

The `text2d` and `text3d` procedures use the flag `more` to indicate when the text specified in the procedure is to be buffered. The `append_text` procedure is then used to append text to the buffer.

- If `more` = `FALSE` or `0`, no more text is expected, and the text is sent to the specified device.
- if `more` = `TRUE` or not equal to `0`, more text is to follow and the text is buffered. The `append_text` procedure is used to append more text to a `text2d` or `text3d` buffer. This flag is cancelled and any buffered text is sent to the specified device if a `text2d` or `text3d` procedure is executed.

Character sets are directories stored in `/usr/lib/starbase/stroke`. Fonts are stored in the character set directories. For example, the fixed-width, stroked USASCII font is the file named `1` and is located in:

```
/usr/lib/starbase/stroke/usascii/1
```

Template Programs

The following programs may be used as templates for the example programs to follow. To emphasize the text procedures being discussed, only those procedures will be listed. These procedures, when added to the templates, will create the complete programs used to draw the examples shown.

The `gopen` function opens an I/O path the the specified graphic device and returns and integer file descriptor used to identify that device until the path is closed with the `gclose` function.

Mapping mode defines whether the mapping of the vdc extent to the viewport is isotropic (`NONDISTORTED`) or anisotropic (`DISTORTED`).

A Template for C Programs

This program is `/user/lib/starbase/demos/text0c.c`. The default file descriptor is assigned to an HP 2623 or HP 2627 Graphics Terminal. In either case, the device is specified and and output device and is initialized.

This program is stored as `example0c.c` in the `/usr/lib/starbase/demos` directory.

```
#include <starbase.c.h>

main(argc,argv)
int argc; char *argv[];
{
   int fildes;

   if (argc > 2) fildes=gopen(argv[1],OUTDEV,argv[2],INIT);
   else fildes=gopen("/dev/tty",OUTDEV,"hp262x",INIT);
   if (fildes == -1) exit(-1);

   /* Starbase Procedures Go Here */

   gclose(fildes);
}
```

A Template for Fortran77 Programs

This template is `/user/lib/starbase/demos/ftemplate.f`. This template will only work with an HP 2623 or HP 2627 Graphics Terminal. To use this template on another device, you must use the appropriate parameters shown bolded in the `fildes` line shown below.

This program is stored as `example0f.f` in the `/usr/lib/starbase/demos` directory.

```
   include '/usr/include/starbase.f1.h'
   program template
   character NULL
   parameter(NULL = char(0))
   include '/usr/include/starbase.f2.h'

   integer*4 fildes,status

   fildes = geopen('/dev/tty'//NULL,OUTDEV,'hp262x'//NULL,INIT)
   if (fildes .eq. -1) stop

C  Starbase Procedures Go Here

   status = gclose(fildes)
   end
```

For example, to use the above template with an HP 98700 Device, the `fildes` line would look like this:

```
fildes = geopen('/dev/98700'//NULL,OUTDEV,'hp98700'//NULL,INIT)
```

A Template for Pascal Programs

This template is stored in the file `/user/lib/starbase/demos/ptemplate.p`. This template will only work with an HP 2623 or HP 2627 Graphics Terminal. To use this template on another device, you must use the appropriate parameters for the ones bolded in the `fildes` line shown below.

This program is stored as `example0p.p` in the `/usr/lib/starbase/demos` directory.

```
program main(input,output);

$include '/usr/include/starbase.p1.h'$

var
   fildes, status : integer;

$include '/usr/include/starbase.p2.h'$

procedure exit(result : integer);external;

begin {main}

   fildes := gopen('/dev/tty',OUTDEV,'hp262x',INIT);
   if fildes = -1 then exit(-1);

    { Starbase Procedures Go Here }

   status := gclose(fildes);
end.
```

For example, to use the above template with an HP 98700 Device, the `fildes` line would look like this:

```
fildes := gopen('/dev/98700',OUTDEV,'hp98700',INIT);
```

Default Text

The data values which define the characters that make up the text are all transformed by an internally calculated matrix into Virtual Device Coordinate values. For a discussion of the coordinate systems used by Starbase, see section 1 of this manual.

The following example shows what default text looks like. Both the `append_text` and `text2d` procedures are used.

This example was created by the program `text1c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
mapping_mode(fildes,TRUE);

move2d(fildes,0.10,0.10);
append_text(fildes,"Append_text defaults to Current Pen Position.",
VDC_TEXT,FALSE);

text2d(fildes,0.10,0.90," Default Text with  ",VDC_TEXT,TRUE);
append_text(fildes," Appended Text ",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
      call mapping_mode(fildes,1)

      call move2d(fildes,0.10,0.10)
      call append_text(fildes,'Append_text defaults to Current Pen Position.'
     c //NULL, VDC_TEXT,FALSE)

      call text2d(fildes,0.01,0.95,' Default Text with  '//NULL,VDC_TEXT,TRUE)
      call append_text(fildes,' Appended Text '//NULL,VDC_TEXT,FALSE)
```

Pascal Syntax

```
mapping_mode(fildes,1);

move2d(fildes,0.10,0.10);
append_text(fildes,'Append_text defaults to Current Pen Position.',
VDC_TEXT,FALSE);

text2d(fildes,0.01,0.95,' Default Text with  ',VDC_TEXT,TRUE);
append_text(fildes,' Appended Text ',VDC_TEXT,FALSE);
```

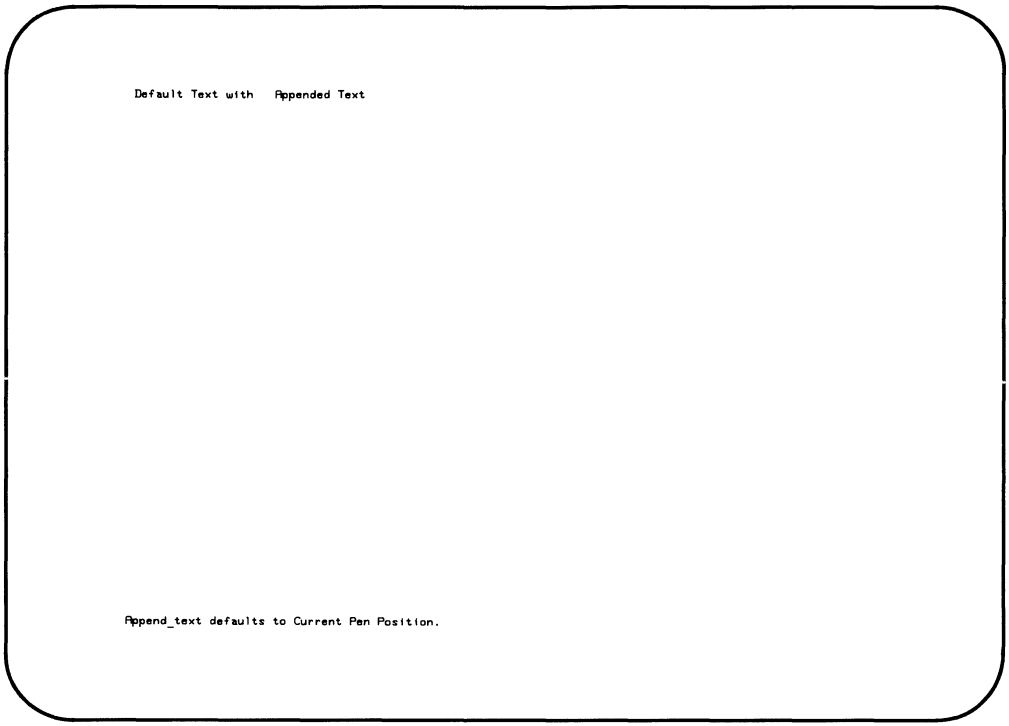


Figure 2-1. Default Text (text1c.c)

Larger Text

The previous example shows how default text can be quite small. For larger and more readable text, use the `character_height` procedure.

The `fildes` parameter identifies the output graphic device. The `height` parameter is a fraction of the Y-axis length for the Virtual Device Coordinate extent and specifies the character's height.

Changing character height may also change character width. The default ratio of character height to character width is maintained at 1.0 unless changed with the `character_expansion_factor` procedure. Character width may also be changed with the `character_width` procedure.

Changing a character's width does **not** change the character's height.

Our example will specify a character height of `0.05`. This example was created by the program `text2c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
character_height(fildes,0.05);
```

Fortran77 Syntax

```
call character_height(fildes,0.05)
```

Pascal Syntax

```
character_height(fildes,0.05);
```

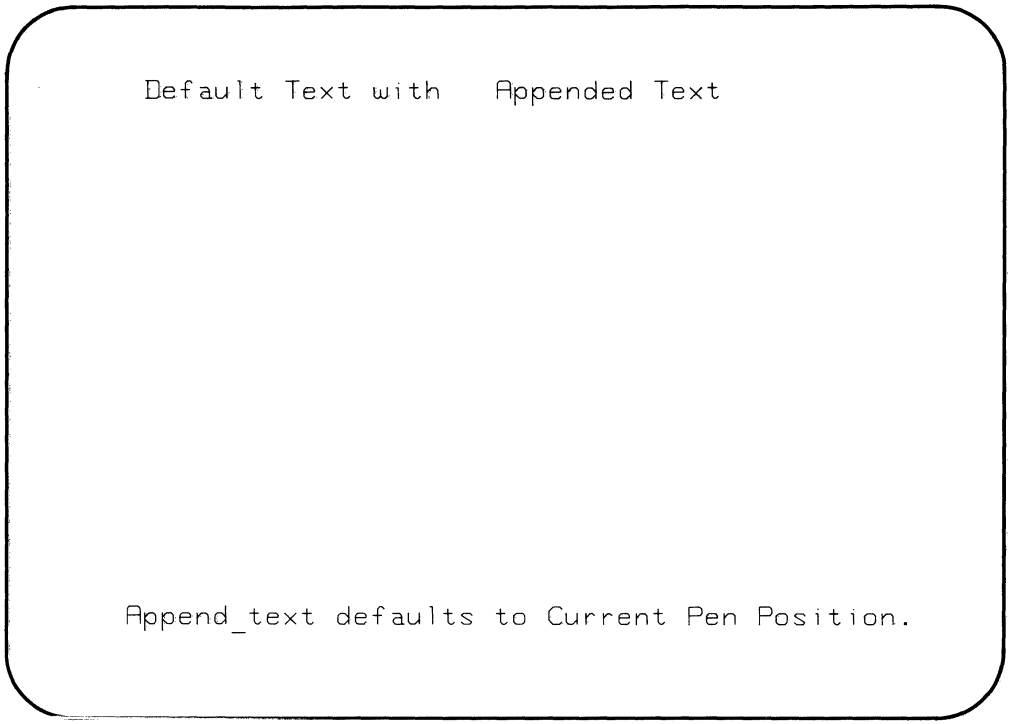


Figure 2-2. Larger Text (text2c.c)

Larger Text Continued

A better look at how `character_height` changes the size of text can be seen with the following example. The character width is automatically changed by `character_expansion_factor` as the height changes. This factor is the ratio of character height to character width. The default ratio is 1.00, so the higher the character, the wider it becomes to keep the factor at 1.00. An example of how the `character_expansion_factor` procedure works is presented later in the section.

This example was created by the program `text3c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
text2d(fildes,0.0,0.8," Default Character Height (0.02)",VDC_TEXT,FALSE);

character_height(fildes,0.04);
text2d(fildes,0.0,0.7," Character Height of 0.04",VDC_TEXT,FALSE);

character_height(fildes,0.06);
text2d(fildes,0.0,0.6," Character Height of 0.06",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
      call text2d(fildes,0.0,0.8,' Default Character Height (0.02)'//NULL,
     c VDC_TEXT,FALSE)

      call character_height(fildes,0.07)
      call text2d(fildes,0.0,0.4,' Character Height of 0.04'//NULL,VDC_TEXT,
     c FALSE)

      call character_height(fildes,0.06)
      call text2d(fildes,0.0,0.6,' Character Height of 0.06'//NULL,VDC_TEXT,
     c FALSE)
```

Pascal Syntax

```
text2d(fildes,0.0,0.8,' Default Character Height (0.02)',VDC_TEXT,FALSE);

character_height(fildes,0.04);
text2d(fildes,0.0,0.7,' Character Height of 0.04',VDC_TEXT,FALSE);

character_height(fildes,0.06);
text2d(fildes,0.0,0.6,' Character Height of 0.06',VDC_TEXT,FALSE);
```

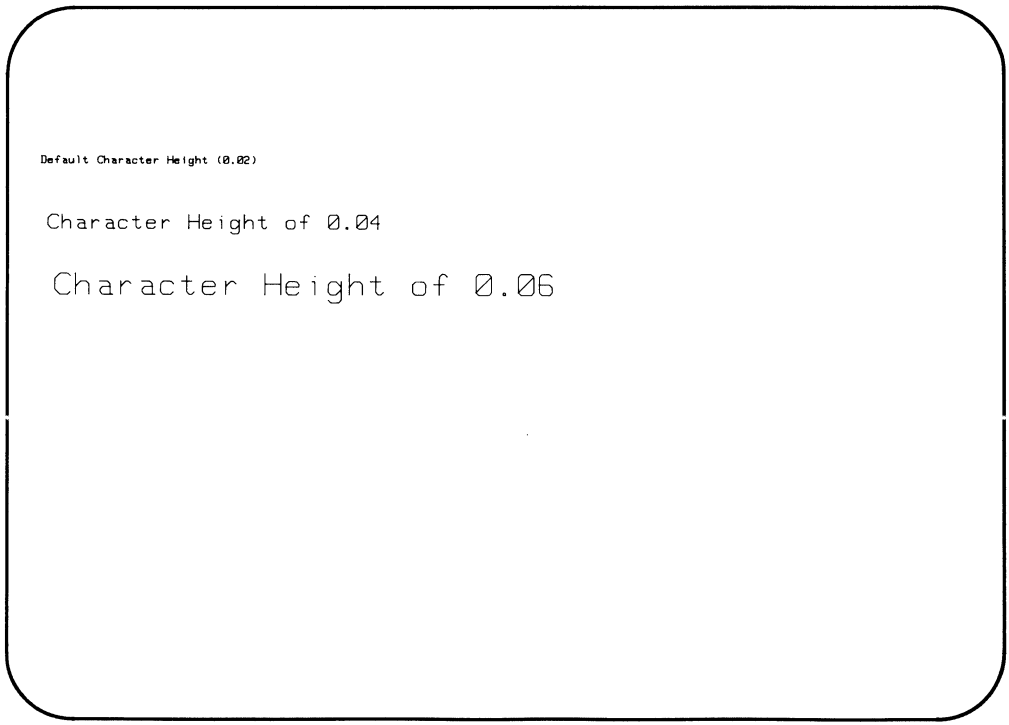


Figure 2-3. Larger Text Continued (text3c.c)

Text Precision

Text precision is selected with the `text_precision` procedure. This procedure has two parameters, *fildes* and *precision.* The first parameter identifies the graphics output device that will have the precision specified in the second parameter.

The second parameter can be one of the three following precisions:

- STRING_TEXT - the device's text capabilities are used. Hardware text generation, in general, provides only limited approximations to the specified text attributes.
- CHARACTER_TEXT - currently the same as STROKE_TEXT.
- STROKE_TEXT - causes the characters to be drawn at a higher precision that hardware text, but may be slow on some devices. This is the default text precision.

Depending upon the device, the visual differences between the various text precisions may or may not be noticable. With STRING_TEXT, changes in an attribute may seem erratic. For example, if you are changing the slant of some test characters. As you change the angle of the slant, the text on your screen my not seem to change until some angle is reached, then the text on you screen will change markedly. This change is device dependent.

Wide and Narrow Characters

Characters can be made wider without changing their height. This is done with the `character_width` procedure.

This example was created by the program `text4c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
character_height(fildes,0.04);

text2d(fildes,0.0,0.50," Default Width of 0.50",VDC_TEXT,FALSE);

character_width(fildes,0.010);
text2d(fildes,0.0,0.10," Character Width of 0.010",VDC_TEXT,FALSE);

character_width(fildes,0.020);
text2d(fildes,0.0,0.20," Character Width of 0.020",VDC_TEXT,FALSE);

character_width(fildes,0.030);
text2d(fildes,0.0,0.30," Character Width of 0.030",VDC_TEXT,FALSE);

character_width(fildes,0.040);
text2d(fildes,0.0,0.40," Character Width of 0.040",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
  call character_height(fildes,0.04)

  call text2d(fildes,0.0,0.50,' Default Width of 0.50'//NULL,VDC_TEXT,
c FALSE)

  call character_width(fildes,0.010)
  call text2d(fildes,0.0,0.10,' Character Width of 0.010'//NULL,VDC_TEXT,
c FALSE)

  call character_width(fildes,0.020)
  call text2d(fildes,0.0,0.20,' Character Width of 0.020'//NULL,VDC_TEXT,
c FALSE)

  call character_width(fildes,0.030)
  call text2d(fildes,0.0,0.30,' Character Width of 0.030'//NULL,VDC_TEXT,
c FALSE)

  call character_width(fildes,0.040)
  call text2d(fildes,0.0,0.40,' Character Width of 0.040'//NULL,VDC_TEXT,
c FALSE)
```

Pascal Syntax

```
character_height(fildes,0.04);

text2d(fildes,0.0,0.50,' Default Width of 0.50',VDC_TEXT,FALSE);

character_width(fildes,0.010);
text2d(fildes,0.0,0.10,' Character Width of 0.010',VDC_TEXT,FALSE);

character_width(fildes,0.020);
text2d(fildes,0.0,0.20,' Character Width of 0.020',VDC_TEXT,FALSE);

character_width(fildes,0.030);
text2d(fildes,0.0,0.30,' Character Width of 0.030',VDC_TEXT,FALSE);

character_width(fildes,0.040);
text2d(fildes,0.0,0.40,' Character Width of 0.040',VDC_TEXT,FALSE);
```

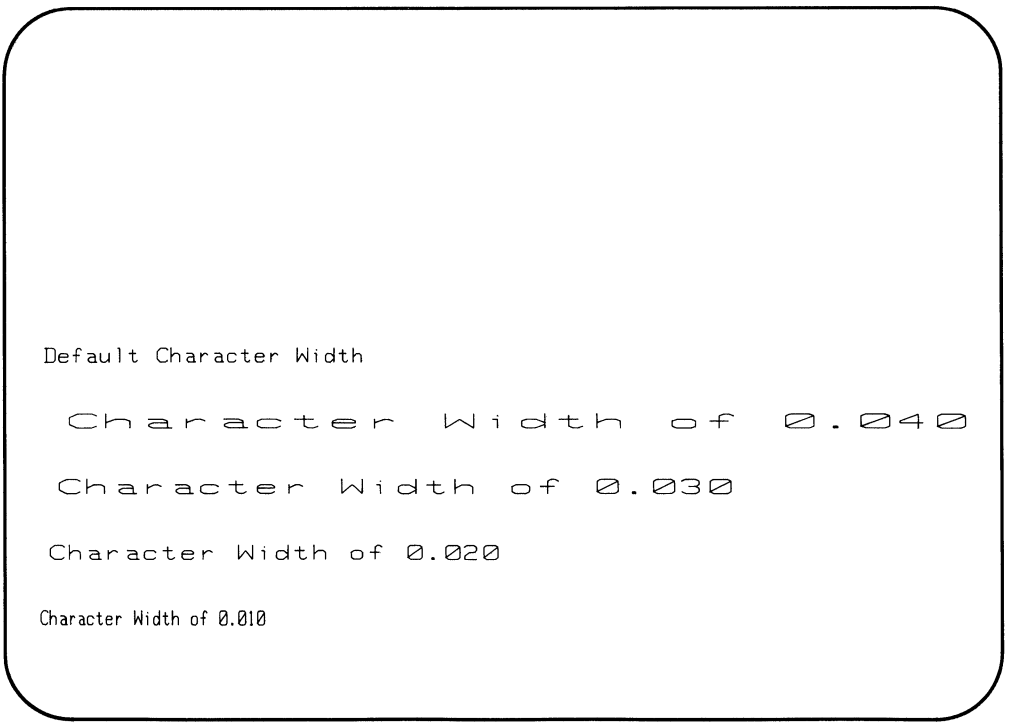


Figure 2-4. Wide and Narrow Characters (text4c.c)

Character Expansion Factor

The `character_expansion_factor` and `character_width` procedures are interactive with one another. The procedure to be executed last supersedes the former. If the last procedure was `character_expansion_factor`, then the width is adjusted with respect to the height of the characters. If the last procedure was `character_width`, then the width is adjusted without respect to the character's height.

This example was created by the program `text5c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
character_height(fildes,0.04);

character_expansion_factor(fildes,0.75);
text2d(fildes,0.05,0.80,"Character expansion factor 0.75",VDC_TEXT,FALSE);

character_width(fildes,0.03);
text2d(fildes,0.05,0.70,"Character Width of 0.03",VDC_TEXT,FALSE);

character_width(fildes,0.040);
character_expansion_factor(fildes,0.75);
text2d(fildes,0.05,0.60,"Character expansion factor 0.75",VDC_TEXT,FALSE);
text2d(fildes,0.05,0.55,"superseding width change",VDC_TEXT,FALSE);

character_expansion_factor(fildes,1.5);
character_width(fildes,0.030);
text2d(fildes,0.05,0.40,"Character Width of 0.030",VDC_TEXT,FALSE);
text2d(fildes,0.05,0.35,"superseding expansion change",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
      call character_height(fildes,0.04)

      call character_expansion_factor(fildes,0.75)
      call text2d(fildes,0.05,0.80,'Character expansion factor 0.75'//NULL,
     c VDC_TEXT,FALSE)

      call character_width(fildes,0.03)
      call text2d(fildes,0.05,0.70,'Character Width of 0.03'//NULL,VDC_TEXT,
     c FALSE)

      call character_width(fildes,0.040)
      call character_expansion_factor(fildes,0.75)
      call text2d(fildes,0.05,0.60,'Character expansion factor 0.75'//NULL,
     c VDC_TEXT,FALSE)
      call text2d(fildes,0.05,0.55,'superseding width change'//NULL,VDC_TEXT,
     c FALSE)
```

```
   call character_expansion_factor(fildes,1.5)
   call character_width(fildes,0.030)
   call text2d(fildes,0.05,0.40,'Character Width of 0.030'//NULL,VDC_TEXT,
  c FALSE)
   call text2d(fildes,0.05,0.35,'superseding expansion change'//NULL,
  c VDC_TEXT,FALSE)
```

Pascal Syntax

```
   character_height(fildes,0.04);

   character_expansion_factor(fildes,0.75);
   text2d(fildes,0.05,0.80,'Character expansion factor 0.75',VDC_TEXT,FALSE);

   character_width(fildes,0.03);
   text2d(fildes,0.05,0.70,'Character Width of 0.03',VDC_TEXT,FALSE);

   character_width(fildes,0.040);
   character_expansion_factor(fildes,0.75);
   text2d(fildes,0.05,0.60,'Character expansion factor 0.75',VDC_TEXT,FALSE);
   text2d(fildes,0.05,0.55,'superseding width change',VDC_TEXT,FALSE);

   character_expansion_factor(fildes,1.5);
   character_width(fildes,0.030);
   text2d(fildes,0.05,0.40,'Character Width of 0.030',VDC_TEXT,FALSE);
   text2d(fildes,0.05,0.35,'superseding expansion change',VDC_TEXT,FALSE);
```

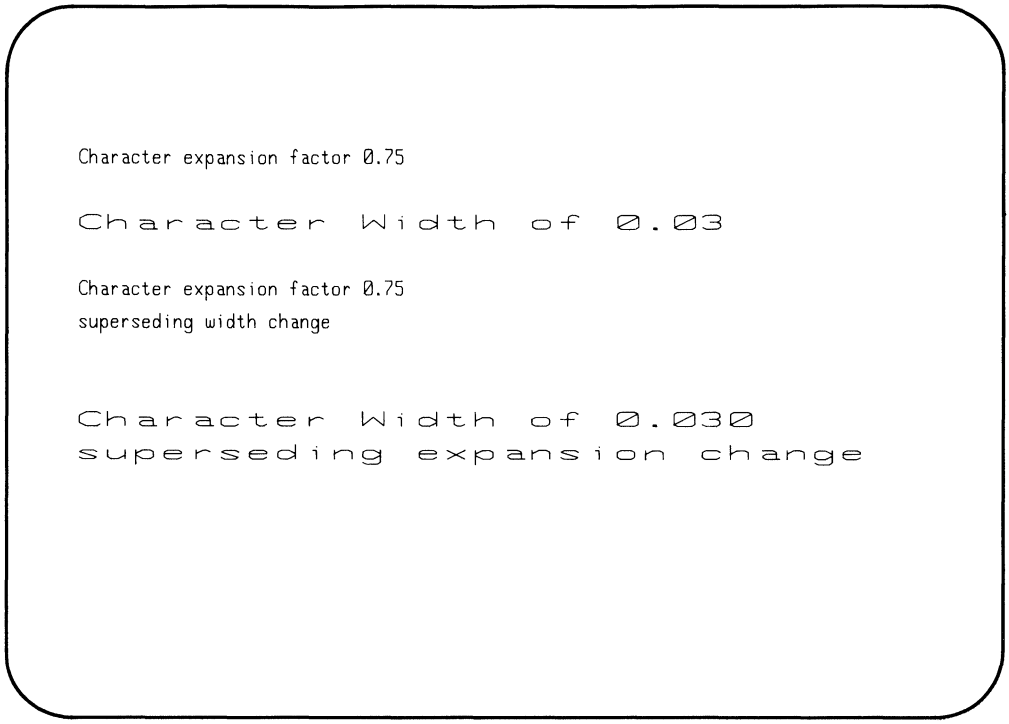


Figure 2-5. Character Width and Expansion Factor (text5c.c)

Intra Character Space

This procedure defines the space (distance) between character cells. This distance is a fraction of character height.

This example was created by the program `text6c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
character_height(fildes,0.04);

text2d(fildes,0.0,0.0,"Default space of 0.0",VDC_TEXT,FALSE);

intra_character_space(fildes,0.2);
text2d(fildes,0.0,0.2,"Character space of .2",VDC_TEXT,FALSE);

intra_character_space(fildes,0.4);
text2d(fildes,0.0,0.4,"Character space of .4",VDC_TEXT,FALSE);

intra_character_space(fildes,0.6);
text2d(fildes,0.0,0.6,"Character space of .6",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
      call character_height(fildes,0.04)

      call text2d(fildes,0.0,0.0,'Default space of 0.0'//NULL,VDC_TEXT,
     c FALSE)

      call intra_character_space(fildes,0.2)
      call text2d(fildes,0.0,0.2,'Character space of .2'//NULL,VDC_TEXT,FALSE)

      call intra_character_space(fildes,0.4)
      call text2d(fildes,0.0,0.4,'Character space of .4'//NULL,VDC_TEXT,FALSE)

      call intra_character_space(fildes,0.6)
      call text2d(fildes,0.0,0.6,'Character space of .6'//NULL,VDC_TEXT,FALSE)
```

Pascal Syntax

```
character_height(fildes,0.04);

text2d(fildes,0.0,0.0,'Default space of 0.0',VDC_TEXT,FALSE);

intra_character_space(fildes,0.2);
text2d(fildes,0.0,0.2,'Character space of .2',VDC_TEXT,FALSE);

intra_character_space(fildes,0.4);
text2d(fildes,0.0,0.4,'Character space of .4',VDC_TEXT,FALSE);

intra_character_space(fildes,0.6);
text2d(fildes,0.0,0.6,'Character space of .6',VDC_TEXT,FALSE);
```

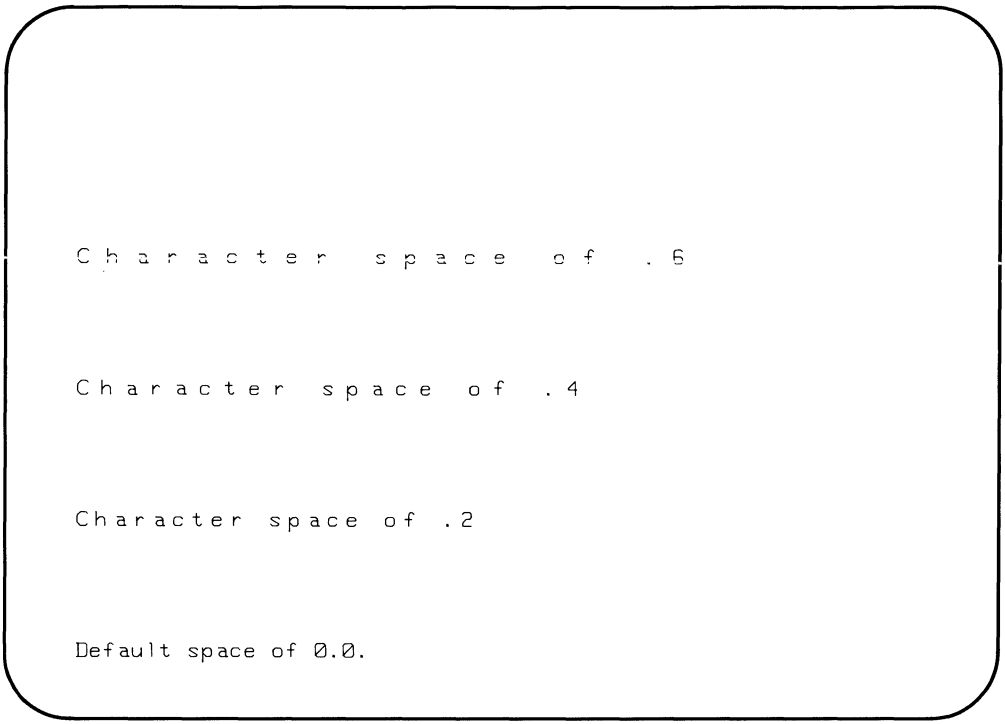


Figure 2-6. Character Space (text6c.c)

Character Slant

You can alter the appearance of your text by changing the slant of its characters. Larger characters are shown for better visual clarity.

This example was created by the program `text7c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
character_height(fildes,0.04);

character_slant(fildes,0.0);
text2d(fildes,0.0,0.15," Slant of 0, default",VDC_TEXT,FALSE);

character_slant(fildes,.268);
text2d(fildes,0.0,0.25," Slant of 0.268 or 15 degrees",VDC_TEXT,FALSE);

character_slant(fildes,0.577);
text2d(fildes,0.0,0.35," Slant of 0.577 or 30 degrees",VDC_TEXT,FALSE);

character_slant(fildes,1.0);
text2d(fildes,0.0,0.45," Slant of 1.00 or 45 degrees",VDC_TEXT,FALSE);

character_slant(fildes,1.73);
text2d(fildes,0.0,0.55," Slant of 1.73 or 60 degrees ",VDC_TEXT,FALSE);

character_slant(fildes,3.73);
text2(fildes,0.0,0.65," Slant of 3.73 or 75 degrees ",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
      call character_height(fildes,0.04)

      call character_slant(fildes,0.0)
      call text2d(fildes,0.0,0.15,' Slant of 0, default'//NULL,VDC_TEXT,FALSE)

      call character_slant(fildes,.268)
      call text2d(fildes,0.0,0.25,' Slant of 0.268 or 15 degrees'//NULL,
     c VDC_TEXT,FALSE)

      call character_slant(fildes,0.577)
      call text2d(fildes,0.0,0.35,' Slant of 0.577 or 30 degrees'//NULL,
     c VDC_TEXT,FALSE)

      call character_slant(fildes,1.0)
      call text2d(fildes,0.0,0.45,' Slant of 1.00 or 45 degrees'//NULL,
     c VDC_TEXT,FALSE)
```

```
  call character_slant(fildes,1.73)
  call text2d(fildes,0.0,0.55,' Slant of 1.73 or 60 degrees '//NULL,
c VDC_TEXT,FALSE)

  call character_slant(fildes,3.73)
  call text2d(fildes,0.0,0.65,' Slant of 3.73 or 75 degrees '//NULL,
c VDC_TEXT,FALSE)
```

Pascal Syntax

```
  character_height(fildes,0.04);

  character_slant(fildes,0.0);
  text2d(fildes,0.0,0.15,' Slant of 0, default',VDC_TEXT,FALSE);

  character_slant(fildes,.268);
  text2d(fildes,0.0,0.25,' Slant of 0.268 or 15 degrees',VDC_TEXT,FALSE);

  character_slant(fildes,0.577);
  text2d(fildes,0.0,0.35,' Slant of 0.577 or 30 degrees',VDC_TEXT,FALSE);

  character_slant(fildes,1.0);
  text2d(fildes,0.0,0.45,' Slant of 1.00 or 45 degrees',VDC_TEXT,FALSE);

  character_slant(fildes,1.73);
  text2d(fildes,0.0,0.55,' Slant of 1.73 or 60 degrees ',VDC_TEXT,FALSE);

  character_slant(fildes,3.73);
  text2d(fildes,0.0,0.65,' Slant of 3.73 or 75 degrees ',VDC_TEXT,FALSE);
```

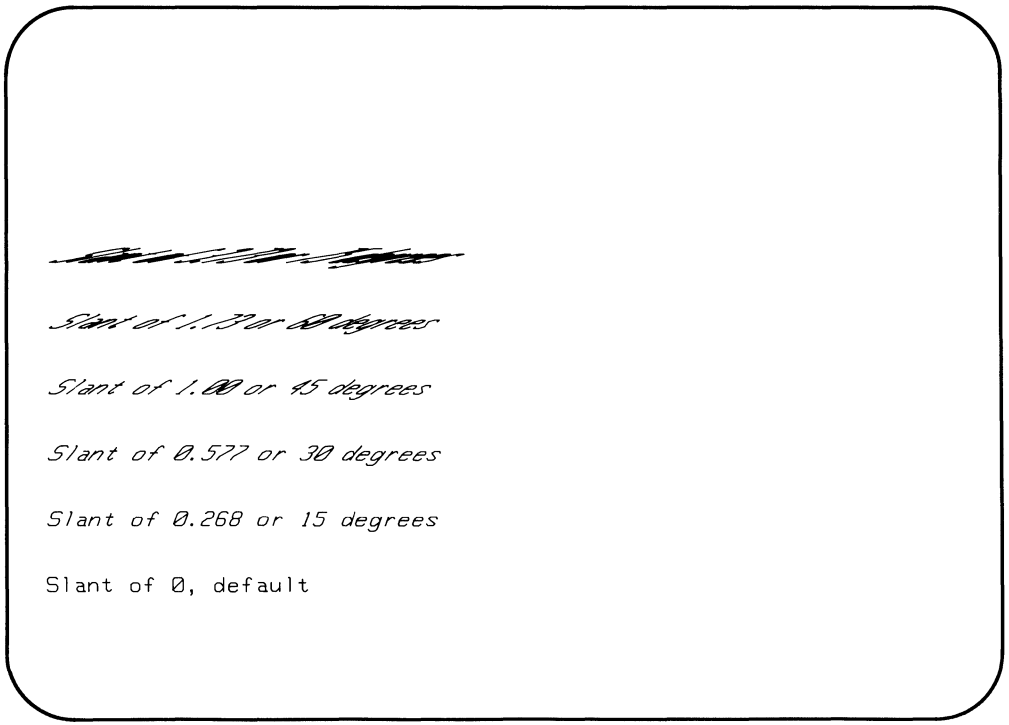


Figure 2-7. Character Slant (text7c.c)

Text Path

This procedure defines where the next character of a string is to be placed with respect to the currently drawn character. The second and following characters can be drawn in the following directions:

- PATH_RIGHT
- PATH_LEFT
- PATH_UP
- PATH_DOWN

The resulting text is aligned according to the assignment made by the `text_alignment` procedure. The default is TA_NORMAL_HORIZONTAL and TA_NORMAL_VERTICAL.

This example was created by the program `text8c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
character_height(fildes,0.04);

text2d(fildes,0.55,0.55,"Path right");

text_path(fildes,PATH_LEFT);
text2d(fildes,0.45,0.55,"Path left",VDC_TEXT,FALSE);

text_path(fildes,PATH_UP);
text2d(fildes,0.50,0.65,"Path up",VDC_TEXT,FALSE);

text_path(fildes,PATH_DOWN);
text2d(fildes,0.50,0.45,"Path down",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
call character_height(fildes,0.04)

call text2d(fildes,0.55,0.55,'Path right'//NULL,VDC_TEXT,FALSE)

call text_path(fildes,PATH_LEFT)
call text2d(fildes,0.45,0.55,'Path left'//NULL,VDC_TEXT,FALSE)

call text_path(fildes,PATH_UP)
call text2d(fildes,0.50,0.65,'Path up'//NULL,VDC_TEXT,FALSE)

call text_path(fildes,PATH_DOWN)
call text2d(fildes,0.50,0.45,'Path down'//NULL,VDC_TEXT,FALSE)
```

Pascal Syntax

```
character_height(fildes,0.04);

text2d(fildes,0.55,0.55,'Path right');

text_path(fildes,PATH_LEFT);
text2d(fildes,0.45,0.55,'Path left',VDC_TEXT,FALSE);

text_path(fildes,PATH_UP);
text2d(fildes,0.50,0.65,'Path up',VDC_TEXT,FALSE);

text_path(fildes,PATH_DOWN);
text2d(fildes,0.50,0.45,'Path down',VDC_TEXT,FALSE);
```

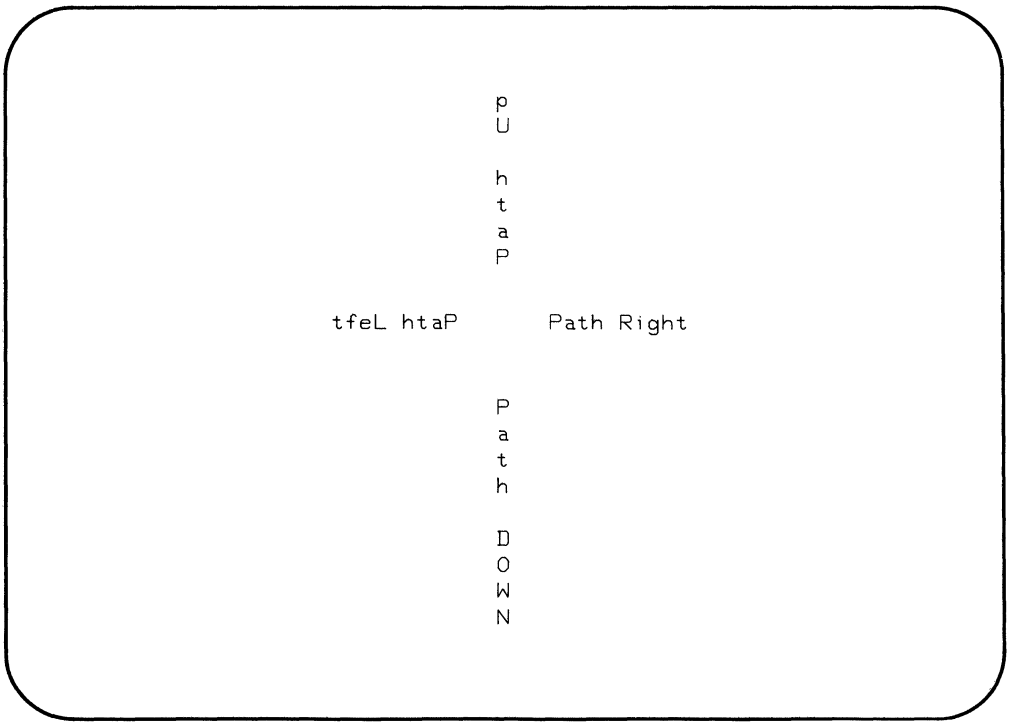


Figure 2-8. Text Path (text8c.c)

Text Line Path

The following program shows how the `text_line_path` procedure works. This procedure defines the position of text following a `line feed`. Possible positions are:

- PATH_RIGHT - Moves text one character width right.
- PATH_LEFT - Moves text one character width left.
- PATH_UP - Moves text one character height up.
- PATH_DOWN - Moves text one character height down.

This example was created by the program `text9c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
character_height(fildes,0.04);

text_line_path(fildes,PATH_RIGHT);
text2d(fildes,0.3,0.4,"Text with\n PATH_RIGHT",VDC_TEXT,FALSE);

text_line_path(fildes,PATH_LEFT);
text2d(fildes,0.3,0.5,"Text with\n PATH_LEFT",VDC_TEXT,FALSE);

text_line_path(fildes,PATH_UP);
text2d(fildes,0.3,0.6,"Text with\n PATH_UP",VDC_TEXT,FALSE);

text_line_path(fildes,PATH_DOWN);
text2d(fildes,0.3,0.7,"Text with\n PATH_DOWN",VDC_TEXT,FALSE);
```

Fortran77 Syntax

Not Supported on Series 500

```
call character_height(fildes,0.04);

call text_line_path(fildes,PATH_RIGHT);
call text2d(fildes,0.3,0.4,'Text with\n PATH_RIGHT'//NULL,VDC_TEXT,FALSE);

call text_line_path(fildes,PATH_LEFT);
call text2d(fildes,0.3,0.5,'Text with\n PATH_LEFT'//NULL,VDC_TEXT,FALSE);

call text_line_path(fildes,PATH_UP);
call text2d(fildes,0.3,0.6,'Text with\n PATH_UP'//NULL,VDC_TEXT,FALSE);

call text_line_path(fildes,PATH_DOWN);
call text2d(fildes,0.3,0.7,'Text with\n PATH_DOWN'//NULL,VDC_TEXT,FALSE);
```

Pascal Syntax

```
character_height(fildes,0.04);

text_line_path(fildes,PATH_RIGHT);
text2d(fildes,0.3,0.4,'Text with'#10' PATH_RIGHT',VDC_TEXT,FALSE);

text_line_path(fildes,PATH_LEFT);
text2d(fildes,0.3,0.5,'Text with'#10' PATH_LEFT',VDC_TEXT,FALSE);

text_line_path(fildes,PATH_UP);
text2d(fildes,0.3,0.6,'Text with'#10' PATH_UP',VDC_TEXT,FALSE);

text_line_path(fildes,PATH_DOWN);
text2d(fildes,0.3,0.7,'Text with'#10' PATH_DOWN',VDC_TEXT,FALSE);
```

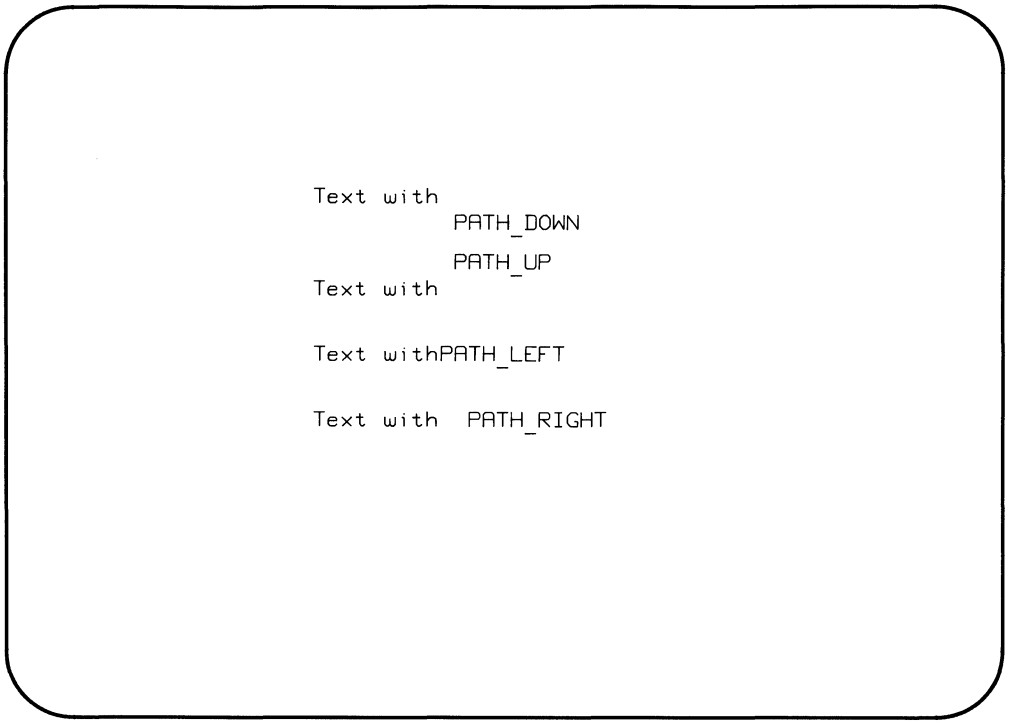


Figure 2-9. Text Line Path (text9c.c)

Text Orientation

Text orientation is specified by two vectors. The *up* vector provides the up component of all vertical strokes that draw the text. The *base* vector provides the the horizontal component of all horizontal strokes that are used to draw the text.

The procedure parameters are:

- fildes - identifies the device the procedure is accessing.
- up_x, up_y - the x and y components of the up vector.
- base_x, base_y - the x and y components of the base vector.

This example was created by the program `text10c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
character_height(fildes,0.04);

text2d(fildes,0.0,0.0,"Default text",VDC_TEXT,FALSE);

text_orientation(fildes,0.25,1.00,1.0,0.0);
text2d(fildes,0.0,0.1,"Up vector now 0.25,1.0.",VDC_TEXT,FALSE);

text_orientation(fildes,0.50,1.00,1.0,0.0);
text2d(fildes,0.0,0.2,"Up vector now 0.50,1.0.",VDC_TEXT,FALSE);

text_orientation(fildes,0.75,1.00,1.0,0.0);
text2(fildes,0.0,0.3,"Up vector now 0.75,1.0.",VDC_TEXT,FALSE);

text_orientation(fildes,0.00,1.00,1.00,0.25);
text2d(fildes,0.0,0.4,"Base vector now 1.00,0.25.",VDC_TEXT,FALSE);

text_orientation(fildes,0.00,1.00,1.00,0.5);
text2d(fildes,0.0,0.5,"Base vector now 1.00,0.50.",VDC_TEXT,FALSE);

text_orientation(fildes,0.00,1.00,1.00,0.75);
text2d(fildes,0.0,0.6,"Base vector now 1.00,0.75.",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
  call character_height(fildes,0.04)

  call text2d(fildes,0.0,0.0,'Default text is shown here.'//NULL,VDC_TEXT,
c FALSE)

  call text_orientation(fildes,0.25,1.00,1.0,0.0)
  call text2d(fildes,0.0,0.1,'Up vector now 0.25,1.0.'//NULL,VDC_TEXT,
c FALSE)

  call text_orientation(fildes,0.50,1.00,1.0,0.0)
  call text2d(fildes,0.0,0.2,'Up vector now 0.50,1.0.'//NULL,VDC_TEXT,
c FALSE)

  call text_orientation(fildes,0.75,1.00,1.0,0.0)
  call text2d(fildes,0.0,0.3,'Up vector now 0.75,1.0.'//NULL,VDC_TEXT,
c FALSE)

  call text_orientation(fildes,0.00,1.00,1.00,0.25)
  call text2d(fildes,0.0,0.4,'Base vector now 1.00,0.25.'//NULL,VDC_TEXT,
c FALSE)

  call text_orientation(fildes,0.00,1.00,1.00,0.5)
  call text2d(fildes,0.0,0.5,'Base vector now 1.00,0.50.'//NULL,VDC_TEXT,
c FALSE)

  call text_orientation(fildes,0.00,1.00,1.00,0.75)
  call text2d(fildes,0.0,0.6,'Base vector now 1.00,0.75.'//NULL,VDC_TEXT,
c FALSE)
```

Pascal Syntax

```
  character_height(fildes,0.04);

  text2d(fildes,0.0,0.0,'Default text is shown here.',VDC_TEXT,FALSE);

  text_orientation(fildes,0.25,1.00,1.0,0.0);
  text2d(fildes,0.0,0.1,'Up vector now 0.25,1.0.',VDC_TEXT,FALSE);

  text_orientation(fildes,0.50,1.00,1.0,0.0);
  text2d(fildes,0.0,0.2,'Up vector now 0.50,1.0.',VDC_TEXT,FALSE);

  text_orientation(fildes,0.75,1.00,1.0,0.0);
  text2d(fildes,0.0,0.3,'Up vector now 0.75,1.0.',VDC_TEXT,FALSE);
```

```
text_orientation(fildes,0.00,1.00,1.00,0.25);
text2d(fildes,0.0,0.4,'Base vector now 1.00,0.25.',VDC_TEXT,FALSE);

text_orientation(fildes,0.00,1.00,1.00,0.5);
text2d(fildes,0.0,0.5,'Base vector now 1.00,0.50.',VDC_TEXT,FALSE);

text_orientation(fildes,0.00,1.00,1.00,0.75);
text2d(fildes,0.0,0.6,'Base vector now 1.00,0.75.',VDC_TEXT,FALSE);
```

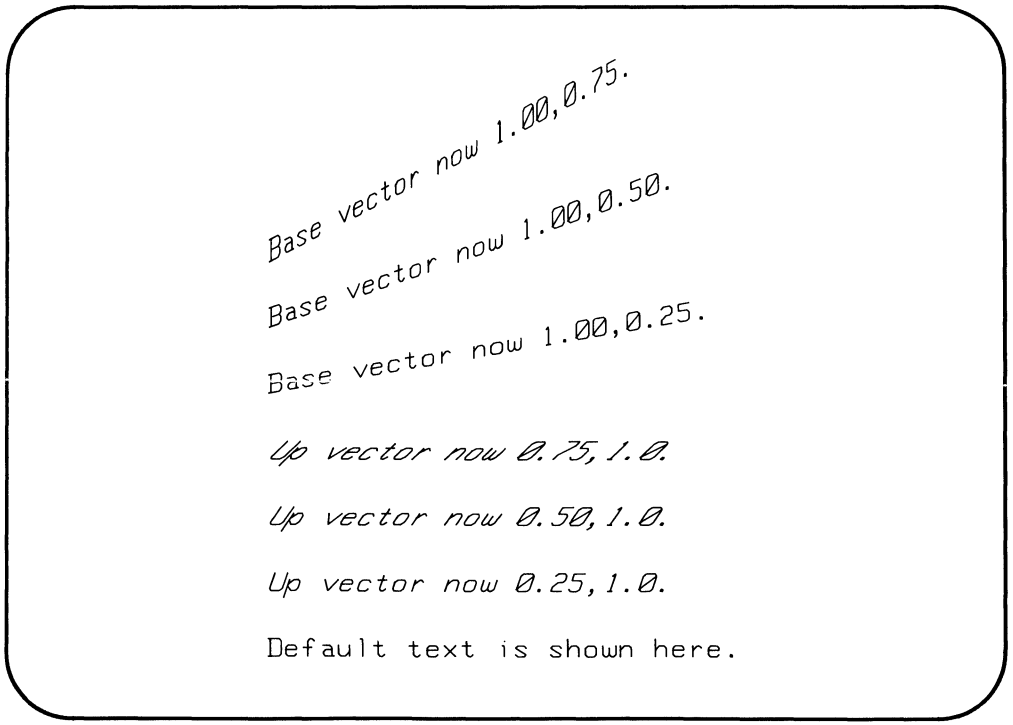


Figure 2-10. Text Orientation (text10c.c)

Text Line Space

This procedure defines the distance between lines when a `line feed` is executed in a string of text. The following example shows several lines with different line space distances.

This work was drawn using `text11c.c` found in `/dev/starbase/demos`.

```
character_height(fildes,0.04);

text2d(fildes,0.0,0.9,"Text with normal\ntext line space",VDC_TEXT,FALSE);

text_line_space(fildes,0.50);
text2d(fildes,0.0,0.7,"Text with 0.50\ntext line space",VDC_TEXT,FALSE);

text_line_space(fildes,1.50);
text2d(fildes,0.0,0.5,"Text with 1.50\ntext line space",VDC_TEXT,FALSE);

text_line_space(fildes,2.00);
text2d(fildes,0.0,0.3,"Text with 2.00\ntext line space",VDC_TEXT,FALSE);
```

Fortran77 Syntax

Not Supported on Series 500

```
  call character_height(fildes,0.04)

  call text2d(fildes,0.0,0.9,'Text with normal text line space',VDC_TEXT,
c FALSE)

  call text_line_space(fildes,0.50)
  call text2d(fildes,0.0,0.7,'Text with 0.50\ntext line space',VDC_TEXT,
c FALSE)

  call text_line_space(fildes,1.50)
  call text2d(fildes,0.0,0.5,'Text with 1.50\ntext line space',VDC_TEXT,
c FALSE)

  call text_line_space(fildes,2.00)
  call text2d(fildes,0.0,0.3,'Text with 2.00\ntext line space',VDC_TEXT,
c FALSE)
```

Pascal Syntax

```
character_height(fildes,0.04);

text2d(fildes,0.0,0.9,'Text with normal'#10'text line space',VDC_TEXT,FALSE);

text_line_space(fildes,0.50);
text2d(fildes,0.0,0.7,'Text with 0.50'#10'text line space',VDC_TEXT,FALSE);

text_line_space(fildes,1.50);
text2d(fildes,0.0,0.5,'Text with 1.50'#10'text line space',VDC_TEXT,FALSE);

text_line_space(fildes,2.00);
text2d(fildes,0.0,0.3,'Text with 2.00'#10'text line space',VDC_TEXT,FALSE);
```

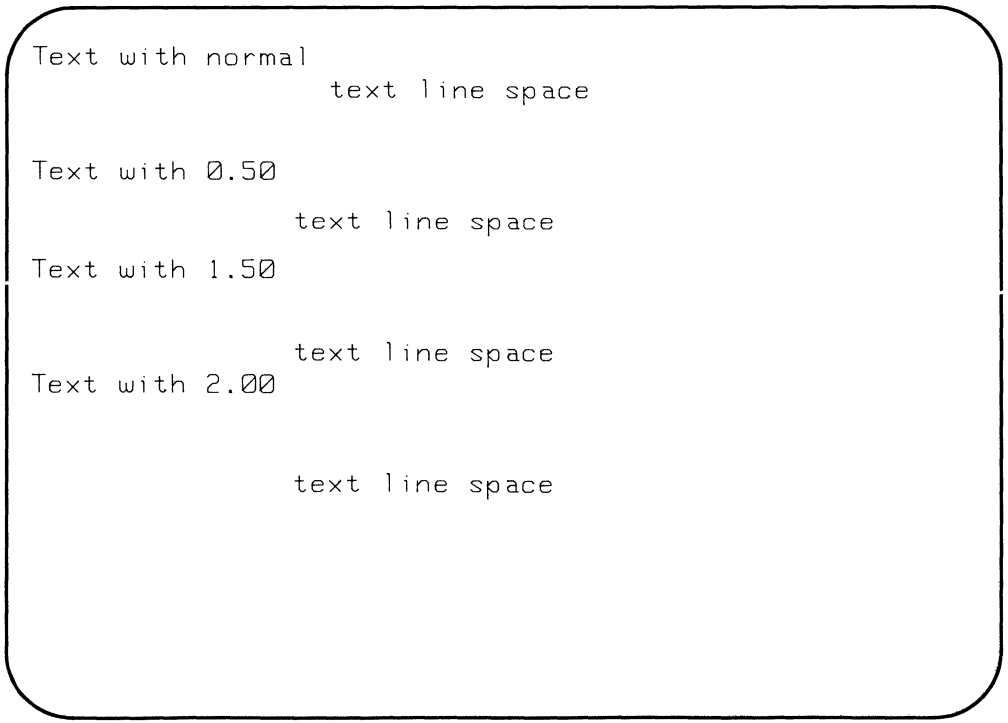


Figure 2-11. Text Line Space (text11c.c)

Text Alignment

This procedure controls text alignment and has the following parameters:

- fildes - as usual, this identifies the target device.
- h_select - this is the horizontal centering and can be one of the following:

1. TA_LEFT
2. TA_CENTER
3. TA_RIGHT
4. TA_CONTINUOUS_HORIZONTAL
5. TA_NORMAL_HORIZONTAL

- v_select - this is the vertical centering and can be one of the following:

1. TA_TOP
2. TA_CAP
3. TA_HALF
4. TA_BASE
5. TA_BOTTOM
6. TA_CONTINUOUS_VERTICAL
7. TA_NORMAL_VERTICAL

- horizontal - the fraction of the face of the text extent box which appears to the negative side of the up vector. (only used with TA_CONTINUOUS_HORIZONTAL)
- vertical - the fraction of the face of the text extent box which appears on the positive side of the base vector. (only used with TA_CONTINUOUS_VERTICAL)

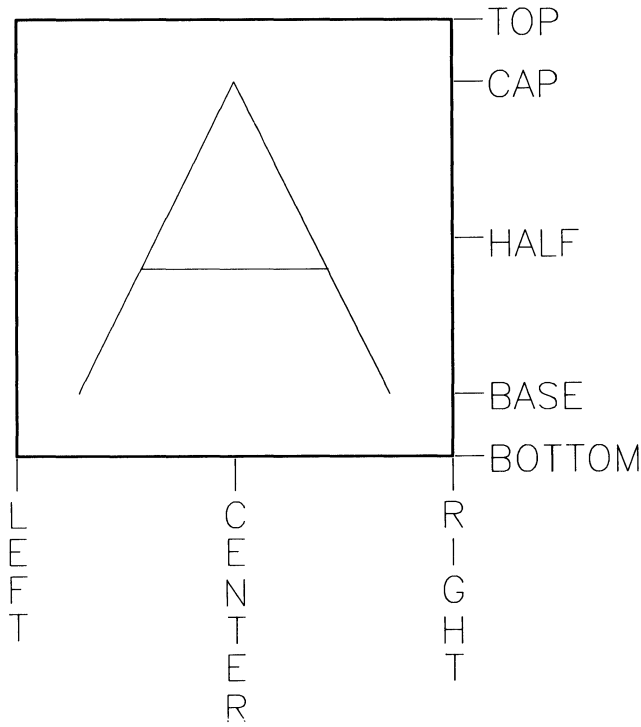


Figure 2-12. A Character

In the examples that follow, lines will be drawn to intersect the location specified for the text to be drawn. The text will consist of the string ABC using large characters.

Text alignment - TA_LEFT

With this text alignment, the lower left point of the first character in the text extent box is on the point specified in the `text2d` or `text3d` statement.

This example was created by the program `text11c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,"TA_LEFT",VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,"With TA_NORMAL_VERTICAL",VDC_TEXT,FALSE);

character_height(fildes,0.18);
text_alignment(fildes,TA_LEFT,TA_NORMAL_VERTICAL,0.0,0.0);
text2d(fildes,0.5,0.5,"ABC",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
call move2d(fildes,0.0,0.5)
call draw2d(fildes,1.0,0.5)
call move2d(fildes,0.5,0.0)
call draw2d(fildes,0.5,1.0)

call character_height(fildes,0.04)
call text2d(fildes,0.55,0.90,'TA_LEFT'//NULL,VDC_TEXT,FALSE)
call text2d(fildes,0.55,0.85,'With TA_NOMAL_VERTICAL'//NULL,VDC_TEXT,FALSE)

call character_height(fildes,0.18)
call text_alignment(fildes,TA_LEFT,TA_NORMAL_VERTICAL,0.0,0.0)
call text2d(fildes,0.5,0.5,'ABC'//NULL,VDC_TEXT,FALSE)
```

Pascal Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,'TA_LEFT',VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,'With TA_NORMAL_VERTICAL',VDC_TEXT,FALSE);

character_height(fildes,0.2);
text_alignment(fildes,TA_LEFT,TA_NORMAL_VERTICAL,0.0,0.0);
text2d(fildes,0.5,0.5,'ABC',VDC_TEXT,FALSE);
```

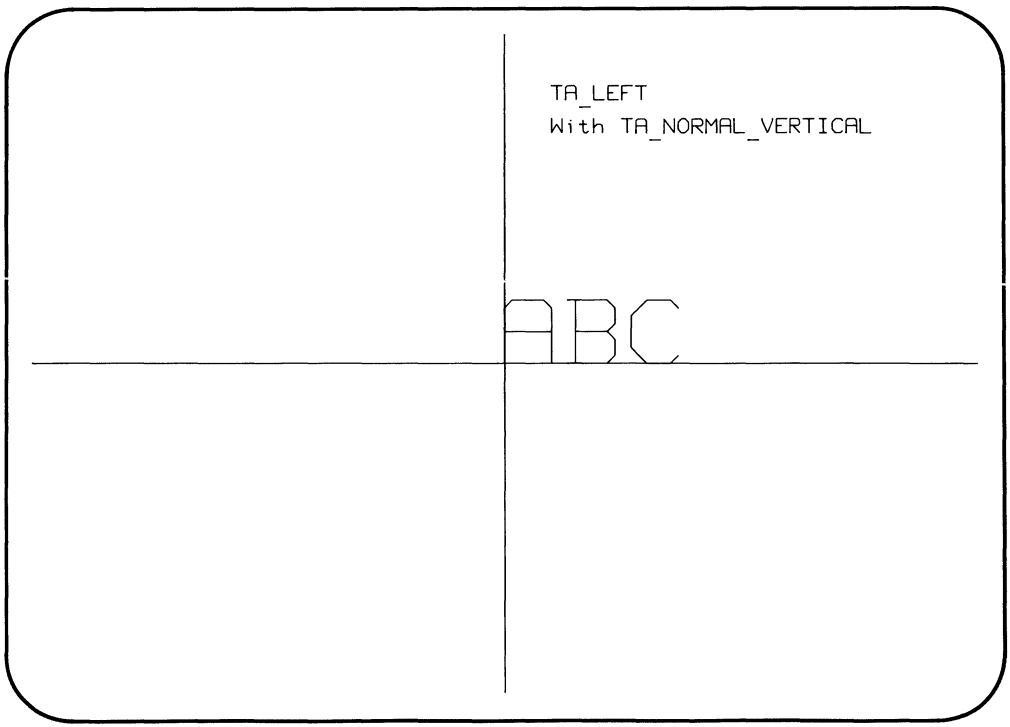


Figure 2-13. Text Alignment - TA_LEFT (text12c.c)

Text alignment - TA_CENTER

The text extent box is centered at the point specified in the `text2d` or `text3d` procedure call.

This example was created by the program `text13c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,"TA_CENTER",VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,"With TA_NORMAL_VERTICAL",VDC_TEXT,FALSE);

character_height(fildes,0.18);
text_alignment(fildes,TA_CENTER,TA_NORMAL_VERTICAL,0.0,0.0);
text2d(fildes,0.5,0.5,"ABC",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
call move2d(fildes,0.0,0.5)
call draw2d(fildes,1.0,0.5)
call move2d(fildes,0.5,0.0)
call draw2d(fildes,0.5,1.0)

call character_height(fildes,0.04)
call text2d(fildes,0.55,0.90,'TA_CENTER'//NULL,VDC_TEXT,FALSE)
call text2d(fildes,0.55,0.85,'With TA_NOMAL_VERTICAL'//NULL,VDC_TEXT,FALSE)

call character_height(fildes,0.18)
call text_alignment(fildes,TA_CENTER,TA_NORMAL_VERTICAL,0.0,0.0)
call text2d(fildes,0.5,0.5,'ABC'//NULL,VDC_TEXT,FALSE)
```

Pascal Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,'TA_CENTER',VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,'With TA_NORMAL_VERTICAL',VDC_TEXT,FALSE);

character_height(fildes,0.2);
text_alignment(fildes,TA_CENTER,TA_NORMAL_VERTICAL,0.0,0.0);
text2d(fildes,0.5,0.5,'ABC',VDC_TEXT,FALSE);
```

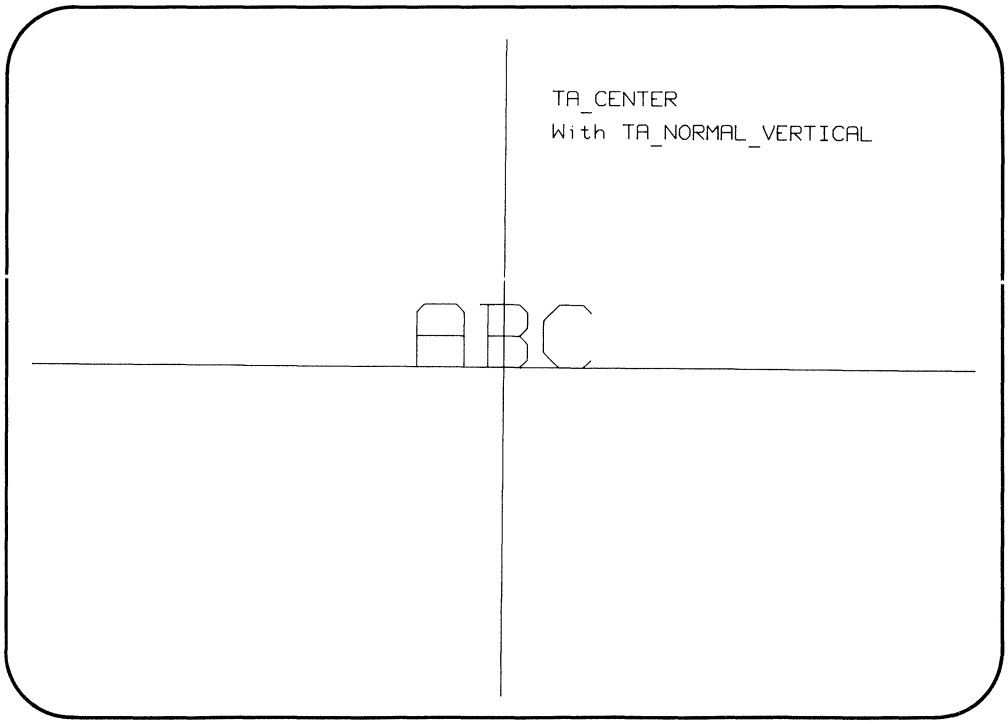


Figure 2-14. Text Alignment - TA_CENTER (text13c.c)

Text alignment - TA_RIGHT

The intersection of the character base vector and the right edge of the most right character is places at the the point specified in the `text2d` or `text3d` procedure call.

This example was created by the program `text14c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,"TA_RIGHT",VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,"With TA_NORMAL_VERTICAL",VDC_TEXT,FALSE);

character_height(fildes,0.18);
text_alignment(fildes,TA_RIGHT,TA_NORMAL_VERTICAL,0.0,0.0);
text2d(fildes,0.5,0.5,"ABC",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
call move2d(fildes,0.0,0.5)
call draw2d(fildes,1.0,0.5)
call move2d(fildes,0.5,0.0)
call draw2d(fildes,0.5,1.0)

call character_height(fildes,0.04)
call text2d(fildes,0.55,0.90,'TA_RIGHT'//NULL,VDC_TEXT,FALSE)
call text2d(fildes,0.55,0.85,'With TA_NOMAL_VERTICAL'//NULL,VDC_TEXT,FALSE)

call character_height(fildes,0.18)
call text_alignment(fildes,TA_RIGHT,TA_NORMAL_VERTICAL,0.0,0.0)
call text2d(fildes,0.5,0.5,'ABC'//NULL,VDC_TEXT,FALSE)
```

Pascal Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,'TA_RIGHT',VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,'With TA_NORMAL_VERTICAL',VDC_TEXT,FALSE);

character_height(fildes,0.2);
text_alignment(fildes,TA_RIGHT,TA_NORMAL_VERTICAL,0.0,0.0);
text2d(fildes,0.5,0.5,'ABC',VDC_TEXT,FALSE);
```

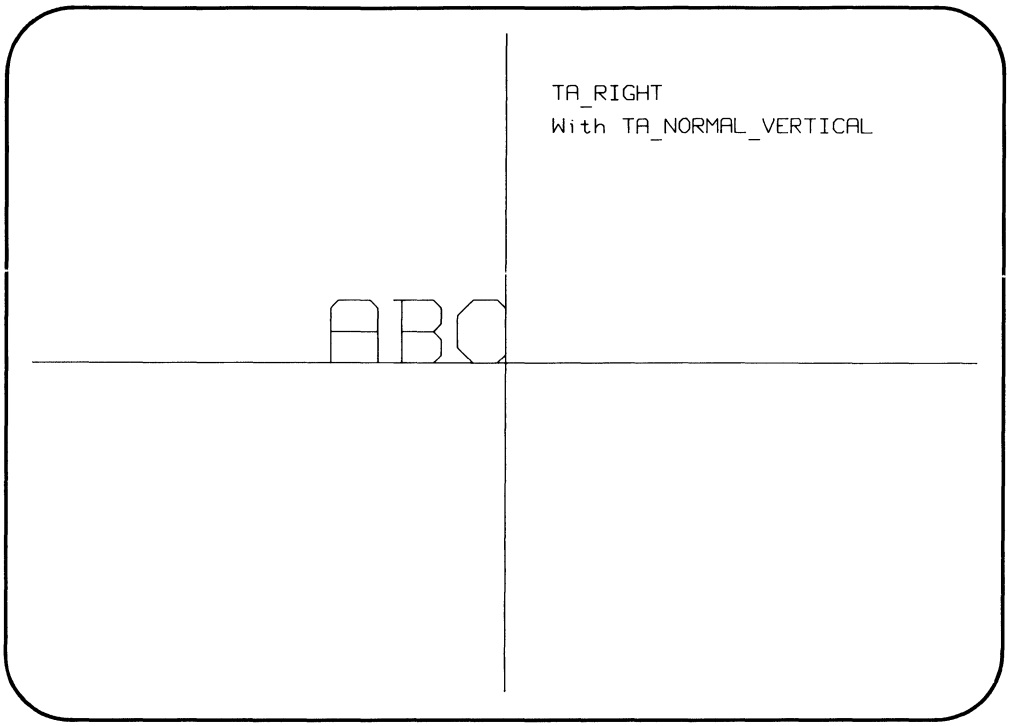


Figure 2-15. Text Alignment - TA_RIGHT (text14c.c)

Text alignment - TA_CONTINUOUS_HORIZONTAL #1

With this procedure, the parameter `horizontal` is used to allow the specification of what percentage (as a fraction) of the the text extent box is to the **positive side** of the point specified in the `text2d` or `text3d` procedure. This percentage can be a real number greater than 1.00. For the first example, a percentage fraction of 0.20 is used.

This example was created by the program `text15c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,"TA_CONTINUOUS_HORIZONTAL",VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,"With horizontal = 0.20",VDC_TEXT,FALSE);
text2d(fildes,0.55,0.80,"and TA_NORMAL_VERTICAL",VDC_TEXT,FALSE);

character_height(fildes,0.18);
text_alignment(fildes,TA_CONTINUOUS_HORIZONTAL,TA_NORMAL_VERTICAL,0.2,0.0);
text2d(fildes,0.5,0.5,"ABC",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
call move2d(fildes,0.0,0.5)
call draw2d(fildes,1.0,0.5)
call move2d(fildes,0.5,0.0)
call draw2d(fildes,0.5,1.0)

call character_height(fildes,0.04)
call text2d(fildes,0.55,0.90,'TA_CONTINUOUS_HORIZONTAL'//NULL,0,0)
call text2d(fildes,0.55,0.85,'With horizontal = 0.20'//NULL,0,0)
call text2d(fildes,0.55,0.80,'and TA_NOMAL_VERTICAL'//NULL,0,0)

call character_height(fildes,0.18)
call text_alignment(fildes,TA_CONTINUOUS_HORIZONTAL,TA_NORMAL_VERTICAL,0.2,0.0)
call text2d(fildes,0.5,0.5,'ABC'//NULL,VDC_TEXT,FALSE)
```

Pascal Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,'TA_CONTINUOUS_HORIZONTAL',VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,'With horizontal = 0.20',VDC_TEXT,FALSE);
text2d(fildes,0.55,0.80,'and TA_NORMAL_VERTICAL',VDC_TEXT,FALSE);

character_height(fildes,0.2);
text_alignment(fildes,TA_CONTINUOUS_HORIZONTAL,TA_NORMAL_VERTICAL,0.2,0.0);
text2d(fildes,0.5,0.5,'ABC',VDC_TEXT,FALSE);
```

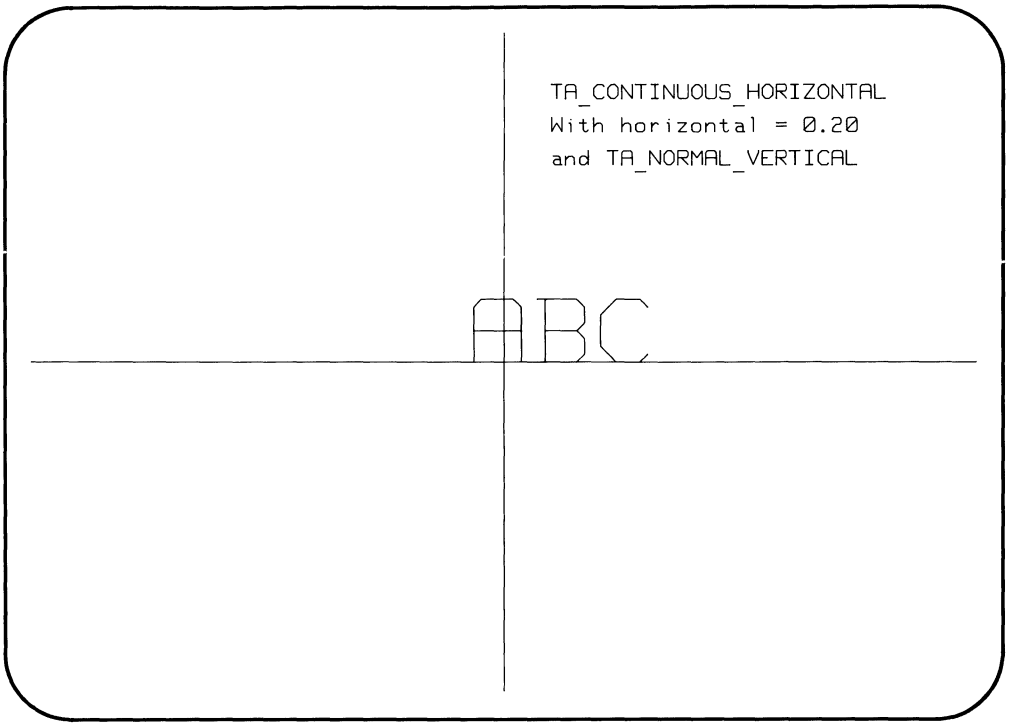


Figure 2-16. Text Alignment - TA_CONTINUOUT_HORIZONTAL Example #1

Text alignment - TA_CONTINUOUS_HORIZONTAL #2

For this example, a `horizontal` value of 0.7 was used.

This example was created by the program `text16c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,"TA_CONTINUOUS_HORIZONTAL",VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,"With horizontal = 0.7",VDC_TEXT,FALSE);
text2d(fildes,0.55,0.80,"and TA_NORMAL_VERTICAL",VDC_TEXT,FALSE);

character_height(fildes,0.2);
text_alignment(fildes,TA_CONTINUOUS_HORIZONTAL,TA_NORMAL_VERTICAL,0.7,0.0);
text2d(fildes,0.5,0.5,"ABC",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
call move2d(fildes,0.0,0.5)
call draw2d(fildes,1.0,0.5)
call move2d(fildes,0.5,0.0)
call draw2d(fildes,0.5,1.0)

call character_height(fildes,0.04)
call text2d(fildes,0.55,0.90,'TA_CONTINUOUS_HORIZONTAL'//NULL,0,0)
call text2d(fildes,0.55,0.85,'With horizontal = 0.7//NULL,0,0)
call text2d(fildes,0.55,0.80,'and TA_NORMAL_VERTICAL//NULL,0,0)

call character_height(fildes,0.2)
call text_alignment(fildes,TA_CONTINUOUS_HORIZONTAL,TA_NORMAL_VERTICAL,0.7,0.0);
call text2d(fildes,0.5,0.5,'ABC'//NULL,VDC_TEXT,FALSE)
```

Pascal Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,'TA_CONTINUOUS_HORIZONTAL',VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,'with horizontal = 0.7',VDC_TEXT,FALSE);
text2d(fildes,0.55,0.80,'and TA_NORMAL_VERTICAL',VDC_TEXT,FALSE);

character_height(fildes,0.2);
text_alignment(fildes,TA_CONTINUOUS_HORIZONTAL,TA_NORMAL_VERTICAL,0.7,0.0);
text2d(fildes,0.5,0.5,'ABC',VDC_TEXT,FALSE);
```

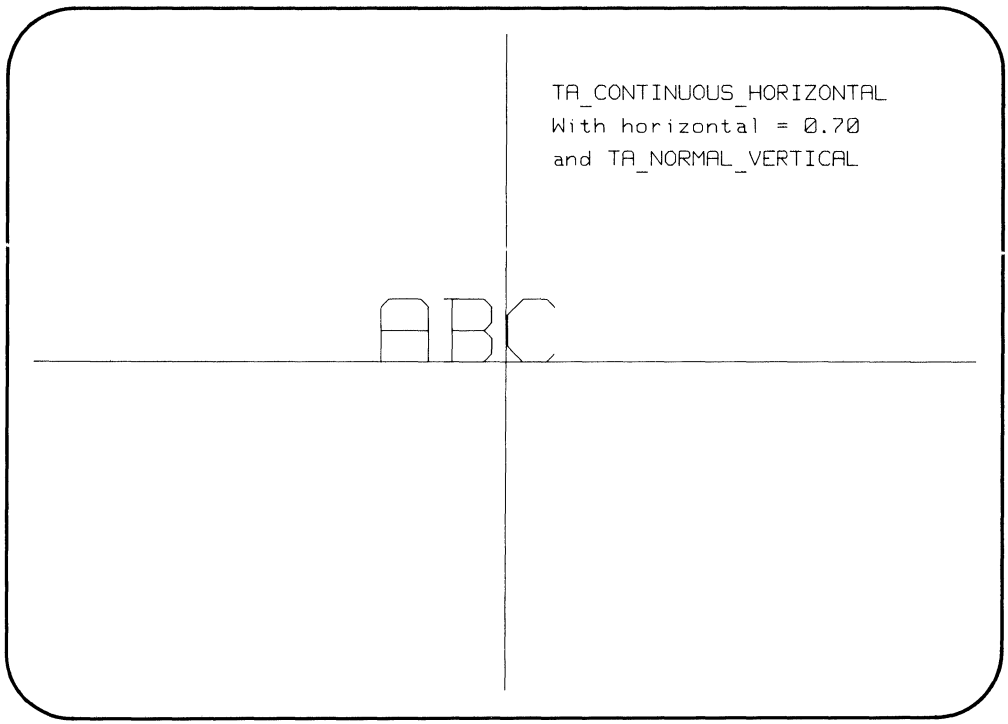


Figure 2-17. Text Alignment - TA_CONTINUOUS_HORIZONTAL Example #2

Text alignment - TA_NORMAL_HORIZONTAL

This example was created by the program `text17c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,"TA_NORMAL_HORIZONTAL",VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,"With TA_NORMAL_VERTICAL",VDC_TEXT,FALSE);

character_height(fildes,0.2);
text_alignment(fildes,TA_NORMAL_HORIZONTAL,TA_NORMAL_VERTICAL,0.0,0.0);
text2d(fildes,0.5,0.5,"ABC",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
      call move2d(fildes,0.0,0.5)
      call draw2d(fildes,1.0,0.5)
      call move2d(fildes,0.5,0.0)
      call draw2d(fildes,0.5,1.0)

      call character_height(fildes,0.04)
      call text2d(fildes,0.55,0.90,'TA_NORMAL_HORIZONTAL'//NULL,VDC_TEXT,
     c FALSE)
      call text2d(fildes,0.55,0.85,'With TA_NORMAL_VERTICAL,VDC_TEXT,FALSE)

      call character_height(fildes,0.2)
      call text_alignment(fildes,TA_NORMAL_HORIZONTAL,TA_NORMAL_VERTICAL,0.0,0.0);
      call text2d(fildes,0.5,0.5,'ABC'//NULL,VDC_TEXT,FALSE)
```

Pascal Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,'TA_NORMAL_HORIZONTAL',VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,'with TA_NORMAL_VERTICAL',VDC_TEXT,FALSE);

character_height(fildes,0.2);
text_alignment(fildes,TA_NORMAL_HORIZONTAL,TA_NORMAL_VERTICAL,0.0,0.0);
text2d(fildes,0.5,0.5,'ABC',VDC_TEXT,FALSE);
```

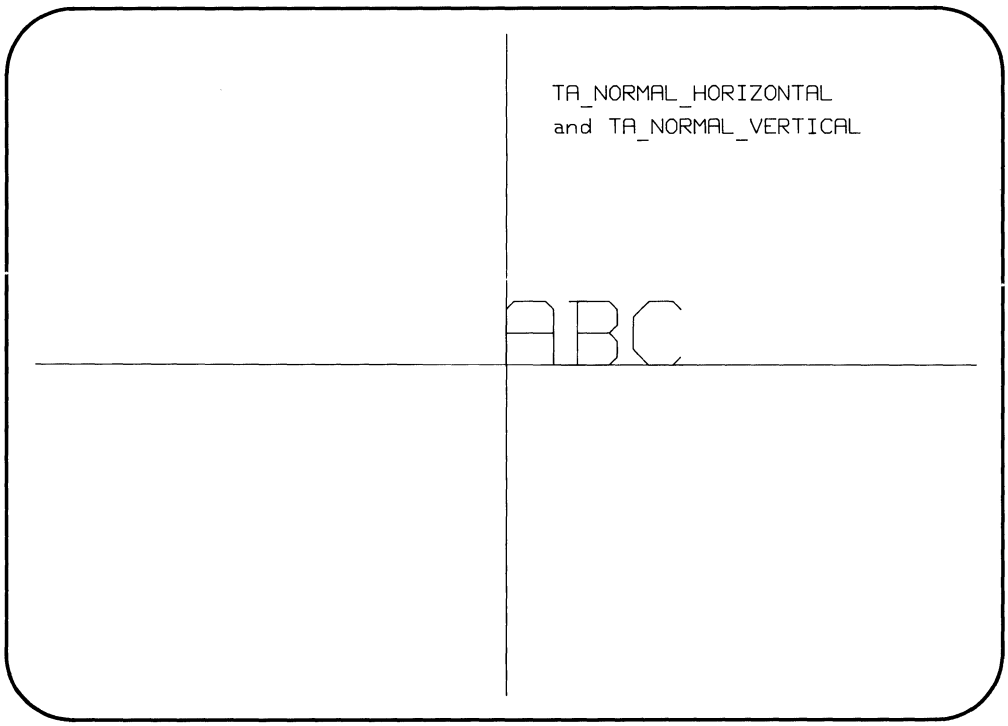


Figure 2-18. Text Alignment - TA_NORMAL_HORIZONTAL (text17c.c)

Text alignment - TA_TOP

Depending upon how the character is defined in its cell, there is usually space between the top of the character (cap) and the top of the character cell (top). With this procedure, the intersection of the top of the text extent box and the right edge of the first character in the string is is aligned with the point specified in the `text2d` or `text3d` procedure call.

This example was created by the program `text18c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,"TA_TOP",VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,"with TA_NORMAL_HORIZONTAL",VDC_TEXT,FALSE);

character_height(fildes,0.2);
text_alignment(fildes,TA_NORMAL_HORIZONTAL,TA_TOP,0.0,0.0);
text2d(fildes,0.5,0.5,"ABC",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
call move2d(fildes,0.0,0.5)
call draw2d(fildes,1.0,0.5)
call move2d(fildes,0.5,0.0)
call draw2d(fildes,0.5,1.0)

call character_height(fildes,0.04)
call text2d(fildes,0.55,0.90,'TA_TOP'//NULL,0,0)
call text2d(fildes,0.55,0.85,'with TA_NORMAL_HORIZONTAL'//NULL,0,0)

call character_height(fildes,0.18)
call text_alignment(fildes,TA_NORMAL_HORIZONTAL,TA_TOP,0.0,0.0)
call text2d(fildes,0.5,0.5,'ABC'//NULL,0,0)
```

Pascal Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,'TA_TOP',VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,'with TA_NORMAL_HORIZONTAL',VDC_TEXT,FALSE);

character_height(fildes,0.2);
text_alignment(fildes,TA_NORMAL_HORIZONTAL,TA_TOP,0.0,0.0);
text2d(fildes,0.5,0.5,'ABC',VDC_TEXT,FALSE);
```

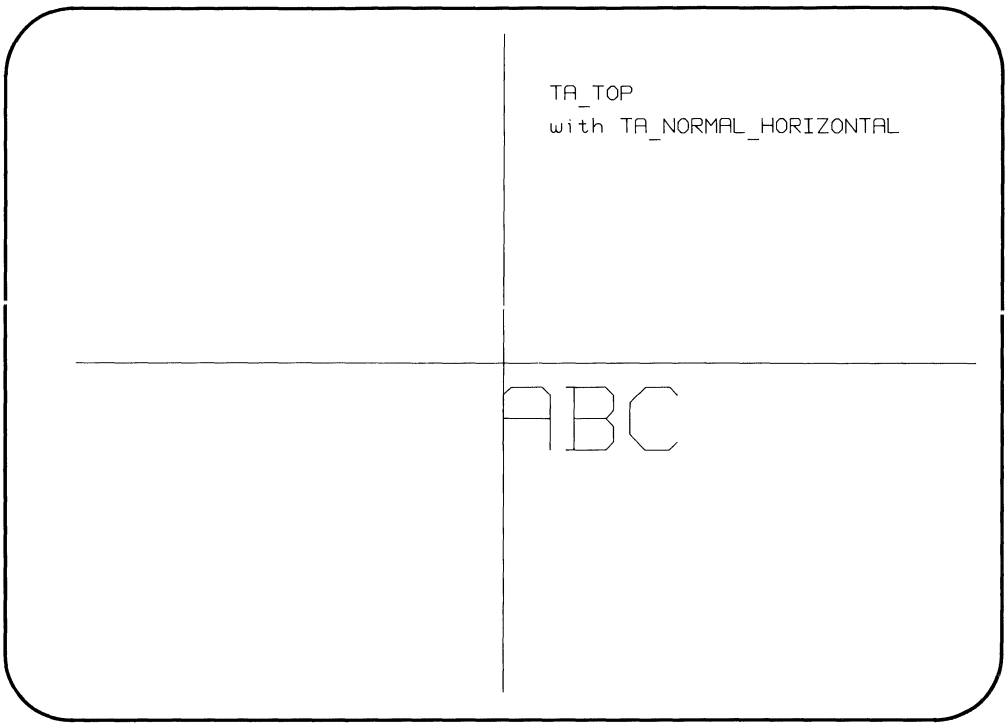


Figure 2-19. Text Alignment - TA_TOP (text18c.c)

Text alignment - TA_CAP

This alignment is the same at `TA_TOP` except that the `CAP` (top of the character) is used instead of the top of the character cell.

This example was created by the program `text19c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,"TA_CAP",VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,"with TA_NORMAL_HORIZONTAL",VDC_TEXT,FALSE);

character_height(fildes,0.18);
text_alignment(fildes,TA_NORMAL_HORIZONTAL,TA_CAP,0.0,0.0);
text2d(fildes,0.5,0.5,"ABC",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
call move2d(fildes,0.0,0.5)
call draw2d(fildes,1.0,0.5)
call move2d(fildes,0.5,0.0)
call draw2d(fildes,0.5,1.0)

call character_height(fildes,0.04)
call text2d(fildes,0.55,0.90,'TA_CAP'//NULL,VDC_TEXT,FALSE)
call text2d(fildes,0.55,0.85,'with TA_NORMAL_HORIZONTAL',VDC_TEXT,FALSE)

call character_height(fildes,0.2)
call text_alignment(fildes,TA_NORMAL_HORIZONTAL,TA_CAP,0.0,0.0)
call text2d(fildes,0.5,0.5,'ABC'//NULL,VDC_TEXT,FALSE)
```

Pascal Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,'TA_CAP',VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,'with TA_NORMAL_HORIZONTAL',VDC_TEXT,FALSE);

character_height(fildes,0.2);
text_alignment(fildes,TA_NORMAL_HORIZONTAL,TA_CAP,0.0,0.0);
text2d(fildes,0.5,0.5,'ABC',VDC_TEXT,FALSE);
```

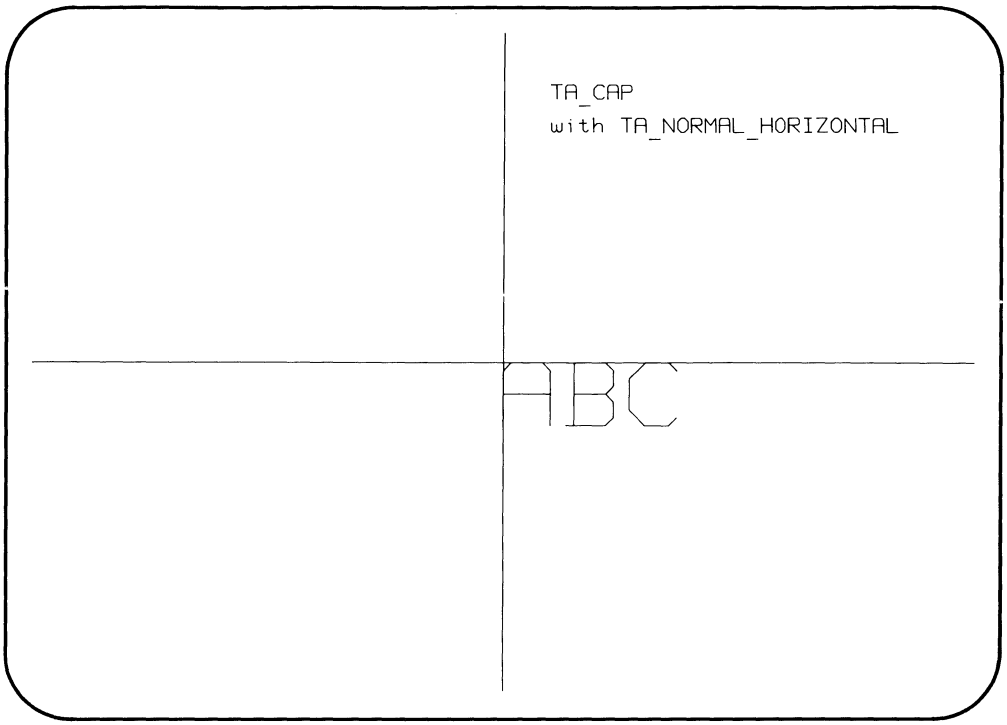


Figure 2-20. Text Alignment - TA_CAP (text19c.c)

Text alignment - TA_HALF

The position `TA_HALF` is located half way between the character cell cap and base. This location may or may not be the center of the drawing character. In this example, the two locations are not the same.

This example was created by the program `text20c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,"TA_HALF",VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,"with TA_NORMAL_HORIZONTAL",VDC_TEXT,FALSE);

character_height(fildes,0.18);
text_alignment(fildes,TA_NORMAL_HORIZONTAL,TA_HALF,0.0,0.0);
text2d(fildes,0.5,0.5,"ABC",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
call move2d(fildes,0.0,0.5)
call draw2d(fildes,1.0,0.5)
call move2d(fildes,0.5,0.0)
call draw2d(fildes,0.5,1.0)

call character_height(fildes,0.04)
call text2d(fildes,0.55,0.90,'TA_HALF'//NULL,VDC_TEXT,FALSE)
call text2d(fildes,0.55,0.85,'with TA_NORMAL_HORIZONTAL',VDC_TEXT,FALSE);

call character_height(fildes,0.18)
call text_alignment(fildes,TA_NORMAL_HORIZONTAL,TA_HALF,0.0,0.0)
call text2d(fildes,0.5,0.5,'ABC'//NULL,VDC_TEXT,FALSE)
```

Pascal Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,'TA_HALF',VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,'with TA_NORMAL_HORIZONTAL',VDC_TEXT,FALSE);

character_height(fildes,0.18);
text_alignment(fildes,TA_NORMAL_HORIZONTAL,TA_HALF,0.0,0.0);
text2d(fildes,0.5,0.5,'ABC',VDC_TEXT,FALSE);
```

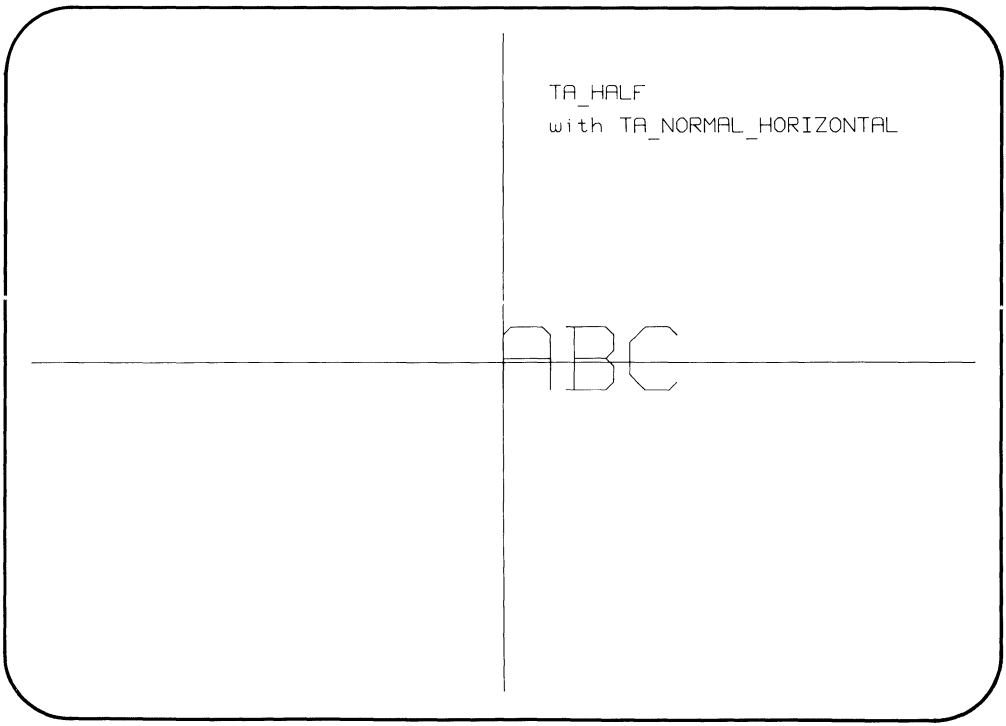


Figure 2-21. Text Alignment - TA_CAP

Text alignment - TA_BASE

This example was created by the program `text21c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,"TA_BASE",VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,"with TA_NORMAL_HORIZONTAL",VDC_TEXT,FALSE);

character_height(fildes,0.18);
text_alignment(fildes,TA_NORMAL_HORIZONTAL,TA_BASE,0.0,0.0);
text2d(fildes,0.5,0.5,"ABC",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
call move2d(fildes,0.0,0.5)
call draw2d(fildes,1.0,0.5)
call move2d(fildes,0.5,0.0)
call draw2d(fildes,0.5,1.0)

call character_height(fildes,0.04)
call text2d(fildes,0.55,0.90,'TA_BASE'//NULL,VDC_TEXT,FALSE)
call text2d(fildes,0.55,0.85,'with TA_NORMAL_HORIZONTAL',VDC_TEXT,FALSE);

call character_height(fildes,0.18)
call text_alignment(fildes,TA_NORMAL_HORIZONTAL,TA_BASE,0.0,0.0)
call text2d(fildes,0.5,0.5,'ABC'//NULL,VDC_TEXT,FALSE)
```

Pascal Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,'TA_BASE',VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,'with TA_NORMAL_HORIZONTAL',VDC_TEXT,FALSE);

character_height(fildes,0.18);
text_alignment(fildes,TA_NORMAL_HORIZONTAL,TA_BASE,0.0,0.0);
text2d(fildes,0.5,0.5,'ABC',VDC_TEXT,FALSE);
```

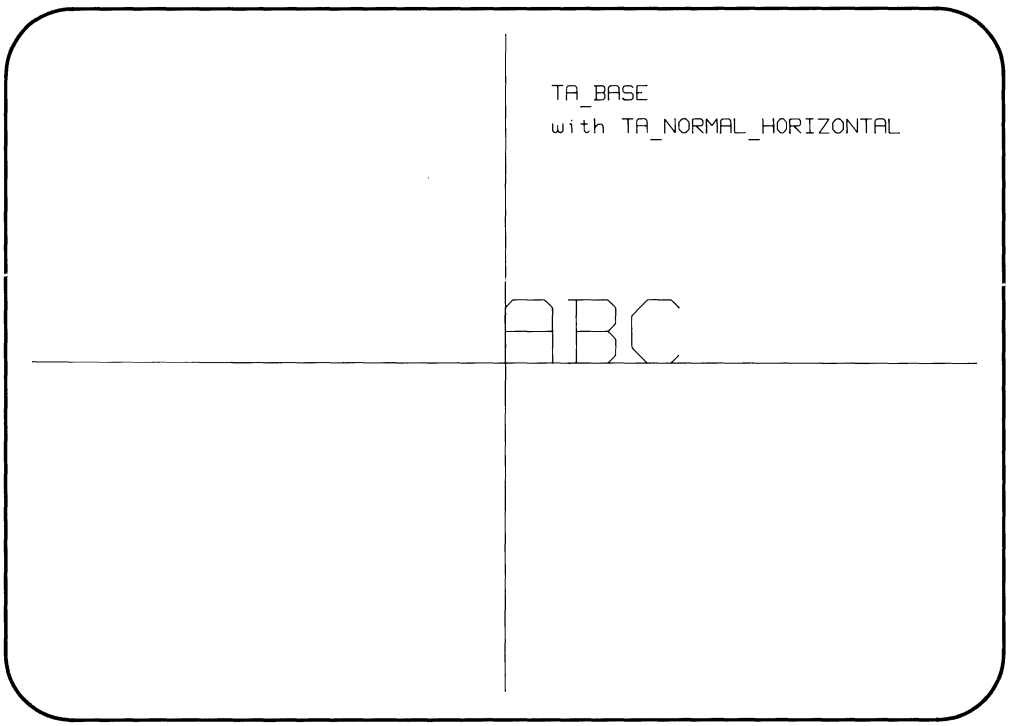


Figure 2-22. Text Alignment - TA_BASE (text21c.c)

Text alignment - TA_BOTTOM

Depending upon how the character is defined in its cell, there is usually space between the bottom of the character (base) and the bottom of the character cell (bottom). With this procedure, the intersection of the bottom of the text extent box and the right edge of the first character in the string is is aligned with the point specified in the `text2d` or `text3d` procedure call.

This example was created by the program `text22c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,"TA_BOTTOM",VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,"with TA_NORMAL_HORIZONTAL",VDC_TEXT,FALSE);

character_height(fildes,0.2);
text_alignment(fildes,TA_LEFT,TA_BOTTOM,0.0,0.0);
text2d(fildes,0.5,0.5,"ABC",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
call move2d(fildes,0.0,0.5)
call draw2d(fildes,1.0,0.5)
call move2d(fildes,0.5,0.0)
call draw2d(fildes,0.5,1.0)

call character_height(fildes,0.04)
call text2d(fildes,0.55,0.90,'TA_BOTTOM'//NULL,VDC_TEXT,FALSE)
call text2d(fildes,0.55,0.85,'with TA_NORMAL_HORIZONTAL'//NULL,VDC_TEXT,FALSE);

call character_height(fildes,0.2)
call text_alignment(fildes,TA_LEFT,TA_BOTTOM,0.0,0.0)
call text2d(fildes,0.5,0.5,'ABC'//NULL,VDC_TEXT,FALSE)
```

Pascal Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,'TA_BOTTOM',VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,'with TA_NORMAL_HORIZONTAL',VDC_TEXT,FALSE);

character_height(fildes,0.2);
text_alignment(fildes,TA_LEFT,TA_BOTTOM,0.0,0.0);
text2d(fildes,0.5,0.5,'ABC',VDC_TEXT,FALSE);
```

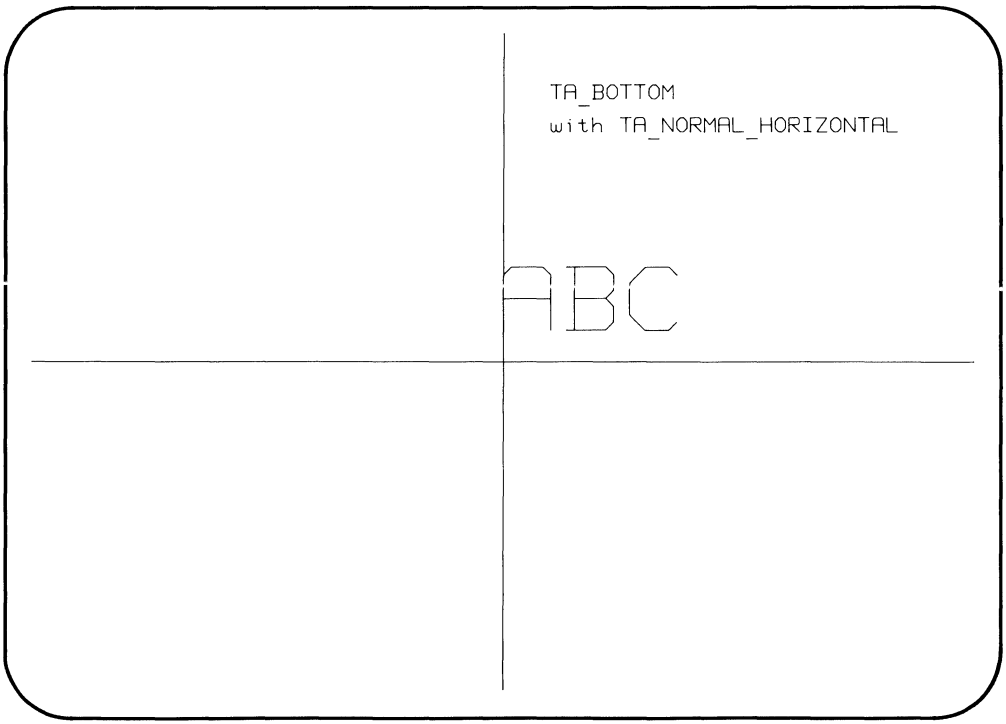


Figure 2-23. Text Alignment - TA_BOTTOM (text22c.c)

Text alignment - TA_CONTINUOUS_VERTICAL Example #1

With this procedure, the parameter `vertical` is used to allow the specification of what percentage (as a fraction) of the the text extent box is to **below** the point specified in the `text2d` or `text3d` procedure. This percentage can be a real number greater than 1.00. For the first example, a percentage fraction of 0.35 is used.

This example was created by the program `text23c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,"TA_CONTINUOUS_VERTICAL",VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,"With vertical = 0.35",VDC_TEXT,FALSE);
text2d(fildes,0.55,0.80,"and TA_CONTINUOUS_HORINTOZAL",VDC_TEXT,FALSE);

character_height(fildes,0.2);
text_alignment(fildes,TA_LEFT,TA_CONTINUOUS_VERTICAL,0.0,0.35);
text2d(fildes,0.5,0.5,"ABC",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
      call move2d(fildes,0.0,0.5)
      call draw2d(fildes,1.0,0.5)
      call move2d(fildes,0.5,0.0)
      call draw2d(fildes,0.5,1.0)

      call character_height(fildes,0.04)
      call text2d(fildes,0.55,0.90,'TA_CONTINUOUS_VERTICAL'//NULL,VDC_TEXT,
     c FALSE)
      call text2d(fildes,0.55,0.85,'With vertical = 0.35//NULL,VDC_TEXT,FALSE)
      call text2d(fildes,0.55,0.80,'and TA_CONTINUOUS_HORINTOZAL'//NULL,0,0);

      call character_height(fildes,0.18)
      call text_alignment(fildes,TA_LEFT,TA_CONTINUOUS_VERTICAL,0.0,0.35);
      call text2d(fildes,0.5,0.5,'ABC'//NULL,VDC_TEXT,FALSE)
```

Pascal Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,'TA_CONTINUOUS_VERTICAL',VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,'with vertical = 0.35',VDC_TEXT,FALSE);
text2d(fildes,0.55,0.80,'and TA_CONTINUOUS_HORINTOZAL',0,0);

character_height(fildes,0.18);
text_alignment(fildes,TA_LEFT,TA_CONTINUOUS_VERTICAL,0.0,0.35);
text2d(fildes,0.5,0.5,'ABC',VDC_TEXT,FALSE);
```

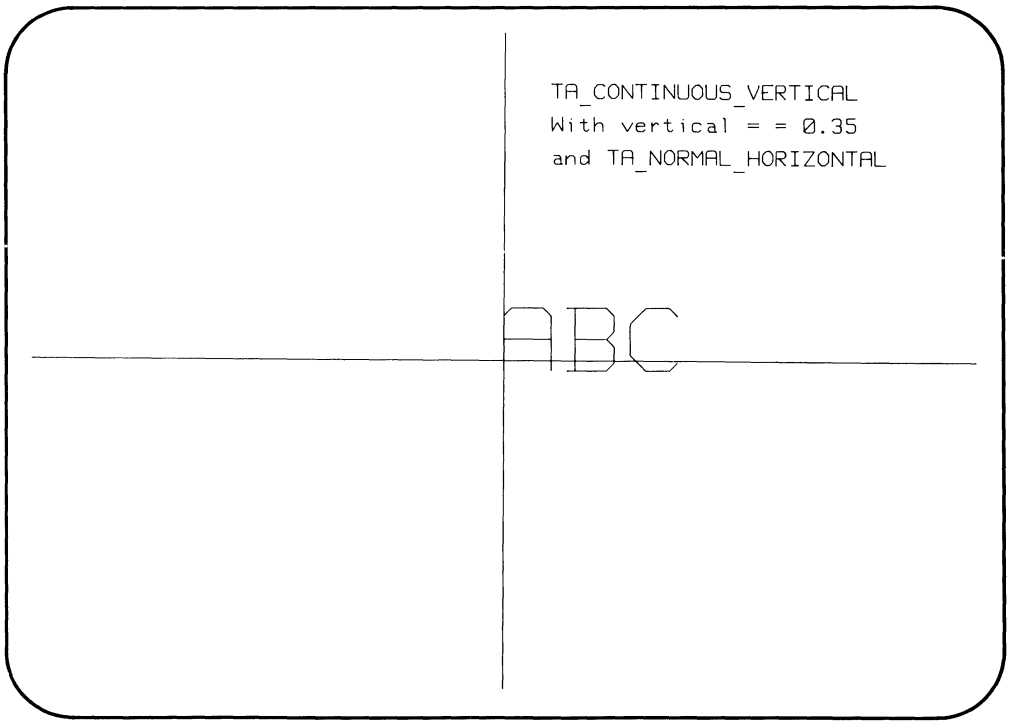


Figure 2-24. Text Alignment - TA_CONTINUOUS_VERTICAL, Example #1

Text alignment - TA_CONTINUOUS_VERTICAL Example #2

As a second example, a `vertical` value of 0.75 is used.

This example was created by the program `text24c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,"TA_CONTINUOUS_VERTICAL",VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,"With vertical = 0.75",VDC_TEXT,FALSE);
text2d(fildes,0.55,0.80,"and TA_NORMAL_HORIZONTAL",VDC_TEXT,FALSE);

character_height(fildes,0.18);
text_alignment(fildes,TA_LEFT,TA_CONTINUOUS_VERTICAL,0.0,0.75);
text2d(fildes,0.5,0.5,"ABC",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
      call move2d(fildes,0.0,0.5)
      call draw2d(fildes,1.0,0.5)
      call move2d(fildes,0.5,0.0)
      call draw2d(fildes,0.5,1.0)

      call character_height(fildes,0.04)
      call text2d(fildes,0.55,0.90,'TA_CONTINUOUS_VERTICAL'//NULL,VDC_TEXT,
     c FALSE)
      call text2d(fildes,0.55,0.85,'With vertical = 0.75//NULL,0,0)
      call text2d(fildes,0.55,0.80,'and TA_NORMAL_HORIZONTAL'//NULL,0,0);

      call character_height(fildes,0.18)
      call text_alignment(fildes,TA_LEFT,TA_CONTINUOUS_VERTICAL,0.0,0.75);
      call text2d(fildes,0.5,0.5,'ABC'//NULL,0,0)
```

Pascal Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,'TA_CONTINUOUS_VERTICAL',VDC_TEXT,0);
text2d(fildes,0.55,0.85,'with vertical = 0.75',VDC_TEXT,0);
text2d(fildes,0.55,0.80,'and TA_NORMAL_HORIZONTAL',VDC_TEXT,0);

character_height(fildes,0.18);
text_alignment(fildes,TA_LEFT,TA_CONTINUOUS_VERTICAL,0.0,0.75);
text2d(fildes,0.5,0.5,'ABC',VDC_TEXT,0);
```

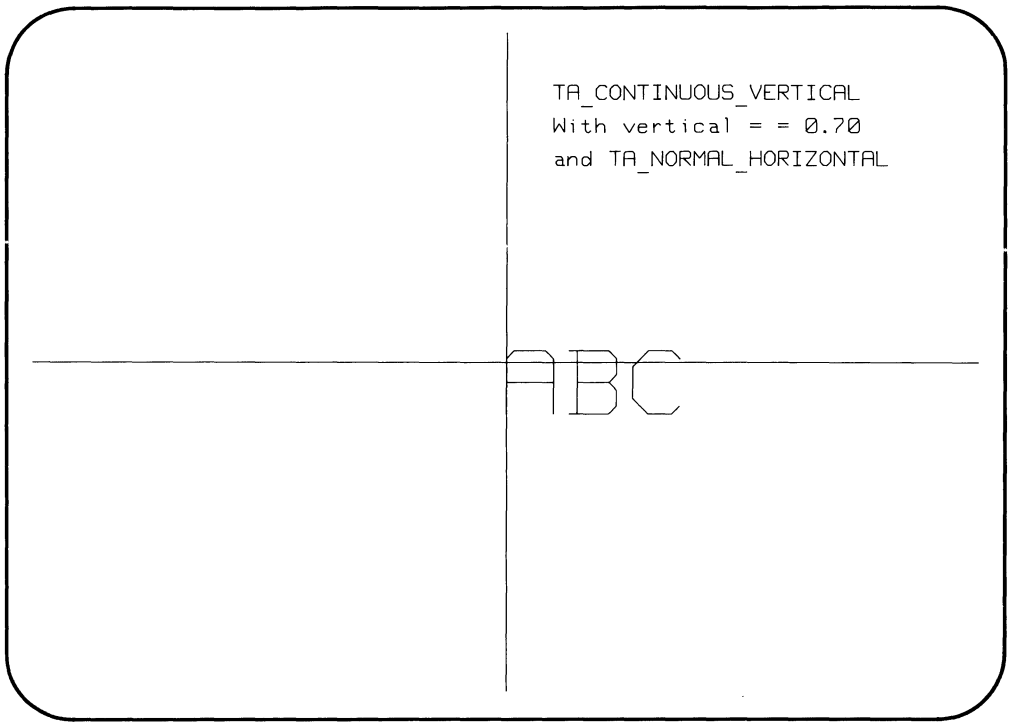


Figure 2-25. Text Alignment - TA_CONTINUOUS_VERTICAL, Example #2

Text alignment - TA_NORMAL_VERTICAL

This example was created by the program `text25c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,"TA_NORMAL_VERTICAL",VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,"TA_NORMAL_HORIZONTAL",VDC_TEXT,FALSE);

character_height(fildes,0.18);
text_alignment(fildes,TA_NORMAL_HORIZONTAL,TA_NORMAL_VERTICAL,0.0,0.0);
text2d(fildes,0.5,0.5,"ABC",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
      call move2d(fildes,0.0,0.5)
      call draw2d(fildes,1.0,0.5)
      call move2d(fildes,0.5,0.0)
      call draw2d(fildes,0.5,1.0)

      call character_height(fildes,0.04)
      call text2d(fildes,0.55,0.90,'TA_NORMAL_VERTICAL'//NULL,VDC_TEXT,
     c FALSE)
      call text2d(fildes,0.55,0.85,'TA_NORMAL_HORIZONTAL'//NULL,VDC_TEXT,FALSE)

      call character_height(fildes,0.18)
      call text_alignment(fildes,TA_NORMAL_HORIZONTAL,TA_NORMAL_VERTICAL,0.0,0.0);
      call text2d(fildes,0.5,0.5,'ABC'//NULL,VDC_TEXT,FALSE)
```

Pascal Syntax

```
move2d(fildes,0.0,0.5);
draw2d(fildes,1.0,0.5);
move2d(fildes,0.5,0.0);
draw2d(fildes,0.5,1.0);

character_height(fildes,0.04);
text2d(fildes,0.55,0.90,'TA_NORMAL_VERTICAL',VDC_TEXT,FALSE);
text2d(fildes,0.55,0.85,'TA_NORMAL_HORIZONTAL',VDC_TEXT,FALSE);

character_height(fildes,0.2);
text_alignment(fildes,TA_NORMAL_HORIZONTAL,TA_NORMAL_VERTICAL,0.0,0.0);
text2d(fildes,0.5,0.5,'ABC',VDC_TEXT,0);
```

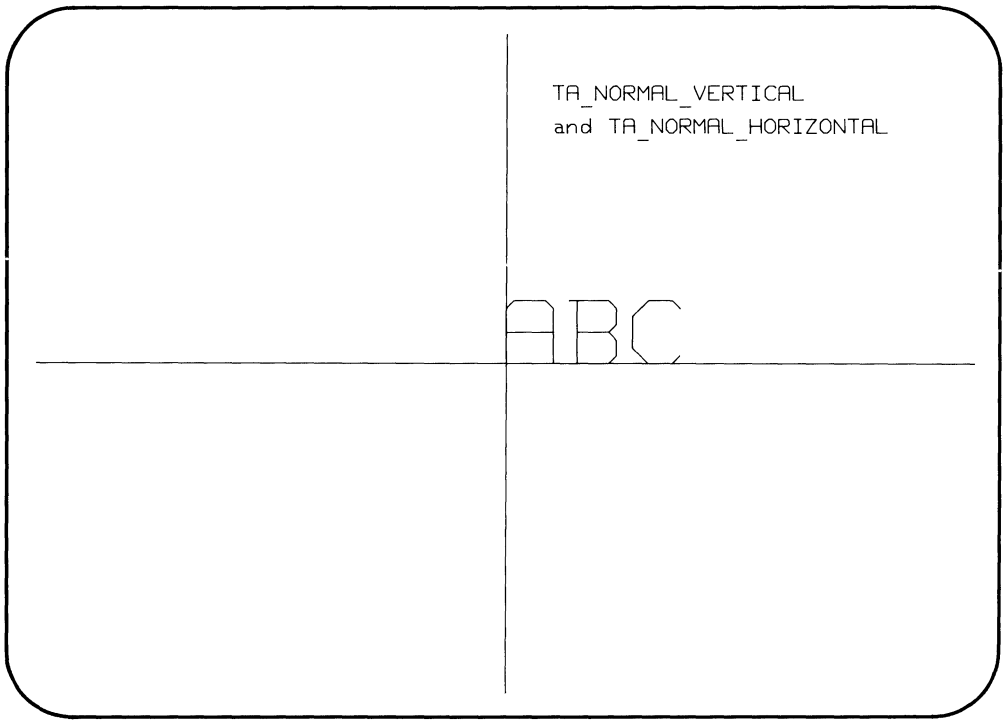


Figure 2-32. Text Alignment - TA_NORMAL_VERTICAL (text25c.c)

Designate Character Set

The following program shows how to select the various character sets.

This example was created by the program `text26c.c` found in `/usr/lib/starbase/demos`.

```
#include <starbase.c.h>
main(argc,argv)
int argc;
char *argv[];
{
int fildes,chset,gset;
float x,y;
char *c;
static char *chsets[] = {
"usascii",
"hproman",
"jisascii",
"katakana"
};
fildes = gopen("/dev/tty",OUTDEV,"hp262x",INIT);
vdc_extent(fildes,0.0,0.0,0.0,13.0,9.0,0.0);
clip_rectangle(fildes,0.0,13.0,0.0,9.0);
character_height(fildes,1.0);
for (chset=0; chset<=3; chset++) {
for (gset=0; gset<=3; gset++) {
designate_character_set(fildes,"",0);
designate_character_set(fildes,"",1);
designate_character_set(fildes,"",2);
designate_character_set(fildes,"",3);
designate_character_set(fildes,chsets[chset],gset);
set_p1_p2(fildes,FRACTIONAL,chset/4.0,gset/4.0,
                                    0.0,(chset+1.0)/4.0,(gset+1.0)/4.0,0.0);
switch (gset) {
case 0:
c="\017";
break;
case 1:
c="\016";
break;
case 2:
c="\033n";
break;
case 3:
c="\033o";
break;
}
text2d(fildes,0.0,0.0,c,0,0);
text2d(fildes,1.0,1.0," !\"#$%&'()*+",0,0);
```

```
text2d(fildes,1.0,2.0,",-./01234567",0,0);
text2d(fildes,1.0,3.0,"89:;<=>?@ABC",0,0);
text2d(fildes,1.0,4.0,"DEFGHIJKLMNO",0,0);
text2d(fildes,1.0,5.0,"PQRSTUVWXYZ[",0,0);
text2d(fildes,1.0,6.0,"\\]^_‘abcdefg",0,0);
text2d(fildes,1.0,7.0,"hijklmnopqrs",0,0);
text2d(fildes,1.0,8.0,"tuvwxyz{|}~| height6pt width4pt depth0pt",0,0);
}
}
gclose(fildes);
}
```

Figure 2-27. Character Sets

The following table shows the escape sequences needed to select a specific character set.

shift function		ISO 7 bit	ISO 8 bit	HP 8 bit
Shift out	SO	0/14	-	0/14
Shift in	SI	0/15	-	0/15
Locking shift zero	LS0	-	00/15	-
Locking shift one	LS1	-	00/14	-
Locking shift one right	LS1R	-	esc 7/14	-
Locking shift two	LS2	esc 6/14	esc 6/14	-
Locking shift two right	LS2R	-	esc 7/13	-
Locking shift three	LS3	esc 6/15	esc 6/15	-
Locking shift three right	LS3	-	esc 7/12	-
Single shift two	SS2	esc 4/14*	08/14	-
Single shift three	SS3	esc 4/15	08/15	-

* If a single byte representation of SS2 is required in 7 bits, it should be coded as 1/9.

Text Font Index

The `text_font_index` procedure allows you to select fonts from the current character set. The default character set has two fonts:

- font index = 1 - A fixed-width character font.
- font index = 2 - A variable-width character font.

This example was created by the program `text27c.c` found in `/usr/lib/starbase/demos`.

C Syntax

```
character_height(fildes,0.04);
text2d(fildes,0.0,0.6,"Fixed Width Text Font.",VDC_TEXT,FALSE);
text2d(fildes,0.0,0.2,"Variable Width Text Font.",VDC_TEXT,FALSE);

character_height(fildes,0.1);
text2d(fildes,0.0,0.5,"i like to bike",VDC_TEXT,FALSE);

text_font_index(fildes,2);
text2d(fildes,0.0,0.1,"i like to bike",VDC_TEXT,FALSE);
```

Fortran77 Syntax

```
      call character_height(fildes,0.04)
      call text2d(fildes,0.0,0.6,'Fixed Width Text Font.'//NULL,VDC_TEXT,FALSE)
      call text2d(fildes,0.0,0.2,'Variable Width Text Font.'//NULL,VDC_TEXT,
     c FALSE)

      call character_height(fildes,0.1)
      call text2d(fildes,0.0,0.5,'i like to bike'//NULL,VDC_TEXT,FALSE)

      call text_font_index(fildes,2)
      call text2d(fildes,0.0,0.1,'i like to bike'//NULL,VDC_TEXT,FALSE)
```

Pascal Syntax

```
character_height(fildes,0.04);
text2d(fildes,0.0,0.6,'Fixed Width Text Font.',VDC_TEXT,FALSE);
text2d(fildes,0.0,0.2,'Variable Width Text Font.',VDC_TEXT,FALSE);

character_height(fildes,0.1);
text2d(fildes,0.0,0.5,'i like to bike',VDC_TEXT,FALSE);

text_font_index(fildes,2);
text2d(fildes,0.0,0.1,'i like to bike',VDC_TEXT,FALSE);
```

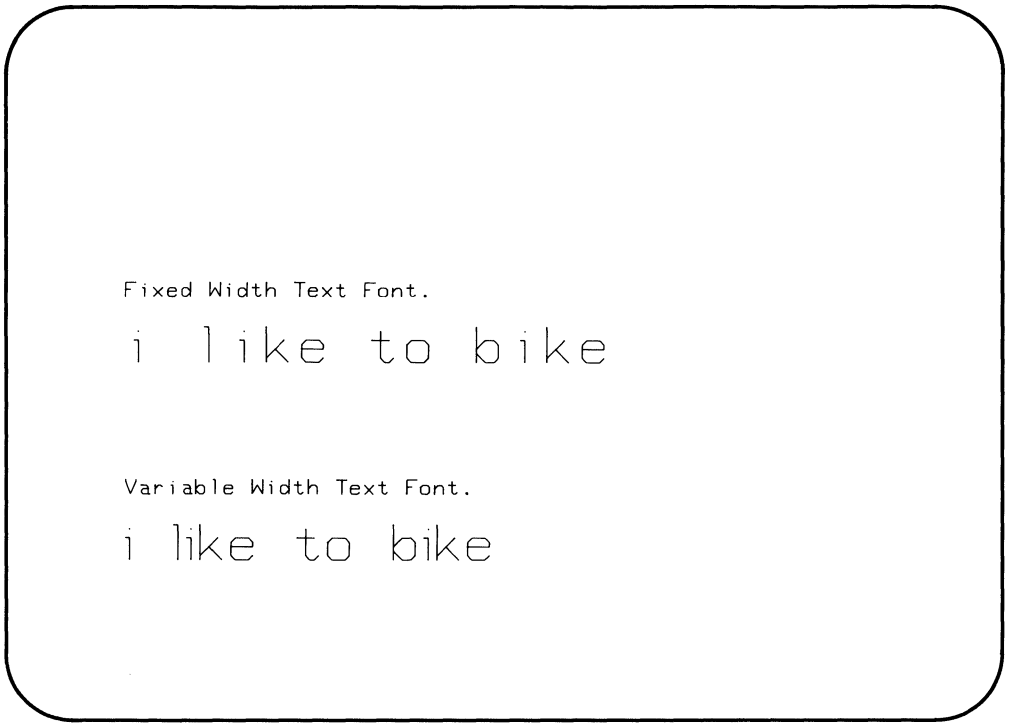


Figure 2-28. Character Fonts (text26c.c_)

Text Color

Text color is selected in the same manner as the fill color, perimeter color and marker colors are selected.

See the section `Color` in this manual for further information on Starbase color.

Notes

Starbase Color Graphics 3

Introduction

The following discussion of color graphics applies to those color graphic devices supported by the Starbase system. Not all devices support all color capabilities described in this chapter. See the *Starbase Device Drivers Library* for detailed information concerning color support for a specific device.

The following procedures are involved with color:

- **Background Color**
  - background_color - sets the background color by specifying the red, green and blue components of the desired color. The `clear_view_surface` procedure paints the background.
  - background_color_index - sets the background color by specifying the color map index of the desired color. The `clear_view_surface` procedure paints the background.
  - clear_view_surface - removes all graphic elements and paints the area specified by the `clear_control` procedure in the color specified by the `background_color` procedure.
  - clear_control - specifies the area of the view surface to be cleared by the `clear_view_surface` procedure. The view surface may be one of the following:
    - CLEAR_VDC_EXTENT - clear the rectangle defined by the `vdc_extent` procedure.
    - CLEAR_CLIP_RECTANGLE - clear the rectangle defined by the `clip_rectangle` procedure.
    - CLEAR_DISPLAY_SURFACE - clear the rectangle defined by the `set_p1_p2` procedure. This is the default.
  - clip_rectangle - defines the current clip rectangle. Graphic elements that extend beyond this rectangle are automatically clipped when clipping is enabled (see clip_indicator).
  - set_p1_p2 - specifies the bounds (p1 and p2) of the device.

- **The Color Table**
  - define_color_table - defines the red, green and blue components of the color table entries.
  - inquire_sizes - returns the devices physical limits, resolution, (p1, p2), and color map size.
- **Line Color**
  - draw2d, draw3d and dcdraw - draws lines between coordinate points. Line color is defined by the `line_color` procedures.
  - line_color - selects the color of lines drawn by Starbase by specifying the lines red, green and blue components.
  - line_color_index - selects the color of lines drawn by Starbase by specifying the the index into the color map.
  - perimeter_color - selects the color of perimeters drawn by Starbase by specifying the lines red, green and blue components.
  - perimeter_color_index - selects the color of perimeters drawn by Starbase by specifying the the index into the color map.
  - polyline - draws line segments defined by a list of points. The color of the lines is defined by the `line_color` procedures.
- **Fill Color**
  - fill_color - selects the red, green and blue components of the fill color. The fill color is used to fill polygons and rectangles.
  - fill_color_index - selects the fill color by an index into the color map. The fill color is used to fill polygons and rectangles.
  - fill_dither - selects the number of colors to be searched for and placed in the dither cell. Only 1, 2, 4, 8 and 16 colors are allowed. Devices may limit the number of options implemented for this parameter depending on device capabilities.
  - rectangle - draws rectangles. Default rectangles are filled and without perimeters.
  - polygon - draws polygons. Default polygons are filled and without perimeters.

- **Marker Color**
  - marker_color - selects the color of markers drawn by Starbase by specifying the lines red, green and blue components.
  - marker_color_index - selects the color of markers drawn by Starbase by specifying the the index into the color map.
  - polymarker - draws markers at designated locations. The color of the markers if defined by the `marker_color` procedures.
  - Text Color
  - text2d, text3d, dctext and append_text - draw textual characters. The color of the characters is defined by the `text_color` procedures.
  - text_color - selects the color of text drawn by Starbase by specifying the lines red, green and blue components.
  - text_color_index - selects the color of text drawn by Starbase by specifying the the index into the color map.

Color Generation

An Overview

All color is device dependent. Each device has its own color capabilities and color map.

Each display pixel is mapped into an address in a section of the computer's internal memory called the *frame buffer.* Each frame buffer address contains a number that is an index into the device's color map. In the following example, the example frame buffer entry contains the color map index value **3**.

The address of the color map is stored as a binary number. For example, the addresses of the HP 2627 Color Map contains 8 bits. Each bit position can be referred to as a **color plane** for that device. All of the least significant bits are plane 0, all of the most significant bits (for this device) are plane 7. Starbase allows you to *disable* color planes, i.e. ignoring any 1's in that column of bits. This limits the possible colors that can be displayed.

The color map can be thought of as a *3 by n array.* The three row entries define a binary value that produce a color intensity from the *color generation hardware.*

The color hardware generation hardware controls the beam of light sent to the display.

The dot on the display is the pixel identified by the frame buffer.

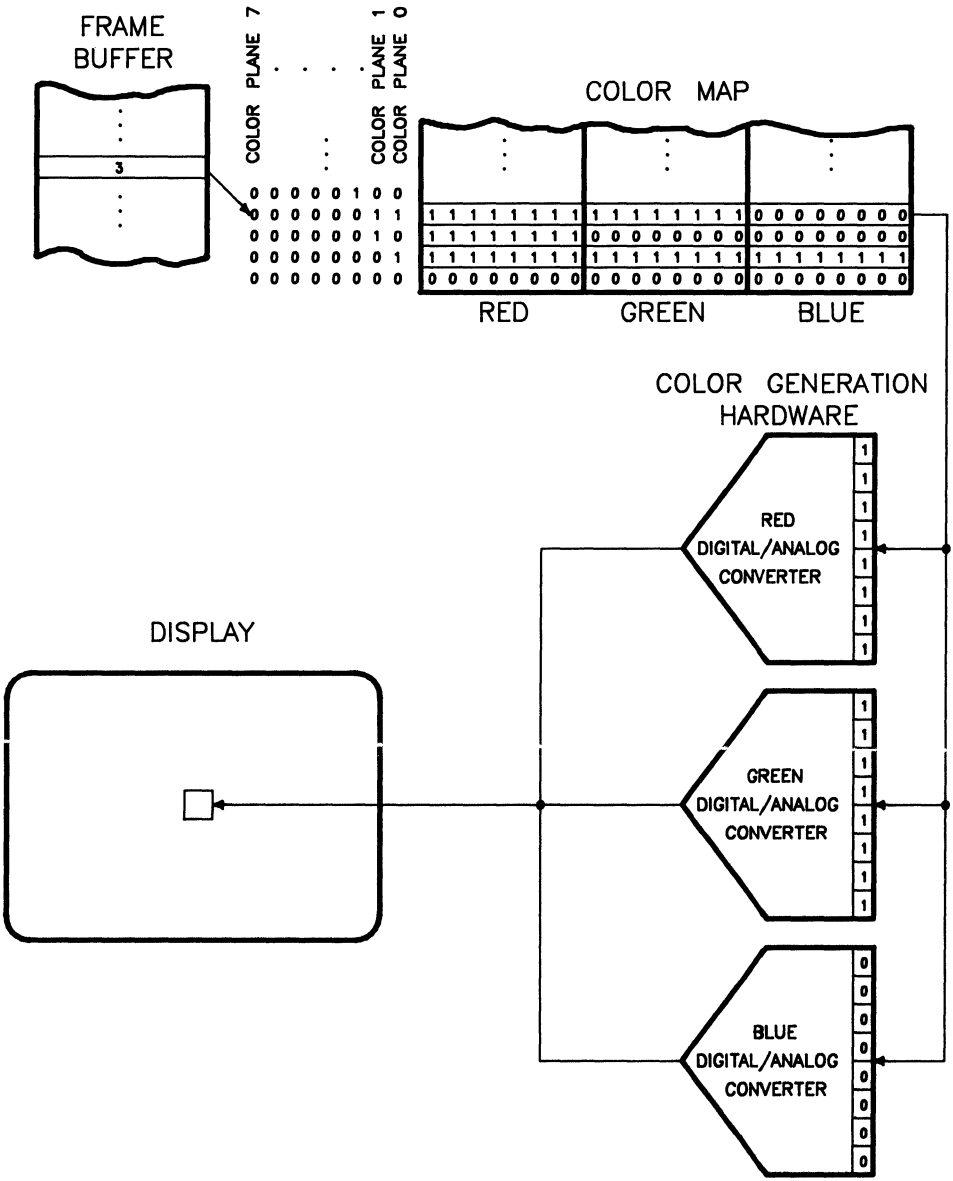


Figure 3-1. The Color Generation Process

The Frame Buffer

Starbase supports bit-mapped color graphics. An area in memory called a *frame buffer* provides a memory location for each pixel on the device's display. Each location contains an integer which is an index into the device's color map and thus defines the color for that pixel. The color description used in the frame buffer is *indirect.* Thus, the value in the frame buffer does not say "use color 12", but rather "use the color described by the color map at index number 12".

Initially, all locations in the buffer contain the value of the currently defined background color. When you draw a line on the screen, what really happens is the values for the correct frame buffer locations are changed to the currently defined line color.

Color Categories

At any given time, the values written to the frame buffer fall into six categories:

- *Background Color* - Whenever `clear_view_surface` is executed, all pixel locations in the frame buffer area to be cleared are set to the background color defined by the `background_color` procedure. The default background color is color table index 0 (zero).
- *Line Color* - Line color is selected with either `line_color` or `line_color_index`. Lines are created with the `polyline`, `draw2d`, `draw3d` and `dcdraw` procedures. The default line color is color table index 1 (one).
- *Perimeter Color* - Perimeter color is selected with either `perimeter_color` or `perimeter_color_index`. Polygons may be created with or without perimeters, depending on the `interior_style` procedure. Perimeters may be drawn with the `polygon`, `rectangle`, and `partial_polygon` procedures. The default perimeter color is color table index 1 (one).
- *Text Color* - Text color is selected with either `text_color` or `text_color_index`. Text is drawn using the `append_text`, `text2d` and `text3d` procedures. The default text color is color table index 1 (one).
- *Fill Color* - Fill color is selected with either `fill_color` or `fill_color_index`. Polygons may be filled or not filled depending upon the `interior_style` procedure. Polygons are created using the `polygon`, `rectangle`, and `partial_polygon` procedures. The default fill color is color table index 1 (one).
- *Marker Color* - Marker color is selected with `marker_color` or `marker_color_index`. Markers are drawn using the `polymarker` procedure. The default marker color is color table index 1 (one).

All of the color control statements listed above are referred to as *modal attributes.* This means that the value established by one of the statements stays in effect for that class of objects until altered by another statement.

The Color Map

Color Planes

The index into the color map is stored as binary values. Each device uses a specific number of binary bits to identify the colors available. For example, the HP 98700 supports 256 entries in the color map, thus needing 8 bits to identify each entry in the map. The HP 2627 supports 8 and requires 3 bits while the HP 2623 supports 2 colors and only one bit. The bit locations for the index values in the color map are called the **color planes** of the device. The colors available can be limited by enabling and disabling these planes.

Color Map Entries

The color map contains a complete description of the colors available for the device. Each color map index identifies three entries, one for each of the primary colors - Red, Green and Blue. The number of colors possible in a device is found by raising the number 2 to the power found by adding up the total number of bits used to specify the three colors. For the HP 98700, this value is over 16 million possible colors, of which, 256 can be active at a time.

Color Generation Hardware

The binary bits stored in the three segments of a color map entry form a pattern that, when applied to the Analog-to-Digital Converters in the raster device, produces a specific voltage level which causes that color to be of a specific intensity. The sum of the three intensities is the color seen on the display.

Selecting a Color

There are two ways to select a color:

- Specify the index into the color map. This is the fastest way to access a color. For example:

```
background_color_index(fildes,2);
```

- Specify the red, blue and green components of the color. Starbase will take your specification and will search the color table for the closest match. The index for that color is then used. This method is slower than the first due to the search time, however, when a specific color is wanted, this gives you the closest match. For example:

```
background_color(fildes,0.20,1.00,0.50);
```

Shell Scripts

The following shell scripts can be used to compile and execute the example programs described in this section. The scripts work with both the Bourne Shell and the C Shell (the C Shell calls the Bourne shell automatically).

The "-l" option is used to reduce typing. This compiler option prepends `/usr/lib/lib` to the file name and appends `.a` to the file name. This completes the path name to the correct file. For example:

```
... -lddhpgl
```

is processed as if it were

```
... /usr/lib/libddhpgl.a
```

A Script for C Programs

To following script will compile and run a C program, linking in the drivers for the HP-GL and HP 2627 devices and the required starbase libraries `sb1` and `sb2`. This script also allows you to easily link in other device drivers.

```
PROG=$1
shift
cc -o $PROG $PROG.c $* -lddhpgl -ldd2627 -lsb1 -lsb2
$PROG
```

This script is available in `/usr/lib/starbase/demos` as the executable file `CC`.

To compile and execute the example program `prog1.c` just type in:

```
CC prog1
```

To execute the same program again, just type in:

```
prog1
```

To link in the device driver for the HP 98700, just recompile with:

```
CC prog1 -ldd98700
```

A Script for Fortran77 Programs

To following script will compile a Fortran77 program, linking in the drivers for the HP-GL and HP 2627 devices and the required starbase libraries `sb1` and `sb2`. This script also allows you to easily link in other device drivers.

```
PROG=$1
shift
fc -o $PROG $PROG.f $* -lddhpgl -ldd2627 -lsb1 -lsb2
$PROG
```

This script is available in `/usr/lib/starbase/demos` as the executable file `FC`.

To compile and execute the example program `prog1.f` just type in:

```
FC prog1
```

To execute the same program again, just type in:

```
prog1
```

To link in the device driver for the HP 98700, just recompile with:

```
FC prog1 -ldd98700
```

A Script for Pascal Programs

To following script will compile a Pascal program, linking in the drivers for the HP-GL and HP 2627 devices and the required starbase libraries `sb1` and `sb2`. This script also allows you to easily link in other device drivers.

```
PROG=$1
shift
pc -o $PROG $PROG.p $* -lddhpgl -ldd2627 -lsb1 -lsb2
$PROG
```

This script is available in `/usr/lib/starbase/demos` as the executable file `PC`.

To compile and execute the example program `prog1.p` just type in:

```
PC prog1
```

To execute the same program again, just type in:

```
prog1
```

To link in the device driver for the HP 98700, just recompile with:

```
PC prog1 -ldd98700
```

Skeleton Programs

The following programs provide the various include files, the procedures that open and close the graphics devices and the other components needed to make working Starbase programs. The appropriate procedures are added and the result will compile and execute.

A Template Program for C

```
#include <starbase.c.h>

main(argc,argv)
int argc; char *argv[];
{
   int fildes;

   if (argc > 2) fildes=gopen(argv[1],OUTDEV,argv[2],NO_INIT);
   else fildes=gopen("/dev/tty",OUTDEV,"hp262x",INIT);
   if (fildes == -1) exit(-1);


   make_picture_current(fildes);
   gclose(fildes);
}
```

A Template Program for Fortran77

```
include '/usr/include/starbase.f1.h'
program chart
character NULL
parameter(NULL = char(0))
include '/usr/include/starbase.f2.h'

integer*4 fildes
integer*4 status

fildes = gopen('/dev/tty'//NULL, OUTDEV, 'hp262x'//NULL, INIT)
if (fildes .eq. -1) stop

status = gclose(fildes)
end
```

A Template Program for Pascal

```
program main(input,output);

$include '/usr/include/starbase.p1.h'$

var
   fildes, status : integer;

$include '/usr/include/starbase.p2.h'$
procedure exit(result: integer); external;

begin {main}
   fildes:=gopen('/dev/tty',OUTDEV,'hp262x',INIT);
   if fildes = -1 then exit(-1);

   status := gclose(fildes);
end.
```

Example Color Programs

The example programs that follow will present different colors depending upon the current definition of the device's color map. The examples were created using the color map for the HP 2627 Color Graphics Terminal.

A Color Map

The following example was used to generate the color map used by the HP 2627 Color Graphics Terminal. This example shows the colors of the map, and also demonstrates how the line, fill and text colors can be selected.

```
int fildes, color=0;
float y, r[1][3];
char c[80];
char x[80];
static char *name[] = {
   "Black",
   "White",
   "Red",
   "Yellow",
   "Green",
   "Cyan",
   "Blue",
   "Magenta"};

mapping_mode(fildes,TRUE);
character_height(fildes,0.06);
```

```
character_width(fildes,0.015);
interior_style(fildes,INT_SOLID,TRUE);
text_alignment(fildes,TA_NORMAL_HORIZONTAL,TA_HALF,0.0,0.0);

for (y=0.0; y<=0.8; y+=0.1) {
   fill_color_index(fildes,color);
   rectangle(fildes,0.60,y,0.73,y+0.08);
   inquire_color_table(fildes,color,1,r);
   text2d(fildes,0.75,y+0.02,name[color],VDC_TEXT,FALSE);
   sprintf(x,"Red = %3.2f Green = %3.2f Blue = %3.2f",
      r[0][0],r[0][1],r[0][2]);
   sprintf(c,"Pen %d",color++);
   text2d(fildes,0.0,y+0.02,x,VDC_TEXT,FALSE);
   text2d(fildes,0.63,y+0.02,c,VDC_TEXT,FALSE);
   }
```

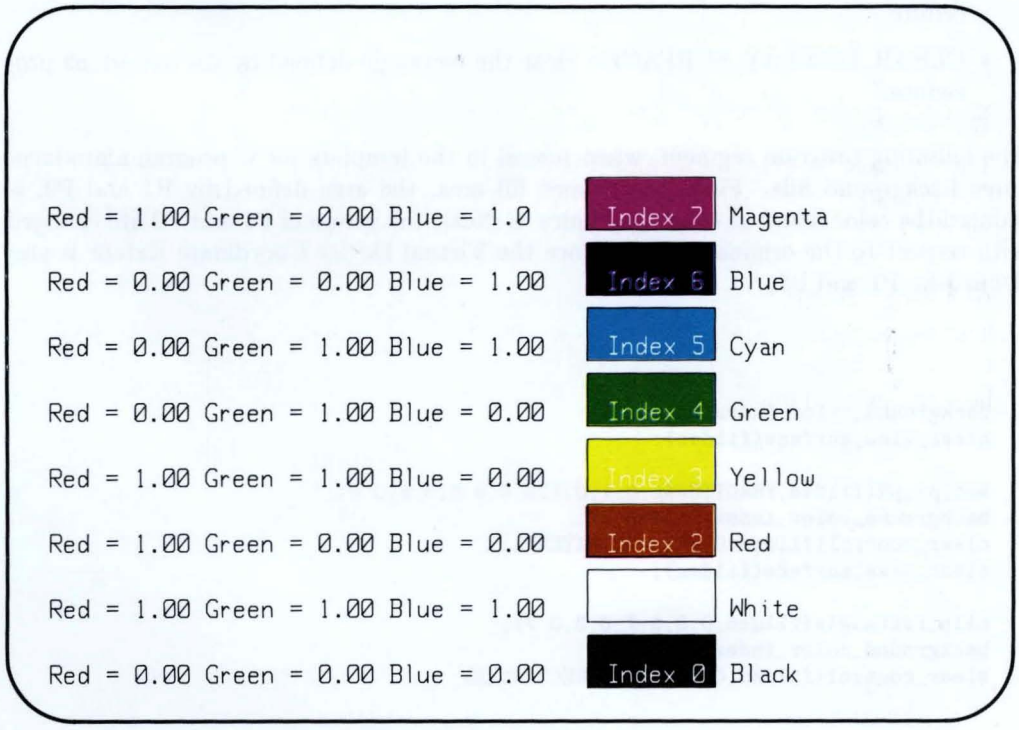


Figure 3-2. A Color Map (color1c.c)

Background Color

These procedures are device dependent and only work with those devices that have backgrounds. An exception is a plotter with automatic paper feed. The plotter will advance the paper whenever a `background_color` procedure is executed.

The background area defined by the `clear_control` procedure is painted the color specified by either the `background_color` or `background_color_index` procedure with each execution of the `clear_view_surface` procedure. This painting causes the entire area to be the color specified.

The `clear_control` procedure uses the following enumerated types to define the areas to be painted:

- CLEAR_VDC_EXTENT - the rectangle defined by the `vdc_extent` procedure.
- CLEAR_CLIP_RECTANGLE - the rectangle defined by the `clip_rectangle` procedure.
- CLEAR_DISPLAY_SURFACE - clear the rectangle defined by the `set_p1_p2` procedure.

The following program segment, when placed in the template for C programs produces three background fills. First, the default fill area, the area defined by P1 and P2, is painted the color found at color map index 4. Next, the values of P1 and P2 are changed with respect to the original values. Since the Virtual Device Coordinate Extent is also defined by P1 and P2,

```
.
.
.
background_color_index(fildes,4);
clear_view_surface(fildes);

set_p1_p2(fildes,FRACTIONAL,0.1,0.1,0.0,0.9,0.9,0.0);
background_color_index(fildes,5);
clear_control(fildes,CLEAR_VDC_EXTENT);
clear_view_surface(fildes);

clip_rectangle(fildes,0.3,0.7,0.3,0.7);
background_color_index(fildes,6);
clear_control(fildes,CLEAR_CLIP_RECTANGLE);
.
.
.
```

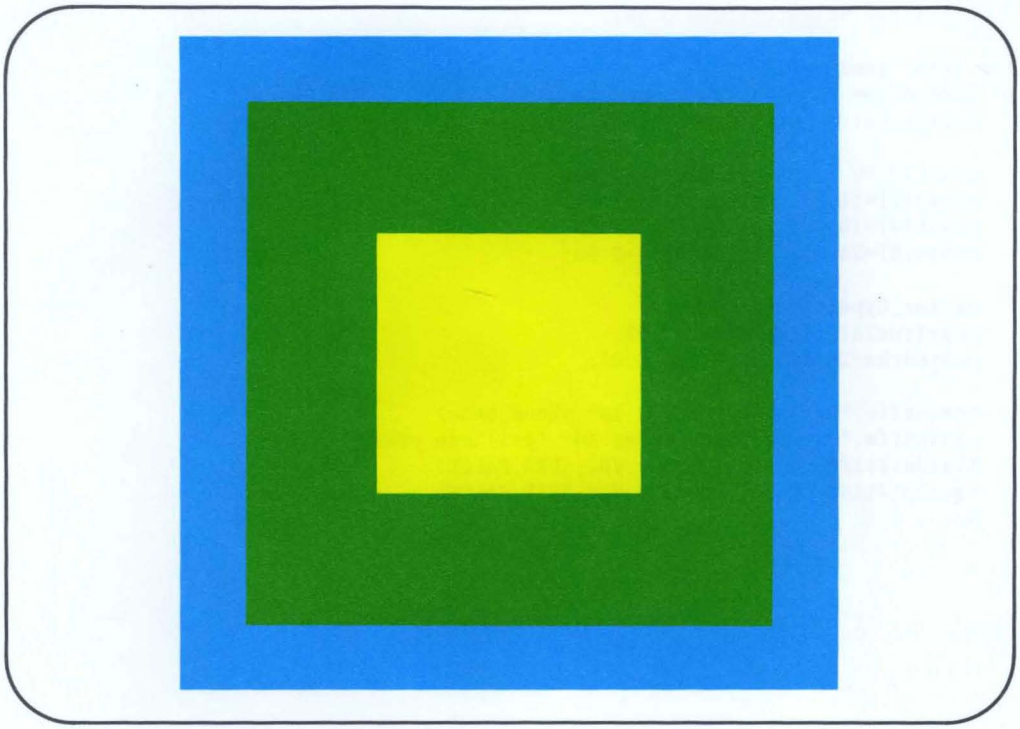


Figure 3-3. Background Color (color2c.c)

Polyline and Polymarker Color

The following example is used to show how polylines and markers can be drawn in different colors.

```
int fildes,y,num_pts;
float clist[8];
float ph_lim[2][3], res[3], p1[3], p2[3];
int num_pens;
char c[80];
char m[80];

inquire_sizes(fildes,ph_lim,res,p1,p2,&num_pens);

vdc_extent(fildes,0.0,0.0,0.0,25.0,10.0,0.0);
clip_rectangle(fildes,0.0,25.0,0.0,10.0);
mapping_mode(fildes,TRUE);

marker_orientation(fildes,0.0,1.0);
```

```
    marker_size(fildes,0.2,1);
    character_height(fildes,0.35);

for (y=0; y<=8; y++){
    line_color_index(fildes,y%num_pens);
    marker_color_index(fildes,(y+1)%num_pens);

    clist[0]=9.0;     clist[1]=y+0.50;
    clist[2]=14.0;    clist[3]=y+1.50;
    clist[4]=19.0;    clist[5]=y+1.50;
    clist[6]=24.0;    clist[7]=y+0.50;

    marker_type(fildes,y);
    polyline2d(fildes,clist,4,0);
    polymarker2d(fildes,clist,4,0);

    sprintf(c,"Line Color Index %d",y%num_pens);
    sprintf(m,"Marker Color Index %d",(y+1)%num_pens);
    text2d(fildes,0.30,y+0.35,c,VDC_TEXT,FALSE);
    text2d(fildes,0.30,y+0.65,m,VDC_TEXT,FALSE);
    }
```

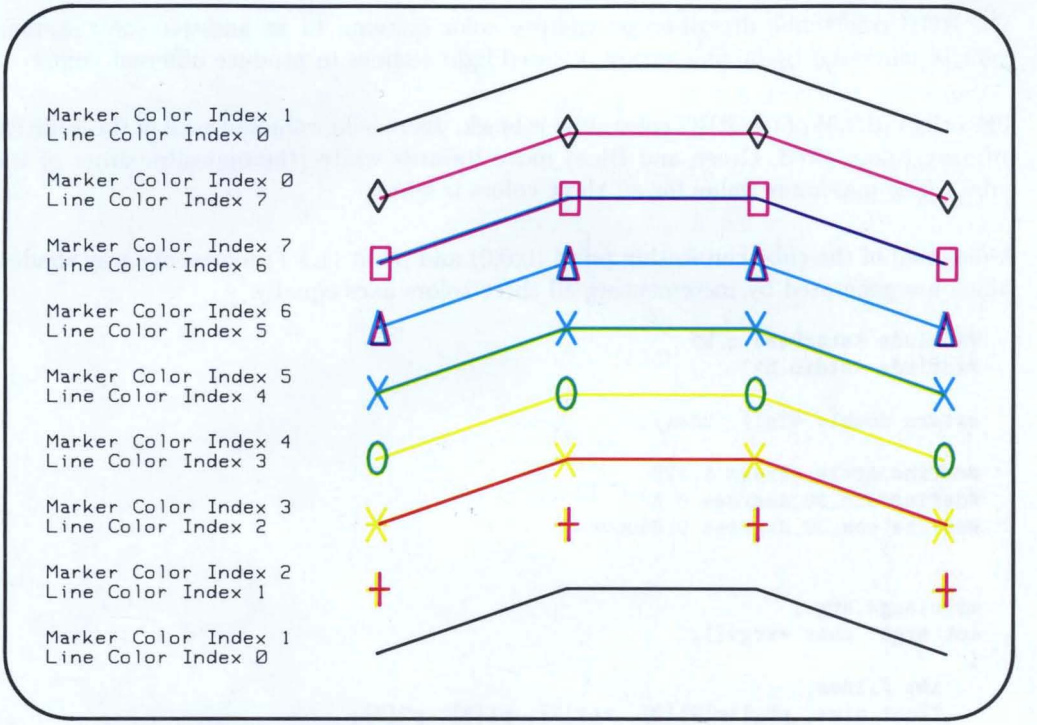


Figure 3-4. Polyline and Polymarker Color (color3c.c)

The Color Cube

The RGB color cube describes an *additive* color system. In an additive color system, color is generated by mixing various colored light sources to produce different colors.

The origin (0,0,0) of the RBG color cube is black. Increasing values of each of the additive primary colors (Red, Green and Blue) move towards white (the opposite corner of the cube.) The maximum value for all three colors is white.

A diagonal of the cube connecting point (0,0,0) and point (1,1,1) represents gray shades, which are generated by incrementing all three colors axes equally.

```
#include <starbase.c.h>
#include <stdio.h>

extern double sin(), cos();

#define scale_screen 1.125
#define sin_30_degrees 0.5
#define cos_30_degrees 0.866025


main(argc,argv)
int argc; char *argv[];
{
   int fildes;
   float size, ph_lim[2][3], res[3], p1[3], p2[3];
   float extent, s, c;
   int cmap_size, dither_size, step, r, g, b, valid;
   char ch;

   if (argc>2) fildes=gopen(argv[1],OUTDEV,argv[2],INIT);
   else fildes=gopen("/dev/tty",OUTDEV,"hp2627",INIT);
   if (fildes == -1) exit(1);

   inquire_sizes(fildes,ph_lim,res,p1,p2,&cmap_size);

   valid = FALSE;
   while (!valid) {
      printf("Enter dither size (1,2,4,8 or 16): ");
      scanf("%d",&dither_size);
      if ((dither_size == 1) || (dither_size == 2) ||
         (dither_size == 4) || (dither_size == 8) ||
         (dither_size == 16)) valid = TRUE;
      }
   fill_dither(fildes,dither_size);

   fflush(stdin);
```

```
    s = sin_30_degrees;
    c = cos_30_degrees;

    extent = cmap_size * scale_screen;
    step = cmap_size/8;

    vdc_extent(fildes,-extent,-extent,0.0,extent,extent,0.0);
    clip_rectangle(fildes,-extent,extent,-extent,extent);

    for (g=0; g<=cmap_size; g=g+step) {
       for (r=0; r<=cmap_size; r=r+step) {
          for (b=0; b<=cmap_size; b=b+step) {
             fill_color(fildes,(float)r/cmap_size,(float)g/cmap_size,
                (float) b/cmap_size);
             polygon(fildes,step,c*r-c*b,g-s*r-s*b);
             }
          }
       }
    make_picture_current(fildes);
    gclose(fildes);
}

    polygon(fildes,scale,x,y)
    int fildes,scale;
    float x, y;
    {
       float clist[14];
       int i;

       clist[0] = x;
       clist[1] = y + scale;
       clist[2] = x + cos_30_degrees * scale;
       clist[3] = y + sin_30_degrees * scale;
       clist[4] = x + cos_30_degrees * scale;
       clist[5] = y - sin_30_degrees * scale;
       clist[6] = x;
       clist[7] = y - scale;
       clist[8] = x - cos_30_degrees * scale;
       clist[9] = y - sin_30_degrees * scale;
       clist[10] = x - cos_30_degrees * scale;
       clist[11] = y + sin_30_degrees * scale;
       clist[12] = x;
       clist[13] = y + scale;

       polygon2d(fildes,clist,7,0);
       make_picture_current(fildes);
    }
```

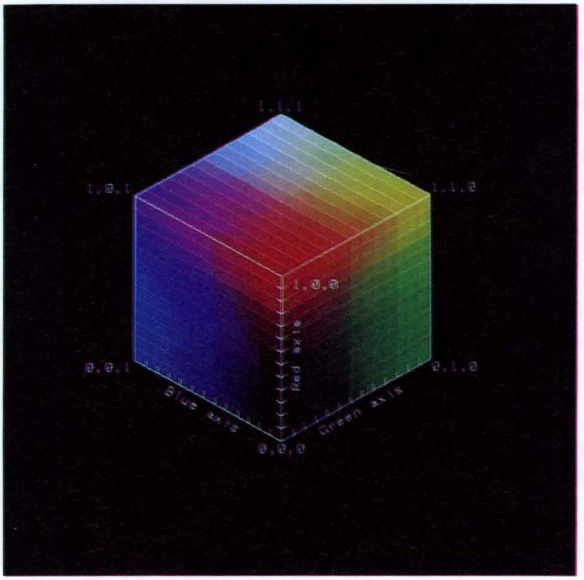


Figure 3-5. Color Cube (color4c.c)

Input 4

Introduction

The Starbase input model involves two classes of input:

- Locator Devices - return x, y, and z coordinates.
- Choice Devices - return a single integer value.

A file descriptor may be associated with any number of locators and choice devices, depending on the nature of the physical device which it represents. The number of locators and number of choice devices associated with a file descriptor may be determined by calling `inquire_input_capabilities`. This call also lists the number of valuators and string devices, but these fields are only present for future extensions; no valuators or string devices are currently implemented.

Locator Devices

Locators return floating point numbers in Virtual Device Coordinate (VDC) values. The range of VDC values is determined separately for each file descriptor and may be different for input and output devices. The range of values may be set to convenient units by using the `vdc_extent` procedure. Usually a locator should be set to use the same units as a related display device. Note that the value returned from a locator is not in World Coordinates and therefore is not affected by the World Coordinate transform stack.

Choice Devices

Choice devices return an integer value, usually describing a set of buttons or keys. One choice device can describe an entire keyboard or set of buttons. Most button devices return button numbers ranging from 1 to N, (N being the number of buttons.) Keyboard devices return ASCII character codes for those keys with ASCII codes. A value might also represent the release of a button, (usually as a negative value.) Some drivers also provide an alternate representation of the same set of buttons. For instance, devices on the HP-HIL interface loop return button numbers on choice device 1, and a bit-wise encoding of the same buttons on choice device 2. The buttons 1 through 32 on an HP-HIL device are represented by the bits 2^0 through 2^31 in the choice value. For each button pressed, a bit is set to 1.

Triggering

Most input devices feature a way to indicate that data is ready. This trigger mechanism is used to proceed when the computer is waiting for a response. For choice devices this is a often a button or key press, (or possibly the release of a button.) For locators, the trigger is usually a button or a switch on the pointing device. One trigger can cause multiple locators and choice devices on a file descriptor to indicate that data is ready. This trigger mechanism is used by the request and event input techniques discussed later in this section. A few devices have no trigger mechanism. These devices are never waited for. They are always read from in a non-blocking manner.

Input Methods

Sampling

Sampling reads the current value of a device without waiting for any trigger action by a user. Repeated samples may be used to monitor a continuously changing value.

Requests

A request reads the value of a device the next time that a trigger indicates data is ready. It may be used to read one value after prompting the user to enter data.

Requests wait for a trigger on an input device, then return the value of the device at the trigger time. A program may either begin a request and proceed to other activities, or begin a request and wait for it to complete. Since requests take only one data value they are well suited to reading a single input after prompting for data. They are not suitable for capturing many closely spaced triggers, because triggers may be missed between the completion of one request and the start of another. Requests can be started on multiple devices, but each device must be tested separately to determine when a request has completed.

Events

The event mechanism reads a device whenever a trigger occurs. The data from a device, (or several devices,) is placed into a first-in-first-out queue for later retrieval. Events can be used in order to be certain that all trigger actions are remembered even if an application program is busy when the triggers occur.

Events use a single queue to save away the values from input devices whenever they are triggered. This allows an application to monitor several input devices at the same time. It also guarantees that triggers will not be missed. Each input device has a daemon process that is started whenever:

- Events are enabled for that device.
- Tracking is enabled for that device.

While requests need to be restarted after completing, events remain enabled until they are explicitly turned off, or the device is closed. This makes event queuing well suited to tasks where no data should be missed, such as reading text from a keyboard. On the other hand, events are not convenient if data is only required as a response to a prompt. The program below uses events to read all locator and choice triggers from one device. It also uses tracking to follow the locator on a display.

Tracking

Tracking is the feedback from a locator device to a display device. An echo position will follow the value of locator device without any need for continuous monitoring by an application program.

Tracking causes a separate process to constantly follow the value of a locator. This value is used to update the position of an echo on a display device. Each input device may track one of it's locators to the first echo of one display. If an input device features more than one locator, then only one of the locators on the device can be tracked at one time. If an output device displays multiple simultaneous cursors, then the tracking activity will affect the first of these cursors.

There is substantial overhead involved with the start of tracking, so it is best to avoid frequently turning tracking on and off. Since tracking always uses the current echo type, setting the echo type to no echo, (0), will stop visual feedback without stopping the tracking activity.

Sampling

The procedures `sample_choice` and `sample_locator` return the current value of a choice or locator device. The device is specified by a file descriptor and ordinal. The call returns a flag, `valid`, which indicates when the device was unable to return a value.

Valid may be FALSE because:

- the device does not exist
- there was a hardware failure
- the device was in a bad state for sampling.

Some devices cannot be sampled while waiting for a trigger. Such a limitation is indicated by the `sample_while_request` flag from the `inquire_input_capabilities` routine. Sample for a choice device may have different meanings for different devices. For some devices, sample will only return the number of the last button pressed. For other devices sample can return the up/down state of all buttons at the current time. Because of these differences in device capabilities, `sample_choice` is not as device independent as other input calls.

Example 1

The program below uses a request and repeated sample calls to enter a data point.

```
#include <starbase.c.h>
main()
   {
   int locator, display, valid, ready;
   float x, y, z;

   locator = gopen("/dev/locator", INDEV, "hp-hil", INIT);
   if (locator == -1) exit(1);
   display = gopen("/dev/crt", OUTDEV, "hp98700", INIT);
   if (display == -1) {
   locator = gclose(locator);
   exit(1);
   }

   sample_locator(locator, 1, &valid, &x, &y, &z);
   echo_type(display, 1, 1, &x, &y, &z);
   initiate_request(locator, LOCATOR, 1, &valid);
   do {
   sample_locator(locator, 1, &valid, &x, &y, &z);
   echo_update(display, 1, x, y, z);
   inquire_request_status(locator, LOCATOR, 1, &ready);
   } while (!ready);
      request_locator(locator, 1, 1.0, &x, &y, &z);
      echo_type(display, 1, 0, x, y, z);
      printf("%f, %f, %f\n", x, y, z);
      locator = gclose(locator);
      display = gclose(display);
   }
```

Example 2

The next program uses events and tracking to read 20 values from both a locator and a choice device.

```
#include <starbase.c.h>
main()
   {
   int locator, display;
   int valid, class;

   printf("Demonstration of events.\n");

   locator=gopen("/dev/locator",INDEV,"hp-hil",INIT);
   if (locator == -1) exit(1);
   display = gopen("/dev/crt",OUTDEV,"hp98700",INIT);
   if (display == -1) {
      locator = gclose(locator);
      exit(1);
   }

   clear_view_surface(display);
   enable_events(locator,LOCATOR,1);
   enable_events(locator,CHOICE,1);

   track(locator, display, 1);

   count = 0;
   do {
      await_event(-1,2.0,&valid,&class);
      if (valid) {
         count = count + 1;
         printf("await_event- (valid)  class: %d\n", class);
         switch (class) {
            case LOCATOR:
                  check_locator(display);
                  break;
            case CHOICE:
                  check_choice();
                  break;
                        }
                  }
            else
               printf("await_event- (not valid)\n");
               } while (count < 20);

               disable_events(locator,ALL,1);
               track_off(locator);
               gclose(display);
               gclose(locator);
```

```
            exit(0);
            }

            check_locator(display)
            int  display;
            {
            int valid, fildes, ordinal, status, message_link;
            floatx=0.5, y=0.5, z=0.0;

            valid = !read_locator_event(-1,&fildes,&ordinal,&x,&y,&z,&status,
&message_link);
            printf("Locator:\n");
            printf("valid:%d,fildes:%d,ordinal:%d,x:%f,y:%f,z:%f\n",valid,
                    fildes,ordinal,x,y,z);
            printf("status: ");
            switch (status) {
               case EMPTY_NO_OVERFLOW: printf("empty, no overflow\n"); break;
               case NOT_EMPTY_NO_OVERFLOW:printf("not empty, nooverflow\n");
                    break;
               case EMPTY_OVERFLOW: printf("empty, overflow\n"); break;
               case NOT_EMPTY_OVERFLOW:printf("not empty, overflow\n");break;
               }
            printf("message_link: ");
            switch (message_link) {
               case SINGLE_EVENT: printf("single event\n"); break;
               case SIMULTANEOUS_EVENT_FOLLOWS:printf("simultaneous event
                                                        follows\n");
               }
            if (valid) {
            move2d(display, x - 0.01, y - 0.01);
            draw2d(display, x + 0.01, y - 0.01);
            draw2d(display, x + 0.01, y + 0.01);
            draw2d(display, x - 0.01, y + 0.01);
            draw2d(display, x - 0.01, y - 0.01);
            make_picture_current(display);
            if (x < 0.1 && y < 0.1)
                        }
            }

            check_choice()
            {
            int valid, fildes, ordinal, value, status, message_link;

            valid=!read_choice_event(-1,&fildes,&ordinal,&value,&status,
                   &message_link);
            printf("Choice:\n");
            printf("Choice:\nvalid:%d,fildes:%d,ordinal:%d,value:%d\n",
                    valid,fildes,ordinal,value);
            printf("status: ");
            switch (status) {
```

```
    case EMPTY_NO_OVERFLOW: printf("empty, no overflow\n"); break;
    case NOT_EMPTY_NO_OVERFLOW:printf("not empty, no overflow\n");
         break;
    case EMPTY_OVERFLOW: printf("empty, overflow\n"); break;
    case NOT_EMPTY_OVERFLOW: printf("not empty, overflow\n");
         break;
                }
printf("message_link: ");
switch (message_link) {
    case SINGLE_EVENT: printf("single event\n"); break;
    case SIMULTANEOUS_EVENT_FOLLOWS: printf("simultaneous event
                                                    follows\n");
                    }
}
```

Modeling and Viewing

Introduction

The viewing transformations that occur in the starbase graphics library are outlined in the diagram below.

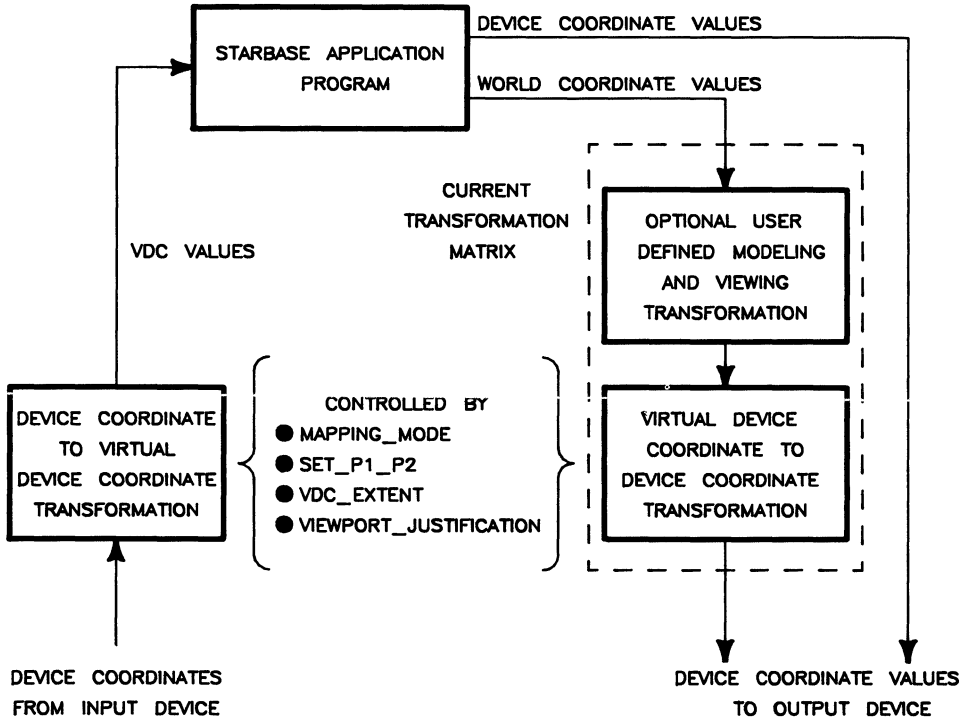


Figure 4-1. Transformation Diagram

Graphics transformations are commonly explained in terms of a modeling transformation which moves objects/subobjects in the user's coordinate system followed by a viewing transformation which determines what subset of the users coordinate system is visible, and a normalized device coordinate transformation which scales to the device's native coordinate system. Starbase combines these three logical steps into one actual transformation, so that there is only one current transformation.

There are three coordinate systems that are of primary interest:

- Device Coordinates (DC)
- Virtual Device Coordinates (VDC)
- World Coordinates (WC)

Utilities that convert between these coordinate systems make it possible for the user to work easily in any system needed.

Two-Dimensional Viewing

Two-dimensional viewing is best explained with a simple windowing transformation. A window can be specified in an arbitrary coordinate system (the World Coordinate System) defined by the application program. These units can be any units that are convenient for the user's program; i.e., inches, feet, millimetres, years, dollars, etc.

The portion of the graphical data which is within this window will be visible on the screen (physical display surface) within the current viewport (subset of screen). The manner in which this happens is that the full extent of the window maps to the full extent of the viewport.

Any portions of the graphical data which lie outside of the window are clipped (discarded from the picture).

Using starbase, there is more than one way to do windowing. The simplest is to use the `vdc_extent` procedure to scale the range of virtual device coordinates to be that of the window. The viewport position and size can be changed with the `set_p1_p2`, `viewport_justification`, and `mapping_mode` procedures.

Example 1:

In the following example, a viewport which fills the entire HP 98710 display is set to receive data with the X-axis defined in years from 1980 thru 1985 and the Y-axis defined in dollars from $0.00 thru $100.00.

```
gopen(fildes,"/dev/crt",OUTDEV,"hp98710",INIT);

vdc_extent(fildes,1980.0,0.0,0.0,1985.0,100.0,0.0);
clip_rectangle(fildes,1980.0,1985.0,0.0,100.0);
mapping_mode(fildes,ISOTROPIC);

set_p1_p2(fildes,FRACTIONAL,0.0,0.0,0.0,1.0,1.0,1.0);
```

Example 2:

In this second example, a window is created in the upper-right quarter of the display.

```
gopen(fildes,"/dev/crt",OUTDEV,"hp98710",INIT);

/* Set up window.  X and Y axes are both inches. */

vdc_extent(fildes,500.0,900.0,0.0,200.0,400.0);
clip_rectangle(fildes,500.0,900.0,200.0,400.0);

/* Viewport is restricted to upper right quarter of screen. */

mapping_mode(fildes,ISOTROPIC);
set_p1_p2(fildes.FRACTIONAL,0.5,0.5,0.0,1.0,1.0,0.0);

/* Viewport is centered in x and y within p1_p2 area to */
/* maintain isotropic units. */

viewport_justification(fildes,0.5,0.5);
```

A more elaborate way of doing windowing is to set up vdc_extent to a fixed normalized device coordinate range and window changes can be implemented using a 2-dimensional transformation matrix. The mapping and positioning defined by the above equations are simply the scaling and translations terms of a 2-dimensional matrix. This second method has the advantage that the window can also be made to rotate within the user's arbitrary coordinate system by using the rotation components of the matrix. The viewport can then be changed by just changing the clip_rectangle and appropriate scaling and translation terms of the matrix.

Using The Transformation Matrix Stack

Starbase has a transformation stack that can be used to concatenate and/or stack graphics viewing/modeling matrix operations. This feature is useful for composing a viewing matrix, instancing objects/subobjects, and temporarily changing the current transformation and returning to the previous state. It can also be used as general-purpose 4x4 transformation engine for such things as bspline curve generation.

The top of the matrix stack is the current transformation matrix, through which all output primitives are transformed. Conceptually, and in the actual implementation, the vdc-to-device coordinate transformation is a transformation matrix also. It can be thought of as the matrix at the bottom of the matrix stack which can never be popped or replaced using the matrix operations. It can be changed by using the vdc operations specified in the manual(vdc_extent, set_p1_p2, mapping_mode, and viewport_justification). The vdc-to-device matrix is the current transformation matrix immediately after a gopen since the matrix stack is empty.

The push and replace matrix operations are always concatenated with the current vdc-to-device matrix. This provides device independence when using the matrix stack. The matrix concatenation operations will concatenate with the current transformation matrix. If the matrix stack is empty, this concatenation is performed with the vdc-to-device matrix.

The post-concatenation option on matrix concatenation operations is NOT recommended for the novice user. The results are device dependent because the matrix is post-concatenated after the vdc-to-device transformation.

The push_vdc_matrix is a convenient and efficient means of getting the vdc-to-device matrix onto the top of the matrix stack. Changes to the vdc-to-device matrix DO NOT affect any matrices on the stack that were concatenated with the vdc-to-device matrix OR any matrices on the stack that were put there using push_vdc_matrix. It is highly recommended to flush the matrix stack before changing the vdc-to-device matrix.

Now that the matrix stack usage has been reviewed, the second method of doing a 2-dimensional windowing transformation can be demonstrated.

Example 3:

```
/* Initialize the range of virtual device. */

gopen(fildes,"/dev/crt",OUTDEV,"hp98710",INIT);
vdc_extent(fildes,0.0,0.0,0.0,1.0,1.0,1.0);
mapping_mode(fildes,ISOTROPIC);
push_vdc_matrix(fildes);

/* Each time the window is changed, the following must be done. */

clip_rectangle(fildes,Vxl,Vxr,Vyb,Vyt);
Sx = (Vxr - Vxl)/(Wxr - Wxl);
Sy = (Vyt - Vyb)/(Wyt - Wyb);
winxform[0][0] = Sx;
winxform[0][1] = 0;
winxform[1][0] = 0;
winxform[1][1] = Sy;
winxform[2][0] = Sx*Wxl + Vxl;
winxform[2][1] = Sy*Wyb + Vyb;
replace_matrix2d(fildes,winxform);
```

In the above example, if Wxl=400, Wxr=900, Wyb=300, Wyt=700, and Vxl=0.5, Vxr=1.0, Vyb=0.5, Vyt=1.0, then the range of window coordinates 400 to 900 in x and 300 to 700 will map to the upper left quarter of the screen.

The other operation besides windowing that the transformation stack is useful for is modeling transformations. By setting up the viewing operation first and then concatenating modeling matrices onto the top of the matrix stack, objects can be moved or "instanced" in different positions in the user's coordinate system. Also, subobjects can be made a part of other objects by making the subobject's position and orientation relative to the object's position and orientation. This is done by concatenating the subobject's transformation onto that of the object before drawing the subobject. Then the subobject's transformation is popped off the stack at completion of subobject.

Devices that implement matrix stacks in hardware have a limit to the number of matrices that can be on the matrix stack at any one time.

Three-Dimensional Viewing

Before doing any three-dimensional primitives, the device must be "gopened" with the THREE_D mode.

Three-dimensional viewing is best explained with a simple camera model. This camera can be positioned in an arbitrary 3D coordinate system defined by the application program. These units can be any units that are convenient for the user's program; i.e., inches, feet, millimetres, years, dollars, etc. The portion of the graphical data which is within the camera view will be visible on the screen within the current viewport. The camera model discussed here is implemented in the file /usr/lib/starbase/demos/sb.3d.c and used in several of the demos in the directory /usr/lib/starbase/demos. The parameters which will be used on the camera routine will be position of the camera (cx,cy,cz) and position that the camera is pointed at, or the reference point (rx,ry,rz). The transformation matrix that accomplishes this camera operation is the concatenation of several matrices. First a translation to the camera position, then a rotation to rotate XYZ into the eye coordinate system, and finally a perspective transformation. The actual mechanics of the individual matrix operations can be investigated in depth by reviewing a graphics text book. The camera transformation is accomplished using starbase by first pushing the perspective matrix onto the stack(matrix operations occur in reverse order.) Since the intermediate results of the three matrix steps outlined above are of no interest, the rotation is pre-concatenated with the perspective and then replaces the perspective matrix on the top of stack. Similarly, the translation matrix is pre-concatenated with the top of the stack and the result replaces the top of the matrix stack.

Once the camera is positioned, the matrix stack only has one matrix in it, then the remainder of the matrix stack can be used for the modeling transformations needed to move objects around in the world coordinate system. For example, suppose we have a data base that describes a jet, as in the fighter demo (fighter.c). The camera can be positioned only once and the jet itself can be rotated with modeling transformations. In the fighter demo, it is not obvious whether the camera or the fighter is being rotated. If a background of trees or something similar was drawn in a fixed position in the users coordinate system, it would be more obvious that the jet was moving, not the camera.

The tractor demo (trac.c) is an even better example of the use of both viewing and modeling transformations. The camera position can be changed using the input device while the tractor itself moves down the bumpy road with modeling transformations. The tractor body moves down the road while each of the wheels is just another instance of the "wheel" database with different positions and rotations. The road remains in place the entire time. The camera is always pointed at the middle of the tractor for convenience.

Multiple Active Devices

When using more than one starbase device at a time, it is possible to establish a different viewing operation on each device since all state is maintained per device. Although this may prove useful in some cases, it is not the general usage. In general, it is advisable to set up the vdc to be equal on the two devices. That way input values can be read from the input device in vdcs and converted to world coordinates using the output device's matrix stack, using vdc_to_wc. Thus only the output device will need to maintain a matrix stack.

Since distortion on an input device is not a problem unless digitizing an object, it is probably best to always set the mapping mode to isotropic with

```
mapping_mode(fildes,ISOTROPIC);
```

for input devices. This will allow use of the entire device range.

Notes

Manual Comment Sheet Instruction

If you have any comments or questions regarding this manual, write them on the enclosed comment sheets and place them in the mail. Include page numbers with your comments wherever possible.

If there is a revision number, (found on the Printing History page), include it on the comment sheet. Also include a return address so that we can respond as soon as possible.

The sheets are designed to be folded into thirds along the dotted lines and taped closed. Do not use staples.

Thank you for your time and interest.

Manual Comment Card

If you have any comments or questions regarding this manual, write them on this comment card and place it in the mail. Include page numbers with your comments wherever possible. Enter the last date from the Printing History page on the line above your name. Also include a return address so that we can respond as soon as possible.

HP-UX Concepts and Tutorials
Vol. 6: Graphics

97089-90070 April 1985

Last Date: _____

(See the Printing History in the front of the manual)

Name: _____

Company: _____

Address: _____

_____

Phone No: _____

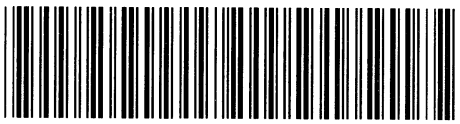
NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 37 LOVELAND, COLORADO

POSTAGE WILL BE PAID BY ADDRESSEE

Hewlett-Packard Company
Fort Collins Systems Division
Attn: Customer Documentation
3404 East Harmony Road
Fort Collins, Colorado 80525



Reorder Number
97089-90070
Printed in U.S.A. 4/85

97089-90605
Mfg. No. Only