

RIPAIN 2 USER'S GUIDE

Version 2.00
November 1, 1995

TeleGrafix Communications, Inc.

Copyright © 1992-95 TeleGrafix Communications, Inc.
All Rights Reserved

First Edition

Please don't pirate this software. We all quit good jobs to follow our dream and start this company. As with most dreams, you need the support of those around you. Pirating the software is the same as robbing us of the dream.

Would you want someone else to ruin your dream.

No part of this manual or the accompanying software may be reproduced or transmitted by any means without the express written consent of TeleGrafix Communications, Inc. Duplication or translation of any part of this manual for any use, other than personal, is a violation of United States copyright law.

TeleGrafix Communications, Inc. has made every reasonable effort to ensure the accuracy of the contents of this manual. However, TeleGrafix Communications, Inc. assumes no responsibility for the damages due to errors and omissions, and disclaims any implied warranty of merchantability or fitness for a particular purpose.

Printed in the United States of America

Trademark Information

RIPscrip, RIPaint, the RIPaint logo, RIPterm, and the TeleGrafix logo are trademarks of TeleGrafix Communications, Inc.

All other trademarks are the property of their respective owners. Their use in this guide does not constitute an attempt by TeleGrafix to claim them as their own, nor does TeleGrafix specifically endorse them.

Graphics Code Copyright 1993-95 MetaGraphics Software, Inc.
Screen Font Code Copyright 1993-95 Ancier Technology, Inc.
Digitized Audio Code Copyright 1993-95 SOS/Human Machine Interfaces, Inc.
Protected Mode Code Copyright 1991-95 Tenberry Software, Inc.
Vector Font Code Copyright 1988-95 Borland International, Inc.

Software Licensing Agreement

DISCLAIMER

USE OF THE SOFTWARE PROGRAM ON THE ENCLOSED DISKS IS SUBJECT TO THE TERMS OF THIS LICENSING AGREEMENT. YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS BEFORE INSTALLING OR USING THIS SOFTWARE. INSTALLING OR USING THIS SOFTWARE INDICATES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS. IF YOU DO NOT AGREE WITH THEM, YOU SHOULD RETURN THIS SOFTWARE WITHIN 30 DAYS OF THE ORIGINAL DATE OF PURCHASE, AND THE PRICE OF THE PRODUCT WILL BE REFUNDED TO YOU. DO NOT USE THIS SOFTWARE UNLESS YOU AGREE TO BE BOUND BY THE TERMS OF THIS LICENSING AGREEMENT

DEFINITIONS

You and Your shall be taken as referring to the person or business entity who purchased this License to use this Software or for whom such License was purchased.

Software shall be taken as referring to the files supplied on the diskette(s) inside the package, and to any and all copies, updates, modifications, functionally-equivalent derivatives, or any parts or portions thereof.

LICENSE

You may:

1. Install and use one copy of this Software on a single Computer.
2. Copy this Software into machine-readable or printed form, for backup or archival purposes in support of your use of this Software, provided any copy must contain all of the original Software's copyright and proprietary notices.
3. Transfer this Software and License to another party if the other party agrees to accept the terms and conditions of this Agreement. If the enclosed Software is an update, any transfer must include the updated and all prior versions. If you transfer the Software, you must at the same time either transfer all copies, whether in machine-readable or printed form, to the same party, or destroy any copies not transferred.

If this Software package contains both 3.5 and 5.25 diskettes, only a single Software License is created. All enclosed diskettes are covered under, and restricted by, the terms of this single Software License Agreement.

If you receive your first copy of the Software electronically, and a second copy on media, the second copy may be used for archival purposes only. This license does not grant you any right to any enhancement or update.

Title, ownership rights, and intellectual property rights in and to the Software shall remain in TeleGrafix and/or its suppliers. The Software is protected by the copyright laws of the United States and international copyright treaties. Title, ownership rights, and intellectual property rights in and to the content accessed through the Software is the property of the applicable content owner and may be protected by applicable copyright or other law. This License gives you no rights to such content.

YOU MAY NOT USE, COPY, MODIFY, TRANSLATE, REVERSE ENGINEER, DECOMPILE, DISASSEMBLE, OR TRANSFER THIS SOFTWARE, OR ANY COPY, MODIFICATION, OR MERGED PORTION, IN WHOLE OR IN PART, EXCEPT AS EXPRESSLY PROVIDED FOR IN THIS LICENSE, OR IN AMENDMENTS SIGNED BY AN OFFICER OF TELEGRAFIX COMMUNICATIONS, INC. (TeleGrafix). IF YOU TRANSFER POSSESSION OF ANY COPY OF THIS SOFTWARE, OR ANY FUNCTIONALLY-EQUIVALENT DERIVATIVE, OR ANY PORTION OR MODIFICATION THEREOF, TO ANOTHER PARTY, OR REMOVE AND COPYRIGHT OR PROPRIETARY NOTICES OR LABELS ON THE SOFTWARE OR ANY COPY, YOUR LICENSE IS AUTOMATICALLY TERMINATED.

THE EXCEPTION TO THE ABOVE CONDITION IS THAT THE BUTTONS, ICONS, PALETTES, PATTERNS, AND RIPSCRIP SCENES INCLUDED WITH THE SOFTWARE MAY BE MODIFIED AND USED FREELY IN YOUR SCENE FILES WITHOUT RESTRICTION.

TERM

This license is effective until terminated. You may terminate it at anytime by destroying all copies of the Software covered by this Agreement. It will also terminate upon conditions set forth elsewhere in this Agreement or if you fail to comply with any term or condition of this Agreement. You agree upon such termination to destroy this Software, including all copies, functionally-equivalent derivatives, and all portions and modifications thereof in any form.

LIMITED WARRANTY

THIS SOFTWARE IS PROVIDED AS IS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO , THE IMPLIED WARRANTIES OF MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE SOFTWARE IS WITH YOU. SHOULD THE SOFTWARE PROVE DEFECTIVE, YOU (NOT TELEGRAFIX) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR, OR CORRECTION.

SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

TeleGrafix does not warrant that the functions contained in this software will meet your requirements or that the operation of this Software will be uninterrupted or error-free. However, TeleGrafix does warrant the diskette on which the Software is furnished to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of delivery to you.

LIMITATIONS OF REMEDIES

TeleGrafix's entire liability and your exclusive remedy shall be:.a)The replacement of any diskette not meeting TeleGrafix Limited Warranty and which is returned to TeleGrafix , or b)If TeleGrafix is unable to deliver a replacement diskette which is free of defects in materials or workmanship, you may terminate this Agreement by returning this Software and your money will be refunded.

IN NO EVENT WILL TELEGRAFIX BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE EVEN IF TELEGRAFIX OR ITS AUTHORIZED REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY. SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

GENERAL

You may not sublicense, assign or otherwise transfer this License or Software except as expressly provided in this Agreement. Any attempt

to otherwise sublicense, assign, or transfer any of the rights, duties or obligations hereunder is expressly prohibited and will terminate this Agreement immediately. All Agreements covering this Software (including but not limited to any and all updates, upgrades, and enhancements to this Software or any portion thereof, bearing the same registration number) shall be deemed to be counterparts of one and the same License Agreement instrument. TeleGrafix retains the right to change the terms and conditions of this License Agreement at any time without prior notice.

BY INSTALLING OR USING THIS SOFTWARE, YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT, AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU FURTHER AGREE THAT IT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF THE AGREEMENT BETWEEN US, WHICH SUPERSEDES ANY PROPOSAL OR PRIOR AGREEMENT, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN US RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

U.S. GOVERNMENT RESTRICTED RIGHTS

Use, duplication or disclosure by the Government is subject to restrictions set forth in subparagraphs (a) through (d) of the Commercial Computer-Restricted Rights clause at FAR 52.227-19 when applicable, or in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, and in similar clauses in the NASA FAR Supplement. Contractor/manufacturer is

TeleGrafix Communications Inc.,
16458 Bolsa Chica Road, Huntington Beach, CA 92649.

EXPORT CONTROLS

None of the Software or underlying information or technology may be downloaded or otherwise exported or reexported (i) into (or to a national or resident of) Cuba, Iraq, Libya, Yugoslavia, North Korea, Iran, Syria or any other country to which the U.S. has embargoed goods; or (ii) to anyone on the U.S. Treasury Department's list of Specially Designated Nationals or the U.S. Commerce Department's Table of Deny Orders. By downloading or using the Software, you are agreeing to the foregoing and you are representing and warranting that you are not located in, under the control of, or a national or resident of any such country or on any such list.

TERMINATION

This license will terminate automatically if you fail to comply with the limitations described above. On termination, you must destroy all copies of the Software.

MISCELLANEOUS

This Agreement represents the complete agreement concerning this license between the parties and supersedes all prior agreements and representations between them. It may be amended only by a writing executed by both parties. If any provision of this Agreement is held to be unenforceable for any reason, such provision shall be reformed only to the extent necessary to make it enforceable. This Agreement shall be governed by and construed under the laws of the State of California and the United States of America. Venue for litigation concerning this Agreement shall be the courts serving Orange County, California. The application of the United Nations Convention of Contracts for the International Sale of Goods is expressly excluded.

Table of Contents

Chapter one: Introduction	1-2
How does RIPscrip work	1-3
Differences from 1.54 and 2.0 formats	1-4
Backward Compatibility topics	1-4
Chapter Two: Installation	2-2
Chapter Three: Tutorial	3-2
Chapter Four: Buttons and Mouse field s	4-2
Chapter Five: Drawing tools	5-1
Arc/Wedge	5-2
Article	5-4
BMPs/Icons	5-5
Button	5-8
Columns	5-10
Closed Curve	5-11
Closed Polyline	5-15
Edit mode	5-18
Eye Dropper	5-20
Line	5-22
Mouse Field	5-24
Oval/Circle	5-26
Photo/JPEG	5-29
Point	5-32
Polygon	5-34
Polyline	5-37
Rectangle/Square	5-40
Round Rectangle	5-43
Sound files	5-46
Snapshot	5-48
Text	5-52
Text Window	5-56
Chapter six Editing Tools	6-1
Button Designer	6-2
Fill Style Editor	6-5

Table of Contents

Font Selector	6-7
Grid Snap Editor	6-10
Icon/BMP Editor	6-12
Line Style Editor	6-15
Mouse limits	6-17
 Chapter Seven: Using Text Variables	 7-1
 Chapter Eight: Misc. Host commands	 8-1
 Chapter Nine: Using Templates	 9-1
 Chapter Ten: Advanced Templates	 10-1
 Chapter Eleven: Host commands, what can go where	 11-1
 Chapter Twelve: Coordinate Systems in RIPscrip	 12-1
 Chapter Thirteen: Graphical Viewports	 13-1
 Chapter Fourteen: Color under RIPscrip	 14-1
 Chapter Fifteen: Data Tables in RIPscrip	 15-1
 Chapter Sixteen: Data Save Areas	 16-1
 Appendix A: Text Variable Reference	 A-1
\$ADOW\$	A-5
\$ALARM\$	A-5
\$AMPM\$	A-5
\$APP\$	A-6
\$APPx\$	A-6
\$ATW\$	A-7
\$AVP\$	A-7
\$BACKSTAT\$	A-8
\$BASEMATH\$	A-9
\$BAUDEMUL\$	A-10
\$BEEP\$	A-11
\$BLIP\$	A-12
\$CLS\$	A-12
\$COFF\$	A-13

Table of Contents

\$COLORMODE\$	A-13
\$COLORS\$	A-15
\$COMPAT\$	A-15
\$CON\$	A-16
\$COORDSIZE\$	A-17
\$COPY\$	A-17
\$CUR\$	A-22
\$CURSOR\$	A-24
\$CURX\$	A-24
\$CURY\$	A-25
\$D\$	A-25
\$DATE\$	A-26
\$DATETIME\$	A-26
\$DAY\$	A-26
\$DOW\$	A-27
\$DOY\$	A-27
\$DTW\$	A-27
\$DVP\$	A-28
\$DWAYOFF\$	A-29
\$DWAYON\$	A-30
\$EGW\$	A-30
\$ETW\$	A-31
\$FIELDID\$	A-32
\$FILEDEL\$	A-32
\$FYEAR\$	A-33
\$HKEYOFF\$	A-33
\$HKEYON\$	A-34
\$HOUR\$	A-34
\$IFS\$	A-35
\$IMGSTYLE\$	A-37
\$INUSE\$	A-40
\$ISEXTWIN\$	A-40
\$ISPALETTE\$	A-41
\$ISPROT\$	A-42
\$M\$	A-42
\$MCURSOR\$	A-43
\$MHOUR\$	A-43
\$MIN\$	A-44
\$MKILL\$	A-44
\$MONTH\$	A-45
\$MONTHNUM\$	A-45
\$MSTAT\$	A-45

Table of Contents

\$MTW\$	A-46
\$MUSIC\$	A-47
\$MVP\$	A-47
\$NOREFRESH\$	A-48
\$NULL\$	A-48
\$OFFSCREEN\$	A-48
\$OPTION\$	A-49
\$PAENTRY\$	A-51
\$PCB\$	A-53
\$PHASER\$	A-54
\$PORTH\$	A-55
\$PORTW\$	A-55
\$PORTX0\$	A-57
\$PORTX1\$	A-59
\$PORTY0\$	A-60
\$PORTY1\$	A-61
\$PROT\$	A-61
\$RBS\$	A-62
\$RCB\$	A-62
\$RCP\$	A-63
\$RENV\$	A-64
\$REFRESH\$	A-64
\$RESET\$	A-65
\$RESTORE\$	A-72
\$RESTOREX\$	A-73
\$RESTOREALL\$	A-74
\$RESX\$	A-75
\$RESY\$	A-75
\$REVPHASER\$	A-75
\$RGS\$	A-76
\$RIPVER\$	A-77
\$RMF\$	A-77
\$RTW\$	A-78
\$SAVE\$	A-79
\$SAVEx\$	A-80
\$SAVEALL\$	A-81
\$SBAROFF\$	A-82
\$SBARON\$	A-82
\$SBS\$	A-82
\$SCB\$	A-83
\$SCP\$	A-84
\$SEC\$	A-84

CHAPTER 1

Introduction

RIPaint 2

TeleGrafix Communications, Inc.

Last updated on 10/20/95

INTRODUCTION TO RIP*scrip* GRAPHICS

RIPscrip graphics is an innovative way of using online services graphically. It is used heavily in the online bulletin board community and increasingly in the UNIX/Internet world as a method of standardizing online services.

INTRODUCTION

As system operators of many bulletin board systems, we've often wished for some form of Graphical User Interface for our boards. Like most Sysops, we've come across many solutions. But they all seemed to fall short in one way or another: inadequate for THIS system, incomplete, difficult to implement, too complex, or lacking in graphics development tools. In short, we became frustrated.

So, we decided to write our own graphical script language.

RIP*scrip* stands for "Remote Imaging Protocol script" language. This graphical language is our answer to the graphics needs of the BBS community and has serious tools for implementation and practical use.

For more information on RIPaint, RIPdraw, RIPTerm or RIP *scrip* development tools (RIP2C, RIP2PAS, etc.), contact:

TeleGrafix Communications, Inc.
16458 Bolsa Chica #15
Huntington Beach, CA 92649

VOICE (714) 379-2131
FAX (714) 379-2132
BBS (714) 379-2133

Internet: rip.support@telegrafix.com

DEFINITION

RIP*scrip* is a text based Script language for displaying online graphics. The script language conforms to 7-bit ASCII, avoiding the use of Extended ASCII characters. This allows transmission over X.25 networks and other carriers that do not support full 8-bit binary transfers easily. RIP*scrip* allows RIP*scrip* graphical statements to be mixed with printable ASCII text and [de facto standard] ANSI/VT-100 directives. RIP*scrip* can dynamically determine what is graphics and what is text and display them appropriately in separate windows (a graphics window and a text window). And if you must have your own proprietary commands, RIP*scrip* has room for that too.

HOW DOES RIP*scrip* WORK?

RIP*scrip* uses a flexible, and very efficient script language for its graphical statements. Its efficiency stems from its compactness and developmental planning. It is entirely Object Oriented instead of Raster Oriented for efficient transmission of data and powerful editing capabilities (using RIPaint or RIPdraw for example). The language is open ended enough so that literally billions of different graphics commands can be implemented as needed. RIP*scrip* is not a proprietary protocol standard, otherwise you wouldn't be reading this document. It is open to suggestion from the rest of the world.

Earlier Graphical Script Languages (Avatar and Skypix among others), utilize special command characters to indicate which graphics command is to be executed. This allowed for their use on systems that are limited to ASCII printable text. Traditional script languages use English words to accomplish things (e.g., "BOX 0,0 100,50"). This kind of thing is incredibly bulky, especially when you consider that pictures are usually not simple things, but comprised of hundreds or thousands of individual graphical operations (e.g., lines, circles, curves, text, etc.). With this in mind, a human-readable script language was completely inappropriate for the relatively limited bandwidth of conventional modems.

So, one of our main strategies for this language was to make it as efficient as possible without going completely binary. This allows the immediate installation of the protocol onto any ASCII text-based host system -- because the language consists entirely of ASCII printable

characters. We justify the unreadability of the language by pointing out the limitations of today's modems and phone lines -- the language must be compact.

DIFFERENCES FROM 1.54 AND 2.00 FORMATS

In previous versions of *RIPscrip*, we used a simple Icon file format for its disk based bitmap icons. In 2.00 we introduce a newer, more superior file format (the BMP/DIB file format). This is a device independent bitmap format that accommodates monochrome images, 16 color images, 256 color images and 24 bit images. There is no real form of compression internally but the capability for compression exists for future expandability. The reason for changing file formats was a very important one because the older format did not have the built-in facilities for resolution and color palette independence. This new format does. We chose the BMP file format because it is truly a device independent bitmap format. For details about the actual file format specification, see the end of this document.

Any place in the *RIPscrip* specification that used to refer to .ICN files now will use .BMP files. The older .ICN file format will no longer be supported in 2.0 and later revisions.

BACKWARD COMPATIBILITY TOPICS

With this release, XOR write modes now apply to all graphical primitives (also fill patterns). Overall this shouldn't pose any problems for 99% of all *RIPscrip* 1.54 files out there, but there might be a few that are affected - if in *RIPaint*, you enable XOR write mode and do a couple lines and rectangles, then do a circle or filled rectangle, the circle or filled rectangle will be drawn in COPY mode even though XOR mode is active. In a 2.0 terminal, this would cause the circles and filled in areas to be XOR'ed as well. In this regard, 2.0 is not 100.0% fully backward compatible.

Also with this release, the world coordinate/resolution independent aspect of the language handles filled-in areas a bit differently (see the world coordinate system section below). Filled-rectangles don't fill all the way to the right or bottom borders of the rectangle - they are inset by one pixel from the right and up one line from the bottom. The reasons for this are described in the world coordinate section of this document.

This can cause one pixel gaps in 1.54 files that wouldn't happen in 2.0 related files.

We have introduced a system of graphical ports, or drawing ports. These drawing ports are areas where actual drawing takes place. In 1.54 and prior versions, any graphics that are drawn (e.g., lines, circles, etc.) are drawn to the screen (the screen port). Under 2.0 you can define multiple drawing ports which may or may not be on the screen (e.g., screen ports, or offscreen "clipboard" ports). Each drawing port can have one viewport associated with it, allowing you to modify the sub-region inside the port that drawn objects will be "clipped to" (a clipping rectangle). The screen is always port #0 and is formally known as the default screen port. Under 1.54 you were allowed to have an offscreen bitmap area known as the clipboard. Under 2.0, the clipboard is simply an offscreen "clipboard" port. You may switch to any defined port and subsequent drawing operations will apply to that port instead of the screen (unless the new port is a screen port).

The older *RIPscrip* commands that deal with the clipboard are in essence, obsolete. When they are instructed to work with the clipboard, they work with the first offscreen "clipboard" port that they find. Some newer commands have been included, which are more powerful, that allow you to specifically designate a given port number to work with. You are allowed up to 36 separate graphical drawing ports, which port #0 is the screen and cannot be deleted.

CHAPTER 2

RIPaint 2 Installation

RIPaint 2

TeleGrafix Communications, Inc.

Last updated on 10/20/95

RIPaint 2 Installation

RIPaint must be installed on your hard drive. It must be installed from a 3 1/2, 1.44 floppy.

To install RIPaint :

- 1.) Insert RIPaint disk one into your floppy disk drive
- 2.) If you put it in your A: drive type A:Install If you put it in your B: drive type B:Install
- 3.) You will be asked the following questions.
 - A.) Do you wish to install RIPaint? Press Enter or Y to continue .
 - B.) What drive do you wish to put RIPaint on? This is your destination drive. Press the Enter key to select drive C:. Use the arrow keys to select alternate drives. Once you select the drive hit Enter.
 - C.) What subdirectory do you wish to install RIPaint in? The default subdirectory is \RIPaint. If you already have RIPaint 1.54 you may want to select a different subdirectory to avoid overwriting RIPaint 1.54. To select the default subdirectory hit the Enter key. To select a alternate subdirectory type the path to that directory. Do not include the drive letter as it is already added.
Example: GRAPHICS\BBS\RIPAINT
 - D.) Confirming your path\subdirectory choice. Please read your path statement and select the Enter key if correct or press N or use arrow keys to select No.

- E.) Please enter your name. Type the name to whom the product belongs.
- F.) Please enter your serial number. Your serial number may be found in three places on your software. 1.) On the registration card on the upper right hand corner. 2.) On disk one. 3.) On the outside of the licensing envelope that your disks came in. Type in your serial number. Be careful to use the spaces or the program will not accept the serial number as valid. Also be aware that the O, 0, and Q can look alike.
- G.) Confirming your name and serial number. Please proof read both of these before you hit the enter key to confirm or use the arrows keys or the N key to say no. If you make a mistake enter no and retype your information. If you read the serial number off disk one, be sure to replace it in the floppy drive before confirming the information.
- H.) At this time the program will install. It will ask you to insert disk two & disk three to continue the installation.

Once *RIPaint* is installed go to the C:\RIPaint directory or the directory that you chose and type *RIPaint*. It will ask you the following questions.

- 1.) Video & Mouse set up.
Use the auto-sensing for the video & mouse drivers or use the arrow key to select the video card best suited to your system. The VESA super VGA 640 X 480 is the common choice. Then use the tab key to change to the Mouse Driver selections. Look carefully at the mouse selections. Note that some drivers specify the COM ports that the mouse is on. If you do not know what COM port your mouse is on you can look it up in MSD. (MSD is MicroSoft Diagnostics and will be in computers with DOS 5.0 or better.) MSD can be reached by going to your C: and typing MSD. (Some people may need to go to their C:\DOS prompt before typing MSD.) MSD will give you a screen

with gray boxes. The Mouse box is on the left side, second from the bottom. Click on this. The sixth line down will list Mouse IRQ. As most computers have the mouse on COM port one the standard IRQ is 4. If your IRQ is 3 your mouse is normally on COM Port 2 or 4. Some drivers like the PS2 use IRQ 12 and are normally found on COM one. Use the Enter key only after having made both selections. If you make a mistake you can exit back to your C:\RIPAIN directory and delete the RIPAIN.CNF file to start the setup again.

2:) What audio card do you have?

Use the auto-sensing mode to detect your sound card and it's settings. RIPaint has all the popular sound card and several uncommon ones if you want to select you sound card manually use you arrow keys to find the right one and hit enter to select.

3:) What printer are you using?

Use your row keys or your mouse to select the driver for your printer or hit enter if you have no printer. At this time do not set the advanced settings. You can go back in at any time and make changes by accessing FILE/PRINTER SET UP in the RIPaint program.

Once this information is processed you will be in the drawing screen for RIPaint.

CHAPTER 3

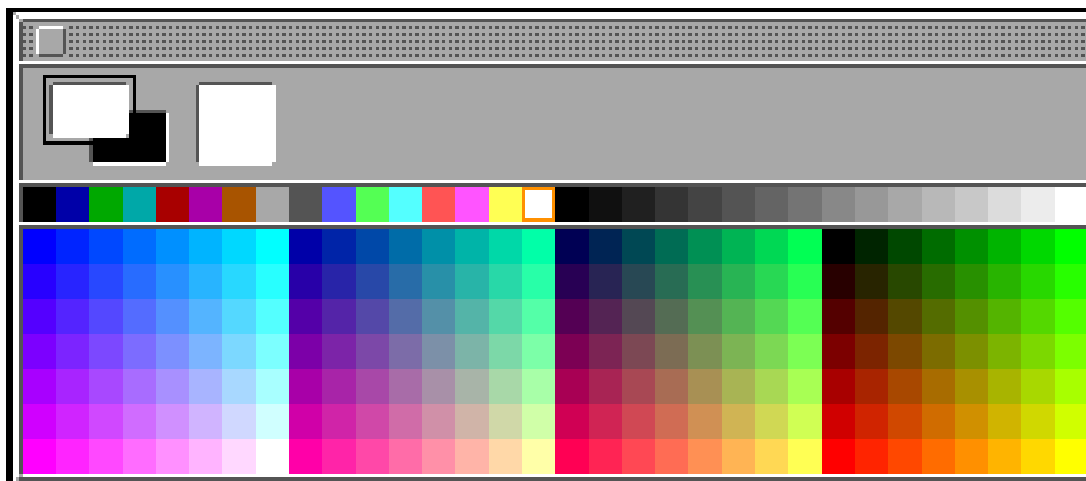
Tutorial

RIPaint 2

TeleGrafix Communications, Inc.

Last updated on 10/20/95

Tutorial



This is the Color Picker. Do you see the little grey square in the corner? That is what you use to make the Color Picker go away. Now move the mouse cursor until it is over the center of the dark grey bar at the top of the Color Picker. Make sure the little grey square isn't under the cursor. Now press down and hold the left mouse button. Move the mouse around. See how the Color Picker is following the cursor around? When the Color Picker is where you want it release the left mouse button. What you just did is called dragging an object.

Dragging an item - Move the mouse until the mouse cursor is over the item and press and HOLD the left mouse button. Now move the mouse cursor until the item is where you want it to be. Now release the left mouse button. You drag an item to move it to a new location. Some items can only be picked up on certain sections of them. For example, the Color Picker and Toolbox can only be picked up in the middle of the dark grey bar on the top of them.

Notice how the Color Picker has three rectangles on the top part of it. Do you see any where on the status bar that has the same three rectangles? Right, There in the middle. Notice that the colors of the rectangles are the same as they are on the Color Picker. When the Color Picker isn't on the screen you can look down at the Status bar and see what the colors are set to. Remember we mentioned the small grey square in the upper left hand corner of the Color Picker? Now, I want you to move the mouse pointer over the grey square and press and release the left mouse button.. Notice the Color Picker disappeared.

Clicking on an item - Move the mouse until the mouse cursor is over the item and press and release the left mouse button. You click on items to select them.

In order to bring the Color Picker back, click the mouse on the area of the three rectangles on the status bar. The first rectangle on the far left shows what the Drawing color is currently. The Drawing color is used to draw lines, rectangles, and the borders of filled objects. The one slightly behind it is the Background color. Some commands use a second color in the background of the line. The rectangle on the far right shows the current Fill color. This is the color that filled objects are filled with. There are many colored squares below these rectangles representing the possible colors available. To change a color simply move the cursor over the rectangle you want to change and press the left mouse button.

Notice that after you clicked on the rectangle that there is now a black border around the it. Now click on any of the small squares below the rectangles. Notice how the rectangle changed to the color you just clicked on? Congratulations you just changed your first color.

Try some of the following exercises:

Change the drawing color to a shade of green.
Change the background color to a shade of blue.
Get rid of the Color Picker and then get it back.

Now that we know how to use the Color Picker, let's learn about the Toolbox. Here is what it looks like.



Icons - Little pictures that represent something else. For example, the letter T on the Toolbox represents “Text mode”, a mode where you can

Tutorial

draw text on the screen. Icons can also represent choices that you can select by clicking on it..

Try clicking on some of the items on the Toolbox. Notice how the one that you click on looks darker than the others? That tells you that is the tool you are currently using. Also look at the status bar, do you see the same tool down there? The status bar also shows what tool you are currently using. As you can see there are a lot of different tools that you can use. I'll be showing you how to use them a little later. Just like the Color Picker, you can remove the Toolbox from the screen. Try getting rid of the Toolbox by clicking on the little grey box in the upper left hand corner of the Toolbox like you did with the Color Picker. Now how do you think you would get the Toolbox to reappear? If you guessed clicking on the status bar where it shows what tool you are using, then you're doing great. Another way to make the Toolbox reappear is to hit the TAB key. Try removing the Toolbox and making it reappear with the TAB key.

On the bottom of the screen there should be a row symbols and text. This is called the Status Bar. It shows you what the current status of some things are as well as allow you to change some of them. On the far left there should be an row of text. This is the name of the font that RlPaint is currently using. In parenthesis is the current size of the font. If you click on this rectangle, a dialog box will pop up and allow you to select a new font and/or font size. Next there are four squares with letters in them. These squares stand for Bold, Italic, Underline and Strikeout. If they are active they are a darker color. If you click you mouse on one of them you'll notice that the change from a light background to a dark background. For example, if the B has a dark background, any text you put on the screen will drawn in boldface.

The next square shows the current fill pattern. This is the pattern that is used when you draw filled objects. If you click on it, the Fill Pattern Editor will be brought up. To the right is an icon that shows the current Line pattern and width. When this icon is clicked, the Line pattern editor is activated. Next is a long rectangle with three colored rectangles in it. These show the current colors for RlPaint. From left to right, they are Drawing Color, Background Color, and Fill Color. If you click in this rectangle the Color Picker will be displayed, if it isn't already.

Next is the icon that shows the active drawing tool. If this is clicked on, the Toolbox will be displayed, if it isn't already. The next icon has the word Undo on it in blue. If you click on this icon, RIPaint will remove the last drawing command from your scene. You can click on this as many times as you need to remove drawing commands.

If you look to the right, you'll see an icon with a single arrow pointing to the right. If you click on this, RIPaint will ask you what speed you would like to play the file back at, and then display it at that speed. The double arrow next to it will replay the scene at full speed. Next is what looks like a raised button that is half grey and half blue. If you click on this, RIPaint will bring up the Button Style Editor.

The Grid to the right will activate the Grid Snap editor. The next icon will put you in the object editor where you can edit individual files. Next we see the icon that if clicked will bring up a dialog allowing you to change your settings. The question mark when clicked gives you a help screen on the current drawing mode. On the far right side, is a rectangle that shows the current position of the mouse on the screen. Clicking on it will switch between showing the position in World Coordinates, Video coordinates (has a blue border), and no coordinate display.

Now let's open a file. I need you to hold down the ALT key and then press the L key and then release both. Now you can see an example of a dialog box. In the upper left is an Edit box, you can type in the name of a file you want to load here. Click in the rectangle to the right of the text File name. You should now have a cursor in there. Now type the following: BEGIN.RIP and hit ENTER. Now you should hear your disk drive spin and some graphics should display on the screen.

Another way to load a file is to use the menuing system. To do that click on the right mouse button. A menu bar will appear on the top of the screen. Click the left mouse button on the word file on the menubar. Now click on the word open. This also brings up the dialog to load a file. Try loading one of the RIP files that came with RIPaint 2

This is a sample of what you can do with RIPaint 2.0. You'll notice there are pictures, bar and pie graphs. Now I'm going to have you click the mouse on the status bar again. Click on the two triangles facing to the right. The screen just redrew itself. If you ever want to redisplay the screen just click on that icon. Now I'll show you a neat little trick. Since

RIPscrip graphics are made to be sent over any type of connection, you might want to see how the screen will look to your users at different transmission speeds. If you click on the single right point triangle, you will get a dialog with a list of choices for speed. This kind of dialog is called a pick list, because it gives you a list of options and you click on one of them to continue. If you click on the word Exit, you cancel your request to redraw the file.

Baud - Compu-speak for speed of transmission. In most cases, all you need to know is that the higher the number the faster it can send a file. However, just to keep you on your toes, they often refer to the higher speeds with a decimal point. For example, the speed 14,400 baud is often known as 14.4 kilobaud, you'll also see 28.8, and 56.7. These numbers have been divided by 1000 to keep the number simpler. The word kilobaud, which is often left out, tells you that you need to multiply it by 1000 to compare it to a normal number. To keep this all straight, remember the larger the number the faster the transmission, and that numbers with a decimal point need to be multiplied by 1000 to give you the true speed.

Now I want you to click on the speed listed as 14,400 baud. This was a very common speed at the time this manual was written. The screen will redraw after you clicked on 14,400, but notice that the speed is a little slower than it was the first time. The first time, since all the data was here, we did not have to wait for the RIPscrip file to transfer. The second time we asked RIPaint to act like it was being transmitted to it at that speed. Now try redrawing the screen at 2400 baud, another popular modem speed. It was a LOT slower that time, in fact 14,400 baud transmits data over 4 times faster than 2400 baud. You may want to keep this in mind if a lot of your users are using older slower 2400 baud modems. Some Host operators, just don't let people log on at speeds below 14.4 kilobaud, others just simply create their displays so that the transmission won't take as long. No matter which you chose, you can tell what it will look like on your users end with the single arrow icon.

Sometime you make a mistake and you need to erase it and do it over. If you made the error on the last drawing command you entered, there is a short cut to erase it. This command is called Undo. If you look at the status bar you'll notice that towards the middle is an icon with the word

Undo in blue text on it. If you click on the word Undo, it will remove the last command from the RIPscrip file.

Warning:

When you Undo a command all information about it is lost. If you want to put the command back you need to create it from scratch like you did it in the first place.

Each time you click on Undo it will remove another command until there are no more commands to remove. You have what is called unlimited Undoes. It's similar to knitting where you can unravel the last knot if you made a mistake, or more then one if the mistake is a few stitches back. Sometimes it won't look like the screen changes at all when you click on Undo, because not all commands affect the screen directly. For example, a change drawing color command just sets the color. It doesn't actually draw anything on the screen.

You can save a file by holding down the ALT key and hitting the S key. A save file dialog will appear, click on the rectangle next to the name Filename. Now type in MY.RIP and click on OK. You just saved this file to the file MY.RIP. You can try loading it back in if you like.

Next, we need to get rid of the file we have in memory to get ready for the next chapter. To do this type ALT-N. A dialog box will pop up asking if you want to save the changes made to this file first. If you click on cancel, then RIPaint will know you changed your mind about starting a new file. If you click on YES, RIPaint will save the current file and then clear the buffer and set everything up for a blank file. NO simply clears the buffer and sets up with a blank file.

Warning:

If you exit RIPaint or start a new file and answer NO to the dialog asking if you want to save the changes, you will lose any changes made to the file since the last save.

Now we have a blank screen, and can start drawing. Bring up the Toolbox if it's not visible. Now I want you to click on the icon that looks like a line going from the lower left to the upper right. You just told RIPaint to use the Line tool. Now select a color to draw with.

Remember the rectangle on the left of the Color Picker tells you what color RIPaint will draw with. If you don't remember, chapter 1 will refresh your memory. Okay we have the line tool and a color picked out. Let's draw a line.

Move the mouse cursor away from the Color Picker and Toolbox. Think about a line you want to draw. Move the cursor to the beginning point of the line you are thinking of. When you are there, press down the left mouse button and hold it. See how there is a pink line between the beginning point and the mouse cursor? That is called a rubber band line, and it shows you what the line would look like if you were to release the mouse at that point. Now move the cursor to where the line should end and release the left mouse button. You might have noticed that that what you just did was very similar to the way that you dragged the Toolbox and Color Picker around. This is also a dragging operation.

Now try doing the following:

Remember that you can Undo the last command with the Undo icon on the status bar.

- Draw a simple house in grey with 5 lines.
If you make a mistake just click on Undo.
- Add a blue outline of a window with 4 lines.

Did you have some problems making the lines straight on the house? If so you're not alone. While it may not be that hard it is a pain in the neck to do. For example, to make a horizontal line, hold down the shift key, move to the beginning of the line press and hold the left mouse button and immediately move the mouse to the right or the left. The mouse pointer will not be connected to the rubber band line this time because RIPaint is forcing it to be a horizontal straight line no matter where the mouse is. Release the left mouse button, when the pink rubber band line is where you want it. RIPaint will then erase the pink line and draw the line in the current drawing color. You can release the SHIFT key after the final line is drawn. To make a vertical line, move up or down instead of left or right.

Now try drawing that house again this time using the SHIFT key to keep the lines straight.

- Draw a simple house outline with 5 lines in red.
- Add a green door with 4 lines

- Add a blue window with 4 lines

Secrets of the RIPmasters

You can tell RIPaint to force a line to be horizontal or vertical by pressing and holding the SHIFT key at any time before you release the left mouse button. If you change your mind, you can release the shift key at any time and it will change back to a normal line.

You can draw a line or any other object over the top of the Toolbox, as long as the line or object is started away from the Toolbox. Just create your object as normal. Releasing the left mouse button either on the Toolbox or on the other side of it. The Toolbox will flash as the object is drawn but quickly reappears. The same is true of the Color Picker.

Do you see that horizontal line on the status bar, try clicking on it. A dialog box popped up with different lines on it. Click on one of the lines on the right hand side. Did you notice how that changed the width of the lines in the example box on the right? Now select OK and try drawing a line. Notice the difference in how thick the line is? Look at that horizontal line on the status bar, see how it's thicker? That shows you what your current line pattern and style look like. Click on it again. This dialog that just popped up is called the Line pattern editor. Okay, try clicking on the button marked Dashed. Can you see the changes that made in the examples box? Notice how the top objects are now dashed? They obey both the line pattern and the line thickness. The objects on the bottom only obey the line thickness.

Now click on OK and make another line. Notice how it is a dashed line now. If you look at the status bar you'll notice that the horizontal line is dashed down there as well. Click on it again, this time click on the arrows on the right side of the rectangle that is labeled Size. Click on the up arrow until the number says 10. Now click on okay and draw another line. See how much thicker the line is. Now change the line style back to Solid and a thickness of 5. You can set your own custom line patterns. For more information, lookup Line pattern editor in the dialog reference.

Now I'll show you something new. do you see the icon on the Toolbox that looks like a dot? Click on it. You now are using the Point tool. now click on somewhere on the screen. See the nice large point it made?

Okay, now change the line thickness to 10 again, and draw another point. See how the point is affected by the line thickness? The Point tool uses the line thickness setting to tell it how large to make the point.

Sometimes you'll want to draw just a single pixel. To do that hold down the CTRL key when you click on the location where you want the pixel to be. If you want to place a whole bunch of points, in a freehand drawing mode, hold down the SHIFT key and the left mouse button and move the mouse around.

Secrets of the RIPmasters

Avoid using the SHIFT key with the Point tool too much. It will make your file a lot larger. Each point requires many characters to be transmitted. It is included for those times when you need this ability. If covering a large area it is always much better to create a custom fill pattern or two and to fill the areas of the screen where it's needed.

Now try this:

Start a new file and draw a stick man with a large point for the head and two black pixels for his eyes.

Try giving the stick man some hair using the SHIFT key.

Now let's try another tool, click on the empty square on the Toolbox. Now, move to a point you want to put a rectangle and press down and hold the left mouse button. Now move the mouse around some. The pink rectangle that you are moving around shows where the rectangle would be if you released the left mouse button right then. When you have the rectangle you want, release the left mouse button and the rectangle will be drawn in the current drawing color. Try changing the drawing color and make another rectangle. What happens if you change the line style and width. Notice, the rectangle's sides obey both the line style and the line width.

There are two keys that change how you place a rectangle. The first, SHIFT forces the rectangle to be a square. Try it. Notice what happens if you press and release the SHIFT key while you are placing a rectangle. It switches back and forth between being a square and a rectangle. The other key, CTRL when pressed centers the rectangle or square from the first point you selected. This will allow you to nestle

rectangles together easily, by using the CTRL key and always starting from the same initial point.

Now that we understand that command let's try using the icon of a filled square with a border. Try making a square in the center of the screen. See how the center is filled with the current fill color? Try changing the fill color from the Color Picker. Remember the fill color is the rectangle on the right hand side of the Color Picker. It may be that the rectangle you just drew wasn't filled with a solid color, in that case, a fill pattern was set. Look at the icon at the left of the line style on the status bar. Notice how it has the pattern and color as the square you just drew. Try clicking on it. You'll notice another dialog box just popped up. This is called the Fill Pattern Editor. Click on one of the patterns, other than the one marked custom. Now click on OK, Now draw another rectangle. Notice how it is filled with the new fill pattern? You can create you own custom fill pattern. For more information, look up Fill Pattern editor in the dialog reference.

Now try using the tool with the icon of a filled square with no border. Try drawing a rectangle from one corner of the screen to the other. Notice how this time the rectangle didn't have a border drawn around it? The three different tools allow you quickly and easily pick what kind of Rectangle tool you want to use. You simply have to select the one that has the features you want. You'll notice there are three varieties of many of the tools. In these cases, all three are the same object type and only differ in whether they are unfilled, filled with a border, or just plain filled.

Now you'll notice there's another icon that looks like a square but has rounded corners. That is called a rounded rectangle. First select an area on the screen just like you did with the normal rectangle. After you release the left mouse button, you'll notice that it drew a rectangle with rounded corners. In each corner you'll see a circle, and in the middle of the circle will be a dot. Move over anyone of the dots and press and hold the left mouse button. Now move the mouse around. See how the size of the circles are changing and so are the corners of the rectangle. When the corners of the rectangles look right, release the left mouse button. Now the rounded rectangle is drawn using the current colors. You can use the SHIFT and ALT keys when placing the rounded rectangle just like you did with normal rectangles. For example, if you hold down the CTRL it will be forced to be a square.

Now that you know how to place a rounded rectangle, try to make a filled rounded square with a border. You need to select the rounded rectangle icon that shows both the border and the center as being filled.

Remember to hold down the CTRL key to force the rectangle to form a square. You can make a plain filled rounded rectangle by simply selecting the bottom rounded rectangle icon.

Now that you know how to work with rectangles we can do some interesting stuff. I'm going to show you how to load a BMP. I need you to hold down both the CTRL and the ALT keys and hit the B key. This will bring up a dialog asking you which BMP you'd like to load. Click on the word `ALCHEMY.BMP`. Now you are put back onto the editing screen with a crosshair for a cursor. Move the cursor to the upper left hand corner of the area where you want to display this BMP. Press and hold down the left mouse button. Pretend that you are dragging a rectangle where the BMP will be displayed. When the rectangle with the X in it is correct, release the left mouse button. Now a dialog will appear. For now, just select OK. Now there is a pretty icon with a bunch of scientific equipment where you chose to put the BMP.

Now when you are using BMP's you need to make sure they are on your user's hard drive, either in their `ICONS` directory, or in the system directory for your BBS. Many 3rd party manufacturers make doors and programs that will take care of updating your BMP's for you. Some host software will do it for you automatically if you have it turned on. If you don't put the BMP's you use in your user's hard drive, they will see a red dashed rectangle instead of the BMP. Try putting `ALCHEMY.BMP` in different places and at different sizes. See how easy that is? Type CTRL-ALT-B again. Select `BLAST.BMP` this time and select a place for it on the screen. Now let's look a little closer at the settings on that dialog box.

If you think that you may have users using a video mode with fewer colors than you have, set Dithering on and activate palette off. If they have enough colors, no dithering will be done.

If you want your BMP to look exactly the way you made it, set dithering off and activate palette on. This will change the current video palette to match the palette of the BMP. Note: Objects on the screen may change colors because the palette entries have been changed. If you want to do

this, it is best to load this BMP first and then draw any other objects you need.

If you want to experiment with these settings, try changing your video modes and see what happens with the different settings. I'll leave that to you for extra credit.

One of the nicest features of BMP's is that they can have sections that are transparent. In this case, blast.bmp is a BMP you can use with transparency. Set transparency on. And set the transparency color to black.

Now click on OK. You may not notice a difference if the background of the screen is black right now. Try displaying blast over a filled rectangle with the transparency set on. You'll see that the color of the rectangle shows through in places around and in the BLAST.BMP. Remember all BMP's are rectangular. The reason that this looks irregular is that many of the pixels are set to black, which we told RlPaint we wanted to be transparent. If none of the pixels in a BMP match the transparency color then it will display normally.

What happens is that when the BMP is displayed with transparency on, is that any pixel that is the same color as transparency color will not be displayed. In this cases the transparency color is black. Transparency allows you to have irregularly shaped objects, and ones that seem to have windows in them.

Try doing this:

Draw a green filled rectangle with no border that covers the bottom two thirds of the screen. Draw a blue filled rectangle with no border that covers the upper third of the screen. Make a house and practice bombing it with the BLAST.BMP.

Are we having fun? I hope so. Now let's get really tricky and do some animation. We need to be able to place things precisely to do animation.. Click on the grey grid of lines on the status bar. You should see a bunch of criss crossing lines on the screen now. These lines aren't actually a part of your scene, they just help you positions things. In fact, you can only click your mouse on the intersections of the crossed lines. If you click the mouse anywhere else, the closest intersection is selected instead. Try making a filled rectangle now. See

how the lines limit you? To turn off the grid snap click on the grey crossed lines on the status bar again.

Now for the madness and mayhem. Okay turn the grid snap back on, by clicking on the grey crossed lines on the status bar. OK, now make a green filled rectangle that covers most of the screen. Now, load the BLAST.BMP and select a place to display it. Remember where you selected. I'm going to ask you to put the BMP exactly there a little later. Now when the dialog pops up, don't select transparency this time. Instead click on the arrow next to drawing mode and click on XOR. Now select okay.

Not too exciting you say. It looks like it did before. Well the magic happens now. Now load the BLAST.BMP again and display it EXACTLY where you put it the first time. Use the same settings, no transparency and drawing mode XOR. When you select OK. The image of BLAST.BMP will disappear! You see XOR is a drawing mode that if you draw something in the same exact place twice. it will disappear without destroying what ever was on the screen before it was first displayed. When you XOR an image, all black pixels will be transparent, just as if you had transparency set on. This has to do with the way it is displayed on the screen. You will notice that the colors in some icons will change, if placed over a colored filled area. This happens because of the way the image is displayed the only thing you can do is change the colors so that when displayed on that solid color they look right.

Now that you know how to do that we can start to play with animation. Clear out this scene and begin a new one, by typing ALT-N. Okay, now click on the grid snap icon on the status bar. Remember, that the one with all the grey crossed lines. Now I want you to load the BMP 3&HALF.BMP and put it in the upper left hand corner. Make it 3 squares wide with the grid snap and 3 squares tall. Draw it using XOR drawing mode. Now is the trick, we use a special text variable to tell RIPaint that we want the terminal to put in a delay. You do that by holding down the ALT key and hitting the 0 (Zero). A dialog will pop up. In the text area, enter the following \$D(60)\$, and then click on OK. You just told RIPaint to wait 1 second before processing the next command. Now go back and draw the 3&HALF.BMP on top of the first one with XOR mode. Now we want to move the BMP. Okay now draw the 3&HALF.BMP starting down one square to the right. Remember to use XOR drawing mode. Now we want to put in another delay of 1 second. To do that hit

ALT-0 again and type in \$D(60)\$. Now erase the image of 3&HALF.BMP by drawing over it at the exact same location with XOR. Draw the BMP again starting one square to the right. Put in another delay of 1 second. Do this several more times. When you think you have enough, click on the double arrow on the status bar. When the scene plays back it will look like the diskette is moving across the screen.

The basic technique of animation is

1. Draw the BMP using XOR and with transparency turned off.
2. Wait for some period of time, using the \$D(<number>)\$ command. Where (<number>/60) is the number seconds to wait. You can use any setting you want, but remember that the smaller this number is the more images you'll have to draw fill in a certain length of time.
3. Erase the BMP by drawing it in the exact same location with XOR drawing mode and transparency turned off.
4. Draw the BMP in it's new location using XOR, without transparency and continue at step 2.

Now keep in mind that you can move the BMP any direction you want. As long as you always erase it correctly. The key to remember is to put the delay command between the time the BMP is drawn and when it is erased. This makes sure that the BMP is visible most of the time and cuts down on the flickering.

You can use any of the drawing commands in animation. Before you draw anything, click on the icon to the left of the question mark on the status bar. Set the drawing mode to XOR. Click on OK. Now try drawing a line. Put in a one second delay and then draw it again with XOR on. Remember to turn on XOR before each drawing command. Also remember that any BMP's you use must be in your user's ICONS directory or preferably in the system directory for you BBS.

Now let's show you how to load in photos to jazz up your scenes. To add a photo, press down the CTRL and the ALT keys and hit the H key. A dialog will pop up asking you which image you want to load. Select JUPITER.JPG this time, and click on OK. You are back in edit mode

with the crosshair cursor again. Drag a rectangle to show RIPaint where you want the image to be displayed. Once again a dialog box will pop up, and ask you a few questions about how the image would be displayed. In this case, just click on OK. Now a beautiful image of Jupiter will appear on the screen.

You can set the size and position to anything you want, and it will be either shrunk or expanded to fit that area. Sometimes you'll have an image that looks stretched out in one direction or the other. In this case, select Aspect ratio on the image options dialog. This forces the image to not be unevenly stretched.

Here is an explanation of the image options dialog's settings.

Erase image area - If this is selected, the area where the photo is to be placed is cleared to the current background color, color 0 usually black, before the photo is displayed.

Don't erase area - When selected, the area where the photo is to be shown is not cleared first.

Aspect ratio - If this is set, the photo will be displayed in the same aspect ratio as the original. This keeps the photo from looking stretched in one direction. If set, the photo may not cover the entire area you selected, this will result in black spaces where the photo doesn't cover.

Kill file after - If this is set, the photo file is deleted after it is displayed. Only use this if you are sending the image file over the modem. If you are displaying an image from the user's hard drive this will delete it after it is displayed.

Commit palette - When selected, the palette of the photo will be remapped to colors in the current color palette. This is only used with GIF images. This allows the photo to look as close to original as possible without changing the colors of other graphics on the screen.

Now your image file should be displayed on the screen. Try loading different images with the different settings.

There is one more topic we'll cover in this tutorial, multi column text. Have you ever wanted to make a screen that would have several

columns on it that you could easily display text in? Now with RIPaint 2, you can do that easily.

The first thing you need to do is select an article. You have a choice of article 1-35. Let's select article one to start with. To select an article, hold down the SHIFT key and hit the F3 function key. Select article 1 on the dialog and click on Ok. Each article is a separate text file that can have up to 35 columns associated with it. Now we need to create some columns for our article to be displayed in. To do this, hold down the SHIFT key and hit the F4 function key. Now you need to drag a rectangle that covers the place you want the first column of the article to be. When asked, make the first column 0, second column 1, etc.

After you have made a few columns, you'll want to load a text file into it. Hold down the ALT key and hit the letter T, this lets you load a text file from the DOC subdirectory under RIPaint. This text file should be a standard ASCII file with only the following formatting information. Paragraphs are marked by a blank line. The possible extensions are TXT and DOC. When the file is loaded into the column, all tabs and multiple spaces are ignored. Paragraphs are automatically indented and two spaces are placed after periods, colons, exclamation points and question marks. This means that charts are not a good candidate for use in these columns.

Now that the entire file is loaded into RIPaint and is being displayed, you may notice that not all of the text is being displayed. This text is called Overflow text, there isn't enough columnar space to display it. You can either add another column segment to the article or copy this overflow text into a text file. You do this by holding down the ALT key and hitting the V key. RIPaint will ask you for a name for the file, and then save the overflow text into it, when you click on Ok. You can load this text into another screen if you want to.

As you can see RIPaint 2 is easy to use and extremely powerful. The following chapters give you more information on the many different tools and features of RIPaint

CHAPTER 4

Buttons and Mouse fields

RIPaint 2

TeleGrafix Communications, Inc.

Last updated on 10/20/95

Now let's learn how we can make these screens interactive.

First, let's cover some new terminology. When you create a button, you will position a pink rectangle where you want the button to be. This is known as the Button area.

The first choice you need to make about a button style is what type of button it will be. There are three button types. The first is called the plain button type, it is a simple button often with a text label. The second button type is the Icon button, it is a button that displays an Icon/BMP from the disk in the Button Area, the area selected when making the button. The third and last button type Clipboard also uses an picture, however it gets the image from the last snapshot taken.

If you want a button that does not have a picture on it, use plain button. If you want a button that uses a picture, in the form of a BMP on the user's hard drive, then select the Icon button style. If you have previously taken a snapshot of a part of the screen and you want to use that on the button, select the Clipboard button.

You can set a standard size for any buttons you make. To do that set the height and the width to something other than zero. Now whenever you make a button it will always be the same size. This is very helpful when you want the buttons to be uniform and line up properly. The Button area will be set to the size of the height and the width. The next time you create a button, instead of having to drag a rectangle to mark the position and size of the button, all you have to do is move a pink rectangle to where you want to put it. This pink rectangle is already the size you set here.

On the far upper right of the Button Style Editor are some settings that affect the look of the button area. The most important way to learn about these settings is trying them in combinations and looking at the example button to see if you like the results.

First, another term I'll be using, shadowed rectangle, a rectangle with different colors on each edge to make it look like it is a shadow. Shadow one has the right and top side of the rectangle drawn with the Bright color. The lower and left side of the rectangle is drawn with the Dark color. This makes the area inside the rectangle look like it is above the surface.

Shadow two has the right and top side of the rectangle drawn with the Dark color. The lower and left side of the rectangle is drawn with Bright color. This makes it look like the area inside the rectangle, is below the surface.

If Chisel is set, RIPterm will draw a shaded rectangle, of the shadow two type, several pixels inside of the Button area. Right inside that rectangle the button will have a shaded rectangle of type two. The two together give the impression that there is a groove present. This makes it look like a 2-pixel groove was chiseled out around the inside edge of the button. Usually you will only want to use this effect with plain buttons, because it will draw the Chisel on top of the image. Chisel uses only the Bright and Dark color settings.

If Sunken is set, the button will look like the button is below the surface of the screen. This is done by drawing a shadowed rectangle, of type one, on the very edge of the Button area.

If Recess is set, the button will look like the button is sticking out above the surface of the screen. The button will have a black rectangle around it just outside of the Button area. If you also have Bevel set, this rectangle will be just outside of the bevel instead. Around that black rectangle is a shadowed rectangle, of type one, that makes it look like the button is sticking out above the surface. A button with Recess enabled is actually 2 pixels wider on each side. Keep this in mind if you use the gridsnap editor.

If Bevel is set, the button will look like the edges slope down. The size of the bevel is set by the Size spinner box. The larger the size the more edges appear to slope. A series of shadowed rectangles, of type two, are drawn around the button area. The corners are marked with lines of color Corner. The bevel is drawn just outside of the Button area, so the button will be larger than the button area you selected by about twice the size of the current bevel. Remember to add two times the bevel size to the height and width to calculate the size needed for the Gridsnap editor. Remember, that the Gridsnap editor uses device co-ordinates and not world coordinates. This means you'll need to convert the width and height into device coordinates.

Buttons

One last choice you have to answer about the look of the button is what the surface should look like. You can set this with the interior *combo box*. Your choices are Normal, where the surface is the same color as the surface color. Or you can chose Clear, where the surface of the button is transparent. And last but not least you can have it filled with the current fill color by selecting Filled.

There are two special settings that might help you. The first, Hot Icon allows you to have two versions of the button's icon. One is used when the Button is not selected, and the other is used when it is. For example, if I have an Icon named MAIL.BMP, and Hot Icon is set. When the button is selected RIPterm will use the Icon called MAIL.BMH instead. Note how they both have the same filename but have different extensions. Both are normal BMP's the only difference is that one is saved with the extension .BMH and not BMP.

Secrets of the RIPmasters

Hot icons - If you want some neat special effects try the following, create two versions of your icons/BMPs: one to be shown normally and one to show when the button is inverted. Give the inverted version the file extension of .BMH instead of BMP. Make sure both the names are the same though. Now select Hot icon when you create the button style. Now when someone clicks on your button, instead of that ugly inverted version of your icon, you can have one that looks just like you want it to. It could even be a completely different image. Imagine a BMP of a puppy that sticks it's tongue out when you click on it. Hot icons can't be used with Autoclip.

The second is called AUTOCLIP. Using Autoclip can significantly speed up button display if you are using icon buttons. The first time you create a button after setting Autoclip, RIPaint/RIPterm will copy the image of the icon onto the clipboard, including any special effects such as Bevel, Chisel, and Sunken special effects. Drawing such special effects can take a bit of time. The next button you create will simply copy the image off the clipboard to the new location and add the label. When using Autoclip, the name of the icon file is only sent once and read off the disk once. This leads to significant savings in both transmission and display time. If you are using autoclip, you can't select Hot icons.

Most buttons will have a text label of one kind or another, even icon buttons. You set the text for the label when you create the button. You do however set the way that text will be displayed as a label. You need to decide what color it will be, whether it will have a shadow, where in relation to the rest of the button it will be drawn etc. Any changes you make will be reflected in the Example button in the lower left hand corner of the Button Style Editor.

The first important thing to know is how to set the position of the label relative to the button area. The button area is the area of the mouse that can be selected by a mouse. You set it when you create the button, the rectangle you placed marks it.

Now let's look at how you place the text in relation to the button. On the Button Style Editor there is a *combo box* called Label Orient . You have four options, Center, Right, Left, Up, Down.

Up
Right Center Left
Down

The text will be displayed where the option name is in this diagram. The rectangle in this diagram shows the area that has been selected to

be the button area with the mouse. That means that unless you set Label Orient to Center, the label will not be on the button. Here are some examples:

Right:	Label		Up:	Label
Left:			Down:	Label
Center:				Label

Not only can you control where the label will appear in relation to the button you can also set whether it is right, left or center justified in the labels area. This is set using the *combo box* labeled Label Align . Here are some examples, showing the three different possible settings of Label Align:

	Left	Center
Right		

Buttons

The last item affecting the positioning of the label text is controlled by the *combo box* Text adj Center. If Label Orient is set to Right, Left, or Center, the RIPscrip compatible terminal will vertically center the label from the center of the button area. Text adj Center is used to control how that is done. If Text adj Center is on, then it will be centered using desenders. If off, the desenders don't affect how the label is centered.

You should set Label adj Center on, if you have a button that doesn't look like it's centered properly. If you have a horizontal row of buttons, to make sure the baselines all line up, set Label adj Center off. In most cases, it doesn't hurt to have Label adj Center set off, just turn it on when you need it.

Here is a chart listing all possible combinations of Label Orient , Label Align and Label adj Center :

Label Orient	Label Align	Label adj Center	Label looks like:
Up	Right	N/A	Label
Up	Center	N/A	Label
Up	Left	N/A	Label
Center	Right	Yes*	Label
		No**	
Center	Center	Yes*	Label
		No**	
Center	Left	Yes*	Label
		No**	
Down	Right	N/A	Label

Down	Center	N/A		
			Label	
			ï	
			æ	Left
				N/A
			Label	
Left	N/A	Yes*		
	Label	No**		
Right	N/A	Yes*		Label
		No**		

N/A - Setting is ignored by RIPterm.

* - Label is Centered vertically, taking account for descenders.

** - Label is Centered vertically, ignoring descenders.

The next important thing you have to set is what the actual text of the label is going to look like. Buttons use the current font style, and you can change that if you want by clicking on the Fonts button on the Button Style Editor. Buttons however will ignore all but the following font settings, Font name, Font size, Bold, Italic, Underline, Strikeout and spacing. Dropshadow, Character and String rotation, as well as the rest of the font information is ignored when a button is drawn. The example button in the lower right hand corner will show you what the text will look like.

You can change the color of the label text with the two color buttons Foreground and Background color. To change the current color, click in the circle and select a new color from the Color Picker. The foreground color is used for drawing the Text and the background color is used to draw any dropshadow if you have one enabled. Again, any changes to these settings will be shown in the example button.

Up until now you could make pretty buttons but they didn't do anything. Sure it says "Exit" on it but it doesn't work if you click on it. Now we get to the real power in buttons. The first thing you need to select in the Button Style Editor is the mouse *checkbox*. This will cause parts of the Dialog to change from grey to black. This means you can now use them.

Buttons

When the mouse checkbox is checked, RIPaint will need additional information about each button you make. Not only will it ask you what you want to label the button, it will ask you if you want to set a Hot Key, and what the host return string should be.

A Hot Key is a key on the keyboard that when that key is hit, the button will be triggered just as if it had been clicked on. It is a shortcut way of triggering it. There are three settings on the Button Style Editor that affect how Hot Keys are treated. If the Underline *checkbox* is checked, then if the Hot Key is in the label it will be underlined. If the Highlight *checkbox* is checked, and the Hot Key is contained in the label, then that letter will be drawn with the Highlight color instead of the normal Foreground color. This makes the Hot Keys very visible to the people using them.

The host return string is simply a special string of information that will be sent to the host, when the button is clicked. This string of information can contain some special characters such as RETURN, as well as commands to RIPterm in the form of Text Variables. The host string doesn't have any settings that affect it in the Button Style Editor. I just mentioned it to help you understand what happens when a Mouse Button is selected.

Much more powerful buttons can be made using Templates. (See Chapter 9 Templates) These buttons are called check box and radio buttons.

CHAPTER 5

Drawing Tools

RIPaint 2

TeleGrafix Communications, Inc.

Last updated on 10/20/95

Arcs



Placing an arc

1. Click on the icon on the Toolbox that represents the type of arc you want to draw.
2. Select the oval you want the arc to be based from. (See Oval)
3. The base oval you selected will appear in grey, with two points on it. Between these two points is the arc that is currently selected.
4. If the arc does not cover the proper angle, click and hold the left mouse button on one of the points and move it until the angle is correct.
5. If the angle is correct but the arc is on the wrong section of the oval, press and hold the CTRL key.
6. Move the point until the arc is oriented correctly.
7. Release the left mouse button and/or the CTRL key.

To exit arc placement mode

1. Click on the right mouse button or hit the ESC key.

To move an arc

1. Select the Arrow icon on the Toolbox,, to select Edit Mode.
2. Select the arc you want to move. (See Edit Mode)
3. Once you have selected your arc, move the cursor to the middle of the editing rectangle.
4. Press and hold the left mouse button down.
5. Move the mouse until the arc is the place you want it.
6. Release the left mouse button.

To delete an arc

1. Select the Arrow icon on the Toolbox, to select Edit Mode.
2. Select the arc you want to move. (See Edit Mode)
3. Hit the DELETE key.

To change the size and/or shape of an arc

1. Select the Arrow icon on the Toolbox, to select Edit Mode.
2. Select the arc you want to move. (See Edit Mode)

Drawing Tools

3. Move the cursor over one of the control points around the boundary rectangle.
4. Press and hold the left mouse button.
5. Move the mouse until you have the size and/or shape you want.
6. Release the left mouse button.

To change the color of an arc

1. Select the Arrow icon on the Toolbox,, to select Edit Mode.
2. Select the arc of which you want to change the colors. (See Edit Mode)
3. Bring up the Color Picker. (See Color Picker)
4. If the arc has a border, setting the drawing color will change the color of its border.
5. If the arc is filled, setting the fill color will change the color of the filled area of the arc.

To change the fill pattern of an arc

1. Select the Arrow icon on the Toolbox,, to select Edit Mode.
2. Select the arc of which you want to change the fill pattern. (See Edit Mode)
3. Bring up the Fill Pattern Editor by hitting the F5 function key.
4. Change the fill pattern. (See Editor, Fill Pattern)
5. Click on OK to exit the Fill Pattern Editor.

To change the width of the border of an arc

1. Select the Arrow icon on the Toolbox,, to select Edit Mode..
2. Select the arc, for which, you want to change the border width. (See Edit Mode)
3. Bring up the Line Style Editor by hitting the F4 function key.
4. Change the line thickness to the desired setting. (See Editor, Line Style)
5. Click on OK to exit the Line Style Editor.

Article

Creating an Article

1. Hold down the SHIFT key and hit the F3 function key.
2. Select the Article number you want to work with.
3. Create as many columns as you need for this article. (See Columns)
4. Hold down the ALT key and hit the letter T.
5. Choose the name of the text file you want to load into this article.
6. Click on OK.
7. Hold down the SHIFT key and hit the F1 function key, to end the article.

Deleting an Article

1. Select the column(s) of the article. (See Edit Mode)
2. Hit the Delete key.

To change the font of an Article

1. Select the first column of the article. (See Edit Mode)
2. Hit the F3 function key, to call up the Font Selector.
3. Change the font.
4. Click on the Ok button.

To change the font color of an Article

1. Select the first column of the article. (See Edit Mode)
2. Set the drawing color to the color you want the text to be.
3. Set the background color to the color you want the dropshadow to be.

To add another column to an Article

1. Hold down the SHIFT key and hit the F3 function key.
2. Select the article number you want to add a column to.
3. Add a column. (See Columns)

To delete a column from an Article

1. Select the column you wish to delete.
2. Hit the DELETE key.

BMP's/Icons

Placing a BMP

1. Hit the F9 function key.
2. Select the name of the BMP you wish to load from the File load dialog. (See Load File)
3. Click on OK.
4. Drag a rectangle that covers the area on which you want to display this BMP. (See Rectangle)
5. After you release the left mouse button a dialog box will appear.
6. If you want the BMP to be dithered (if not enough colors are available), make sure there is an X in the box next to Dithered.
7. If you want one of the colors in the BMP to be considered transparent, make sure there is an X in the box next to Transparent. And change the transparency color to match the color you want to be transparent.
8. If you want the BMP to be able to be erased by redrawing it in the exact same place, make sure the Drawing mode is set to XOR.
9. If you want a copy of the BMP to be copied to another port, make sure there is an X by the word Copy, and set Port to the port # to store it.
10. If you want the BMP to be displayed as wallpaper, select Wallpaper.
11. If you want the BMP to be displayed as staggered wallpaper, make sure both Wallpaper and Stagger are selected.
12. Click on OK.

To display a BMP, when a button or mouse field is selected

1. Type the following in the Host string for the button or mouse field you want to trigger the display of this photo: `$IMGSTYLE(CUR,`
2. After that, type in the co-ordinates of the upper left and lower right hand corner of the area the place the BMP is to be displayed, separated by commas. Example: 100, 100, 150, 200
3. Now add: `)$`
4. Finally, add this to the Host string: `$<Filename $` the place *Filename* is the name of your image file.
6. Here is an example of what it could look like
`$IMGSTYLE(CUR, 100, 100, 150, 200))$<TEST.BMP$`
7. Now you can add any characters that the Host will be expecting when this button or mouse field is selected.
8. Remember the BMP must be on the user's hard drive before this command will work.

To have a BMP displayed as wallpaper when a button or mouse field is selected

1. Put the following command in the Host string of the button or mouse field if you want to trigger this.
2. Type the following in the Host string: `$IMGSTYLE(CUR, 0, 0,`
3. After that, type in the width and height in pixels you want the BMP to be, separated by a comma. Example: `100, 100`
4. Add the following to make it display as wallpaper: `, WALLPAP`
5. If you want the image to be staggered every other row, add the following:
`, STAGGER`
6. Now add: `)$`
7. Finally, add this to the Command box: `$<Filename $` the place *Filename* is the name of your BMP.
8. Here is an example of what it could look like
`$IMGSTYLE(CUR, 0, 0, 100, 100, WALLPAP, STAGGER)$`
`$<TEST.BMP$`
9. You can now add any commands that host is expecting to receive when the button or mouse field is selected.
10. Remember the BMP must be on the user's hard drive before this command will work.

To move a BMP

1. Click on the Arrow icon on the Toolbox, to select Edit Mode.
2. Select the BMP you want to move. (See Edit Mode)
3. Move the cursor over the center of the BMP.
4. Press and hold down the left mouse button.
5. Move the editing rectangle to the place you want to put the BMP.
6. Release the left mouse button.

To delete a BMP

1. Click on the Arrow icon on the Toolbox, to select Edit Mode.
2. Select the BMP you want to delete. (See Edit Mode)
3. Hit the DELETE key.

To resize a BMP

1. Click on the Arrow icon on the Toolbox, to select Edit Mode.
2. Select the BMP you want to resize. (See Edit Mode)
3. Move the cursor over one of the eight control points around the editing rectangle.

4. Press and hold down the left mouse button.
5. Move the point on the editing rectangle until it is the size you want.
6. Release the left mouse button.

To change the name of the BMP file to be displayed

1. Click on the Arrow icon on the Toolbox, to select Edit Mode.
2. Select the BMP you want to change. (See Edit Mode)
3. Hit the F2 function key.
4. Change Filename to the new name.
5. Click on OK.

To change the BMP's display attributes, for example, Aspect, Wallpaper etc.

1. Click on the Arrow icon on the Toolbox, to select Edit Mode.
2. Select the BMP you want to change. (See Edit Mode)
3. Hit the F2 function key.
4. Make the changes you want.
5. Click on OK.

Button



Creating a button

1. Use the Button Designer to set the properties you want this button to use. (See Chapter 6 Button Designer)
2. Select the Button Icon from the Toolbox.
3. Drag a rectangle the place you want the button to be.
4. When you release the left mouse button a dialog will appear.
5. Enter the text you want displayed on the label under Label text.
Remember, you can use many Text Variables on the label.
6. If you are creating a clickable button, enter the host command you want sent when the mouse is clicked, next to Host Command.

Remember you can use Text Variables and Templates in this Host Command.

7. If you are creating an Icon button, enter the name of the icon you want displayed as the button.
8. Click on OK to finish.

Moving a button

1. Select the Arrow icon on the Toolbox , to select Edit Mode.
2. Select the button you want to edit. (See Edit Mode)
3. Move the cursor to the center of the Editing rectangle.
4. Press and hold the left mouse button.
5. Move the button to its new locati.
6. Release the left mouse button.

Resizing a button

1. Select the Arrow icon on the Toolbox, to select Edit Mode.
2. Select the button you want to resize.
3. Move the cursor over one of the eight control points on the edge of the editing rectangle.
4. Press and hold the left mouse button.
5. Move the cursor to the new locati.
6. Release the left mouse button.

Deleteing a button

1. Select the Arrow icon on the Toolbox, to select Edit Mode.
2. Select the button you want to edit. (See Edit Mode)
3. Hit the DELETE key.

Changing the Font Attributes of a button

1. Select the Arrow icon on the Toolbox, to select Edit Mode.
2. Select the button you want to change the Font Attributes.
3. Hit the F3 function key, to call up the Font Selector.
4. Make the needed changes.
5. Click on Ok to exit the Font Selector.

Changing the color of the label on a button

1. Select the Arrow icon on the Toolbox, to select Edit Mode.
2. Select the button you want to change the label color. (See Edit Mode)
3. Bring up the Color Picker.
4. If you want to change the color of the label, change the Drawing color to the desired color.
5. If you want to change the color of the label's dropshadow, change the Background color to the desired color.

Columns

Placing a column

1. Hold down the SHIFT key and hit the F3 function key.
2. Select the Article number you want to add a column to.
3. Hold down the SHIFT key and hit the F4 function key.
4. Use the crosshair cursor to drag a rectangle that covers the area where you want to place the column.

If you can't see the outline of the column

1. Hold down the ALT key and hit the F6 function key.

To move the column

1. Click on the Arrow icon on the Toolbox, to select Edit mode.
2. Select the column you want to move. (See Edit Mode)
3. Move the cursor to the center of the edit rectangle.
4. Press and hold the left mouse button down.
5. Move the mouse until the column is in the new location.
6. Release the left mouse button.

To delete the column

1. Click on the Arrow icon on the Toolbox, to select Edit mode.
2. Select the column you want to delete. (See Edit Mode)
3. Hit the DELETE key.

To hand edit the columns information

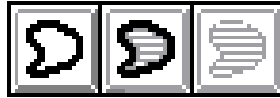
1. Select the column you want to delete. (See Edit Mode)
2. Hit the F2 function key.
3. Make the desired changes.
4. Click on the OK button to exit the editor.

To resize a column

1. Select the column you want to resize. (See Edit Mode)
2. Move the cursor over one of the 8 control points on the edit rectangle.
3. Press and hold down the left mouse button.
4. Move the mouse until the border(s) are the size you want.
5. Release the left mouse button.

Drawing Tools

Closed Curve

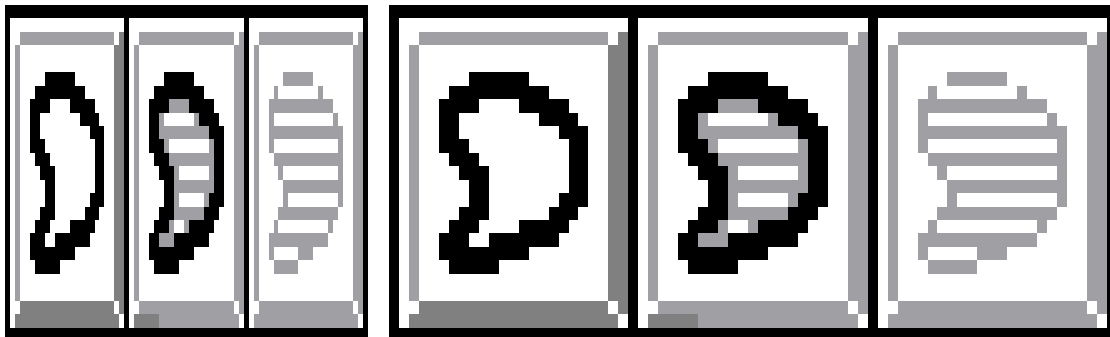


Border: Drawn in current drawing color, obeys Line Thickness

Fill: Drawn in current fill color, obeys Fill Pattern

Placing the first segment

1. Click on the Closed Curve icon, with the features you want, on the Toolbox.
2. Move the cursor to the place you want the first segment of the closed curve to start and press and hold the left mouse button.
3. Move the cursor to the place you want the first segment of the closed curve to end and release the left mouse button.



Parts of a Bezier curve

Differences between smooth and cusp.

Moving an end point of the active segment

1. If you want to move one of the end points and you don't want its joined curve point to move with it, press and hold the CTRL key.
2. Move the cursor over the end point you wish to move.
3. Press and hold the left mouse button.
4. Move the end point to its new location.
5. Release the left mouse button and release the CTRL key if pressed.

Moving a Curve Point

1. If you want the curve point to stay on the line that connects it to its endpoint, press and hold the SHIFT key.
2. Move the cursor over the curve point you want to move.
3. Press and hold the left mouse button.

Drawing Tools

4. Move the curve point to its new locati.
5. Release the left mouse button and release the SHIFT key if pressed.

Adding a segment

1. Any segment will be added from the currently selected segment, if this segment is in the middle of two other segments. The new segment will be placed in between the active segment and one of its neighboring segments.
2. Move the cursor to the place you want the new segment to end.
3. Click and release the left mouse button.

Selecting a segment

1. Click the left mouse button on the end point of the segment.
2. If the wrong segment is made active, click the left mouse button on the opposite side of the chosen segment.

Changing a segment from a curve to a line

1. Select the segment you want to change.
2. Hit the F2 function key.
3. Click on Line on the dialog that appears.

Changing a segment from a line to a curve

1. Select the segment you want to change.
2. Hit the F2 function key.
3. Click on Curve on the dialog that appears.

Changing a joint between two segments from smooth to cusp

1. Select one of the segments near the joint.
2. Hit the F2 function key.
3. Click on Cusp on the dialog that appears.
4. If the wrong joint is changed, return it to what it was and select the segment on the other side of the joint and start aga.

Deleting a segment

1. Select the segment you want to delete.
2. Hit the F2 function key.
3. Click on Remove on the dialog that appears.

To break open the closed polygon

1. Select the segment the place you want to break the polyg.
2. Hit the F2 function key.

Drawing Tools

3. Click on Break on the dialog that appears.

To close an open polygon:

1. Hit the F2 function key.
2. Click on Join on the dialog that appears.

To exit closed curve placement mode

1. Click on the right mouse button or hit ESC.

To move a closed curve

1. Select the Arrow icon on the Toolbox, to select Edit Mode.
2. Select the closed curve you want to move. (See Edit Mode)
3. Once you have selected your closed curve, move the cursor to the middle of the editing rectangle.
4. Press and hold the left mouse button down.
5. Move the mouse until the outline of the closed curve is the place you want it.
6. Release the left mouse button.

To delete a closed curve

1. Select the Arrow icon on the Toolbox, to select Edit Mode.
2. Select the closed curve you want to move. (See Edit Mode)
3. Hit the DELETE key.

To change the size of a closed curve

1. Select the Arrow icon on the Toolbox, to select Edit Mode.
2. Select the closed curve you want to move. (See Edit Mode)
3. Move the cursor over one of the control points around the edge of the editing rectangle.
4. Press and hold the left mouse button.
5. Move the mouse until you have the size you want.
6. Release the left mouse button.

To change the color of a closed curve

1. Select the Arrow icon on the Toolbox, to select Edit Mode.
2. Select the closed curve you want to change the colors. (See Edit Mode)
3. Bring up the Color Picker. (See Color Picker)
4. If the closed curve you chose has a border, setting the drawing color will change the color of its border.
5. If the closed curve is filled, setting the fill color will change the color of the filled area.

To change the fill pattern of a closed curve

1. Select the Arrow icon on the Toolbox, to select Edit Mode.
2. Select the closed curve of which you want to change the fill pattern.
(See Edit Mode)
3. Bring up the Fill Pattern Editor by hitting the F5 function key.
4. Change the fill pattern. (See Editor, Fill Pattern)
5. Click on OK to exit the Fill Pattern Editor.

To change the width of the border of a closed curve

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the closed curve, for which, you want to change the border width.
3. Bring up the Line Style Editor by hitting the F4 function key.
4. Change the line thickness. (See Editor, Line Style)
5. Click on OK to exit the Line Style Editor.

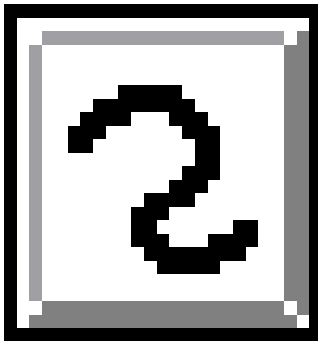
Curved Polyline



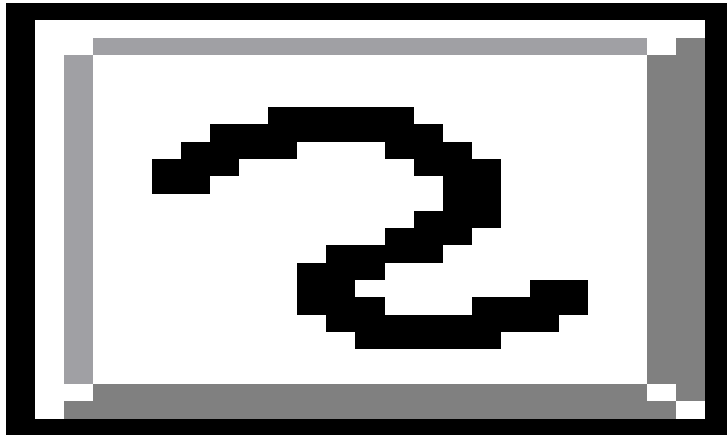
Border : Drawn in current drawing color, obeys Line Thickness

Placing the first segment

1. Click on the curved polyline icon on the Toolbox.
2. Move the cursor to the place you want the first segment of the curved polyline to start and press and hold the left mouse button.
3. Move the cursor to the place you want the first segment of the curved polyline to end and release the left mouse button.



Parts of a Bezier curve



Differences between smooth and cusp joints.

Selecting a segment

1. Click the left mouse button on the end point of the segment.
2. If the wrong segment is made active, click the left mouse button on the opposite side of the chosen segment.

Moving an end point of a selected segment

1. If you want to move one of the end points and you don't want its joined curve point to move with it, press and hold down the CTRL key.
2. Move the cursor over the end point you wish to move.
3. Press and hold the left mouse button.
4. Move the end point to its new locati.

5. Release the left mouse button and release the CTRL key if pressed.

Moving a Curve Point

1. If you want the curve point to stay on the line that connects it to its endpoint, press and hold the SHIFT key.
2. Move the cursor over the curve point you want to move.
3. Press and hold the left mouse button.
4. Move the curve point to its new location.
5. Release the left mouse button and release the SHIFT key if pressed.

Adding a segment

1. Select the segment you want the new segment to be near.
2. Move the cursor to the place you want the new segment to end.
3. Click and release the left mouse button.

Changing a segment from a curve to a line

1. Select the segment you want to change.
2. Hit the F2 function key.
3. Click on Line on the dialog that appears.

Changing a segment from a line to a curve

1. Select the segment you want to change.
2. Hit the F2 function key.
3. Click on Curve on the dialog that appears.

Changing a joint between two segments from smooth to cusp

1. Select one of the segments near the joint you want to change.
2. Hit the F2 function key.
3. Click on Cusp on the dialog that appears.
4. If the wrong joint is changed, return it to what it was and select the segment on the other side of the joint you want to change, and start again.

Deleting a segment

1. Select the segment you want to delete.
2. Hit the F2 function key.
3. Click on Remove on the dialog that appears.

To exit curved polyline placement mode

1. Click on the right mouse button or hit ESC.

To move a curved polyline

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the curved polyline you want to move.
3. Move the cursor to the center of the editing rectangle.
4. Press and hold the left mouse button down.
5. Move the mouse until the editing rectangle is the place you want the polyline to be.
6. Release the left mouse button.

To delete a curved polyline

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the curved polyline you want to delete.
3. Hit the DELETE key.

To change the size of a curved polyline

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the curved polyline you want to change the size.
3. Move the cursor over one of the control points around the editing rectangle.
4. Press and hold the left mouse button.
5. Move the mouse until you have the size you want.
6. Release the left mouse button.

To change the color of a curved polyline

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the curved polyline you want to change the colors.
3. Bring up the Color Picker. (See Color Picker)
4. Setting the drawing color will change the color of the polyline.

To change the width of a curved polyline

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the curved polyline you want to change the width.
3. Bring up the Line Style Editor by hitting the F4 function key.
4. Change the line thickness. (See Editor, Line Style.)
5. Click on OK to exit the Line Style Editor.

Edit Mode



In Edit mode, you select one or more objects you want to edit. A boundary rectangle will be drawn around the boundaries of the selected objects. At the corners and in the middle of each side are control points that can be dragged to resize the rectangle. After you resize the rectangle, all the selected objects will be resized accordingly.

Selecting an object in Edit mode

1. Click on the Edit mode icon on the Toolbox. It looks like an arrow.
2. Click the left mouse on the edge of the object.
4. If more than one item is selected, hold down the SHIFT key and click on the edge of the object you don't want to select with the left mouse button. This will deselect it.

Adding an object to the current selection

1. Hold down the SHIFT key.
2. Click on the edge of the object you want to add with the left mouse button.
3. Release the SHIFT key.

Removing an object from the current selection

1. Hold down the SHIFT key.
2. Click on the edge of the object you want to remove with the left mouse button.
3. Release the SHIFT key.

Selecting a group of objects at once

1. Click on the Edit mode icon on the Toolbox. It looks like an arrow.
2. Drag a rectangle over the objects you wish to select. (See Section 5.11 Rectangle)

Moving objects

1. Click on the Edit mode icon on the Toolbox. It looks like an arrow.
2. Select the objects to move.
3. Move the cursor to the center of the selection rectangle.
4. Press and hold the left mouse button.

Drawing Tools

5. Move the mouse until the objects are the place you want them to be.
6. Release the left mouse button.

Forcing Movement mode

1. Click on the Edit mode icon on the Toolbox. It looks like an arrow.
2. Select the objects you want to move.
3. Press and hold the CTRL button. This forces RIPaint to move the selected objects.
4. Move the cursor to the center of the selection rectangle.
5. Press and hold the left mouse button.
6. Move the mouse until the objects are the place you want them to be.
7. Release the left mouse button.

Deleting objects

1. Select the Edit mode icon on the Toolbox. It looks like an arrow.
2. Select the objects you want to delete.
3. Hit the DELETE key.

Resizing selected objects

1. Select the Edit mode icon on the Toolbox. It looks like an arrow.
2. Select the objects you want to resize.
3. Move the cursor over one of the eight control points.
4. Press and hold the left mouse button.
5. Move the cursor until you have the desired size.
6. Release the left mouse button.

Eye Dropper



While you are in drawing mode. With the eye dropper, you can pick up a color from the current scene to be used as the current drawing color. If you are in editing mode, it will pick up other attributes as well.

Change the Drawing color with the Eye Dropper

1. Bring up the Color Picker.
2. Click the left mouse button on the left most rectangle, to select the Drawing color.
3. Select the Eye Dropper icon from the Toolbox.
4. Move the Eye Dropper until its tip is over the color you want to pick up.
5. Click the left mouse button.

Change the Background color with the Eye Dropper

1. Bring up the Color Picker.
2. Click the left mouse button on the middle rectangle, to select the Background color.
3. Select the Eye Dropper icon from the Toolbox.
4. Move the Eye Dropper until its tip is over the color you want to pick up.
5. Click the left mouse button.

Change the Fill Color with the Eye Dropper

1. Bring up the Color Picker.
2. Click the left mouse button on the rectangle on the right, to select Fill color.
3. Select the Eye Dropper icon from the Toolbox.
4. Move the Eye Dropper until its tip is over the color you want to pick up.
5. Click the left mouse button.

To copy attributes from one object to another object(s) with the Eye Dropper

1. Select the Arrow icon on the Toolbox, to select Edit Mode.

2. Select the objects, to which, you want to apply the attributes. (See Edit Mode)
3. Select the Eye Dropper icon from the Toolbox.
4. Move the Eye Dropper until its tip is over the edge of the object from which you want to copy the attributes.
5. Click the left mouse button.

Line



Line: Drawn in current drawing color, obeys both line pattern and line width.

Placing a line

1. Click on the line icon on the Toolbox.
2. If you want to force the line to be vertical or horizontal, press and hold down the SHIFT key.
3. Move the cursor to the place you want the line to start.
4. Press and hold down the left mouse button.
5. Move the cursor to the place you want the line to end.
6. Release the left mouse button and the SHIFT key if pressed.

To exit line placement mode

1. Hit the right mouse button or hit ESC.

To move a line

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the line you want to move.
3. Once you have selected your line, move the cursor to the middle of the editing rectangle.
4. Press and hold the left mouse button down.
5. Move the mouse until the editing rectangle is the place you want the line to be.
6. Release the left mouse button.

To delete a line

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the line you want to delete.
3. Hit the DELETE key.

To change the size of a line

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)

Drawing Tools

2. Select the line you want to resize.
3. Move the cursor over one of the control points around the editing rectangle.
4. Press and hold the left mouse button.
5. Move the mouse until you have the size you want.
6. Release the left mouse button.

To change the color of a line

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the line of which you want to change the colors.
3. Bring up the Color Picker. (See Color Picker)
4. Setting the drawing color will change the line's color.

To change the width of a line

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the line you want to change the width.
3. Bring up the Line Style Editor by hitting the F4 function key.
4. Change the line thickness. (See Editor, Line Style.)
5. Click on OK to exit the Line Style Editor.

To change the line pattern of a line

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the line you want to change the line pattern.
3. Bring up the Line Style Editor by hitting the F4 function key.
4. Change the line pattern. (See Editor, Line Style)
5. Click on OK to exit the Line Style Editor.

Mouse Field



With this tool you can create transparent areas that when the user clicks on them will send information back to the Host.

Create a mouse field

1. Select the Mouse Field icon on the Toolbox.
2. Drag a rectangle that covers the area the place you want your mouse field to be.
3. When you release the left mouse button, a dialog will appear.
4. Enter the text you want to be sent when this mouse field is selected the place it says Host Command. You can include Text Variables and Templates here. (See Text Variables) (See Templates)
5. Click on Ok.

To have the graphics under the mouse area invert when it is selected

1. Make sure Invert has an X on it, when the dialog pops up.

To have RIPterm do a Reset when a mouse area is selected.

1. Make sure Reset has an X next to it, when the dialog pops up. This is very useful if the next screen to be displayed is not a RIP file.

To have the mouse area triggered if the Enter key is pressed

1. Make sure Default is selected.

To have the mouse area triggered if the ESC key is pressed

1. Make sure Abort is selected, when you create your mouse field.

To have the mouse area triggered if any key is pressed

1. Make sure that Any Key has an X by it when you create your mouse field.

To have the mouse area triggered if specific key is pressed and you know the ASCII code of the key

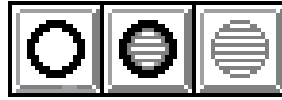
Drawing Tools

1. Set Hot Key value to the ASCII code value of the key you want to trigger it.

To have the mouse area triggered if a specific key is pressed and you can enter it from the keyboard.

1. Set Hot Key Char to the character to trigger the mouse field.

Oval



Border: Drawn in current drawing color, obeys line thickness.

Center: Drawn in current fill color, obeys fill pattern.

Placing the oval

1. Click on the icon on the Toolbox that represents the style of oval you want to draw.
2. Think of the size and location of the rectangle that would just hold the oval you want to draw.
3. Move the cursor to the upper left hand corner of that rectangle.
4. Press and hold the left mouse button.
5. Move the cursor until you have the desired oval.
6. Release the left mouse button.

Placing an oval centered from the initial point

1. Click on the icon on the Toolbox that represents the style of oval you want to draw.
2. Press and hold the CTRL key.
3. Move the cursor to the point you want the oval to be centered.
4. Press and hold the left mouse button.
5. Move the mouse until you have the desired oval.
6. Release the left mouse button and the CTRL key.

Placing a Circle

1. Click on the oval icon on the Toolbox that has the attributes you want.
2. Press and hold the SHIFT key.
3. Think of the size and position of a square that could hold the circle you want to draw.
3. Move the cursor to the upper left hand corner of that rectangle.
4. Press and hold down the left mouse button.
5. Move the cursor until you have the desired circle.
6. Release the left mouse button and the SHIFT key.

Placing a centered Circle

1. Click on the oval icon on the Toolbox that has the attributes you want.
2. Press and hold the SHIFT key.
3. Press and hold the CTRL key.
4. Move the cursor the place you want the center of the circle to be.
5. Release the left mouse button, the SHIFT key and the CTRL key.

Exiting oval placement mode

1. Click on the right mouse button or hit ESC.

To move an oval

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the oval you want to move.
3. Move the cursor to the middle of the editing rectangle.
4. Press and hold the left mouse button down.
5. Move the mouse until the editing rectangle is the place you want the oval to be.
6. Release the left mouse button.

To delete an oval

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the oval you want to delete.
3. Hit the DELETE key.

To change the size and/or shape of an oval

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the oval you want to move.
3. Move the cursor over one of the control points around the editing rectangle.
4. Press and hold the left mouse button.
5. Move the mouse until you have the size you want.
6. Release the left mouse button.

To change the color of an oval

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)

Drawing Tools

2. Select the oval of which you want to change the colors.
3. Bring up the Color Picker. (See Color Picker)
4. If the oval you chose has a border, setting the drawing color will change the color of its border.
5. If the oval is filled, setting the fill color will change the color of the filled area in the oval.

To change the fill pattern of an oval

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the oval you want to change the fill pattern.
3. Bring up the Fill Pattern Editor by hitting the F5 function key.
4. Change the fill pattern to the way you want it. (See Editor, Fill Pattern)
5. Click on OK to exit the Fill Pattern Editor.

To change the width of the border of an oval

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the oval you want to change the border width.
3. Bring up the Line Style Editor by hitting the F4 function key.
4. Change the line thickness. (See Editor, Line Style)
5. Click on OK to exit the Line Style Editor.

Photo

Placing a Photo

1. Hold down the SHIFT key and hit the F9 function key.
2. Select the image you want to place in your scene from the Load File dialog. (See Load File)
3. Click on the OK button when you have selected the file.
4. Drag a rectangle that covers the area the place you want this image displayed. (See Rectangle)
4. If you want the selected area to be cleared to black first, select Erase Image area.
5. If you don't want the selected area to be cleared to black first, make sure Don't Erase Area is selected.
6. If you want to make sure that the image isn't stretched in one direction or another make sure there is an X next to Aspect Ratio.
7. If you want the file to be deleted after it is displayed, select Kill file after. Note: don't use this if you are displaying a file locally, because the file will be deleted after it is displayed.
8. If you want the photo to be wallpapered on the screen, select Wallpaper
9. If you want the photo to be use for staggered wallpaper on the screen, select Wallpaper and Stagger.
10. Click on OK.

To display a photo, when a button or mouse field is selected

1. Type the following in the Host string for the button or mouse field you want to trigger the display of this photo: `$IMGSTYLE(CUR,`
2. After that, type in the co-ordinates of the upper left and lower right hand corners, of the area the place the image is to be displayed, separated by commas. Example: `100, 100, 150, 200`
3. If you want the image to be displayed without stretching in one direction, add the following text: `, ASPECT`
4. Now add: `)$`
5. Finally, add this to the Host string: `$(Filename $ the place Filename`
is the name of the image file you want to display.
6. Here is an example of what it could look like
`$IMGSTYLE(CUR, 100, 100, 150, 200,`
`ASPECT)$(JUPITER.JPG$`
7. Next you can add any characters that the Host will be expecting when this button or mouse field is selected.

Drawing Tools

8. Remember the image file must be on the user's hard drive before this command will work.

To have a photo displayed as wallpaper when a button or mouse field is selected

1. Put the following command in the Host string of the button or mouse field you want to trigger this.
2. Type the following in the Host string: `$IMGSTYLE(CUR, 0, 0,`
3. After that, type in the width and height in pixels you want the image to be, separated by a comma. Example: `100, 100`
4. If you want the image to be displayed without stretching in one direction, add the following text: `, ASPECT`
5. Add the following to make it display as wallpaper: `, WALLPAP`
6. If you want the image to be staggered every other row, add the following:
`, STAGGER`
7. Now add: `)$`
8. Finally, add this to the Command box: `$(Filename $ the place`
`Filename` is the name of the image file you want to display.
9. Here is an example of what it could look like
`$IMGSTYLE(CUR, 0, 0, 100, 100, ASPECT, WALLPAP,`
`STAGGER)$$(JUPITER.JPG$`
10. You can now add any commands that the Host is expecting to receive when this button or mouse field is selected.
11. Remember the image file must be on the user's hard drive before this command will work.

To move a photo

1. Click on the Arrow icon on the Toolbox, to select Edit Mode.
2. Select the photo you want to move. (See Edit Mode)
3. Move the cursor over the center of the editing rectangle.
4. Press and hold down the left mouse button.
5. Move the editing rectangle to the photo's new locati.
6. Release the left mouse button.

To delete a photo

1. Click on the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the photo you want to delete. (See Edit Mode)
3. Hit the DELETE key.

Drawing Tools

To resize a photo

1. Click on the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the photo you want to resize. (See Edit Mode)
3. Move the cursor over one of the eight control points on the edge of the editing rectangle.
4. Press and hold down the left mouse button.
5. Move the editing rectangle until it is the size you want.
6. Release the left mouse button.

If you have black around your photo

You have Aspect and Erase image area set in the photo's settings. Aspect makes sure that the photo is displayed without distortion, and Erase image area clears the selected area to black before displaying the photo. If Aspect doesn't cover the entire area you selected, some of the cleared area will be visible.

1. Turn off the Erase image area setting.
or
1. Turn off the Aspect setting.
or
1. Resize the photo's borders to the size of the photo when it displays on the screen.

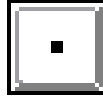
To change the name of the photo file

1. Click on the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the photo you want to change.
3. Hit the F2 function key.
4. Change Filename to the name of the file you want to load.
5. Click on OK.

To change the photo's display attributes, for example, Aspect, Wallpaper etc.

1. Click on the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the photo you want to change.
3. Hit the F2 function key.
4. Change the attributes.
5. Click on OK.

Point



Size of point obeys Line thickness.

To set the size of a point

1. Hit the F4 function key. (See Editor, Line Pattern)
2. Select the desired line thickness.
3. Click on OK.

To place a point

1. Click on the point icon on the Toolbox.
2. If you want use pixels instead of points, press and hold down the CTRL key.
3. If you want to freehand draw, press and hold down the SHIFT key.
4. Move the cursor to the place you want your point to be.
5. Click the left mouse button.

To freehand draw with points

1. Click on the point icon on the Toolbox.
2. Press and hold down the SHIFT key.
3. Move the cursor to the place you want to start freehand drawing.
4. Press and hold down the left mouse button.
5. Move the mouse to place the points.
6. When you are done freehand drawing, release the left mouse button and the SHIFT key.

To place a pixel

1. Click on the point icon on the Toolbox.
2. Press and hold down the CTRL key.
3. Move the cursor to the location you want to put a pixel.
4. Click the left mouse button.

To freehand draw with pixels

1. Click on the point icon on the Toolbox.
2. Press and hold the CTRL key.
3. Press and hold the SHIFT key.
4. Move the cursor to the place you want to start freehand drawing.

Drawing Tools

5. Press and hold down the left mouse button.
6. Move the mouse to place the pixels.
7. When you are done freehand drawing, release the left mouse button, the CTRL key and the SHIFT key.

To exit point placement mode

1. Click the right mouse button or hit ESC.

To move a point

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the point you want to move.
3. Move the cursor to the middle of the editing rectangle.
4. Press and hold down the SHIFT key, to force movement mode.
5. Press and hold the left mouse button down.
6. Move the mouse until the editing rectangle is at the new location for the point.
7. Release the left mouse button and the SHIFT key.

To delete a point

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the point you want to delete.
3. Hit the DELETE key.

To change the color of a point

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the point of which you want to change the color.
3. Bring up the Color Picker. (See Color Picker)
4. Change the drawing color to the color you want the pixel to be.

To change the size of a point

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the point of which you want to change the size.
3. Bring up the Line Style Editor by hitting the F4 function key.
4. Change the line thickness to the desired width. (See Editor, Line Style)
5. Click on OK to exit the Line Style Editor.

Polygon



Border: Drawn in current drawing color, obeys line thickness.

Fill: Drawn in current fill color, obeys current fill pattern.

To place the initial segment of the polygon

1. Click on the appropriate polygon icon on the Toolbox.
2. Move the mouse cursor to the place you want the first segment to start.
3. Press and hold down the left mouse button.
4. Move the cursor to the place you want the first segment to end.
5. Release the left mouse button.

Moving an end point of the currently selected segment

1. Move the cursor over the end point you wish to move.
2. Press and hold the left mouse button.
3. Move the end point to its new location.
4. Release the left mouse button.

Adding a segment

1. Select the segment you want the new segment to be next to.
3. Move the cursor to the place you want the new segment to end.
4. Click and release the left mouse button.

Selecting a segment

1. Click the left mouse button on the end point of the segment you want to select.
2. If the wrong segment is selected, click the left mouse button on the opposite side of the chosen segment.

Deleting a segment

1. Select the segment you want to delete.
2. Hit the F2 function key.
3. Click on Remove on the dialog that appears.

Drawing Tools

To close the polygon

1. Hit the F2 function key.
2. Click on Join on the dialog that appears.

To break open the polygon

1. Select the segment the place you want to break open the polyg.
2. Hit the F2 function key.
3. Click on Break on the dialog that appears.

To exit polygon placement mode

1. Click on the right mouse button or hit ESC.

To move a polygon

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the polygon you want to move.
3. Move the cursor to the middle of the editing rectangle.
4. Press and hold the left mouse button down.
5. Move the mouse until the editing rectangle is the place you want to move the polyg.
6. Release the left mouse button.

To delete a polygon

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the polygon you want to delete.
3. Hit the DELETE key.

To change the size of a polygon

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the polygon you want to resize.
3. Move the cursor over one of the control points on the edge of the editing rectangle.
4. Press and hold the left mouse button.
5. Move the mouse until you have the size you want.
6. Release the left mouse button.

To change the color of a polygon

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)

Drawing Tools

2. Select the polygon of which you want to change the colors.
3. Bring up the Color Picker. (See Color Picker)
4. If the polygon you chose has a border, setting the drawing color will change the color of the border.
5. If the polygon is filled, setting the fill color will change the color of the filled area of the polygon.

To change the fill pattern of a polygon

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the polygon of which you want to change the fill pattern.
3. Bring up the Fill Pattern Editor by hitting the F5 function key.
4. Change the fill pattern. (See Editor, Fill Pattern)
5. Click on OK to exit the Fill Pattern Editor.

To change the width of the border of a polygon

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the polygon of which you want to change the border width.
3. Bring up the Line Style Editor by hitting the F4 function key.
4. Change the line thickness to the desired setting. (See Editor, Line Style)
5. Click on OK to exit the Line Style Editor.

To change the line pattern of the border of a polygon

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the polygon, on which, you want to change the line pattern.
3. Bring up the Line Style Editor by hitting the F4 function key.
4. Change the line pattern to the desired setting. (See Editor, Line Style)
5. Click on OK to exit the Line Style Editor.

Polyline



Border: Drawn in current drawing color, obeys both line thickness and line pattern.

To place the initial segment of the polyline

1. Click on the polyline icon on the Toolbox.
2. Move the mouse cursor to the place you want the first segment to start.
3. Press and hold down the left mouse button.
4. Move the cursor to the place you want the first segment to end.
5. Release the left mouse button.

Moving an end point of the active segment

1. Move the cursor over the end point you wish to move.
2. Press and hold the left mouse button.
3. Move the end point to the place you want it.
4. Release the left mouse button.

Adding a segment

1. Select the segment you want to be next to the new segment.
3. Move the cursor to the place you want the new segment to end.
4. Click and release the left mouse button.

Selecting a segment

1. Click the left mouse button on the end point of the segment.
2. If the wrong segment is selected, click the left mouse button on the opposite side of the chosen segment.

Deleting a segment

1. Select the segment you wish to delete.
2. Hit the F2 function key.
3. Click Remove on the dialog that appears.

To exit polyline placement mode

1. Click on the right mouse button or hit ESC.

To move a polyline

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the polyline you want to move.
3. Move the cursor to the middle of the editing rectangle.
4. Press and hold the left mouse button down.
5. Move the mouse until the editing rectangle is at the new location for the polyline.
6. Release the left mouse button.

To delete a polyline

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the polyline you want to delete.
3. Hit the DELETE key.

To change the size of a polyline

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the polyline you want to resize.
3. Move the cursor over one of the control points on the edge of the editing rectangle.
4. Press and hold the left mouse button.
5. Move the mouse until you have the size you want.
6. Release the left mouse button.

To change the color of a polyline

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the polyline of which you want to change the colors.
3. Bring up the Color Picker. (See Color Picker)
4. Setting the drawing color will change its color.

To change the width of a polyline

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the polyline of which you want to change the width.
3. Bring up the Line Style Editor by hitting the F4 function key.
4. Change the line thickness. (See Editor, Line Style)
5. Click on OK to exit the Line Style Editor.

To change the line pattern of a polyline

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the polyline of which you want to change the line pattern.
3. Bring up the Line Style Editor by hitting the F4 function key.
4. Change the line pattern. (See Editor, Line Style)
5. Click on OK to exit the Line Style Editor.

Rectangle



Border: Drawn in current Drawing color, obeys both line thickness and line pattern.

Fill: Drawn in current Fill color, obeys current fill pattern.

To place a rectangle

1. Click on the appropriate rectangle icon on the Toolbox.
2. Drag a rectangle on the screen. (see below)

To drag a rectangle

1. If you want the rectangle to be a square, press and hold down the SHIFT key.
2. If you want the rectangle to be centered on the initial point, press and hold the CTRL key.
3. If you are not holding down the CTRL key, move the mouse cursor to the place you want the upper left hand corner of the rectangle to be.
4. If you are holding down the CTRL key, move the mouse cursor to the place you want the center of the rectangle to be.
5. Press and hold the left mouse button.
6. Move the cursor until you have the desired rectangle.
7. Release the left mouse button, and the SHIFT and CTRL if they were pressed.

To exit rectangle placement mode

1. Click the right mouse button or hit ESC.

To move a rectangle

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the rectangle you want to move.
3. Move the cursor into the center of the editing rectangle.
4. Press and hold the left mouse button down.
5. Move the mouse until the editing rectangle is at the new location of the rectangle.
6. Release the left mouse button.

To delete a rectangle

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the rectangle you want to delete.
3. Hit the DELETE key.

To change the size of a rectangle

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the rectangle you want to resize.
3. Move the cursor over one of the control points on the edge of the editing rectangle.
4. Press and hold the left mouse button.
5. Move the mouse until you have the size you want.
6. Release the left mouse button.

To change the color of a rectangle

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the rectangle of which you want to change the colors.
3. Bring up the Color Picker. (See Color Picker)
4. If the rectangle you chose has a border, setting the drawing color will change the color of the border.
5. If the rectangle is filled, setting the fill color will change the color of the filled area in the rectangle.

To change the fill pattern of a rectangle

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the rectangle, for which, you want to change the fill pattern.
3. Bring up the Fill Pattern Editor by hitting the F5 function key.
4. Change the fill pattern. (See Editor, Fill Pattern)

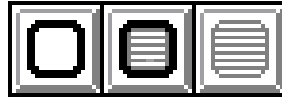
To change the width of the border of a rectangle

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the rectangle you want to change the border width.
3. Bring up the Line Style Editor by hitting the F4 function key.
4. Change the line thickness. (See Editor, Line Style)
5. Click on OK to exit the Line Style Editor.

To change the line pattern of the border of a rectangle

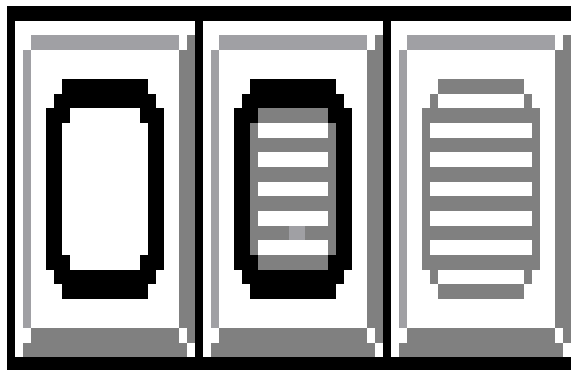
1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the rectangle you want to change.
3. Bring up the Line Style Editor by hitting the F4 function key.
4. Change the line pattern. (See Editor, Line Style)

Round Rectangle



Border: Drawn in current drawing color, obeys both line thickness and line pattern.

Fill: Drawn in current fill color, obeys current fill pattern.



Control points of a round rectangle

To place a round rectangle

1. Click on the appropriate round rectangle icon on the Toolbox.
2. Drag a rectangle on the screen. (See Rectangle)
3. Move the mouse cursor over one of the four control points.
4. Press and hold down the left mouse button.
5. Move the cursor until you get the desired round corners.
6. Release the left mouse button.

To exit round rectangle placement mode

1. Click the right mouse button or hit ESC.

To move a rounded rectangle

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the rounded rectangle you want to move.
3. Once you have selected your rounded rectangle, move the cursor to the middle of the editing rectangle.
4. Press and hold the left mouse button down.

5. Move the mouse until the editing rectangle is at the new location for the rounded rectangle
6. Release the left mouse button.

To delete a rounded rectangle

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the rounded rectangle you want to delete.
3. Hit the DELETE key.

To change the shape or size of a rounded rectangle

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the rounded rectangle you want to resize.
3. Move the cursor over one of the control points on the edge of the editing rectangle.
4. Press and hold the left mouse button.
5. Move the mouse until you have the shape and size you want.
6. Release the left mouse button.

To change the color of a rounded rectangle

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the rounded rectangle of which you want to change the color.
3. Bring up the Color Picker. (See Color Picker)
4. If the rounded rectangle you chose has a border, setting the drawing color will change the border's color.
5. If the rounded rectangle is filled, setting the fill color will change the color of the filled area.

To change the fill pattern of a rounded rectangle

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the rounded rectangle, whose fill pattern you want to change.
3. Bring up the Fill Pattern Editor by hitting the F5 function key.
4. Change the fill pattern to the way you want it. (See Editor, Fill Pattern)
5. Click on OK to exit the Fill Pattern Editor.

Drawing Tools

To change the width of the border of a rounded rectangle

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the rounded rectangle, the border width of which, you want to change.
3. Bring up the Line Style Editor by hitting the F4 function key.
4. Change the line thickness. (See Editor, Line Style)

Sound

To add a sound (.WAV file) to your scene

1. Press the ALT key and hit the F9 function key.
2. Select the .WAV file you want to load from the Load file dialog. (See Section 3.4 Load File)

To stop a sound from playing

1. Hold down the ALT key and hit the 0 (zero)
2. In Command type "\$RESET(SOUND)\$".
3. Make sure Immediately is selected.
4. Click on OK.

To stop a sound from playing when a button or mouse field is clicked

1. Put the following in the host command of the button or mouse field
\$RESET(SOUND)\$
2. You can put any text that needs to be sent to the host after this command.

To have a song play when a button or mouse field is clicked

1. Put \$)*Filename* \$, the place *Filename* is the name of the sound file to play, in the host return string of the button or mouse field.

To remove the command to play a sound

1. Go into the Object Lister by holding down the SHIFT key and hitting the F2 function key. (See Section 6.9 Object Lister)
2. Click the left mouse button on the Play Sound command you want to remove.
3. Once it is highlighted, click on the Delete button at the bottom of the Object lister.
4. Click on the OK button to exit the Object Lister.

To change the name of the sound file to play

1. Go into the Object Lister by holding down the SHIFT key and hitting the F2 function key. (See Section 6.9 Object Lister)
2. Click the left mouse button on the Play Sound command you want, to change the sound file name.
3. Click the left mouse button on the Edit button.

4. Enter the new sound file name in Filename.
5. Click on OK, when the sound file name is correct.
6. Click on OK to exit the Object lister.

To make a sound file play over and over

1. Go into the Object Lister by holding down the SHIFT key and hitting the F2 function key. (See Section 6.9 Object Lister)
2. Click the left mouse button on the Play Sound command you want to loop.
3. Click the left mouse button on the Edit button.
4. Click the left mouse button in the square next to the text Looping Sound.
5. Click on OK.
6. Click on OK to exit the Object Lister.

Snapshot

The ability to take and place Snapshots is a powerful feature. When you take a Snapshot a section of the screen is saved to a special offscreen port. When you Place a Snapshot, it is displayed on the current screen at the location and size you want. Both RIPaint and RIPterm will execute this command. That means you can cut and paste sections of the screen without worrying what is displayed there.

One of the more common uses of these commands is to save a piece of the screen to an offscreen port, display a pop-up dialog or information, and restore the screen when its done.

For example, you could create a RIP file that will save the part of the screen it affects at the beginning of it displays some information and then when the Ok button is pressed it restores the screen to the way it was before this RIP file was executed.

Here are some text variables you can use with Snapshot

`$SMF$` This is a text variable that tells RIPterm to save all the mouse fields on the screen to disk.

`$SMF(PUSH)$` This is a text variable that tells RIPterm to save the current mouse fields in its internal stack.

`$RMF$` This is a text variable that tells RIPterm to restore the mouse fields from the disk to the screen.

`$RMF(POP)$` This tells RIPterm to restore the current mouse fields from its internal stack.

`$MKILL$` This tells RIPterm to kill all mouse fields on the screen.

`$PCB$` This is a text variable that tells RIPterm to put the snapshot data back on the screen the place it took it from.

`$PCB(POP)$` This tells RIPterm to copy the last snapshot data pushed on the stack back to its previous locati.

`$SCB(PUSH)$` This tells RIPterm to push the current Snapshot data onto RIPterms internal stack.

Drawing Tools

To take a snapshot of the screen

1. Hit ESC to exit any drawing mode you might be.
2. Click on the right mouse button
3. Click on the word Edit on the top menu bar.
4. Click on the words Take Snapshot.
5. Drag a rectangle over the area of which you want to take a snapshot.
(See Rectangle)

To place a Snapshot back on the screen

1. Hit ESC to exit what ever drawing mode you are.
2. Click the right mouse button to call up the menu bar.
3. Click on the word Edit on the top menu bar.
4. Click on the words Place Snapshot.
5. Drag a rectangle over the area of which you want the Snapshot to be displayed.

To restore the screen with the snapshot data when a button or mouse field is executed.

1. Put any text variables and text that should be processed before the screen is restored.
2. Then add the following to the Host return string of the button or mouse field to restore the screen: \$PCB\$
3. If you saved the mouse fields before you took your Snapshot, add the following: \$RMF\$
3. After this you can put text variables and text that should be processed after the screen is restored including text that the Host is expecting.

For a dialog box style RIPfile

This is compatible with RIPterm 1.54 and RIPterm 2.0

1. Save all the mouse field data by querying a text variable called \$SMF\$ (See Querying Text Variables)
2. Kill all the mouse fields on the screen by querying a text variable called \$MKILL\$
3. Now take a Snapshot of the area of the screen you will be changing.
4. Now add all the commands to draw your dialog box including buttons.
5. Add the following text to the Host return text for both the OK and Cancel buttons on your dialog: \$PCB\$\$RMF\$

This is compatible with RIPterm 2.0 only!

By using the following method you can add a lot more power. First, instead of saving the mouse field data on disk, we push it onto RIPterm's stack. This speeds up the return from our dialog significantly. Also we free up the Snapshot by storing its information on the stack also. This allows us to use the Snapshot function while we are drawing the dialog. Using the stack like this we could actually have our dialog call another RIP file that contained a dialog etc., without having to worry about using the same backup area.

1. Save all the mouse field data to RIPterm's internal stack, by querying the text variable `$SMF(PUSH)$`
2. Kill all the mouse fields on the screen. (See Section 7.3 Mouse fields)
3. Now take a Snapshot of the area of the screen you will be changing.
4. Now push that Snapshot data onto RIPterm's internal stack by querying the text variable `$SCB(PUSH)$`
5. Now add all the commands you need to draw your dialog box including buttons. Since the Snapshot data has been pushed onto the stack, you can still use Snapshot to make your dialog.
6. Add the following text to the Host return string for both the OK and the Cancel buttons on your dialog: `$PCB(POP)$RMP(POP)$`

Notes

Make sure that all buttons that aren't Radio buttons or Check buttons have the `$PCB(POP)$RMP(POP)$` in their Host return strings to make sure that you leave the stack and the display clean.

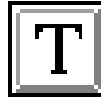
Don't mix `$SMF(PUSH)$` with `$RMP$` or `$SCB(PUSH)$` with `$PCB$`, because this leaves your data on the stack. If someone has already pushed data on the stack, when they try and retrieve it they will get your data instead.

Use `$SMF(PUSH)$`, and kill the mouse fields on the screen. This ensures that the user doesn't set off one of the mouse fields that were on the screen before you took over.

Drawing Tools

Remember to restore the mouse fields with \$RMF(POP)\$ when you are done, or the user will have to have the entire screen redrawn to get the mouse fields back.

Text



Placing text

1. Use the Font Selector to pick the font, and font attributes you want to use. (See Font Selector)
2. Click on the Text icon on the Toolbox, it has a large T on it.
3. Move the I-bar cursor to the place you want the text to be.
4. Click and release the left mouse button.
5. Type in the text you want to be displayed.
6. If you need to edit the text you typed, see Editing Text during initial text placement.
7. If you need to move the text around, see Moving Text during initial text placement.
6. Hit the ENTER key.

Changing text during initial text placement

1. Use the right and left arrow keys to move to the place you want to change the text.
 2. Use the BACKSPACE and DELETE keys to remove unwanted text.
 3. Type in the correct text.
- or
1. Move the I-Bar cursor over the position you want to change in the text.
 2. Click and release the left mouse button.
 3. Use the BACKSPACE and DELETE keys to remove unwanted text.
 3. Type in the correct text.

Moving text during initial text placement

1. Move the I-Bar cursor over the center of the text line.
2. Press and hold the left mouse button.
3. Move the outline rectangle the place you want the text to be.
4. Release the left mouse button.

Exiting out of text placement mode

1. Hit the ESC key.

Drawing Tools

Moving text after it has been placed

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the lines of text you want to move.
3. Move the cursor to the center of the editing regi.
4. Press and hold the left mouse button.
5. Move the outline of the editing region to the place you want to move the text.
6. Release the left mouse button.

Deleting text after it has been placed

1. Select the Arrow icon on the Toolbox, to select Edit Mode.
2. Select the line of text you want to delete.
3. Hit the DELETE key.

Changing Font and Font attributes of text after it has been placed

1. Select the Arrow icon on the Toolbox, to select Edit Mode. It looks like an arrow.
2. Select the line of text you want to change.
3. Hit the F3 function key.
4. Change the Font Selector settings. (See Section 6.4 Font Selector)

Changing the text color

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the text of which you want to change the text color.
3. Bring up the Color Picker. (See Color Picker)
4. Set the drawing color to the color you want the text to be.
5. If your text has a drop shadow, setting the background color will change its color.

Changing the raw text

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the line of text you want to change.
3. Hit the F2 function key.
4. Change the text.
6. Click on OK.

Drawing Tools

You can use any text variables you want in text and they will be processed when the text is displayed. Here are some handy text variables you can put anywhere in any of your text strings.

For example:

Today's date is \$DATE\$. Its \$HOUR\$ 0'clock do you know where your Sysop is?

would be displayed something like this.

Today's date is 12/19/95. Its 12'oclock do you know the place your Sysop is?

\$DATE\$	Replaced with the current date in the form 12/19/93
\$MONTH\$	Replaced with the name of the current month.
\$MONTHNUM\$	Replaced with the number of the current month, January returns 1, December returns 12, Etc.
\$DAY\$	Replaced with current day of the month.
\$DOY\$	Replaced with the number of days that have passed in the current year with leading zeros.
\$YEAR\$	Replaced with the 2 character version of the current year. IE 1995 would return 95.
\$FYEAR\$	Replaced with the full 4 character version of the current year.
\$TIME\$	Replaced with the time that the text variable was processed in the format HH:MM:SS
\$HOUR\$	Replaced with the current hour, 0-12.
\$MHOUR\$	Replaced with the current hour, 00-23.
\$MIN\$	Replaced with the number of minutes passed this hour.
\$SEC\$	Replaced with the number of seconds passed this minute, 00-59.
\$AMPM\$	Replaced with AM or PM depending on what time it is.
\$DATETIME\$	Replaced with Day-of-Week name, Month name, Day of month, Military style time, and full year. Example: Sat Dec 19 14:38:50 1993
\$TIMEZONE\$	Replaced with the time zone that RIPTerm is in, if set.
\$DOW\$	Replaced with the full name of the current day of the week.

Drawing Tools

\$ADOW\$	Replaced with the 3 letter abbreviated name of the current day of the week.
\$WDAY\$	Replaced with the number of the current day of the week. 0 = Sunday.
\$WOY\$	Replaced with the current week number in the year with Sundays being the first day of the week.
\$WOYM\$	Replaced with the current week number in the year with Mondays being the first day of the week.
\$BEEP\$	Beeps when it is processed.
\$BLIP\$	Makes a Blip sound when processed.
\$MUSIC\$	Makes a cheerful sound when processed.
\$ALARM\$	Makes a sound indicating failure.
\$PHASER\$	Makes a ascending sound like a Phaser being fired.
\$REVPHASER\$	Makes a desending sound like a Phaser being fired.

RIPscrip 2.0 only text variables

\$T(F, L)\$	Sounds a tone of F Hertz for L milliseconds. For example, \$T(1000,75)\$ will sound a 1000 Hertz wave for 75 milliseconds.
\$TERMINFO\$	Replaced with the name of the RIPscrip compatible terminal being used.
\$TERMINFO(VERSION)\$	Replaced with the full Version number of the RIPscrip compatible terminal being used.
\$TERMINFO(VENDOR)\$	Replaced by the name of the company that made the RIPscrip compatible terminal being used.

Text Window

The Text Window is the place all non RIPscrip text is displayed. By default, all ANSI codes in the text to be displayed are processed and the results used to display the text in the text window. VT-102 emulation can also be turned on in the Text Window, either by the user or by the Host. Doorway is another emulation mode that RIPterm supports.

To toggle the Text Window border display in RIPaint on and off

1. Hold down the ALT key and hit the F6 function key.

Creating a Text window

1. Hit ESC to end any drawing mode you might be.
2. Click the right mouse button to call up the top menu bar.
3. Click on the word Windows on the menu bar.
4. Click on the word Text Window.
5. Move the cursor to the right and put the cursor over the word Create.
6. Click the left mouse button.
7. Now drag a rectangle that covers the area of the screen the place you want your Text Window to be. (See Rectangle)
8. If you want the Text Window to be erased after it is defined, select Erase Window.
9. If you want the text that is displayed in the window to wrap at the edge of the window, select Wrap Text.
10. If you want the text to be chopped off if it exceeds the boundry of the Text Window, select Chop Text.
11. If you want the cursor to be visable, select Visable.
12. If you want the cursor to be invisible, select Invisible.
13. Select OK.

To move the Text Window

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the Text Window. (See Edit Mode)
3. Move cursor over the center of the editing rectangle.
4. Press and hold down the left mouse button.
5. Move the mouse until the editing rectangle is the place you want the Text Window to be.

To change the size of the Text Window

Drawing Tools

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the Text Window.
3. Move the cursor over one of the control points on the edge of the editing rectangle.
4. Press and hold down the left mouse button.
5. Move the cursor until the editing rectangle is the size you want the Text Window to be.
6. Release the left mouse button.

To delete the Text Window

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the Text Window.
3. Hit the DELETE key.

To change the attributes of the Text Window

1. Select the Arrow icon on the Toolbox, to select Edit Mode. (See Edit Mode)
2. Select the Text Window.
3. Hit the F2 function key.
4. Change the attributes.
5. Click on OK to exit object editor.

To deactivate the Text Window

1. Do a text variable query on the text variable, \$DTW\$. (See Text Variable Query)

To activate the current Text Window

1. Do a text variable query on the text variable, \$ATW\$. (See Text Variable Query)

To erase the contents of the current Text Window

1. Do a text variable query on the text variable, \$ETW\$. (See Text Variable Query)

To deactivate the Text Window when a button or mouse field is selected

1. Put any text variables or text that needs to be processed before the window is deactivated into the Host return string.
2. Add \$DTW\$ to the Host return string.

3. Add any text variables or text that needs to be processed after the window is deactivated.

To activate the Text Window when a button or mouse field is selected

1. Put any text variables or text that needs to be processed before the window is activated into the Host return string.
2. Add \$ATW\$ to the end of the Host return string.
3. Add any text variables or text that needs to be processed after the window is activated.

To erase the contents of the current Text Window

1. Do a text variable query of the text variable \$ETW\$. (See Text Variable Query)

To erase the contents of the current Text Window when a button or mouse field is executed.

1. Add \$ETW\$ to the Host return string.

To disable the cursor in the current Text Window

1. Do a text variable query of the text variable \$COFF\$. (See Text Variable Query)

To enable the cursor in the current Text Window when a button or mouse field is executed.

1. Add \$CON\$ to the Host return string.

To disable the cursor in the current Text Window when a button or mouse field is executed.

1. Add \$COFF\$ to the Host return string.

To process some commands on the BBS without letting the user know he left this menu, when a button or mouse field is executed

1. Add \$DTW\$ to the Host return string.
2. Add all the commands that you need to the Host command string.
For example, D^m d Filename^m x^m. In this case, D^m tells the host you want to go to the download menu item. d Filename^m tells the host you want to download the file called Filename and x^m returns you back to the present menu.
3. Add \$ATW\$ to the Host return string.
4. Add any other text variables or text you need to.

Drawing Tools

Here are some Text Variables that affect the Text Window.

\$DTW\$	Deactivates the current Text Window. All non RIPcode text is ignored and is not displayed.
\$ATW\$	Activates the Text Window. All non RIPcode text will be displayed in the Text Window.
\$COFF\$	Disables the Text cursor.
\$CON\$	Enables the Text cursor.
\$STW\$	Saves the Text window attributes, cursor position, etc to disk.
\$STW(PUSH)\$	*Saves the Text window attributes on RIPterm's stack.
\$RTW\$	Restores the Text window attributes from the disk.
\$RTW(POP)\$	*Restores the Text window attributes from RIPterm's stack.
\$TWERASEEOL\$	*Erases to the end of the current line in the Text window.
\$TWHOME\$	*Moves the text cursor to the upper left hand corner.
\$TWGOTO(X,Y)\$	*Moves the text cursor to the position (X,Y) in the current Text Window.
\$VT102OFF\$	Turns off VT-102 processing in the current Text Window.
\$VT102ON\$	Turns on VT-102 processing in the current Text Window.
\$ETW\$	Erase Text Window
\$TWFONT\$	Returns the current font being used in the Text Window.
\$TWH\$	Returns the height of the current text window in text lines.
\$TWW\$	Returns the width of the current text window in characters.
\$TWIN\$	Returns the current Text Window status.
\$TWX0\$	Returns the X coordinate of the upper left hand corner of the current Text Window.
\$TWY0\$	Returns the Y coordinate of the upper left hand corner of the current Text Window.
\$TWX1\$	Returns the X coordinate of the lower right hand corner of the current Text Window.
\$TWY1\$	Returns the Y coordinate of the lower right hand corner of the current Text Window.
\$ISEXTWIN\$	*Returns if Text Window is Extended Text window.

Drawing Tools

`$CURX$` Returns the current X coordinate of the cursor in the current Text Window.

`$CURY$` Returns the current Y coordinate of the cursor in the current Text Window.

* RIPscrip 2.0 only

CHAPTER 6

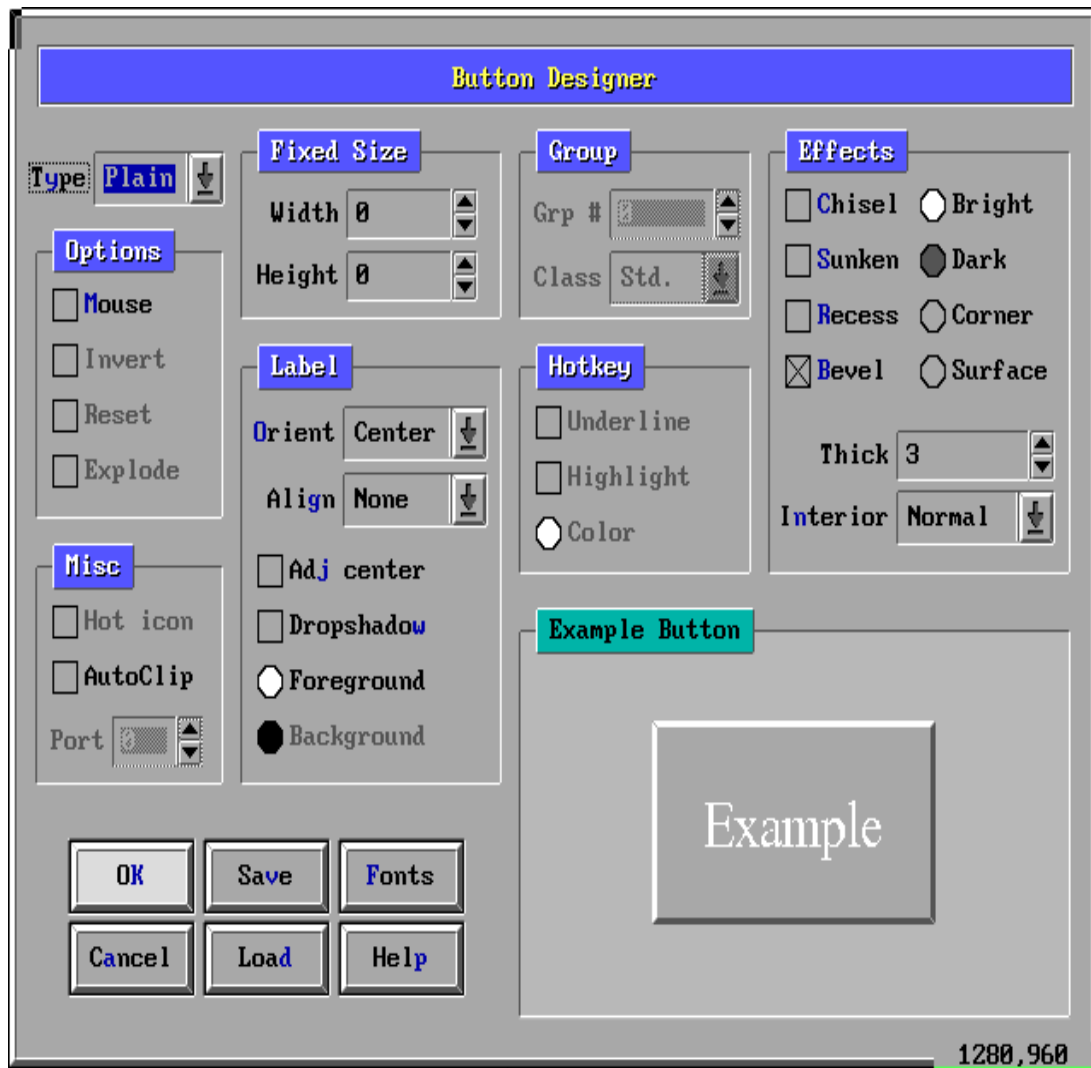
Editing Tools

RIPaint 2

TeleGrafix Communications, Inc.

Last updated on 10/20/95

Button Designer



Type - This lets you set the type of button. Choices are:

Plain - A normal button.

Icon - A button that uses an icon

Clip - A button that uses an image stored in the clipboard.

Fixed Size - These two *combo boxes* let you set a fixed size to all your buttons in pixels.

Mouse - This is a check box. If there is an X in it then a mouse field will be attached to the button allowing it to be clicked on and send text back to the host.

Invert - If checked the button will be inverted when it is clicked by the user. Mouse box must be checked for this to be available.

Reset - If this is checked the button will clear the screen when clicked on by the user. Mouse box must be selected for this to be available.

Explode - If this is checked the button will look like it zooms out to full screen. Mouse box must be selected for this to be available.

Hot icon - If you have an inverted version of your icon/BMP selecting this will use it instead of using a normal invert. It must be an icon button and the Mouse box must be selected for this option to be available.

Autoclip - When selected, the first time you create a button, the icon will be copied to the clipboard. Any buttons created after that will use the image on the clipboard for the icon, at least until the Button style has been changed.

Port - Number of port to get image from.

Label Orient - This is a *combo box* that lets you set where the label will be displayed in relation to the button. Choices are Center, Left, Right, Up, Down.

Label Align - This is a *combo box* that lets you set how the label is displayed in it's area. Choices are Right, Left, and Center.

Label Adj Center - If this is selected then lowercase desenders are taken in account when doing vertical centering. Normally, descenders are ignored to speed up text drawing.

Label Dropshadow - This is a *checkbox* that selects whether the button's label has a dropshadow or not.

Label Foreground - This a color button that shows and sets the current color of the button label text. To change the color, click on the octagon,

and then select the new color from the Color Picker. The octagon will change to the new color as will the example text.

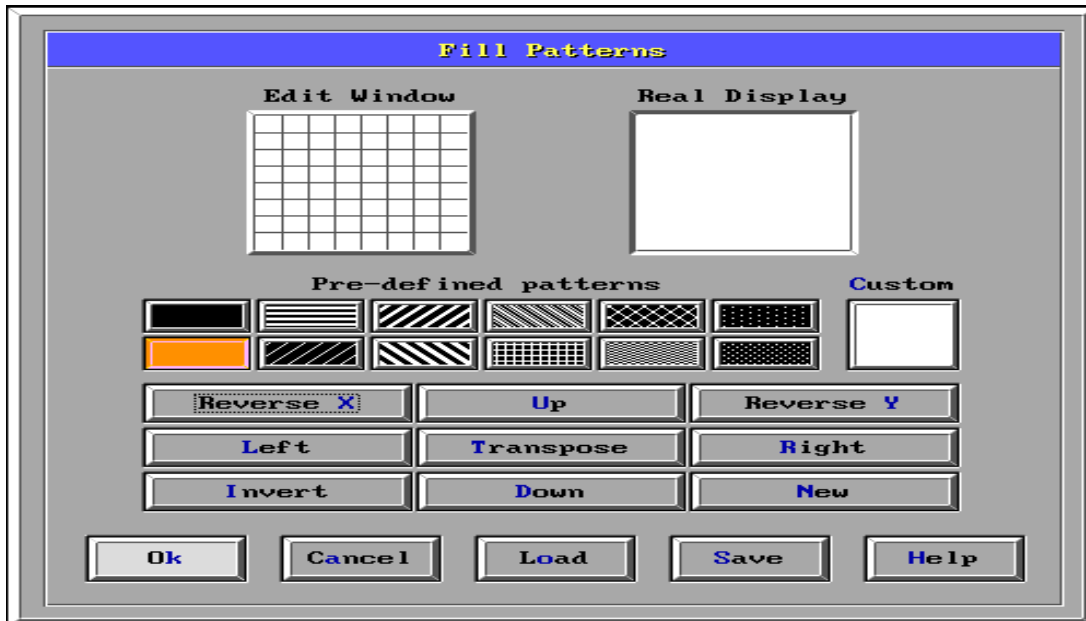
Label Background - This is a color button that shows and sets the current color of the button dropshadow ,if one is selected. To change the color, click on the octagon, and then select the new color from the Color Picker. The octagon will change to the new color as will the dropshadow of the example text, if drop shadow is enabled.

Group # - The group number for this button. Used to group buttons together so they can work together. All buttons with the same group number are related in some way with each other. All buttons in each group must be of the same type. This is not available unless mouse is checked. (See Chapter 9 Templates)

Group class - There are three types of groups that are available, Plain buttons, Radio Buttons, Check Boxes. (See Chapter 9 Templates)

Fill Style Editor

The Fill Style Editor allows you to edit the patterns that are used when filled objects are drawn. To change the fill pattern, either hit the F5 function key or, on the statusbar, click on the box to the right of the S with a slash through it.



Predefined patterns - You can select a preset fill pattern by clicking on it.

Custom - This button displays your current custom fill pattern. Clicking on this button selects your own custom fill pattern.

Edit Window - This is a 8 x 8 group of squares that you can set to your own fill pattern. You can change a square from white to black and vice versa by clicking on it.

Real Display - This is what your pattern looks like when used as a fill pattern.

Reverse X - Exchanges columns 1 & 8, 2 & 7, 3 & 6, and 4 & 5.

Reverse Y - Exchanges rows 1 & 8, 2 & 7, 3 & 6, and 4 & 5.

Up - moves all the rows up one and putting row 1 in row 8.

Down - moves all the rows down one and put row 8 in row 1.

Right - moves all the columns to the right one and putting column 8 in column 1.

Left - moves all the columns to the left one and putting column 1 in column 8.

Transpose - Does the same thing as selecting Reverse X and then Reverse Y.

Invert - Changes all white squares to black and all black squares to white.

New - Sets all the squares to white.

OK - RIPaint updates the settings and returns to Edit mode.

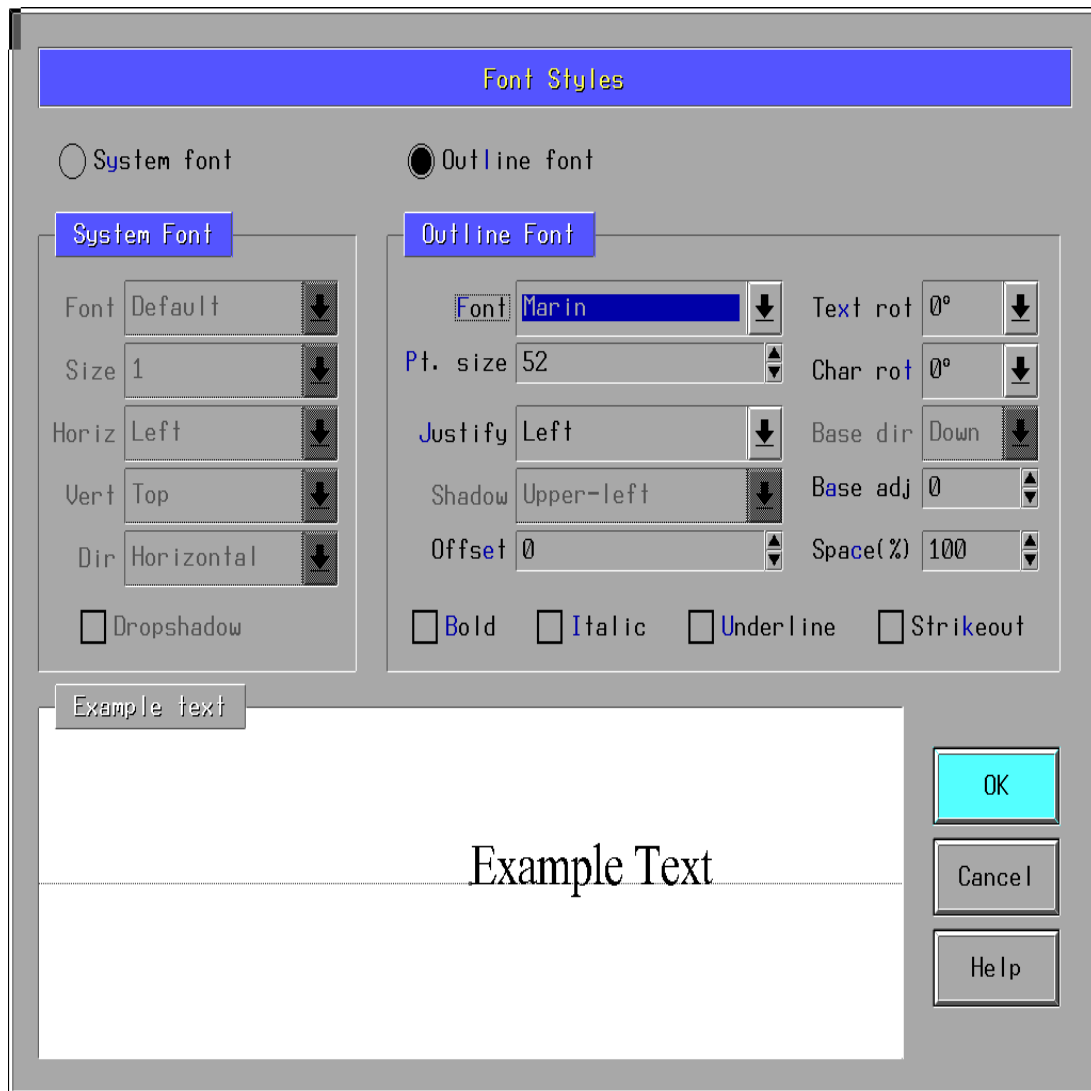
Cancel - RIPaint ignores these settings and returns you to Edit mode using the old settings.

Load - Loads a custom fill pattern from disk file.

Save - Saves current settings to a disk file.

Help - Displays help screens for this dialog.

Font Selector



OK - RIPaint will enable the changes you have made and return to edit mode.

Cancel - RIPaint will ignore the changes you made and return to edit mode.

Help - Brings up a help screen for this dialog.

Example - Shows an example of what text would look like if you were to use these settings.

System font - if selected uses the non-True type/Adobe style fonts. Activates the following controls. All the controls to the right are disabled.

Font - a combo box that allows you to select a font by name.

Size - the size the font should be.

Horiz - the horizontal justification to use. Choices are Left, Right, Center

Vert - Where the base line is in relation to the font. Choices are: Top, Bottom, Baseline, and Middle. Top puts the baseline at the top of the character cells. Bottom puts the baseline at the bottom of the decenders. Baseline puts the baseline at the bottom of the character cell, above the decenders. Middle puts the baseline in the middle of the character cell.

Dir - This is the direction the font moves in. Choices are: Horizontal and vertical.

Dropshadow - If selected the font will drawn with a dropshadow.

Outline font - Set's RIPaint to use Adobe/True type style fonts. Deactivates all controls to the left and activates the following controls.

Font - Lets you select a font by name.

Pt size - Lets you select the point size of the font.

Justify - Here you can select whether the font is to be Right, Left or Center justified.

Shadow - This control becomes active when a value greater than 0 is entered into the Offset *spinner box*. It lets you select where the dropshadow will be in relation to the text.

Offset - If set to greater than 0, it activates the shadow *combo box*. This enables Drop shadow. The higher this number the larger the difference between the Text and the Drop shadow.

Bold - if selected the font will be displayed in Bold type.

Italic - if selected the font will be displayed in Italic type.

Underline - if selected the font will be displayed with underlining.

Strikeout - if selected the font will be displayed with Strikeout enabled.

Text Rot - This sets the rotation of the text in 90 degree increments.

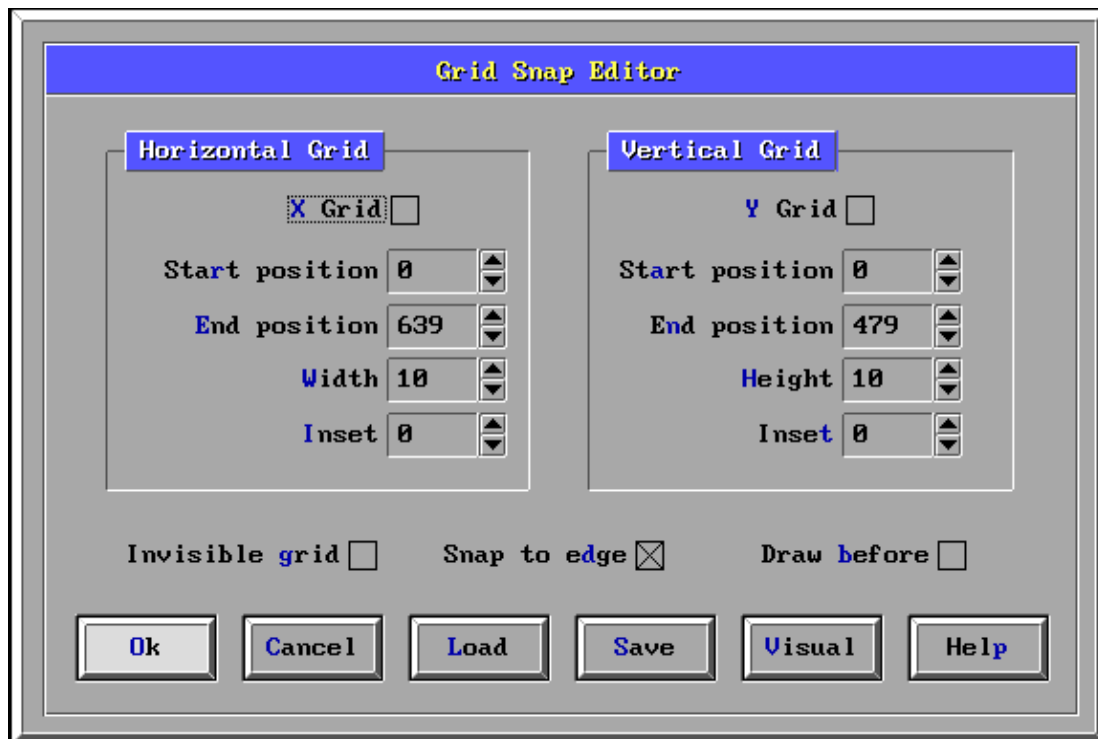
Char Rot - This sets the rotation of the individual characters relative to the text line in general.

Base Dir - This is activated when Base Adj is greater than 0. It sets the location of the Baseline relative to the character boxes.

Base Adj - This activates the combo box Base Dir when greater than 0. This sets the number of pixels the baseline is offset from the character lines.

Spacing - This sets how much spacing is put between each character. If it is set for 100%, it will use the normal amount of space that the font would use. If it is lower than 100%, then the spacing will be that percentage of the normal spacing. If it is higher than 100%, then the spacing will be that percentage of the normal spacing. This allows you to squeeze or expand to make text fit just right.

Grid Snap Editor



Horizontal Grid

X-Grid - When selected RIPaint enables horizontal Grid snap.

Start Position - The location of the first horizontal stop.

End Position - The location of the last horizontal stop.

Width - The number of pixels between each horizontal stop

Inset - If greater than 0, the space between stops is alternated between width and inset.

Vertical Grid

Y-Grid - When selected RIPaint enables vertical Grid snap.

Start Position - The location of the first Grid snap stop.

End Position - The location of the last grid snap stop.

Width - The number of pixels between each vertical stop

Inset - If greater than 0, the space between the stops is alternated between width and inset.

Invisible grid - Selecting this toggles whether the grid snap is drawn visually on the screen or not. If this is selected, an X appears in the box

to the left. When selected the cursor is still locked into only those positions allowed by grid snap, even though the grid snap is not drawn.

Snap to Edge - If selected mouse can only be clicked inside the Grid Snap area.

Draw before - If selected the grid snap is drawn before your scene is displayed, this means that objects will appear on top of the grid. A filled rectangle, for example, would not have grid lines through it. When not selected the grid snap is drawn after your scene.

OK - Clicking on this tells RIPaint to use these settings and returns you to edit mode.

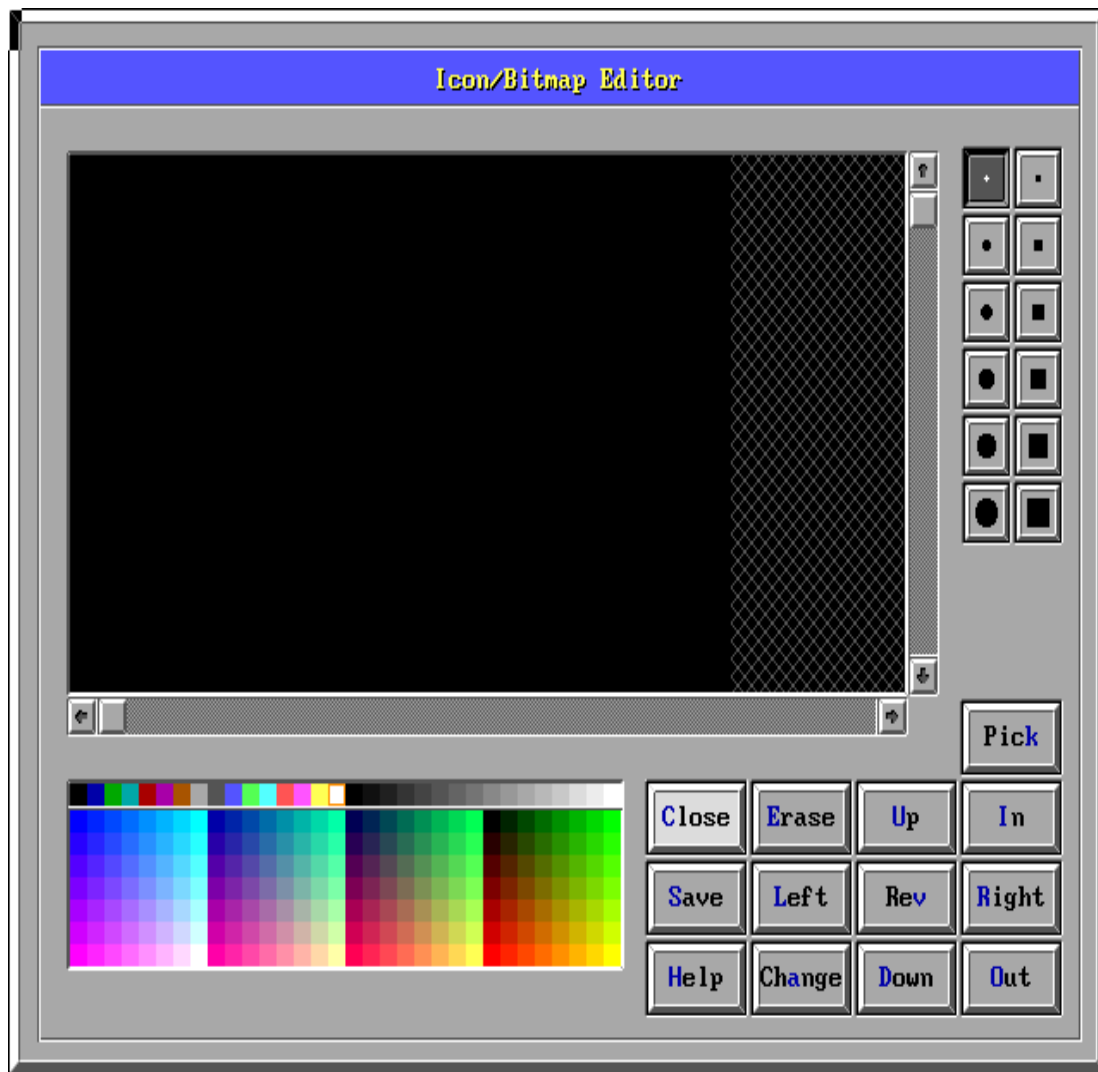
Cancel - Selecting tells RIPaint to ignore any changes in settings and return to edit mode.

Load - Loads in settings that have previously been saved.

Save - Saves settings that are in dialog box to disk.

Help - Shows you help screens for this dialog.

Icon/BMP Editor



The big box on the upper left side - This is where your icon is displayed. You can change pixels here by clicking on them. The pixels are colored based on your current drawing color and brush style. The two scroll bars on the side allow you to view and change any location in your image. By moving the squares you can move to any part of your icon. With the help of two buttons below you can zoom in and out of your icon. This allows you to look at your icon in full size and then zoom out to see it pixel by pixel to edit. Any area that is not part of your icon is displayed as a diamond shaped pattern.

The buttons on the upper right hand side - These are the different brush types and sizes that are available. Click on your choice to change your brush. The larger brushes will cover more than one pixel at a time.

Colored boxes in the lower left corner - Your current drawing color has a square around it. To change your drawing color, simply click on any colored square to choose that color.

Pick - You can pick up a color off of your icon by clicking on this button. Now move the mouse cursor over your icon. Your cursor will change into a hand pointing to the right. . Point the hand at the pixel you want to pick up the color from, and click the left mouse button. Your drawing color will be changed to match the color of that pixel.

Close - RIPaint exits the Icon editor and returns you to the editing screen.

Erase - Sets all the pixels in your icon to the current drawing color.

Up - rotates all of the pixels in your icon up one row and puts the first row in the last row.

Down - rotates all of the pixels in your icon down one row and put the last row in the first row.

Right - rotates all of the pixels in your icon right one column and puts the last column in the first one.

Left - rotates all of the pixels in your icon left one column and puts the first column in the last one.

Save - saves you icon into a disk file.

In - zooms into your icon until it is full size.

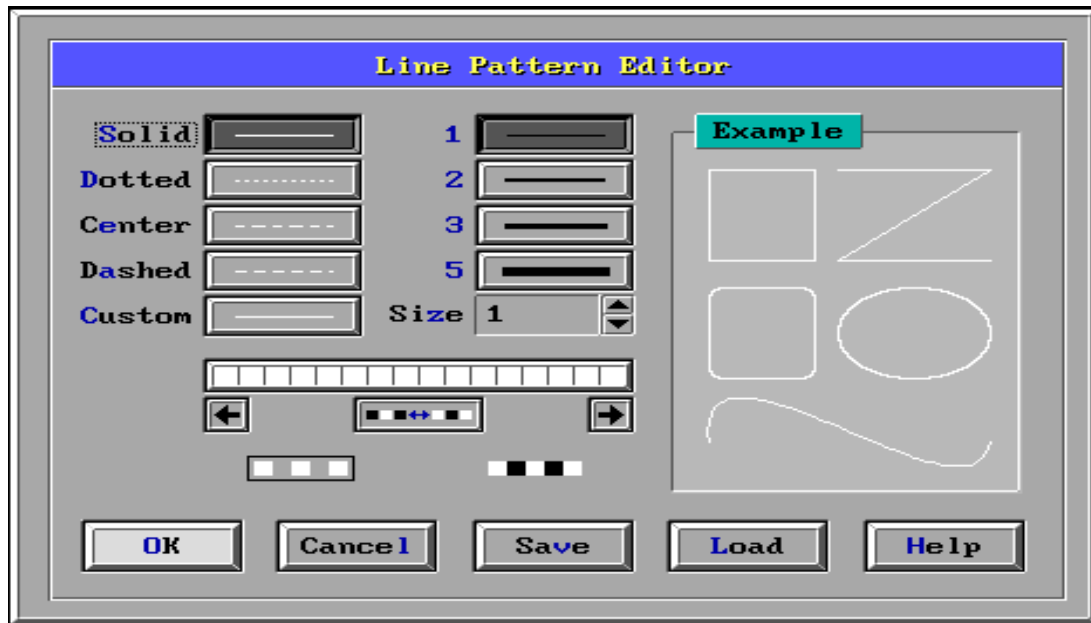
Out - zooms out your icon to make the pixels larger.

Rev - Inverts the colors of all the pixels in your icon. If the SHIFT key is held down while clicking on this it will only do a 16 color inversion.

Help - Displays help screen for this dialog.

Change - Allows you to change all the pixels of one color into another. To do this select the color you want to change as your drawing color. Next click on Change. A Color Picker box will pop up. Click on the color you want to change it to.

Line Style Editor



Solid, Dotted, Center and Dashed - These are preset line patterns simply click in rectangle to the right to chose one.

Custom - This sets the line pattern to the one you have custom made. Below there is a row of 16 boxes. You can click on them and toggle them from black to white. The arrow buttons rotate your pattern left and right. Between the two arrows is a button that will change all the white boxes in to black boxes and vice versa.

Button with white and grey squares - If this is selected, any black boxes in your line pattern will be transparent.

Button with white and black squares - If this is selected, any black boxes in your line pattern will be drawn in the current background color.

1, 2, 3, 5 - These are the preset line thicknesses. Click on the rectangle to the right of your choice to select one.

Size - This allows you to set any line thickness you want.

Example - This shows some sample objects and how they would be affected by the current settings.

OK - Clicking on this button accepts the changes you have made and returns to edit mode.

CANCEL - Clicking on this button ignores the changes you have made and returns to edit mode with the old settings.

SAVE - Clicking on this button allows you to save all the settings in this dialog box so you can load them in later.

LOAD - Clicking here loads previously saved settings into the dialog box.

HELP - Selecting this gives you more help on this dialog box.

Mouse Limits

You can limit the area where a mouse can click in the paint. This does not affect RIPterm, it like Grid Snap, allows you to precisely place objects. When set, RIPaint ignores any mouse clicks outside of the mouse borders.

To set the left mouse border

1. Hold down the ALT key and then hit the F1 function key.
2. Use the mouse to move the blue line to where you want the left side of the mouse border to be.
3. Hit ESC.

To set the right mouse border

1. Hold down the ALT key and then hit the F2 function key.
2. Use the mouse to move the blue line to where you want the right side of the mouse border to be.
3. Hit ESC.

To set the top mouse border

1. Hold down the ALT key and then hit the F3 function key.
2. Use the mouse to move the blue line to where you want the top edge of the mouse border to be.
3. Hit ESC.

To set the bottom mouse border

1. Hold down the ALT key and then hit the F4 function key.
2. Use the mouse to move the blue line to where you want the bottom edge of the mouse border to be.
3. Hit ESC.

To reset all the mouse borders to full screen

1. Hold down the ALT key and then hit the F5 function key.
2. Hit ESC.

CHAPTER 7

Text Variables

RIPaint 2

TeleGrafix Communications, Inc.

Last updated on 10/20/95

AN INTRODUCTION TO TEXT VARIABLES

A text variable represents either a command or string of text that both RIPaint and RIPterm know something about. Each text variable has a name that you use to reference it. A text variable's name has a dollar sign before and after it. For example, the text variable called \$DATE\$ holds the current date on your PC. Some text variables cause a RIPscrip compatible terminal to do something when they are received. \$STATBAROFF\$, for example, when received by a RIPscrip compatible terminal, causes the Status Bar to disappear. The host may ask RIPterm what the values of one or more of these variables are, and if RIPterm has a text variable by that name, it will return the data contained in it to the host.

There are three types of text variables.

1. Built-in text variables that RIP *scrip* products will ALWAYS know about. These include text variables like \$DATE\$ and \$TIME\$ that return a value.
2. Another type of built-in text variables are Active text variables, which perform an action, but return nothing to the host. These include turning the status bar on/off, clearing the graphics screen, and playing some simple sounds, and many more. These variables are a very powerful aspect of RIP *scrip*, providing mechanisms for doing dialog boxes and interactive GUI applications.
3. Then there are also User text variables that can contain a variety of information depending on what the user entered at the time the variable was created. For example, the host might ask you what the contents of the \$FULL_NAME\$ variable is, and if RIPterm doesn't know, it could pop-up a field on the screen and ask you about it. From then on, RIPterm will remember that piece of information for the next time it is needed by a host.

Text Variables

You may use either the pre-defined text variables, or the user text variables at any place that allows text variables.

Some built-in text variables have been extended in RIP *scrip* 2.0 to allow for parameters. This extends text variables functionality in many ways. If a text variable has a parameter, it is enclosed in parenthesis immediately after the text variable name as in the following example:

`$SAVE(8)$`

This works the same as the older `$SAVE8$` text variable which saves the screen to the eighth slot. The new method is more universal in design than having separate text variables for basically identical operations. The older forms of these commands will remain in the RIPscrip language. However their use is not recommended because the new method is far superior.

Not all text variables which take parameters need them. If no parameters are given, default parameters are assumed. In any case, these text variables will have their parameters described in the Appendix A.

If a variable takes more than one parameter, then they are separated by commas between the parenthesis. One example is `$ETW(0,1)$`.

A complete listing of all pre-defined text variables (both data and active) is found in Appendix A.

PRE-DEFINED TEXT VARIABLES

A pre-defined text variable is either a data text variable, or an active text variable. A data text variable is a text variable that inserts a piece of text wherever the text variable is used. For example, the sequence `$DATE$` might get replaced with 09/19/94. This is a simple example of a data text variable. An active text variable on the other hand does something (usually). They normally don't get replaced with other text information. For example, the text variable `$SAVE$` saves the contents of the screen to a disk file that can later be restored with `$RESTORE$`. In these situations, active text variables are removed from whatever text message they are present in (they are still activated though).

Text Variables

A complete list of all pre-defined text variables is found in Appendix A.

WHAT ARE USER VARIABLES?

A user variable is a text variable that RIP *scrip* doesn't know exists. They are custom-defined text variables that contain information that the terminal user will fill in. If a variable already contains information, a host can it can request the data that variable contains without the user having to do anything. Each user variable has the possibility of being secured. If a host requests access to a user variable that has been secured, the RIPscrip compatible terminal will ask for your permission to send the information.

Examples of text variables might be:

Variable name	Variable Contents
\$FULL_NAME\$	George Washington
\$COMPANY_NAME\$	US Army
\$AGE\$	55
\$DATEOFBIRTH\$	2/11/23
\$PHONENUMBER\$	(714) 379-2141

A RIPscrip compatible terminal will "keep track" of these variable names and contents for you. You can tell the terminal to store these values permanently, only during the current session, or only for a brief moment.

When RIPterm needs to know contents of a variable it will display a pop-up dialog box, asking you to enter the information or if the information it has is correct.

Lets take an example. You are the system operator of a large RIP *scrip* host. As you have read, RIP *scrip* can take advantage of database-like ability on the terminal end. If you can alter your host to ask questions with RIP *scrip* text variables built in, you can have the terminal calling your host automatically fill in questionnaires. Imagine if a user could sign-up on your host without having to type more than a single keystroke (i.e., "YES, this information is correct"). With user text variables, you can do this very thing.

QUERYING TEXT VARIABLES

Data Query is a special RIP *scrip* command that can be used to ask the contents of one or more text variables.

Lets take a simple example. You want to ask the terminal program for the user's address information. You could do this with the following query.

```
$FULL_NAME$^m$COMPANY$^m$ST_ADDR$^m$CITY$, $STATE$  
$ZIP$^m
```

Note: ^m is a replaced with a carriage return. It's the same as if you had hit the ENTER key after each value was returned.

This would query from the terminal the contents of 6 text variables, and format them in a manner similar to any normal address on an envelope. The results of this query might send the following back to the host :

```
Joe Sixpack  
ACME Corporation  
13631 Palindrome Parkway  
Surf City, CA 92649
```

If a text variable is queried, and it has not been defined yet, a pop-up dialog will appear asking the user to enter the information.

Under normal situations you reference text variables by simply placing dollar signs (\$) around the variable name. Anywhere where that text variable occurs in a Host Command, Query, or other related place, that can contain text variables, will have its contents replaced with the associated information.

For example, if the text variable \$FIRST_NAME\$ was received, and RIPterm had no information saved for that variable, RIPterm would pop up a dialog asking the following:

Please enter FIRST_NAME:

Text Variables

When the user enters the information, this data is inserted where the `$FIRST_NAME$` variable was located, and the contents of that variable are lost after that moment.

DEFINING PERMANENT VARIABLES

What is truly needed is an ability to preserve that information (either on Disk, or in memory) so that it can be used later on.

There are six basically different text variable reference modes, each of them takes a single command character added between the first dollar sign and the beginning of the variable name. Those characters and their significance are shown in the following chart:

Code	
*	An answer is required
+	Save variable to database permanently
=	Save to internal memory table (lost when RIPterm hangs up)
#	Do not echo keystrokes (show #'s instead). This is useful for things like entering passwords.
-	Used in conjunction with a default response (see below). When this option is used, the value of the variable is set to the default value and the user is not prompted for any data entry (transparent data variable define). Nothing is returned to the host in this mode (unless transparent retrieval mode is used - see below).
&	Transparent data variable retrieval. This allows the host to retrieve a text variable from the terminal and the user is not prompted to modify the information.

If you do not specify the "+" or the "=" directives to actually save the text variable's contents, then the data will be passed on as part of whatever host command the text variable expression was a part of, and will not be saved. To save the text variable permanently, you must specify the " +" command which will store the variable in some kind of internal database file for permanent storage. If you wish to only save the variable temporarily (e.g., for the duration of the current session), use the " =" directive instead - this saves the variable in an internal memory table.

Text Variables

To ask for the FIRST_NAME text variable that must be filled in, and to instruct RIPTerm to save the variable to the local database, you would use the following text variable command syntax:

```
$*+FIRST_NAME$
```

The four command characters (*, +, = and #) can be in any order, but can only appear once in the text variable statement - additional occurrences of them are ignored.

You may specify how wide the data entry field for the text variable is. To do this, simply put the number of columns after the variable name with a comma (,) in between (e.g., \$NAME,10\$).

As you may have noticed earlier, if a text variable is referenced without being previously defined, it will display a generic question to prompt the user. You have the option to specify a custom question. The syntax is similar in nature to the syntax of the host command/text labels of the pop-up pick lists described below. In order to prompt with a particular question, after the variable name place an at-sign (@) followed by the question, then the final dollar sign as in the following example:

```
$FIRST_NAME,20@What's your first name?$
```

In the question text, you are not allowed to use dollar signs at all.

TEXT VARIABLE REFERENCING MODE INTERACTION

Some discussion needs to be made about the six various text variable referencing modes described earlier and how they interact. The two modes " =" and " +" are used to actually store (preserve) the text variable's contents for a period of time.

The transparent data define mode (" -") and transparent data retrieval mode (&) are used by the host to interact with the user without having to prompt the user for any data (e.g., the operation is "transparent" to the user).

There are four basic combinations of transparent operations. Each of which have specific sub-categories depending on the existence of the

variable, existence of the default response, and any form of data security (see the next section):

1. "Transparent Define" and "Transparent Retrieve" Are Both Set:

This combination indicates that the definition of the variable should not involve the user at all. In addition, the final result of the definition should be sent to the host system without the user's involvement. There are basically four general categories for this configuration that depend on the existence (or lack thereof) of the variable itself, and the presence of a default response:

a) Variable Exist and a Default Response Exists

Since both the variable and default response exist, the contents of the variable are supposed to be re-defined with the new default response. If this variable has no security associated with it then its contents are updated with the new value and that value is sent to the host. If security is linked to this variable, then the user is prompted with a dialog informing them that the host wants to re-define the variable and to ask if it should be done. If the user indicates yes, then the variable is re-defined and its new contents are sent to the host. If the user specifies no, then the variable isn't updated and the phrase "DENIED" is sent to the host system to indicate that the user rejected the operation.

b) Variable Exists but the Default Response Doesn't

Since the variable exists, but no default response is specified, this operation is supposed to "clear" the contents of the specified text variable. If no security is associated with the

variable then its contents are cleared (i.e., made blank). The variable is not deleted - it remains in existence, but has no contents. Nothing is sent to the host because the variable now contains nothing. If the variable has security associated with it, then the user is prompted with a dialog stating that the host wants to clear the variable's contents. If the user specifies yes, then the variable is cleared and nothing is sent to the host. If the user chooses no, then the variable is left untouched and the phrase "DENIED" is sent to the host.

c) Variable Doesn't Exist, but the Default Response Does

In this situation, the default response is used to create a new variable with no security associated with it. The default response is sent to the host system.

d) Neither the Variable nor the Default Response Exist

The variable is created with blank contents and no security associated with it. Nothing is sent to the host.

2) "Transparent Define" Set but "Transparent Retrieve" is Not

Under this configuration, a variable is created but nothing is sent back to the host. Even if the operation wasn't allowed for some reason, nothing is sent to the host. This is used in situations where the host doesn't want to know any kind of status of the operation. There are basically four sub-categories for this text variable mode:

a) Variable Exist and a Default Response Exists

Since both the variable and default response exist, the contents of the variable are supposed to be re-defined with the new default response. If this variable has no security associated with it then its contents are updated with the new value. If security is linked to this variable, then the user is prompted with a dialog informing them that the host wants to re-define the variable and to ask if it should be done. If the user indicates yes, then the variable is re-defined to the new contents. If the user specifies no, then the variable isn't updated. Nothing is sent to the host in this mode.

b) Variable Exists but the Default Response Doesn't

Since the variable exists, but no default response is specified, this operation is supposed to "clear" the contents of the specified text variable. If no security is associated with the variable then its contents are cleared (i.e., made blank). The variable is not deleted - it remains in existence, but has no contents. If the variable has security associated with it, then the user is prompted with a dialog stating that the host wants to clear the variable's contents. If the user specifies yes, then the variable is cleared. If the user specifies no, then this operation does nothing. Nothing is sent to the host in this mode.

c) Variable Doesn't Exist, but the Default Response Does

The variable is created with its contents set to the default response. The variable is created with

no security associated with it. Nothing is sent to the host system.

d) Neither the Variable nor the Default Response Exist

The variable is created with its contents blank.
The variable has no security associated with it.
Nothing is sent to the host system.

3) "Transparent Define" not Set but "Transparent Retrieve" is

This mode requests that the terminal read a text variable's contents and send them to the host system without the user's involvement. This is the most frequently used mode for "querying" data from the terminal without the user being involved. In this mode of operation, the default response is ignored because you are not setting the variable to any values. There are basically two situations that can occur under this mode:

a) The Variable Exists

If no security is associated with this variable, then its contents are sent to the host system. If security is associated with the variable, then the user is prompted whether or not it is OK to send the variable to the host system. If the user specifies yes, then the variable's contents are transmitted. If he says no, then the phrase "DENIED" is sent to the host system.

b) The Variable Doesn't Exist

If the variable does not exist, then the phrase "NONE" is sent to the host to indicate that it doesn't exist. Note that this is the only situation where "NONE" is used to indicate that the variable simply doesn't exist.

4) Neither "Transparent Set" nor "Transparent Retrieve" Are Set

a) Variable Exist and a Default Response Exists

The user is prompted with the <default> value. He are allowed to change it. If the user is happy with his operation, the final result is transmitted to the host. If the user cancels the operation, "DENIED" is sent to the host. If the operation was successful, the variable is updated with the new result without changing its security configuration.

b) Variable Exists but the Default Response Doesn't

The user is prompted with the contents of the variable to change. If after possible modification, the user chooses OK, then that final result is sent to the host. If the user chooses cancel, then "DENIED" is sent to the host. If the operation was successful, and the variable is to be "saved," then the variable is updated with the new result without changing its security configuration.

c) variable Doesn't Exist, but the Default Response Does

The user is prompted with the <default> response. If the user chooses OK, then any final result will be transmitted to the host. If he chooses cancel, then the value "DENIED" is sent to the host. If the operation was successful then the variable is created with the final result

and its security status is set to "off" (unless the user had some ability to "force" it on).

d) Neither the Variable nor the Default Response Exist

The user is prompted with a blank data entry field. If the user chooses OK, then any final result will be transmitted to the host. If he chooses cancel, then the value "DENIED" is sent to the host. If the operation was successful then the variable is created with the final result and its security status is set to "off" (unless the user had some ability to "force" it on).

If nothing is sent to the host and the text variable isn't to be saved to the database or to memory, then the text variable operation can be omitted entirely - since it would produce no functional results.

During transparent define or retrieval modes, it is possible for the user to have to be prompted for some information (as we noted earlier). This only occurs when some form of "security" is involved.

The "answer is required" setting forces the user (if prompted) to make some kind of selection (i.e., he can't abort the text variable operation).

POSITIONING THE TEXT VARIABLE QUERY WINDOW

You have the ability to set the X/Y location of the pop-up window that asks for the text variable. This gives you control over the location of the window. The way you do this is by adding some coordinate information before the variable name followed by a colon. An example of this would be as follows:

```
$10,10:FIRST_NAME,20@What's your first name?$(
```

These coordinates are specified in normal decimal format. If one or either of the number are omitted then the dialog is centered either horizontally, vertically or both. Some examples of this are as follows:

Text Variables

<code>\$.50:NAME\$</code>	Centered horizontally
<code>\$50,:NAME\$</code>	Centered vertically
<code>\$.:NAME\$</code>	Centered both horizontally and vertically

If you omit the X/Y specification codes entirely (e.g., `$NAME$`), then the actual location of the pop-up dialog is up to the discretion of the *RIPscrip* software.

X/Y coordinates are in current "world coordinates".

DEFAULT VALUES FOR TEXT VARIABLE QUERIES

You may supply a default value for the text variable. This default value is used if the variable doesn't already exist. When the default value is used it is displayed in the data field's edit region and you have the option of changing it.

To supply a default response, you specify the default contents after an equal sign (=) which must be placed after the variable name (and width parameter and question parameter if applicable). Some examples are:

```
$STATE=Ca$  
$STATE@What state do you live in?=Ca$
```

User defined text variables do not always require a specific response (unless the "*" flag is specified indicating that a response is required). If the user chooses to ignore the request (i.e., hitting CANCEL or whatever), then a value of " NONE" is inserted in place of the user data text variable.

USER DEFINED VARIABLES AND DATA SECURITY

When working with user-defined text variables, the key issue is security. What if you had credit card information stored in a permanent text variable named `$CredCardNo$` and the host system asked your terminal for that information? Obviously, the user should be informed about this request. Situations might arise where security is not a concern - as in a closed environment such as an internal office system using *RIPscrip*.

Text Variables

Some text variables may have data security enabled for them (as in the credit card number above), and others may not. For situations where the host system transparently defines a text variable in the database, it will want to ask the terminal for that variable at various times where the user shouldn't be involved. When an operation occurs where a text variable data entry dialog is displayed asking the user to modify or create a text variable, a check box is present on that dialog which determines the security status of that variable. This allows the user to enable or disable security on a per-variable basis. This lets you enable security for things like credit card information, but turn it off for things like your favorite color (things you don't mind if the host knows about).

DATABASE AND MEMORY VARIABLES - PRECEDENCE

Whenever a data text variable is queried, *RIP scrip* always looks for the variable first in the memory variable list. If it finds the variable there, then that variable's contents are retrieved and used for the operation. If a memory variable doesn't exist, then the variable database is checked.

When variables are defined, you must explicitly state what kind of a variable you want to save (i.e., database or memory). If you specify a database variable and that variable cannot be created (e.g., insufficient disk space, etc.), then the variable will be created in memory instead (a fallback plan).

Let's assume that you have the following variables defined:

Memory Variables	
CITY=New York STATE=NY	CITY=Philadelphia STATE=PA

If you performed the database operation `$&CITY$` (i.e., a transparent retrieve operation), then the text "New York" would be sent to the host system. If on the other hand, you specified the operation `$+CITY$` (fully interactive with saving to the database), then a dialog would appear with "New York" in the data entry field. If the user changed that to say "Chicago" and selected OK, then the value of "Chicago" will be saved in the database variable CITY, overwriting the older "Philadelphia" value. This shows how you can have two variables with the same name in different places - the permanent variable could be thought of as a long-term variable, and the memory variable can be thought of as a "session"

Text Variables

variable. This could be most useful in an example where you have a database system on your host where the user uses a particular database query all the time. If on the other hand, the user wanted to use a temporary query while he/she is on-line, you could set that temporary query up as a memory variable and that one will be found instead of the database variable. When the user logs on to the host next time, the memory variable is gone and they're back to their original "default" query expression.

It should be noted that when deleting a text variable (via the `$RESET(TV,var-name)$` command), memory variables are deleted first and then the database variables. To ensure that all variables (both memory and database) are deleted, you would want to specify the same reset operation twice.

USER DEFINED DATA VARIABLE FORMAT OPTIONS

Just like text variable parameters defined earlier for built-in text variables, user defined text variables have the same luxury. These parameters define the format of the entered data. You are allowed up to two parameters for a user-defined text variable. The first one is the "mode" parameter which designates what type of data the user can enter. The possible settings for mode are:

Mode	
ANY	Any character is allowed
ALPHA	User can enter alphabetic characters only (A-Z, a-z)
NUMBER	User can enter numeric characters only (0-9)
ALPHANUM	User can enter alphanumeric only (A-Z, a-z, 0-9)

The second parameter is an optional conversion designator. The possible settings for this parameter are on the following page:

Description of Conversion	
TONAME	Convert the data to "name" format. The first letter is capitalized and all subsequent characters in a word are set to lower case automatically (e.g., "TOM SMITH" becomes "Tom Smith") No special processing is performed on names like "McDonald", etc.

Text Variables

TOUPPER	Convert the data to upper case.
TOLOWER	Convert the data to lower case

You may omit one or both of the parameters. If both are omitted, then it will default to "ANY" data with no conversion of any kind. You may omit the mode parameter if you wish, and specify only the conversion parameter. Under no circumstances should a conversion parameter be allowed before a mode parameter - this is considered a syntax error.

Some examples of legal and illegal user variable queries might be:

Legal \$USERDATA\$
Legal \$USERDATA()\$
Legal \$USERDATA(ANY)\$
Legal \$USERDATA(ALPHA,TOUPPER)\$
Legal \$USERDATA(TONAME)\$
Illegal \$USERDATA(TONAME,ANY)\$
Illegal \$USERDATA(ANY,ALPHA)\$
Illegal \$USERDATA(TONAME,TOUPPER)\$

By default, a format option of "ANY" is assumed when no parameters are specified.

LIMITS ON LENGTH OF VARIABLE NAMES, ETC.

Valid text variable names may contain alpha characters, numbers, or an underscore ("_"). The first character **MUST** be an alphabetic character. Any text variable name is automatically converted to upper case.

The maximum length of a text variable name is 20 characters. The maximum number of text variable parameters (the ones appearing between parenthesis, not the text string) is 40. The maximum size of a text variable parameter is 12 characters. These are only available to text variables that are part of the RIP *scrip* specification. The maximum size of a question or default response is 100 characters. Anything longer than this length is truncated. The maximum length of a text variable text field is 255 characters. Anything longer than this length is truncated.

EXAMPLES OF USER-DEFINED TEXT VARIABLES

Below are some examples of user defined text variables as might be used in a real world situation:

```
$*+20,50:NAME(ToName),30@What's your name?=John Doe$
```

This is about as complex as they get. Going from left to right, let's look at what all the codes mean. First, the "*" means that a response to this request is required (they cannot hit ESC or CANCEL to get out of it). The "+" means to save the response to the internal database permanently. The sequence "20,50:" means to place the dialog box's upper left corner at location (20,50) on the screen. The "NAME" is the name of the data variable. The format option "(Name)" means that this should be a formatted name field where the first letter of any word should be capitalized and the remaining characters should be set to lower case. The value ",30" after the variable name/format indicates that this field is up to 30 characters in length. "@What's your name?" indicates that the question "What's your name?" should be displayed when the user is prompted for the information. Finally, the "=John Doe" indicates that if the field doesn't exist, it should be filled in with "John Doe" before any editing is allowed. This gives the user the ability to choose a default name if they wish.

```
$*#20,10:PASSWORD,10@Please enter your password$
```

This example is a required data variable that echoes #'s in place of the keystrokes you entered. The field is placed at (20,10) on the screen with a variable name of PASSWORD. The field is 10 characters wide and the prompt is "Please enter your password".

CHAPTER 8

Misc. Host Commands

RIPaint 2

TeleGrafix Communications, Inc.

Last updated on 10/20/95

MISCELLANEOUS HOST COMMANDS

There are many Host commands that allow you to do special things. Some are actually active text variables that let you playback/display different file types locally.

LOCAL RIP *scrip* FILE PLAYBACK

You can re-play a .RIP file that your user has locally on his/her hard disk from any place that allows text variables. After the initial dollar sign (\$), enter the greater-than symbol (>) followed by the filename (with or without the .RIP extension), then end in another dollar sign (\$). Several examples of this are as follows:

```
$>MYFILE.RIP$  
$>FILE1$  
$>FILE1.RIP$$>FILE2.RIP$$>FILE3$
```

Note in the last example, a file extension other than .RIP was used. You are not limited to playing back local .RIP files. In fact, you can playback any file you want. You could load any simple text file, ANSI picture image, or other such thing. If the file is a .RIP file, it will replay any graphics that were in the file and if any Mouse Regions are defined, it will create those fields for you as well, thus allowing you to pop-up dialog screens or other such things that are not built-in to RIPterm normally.

Each "local RIP playback" variable you enter will search for the .RIP file in the current host's icon directory. If it cannot find the file in that directory, it will check the ICONS\ directory.

LOCAL AUDIO FILE PLAYBACK

You can also play a .WAV sound file that is located on the user's hard drive. This command is nearly identical in syntax to the local RIP file playback command with a simple alteration. Instead of using the ">" character, you use the close parenthesis ")" as in the following example:

```
$)AUDIOFILE.FIL$  
$)TRAIN.WAV$
```

The file extension is unimportant.

LOCAL BITMAP/ICON DISPLAY

This command places a local bitmap (.BMP) file onto the screen. The bitmap will be displayed inside the current image settings as defined by the RIP_IMAGE_STYLE command and will adhere to the settings of that command. (See \$IMAGESTY\$ in Appendix A for instructions on setting RIP_IMAGE_STYLE from a Host Command)

For the purposes of universality, the bitmap is shown to the screen using the current screen's color palette and "auto-dithering" mode is used for the viewing of the image. If you need support for some of the other modes for viewing a bitmap image, you will have to use RIPaint's Load BMP command. This command is provided as a simple method of showing a bitmapped image from within a host command.

Note, if no image style definition has been recorded then the bitmap is shown in the maximum size of the current viewport. In other words, it will be scaled to fill up the entire screen

Here's an example:

```
$<FILENAME.BMP$
```

LOCAL PHOTO DISPLAY

This command displays a JPEG file based on the current image style (set with a RIP_IMAGE_STYLE) command. (See \$IMAGESTY\$ in Appendix A for instructions on how to set RIP_IMAGE_STYLE from a Host Command) If no image style is recorded, then the JPEG file will cover the entire screen. This command uses a syntax similar to the local bitmap playback operation but instead uses the "(" character instead of the "<" one.

To playback the local JPEG file MYFILE.JPG, issue the following command:

```
$(MYFILE.JPG$
```

CONTROL CHARACTERS IN HOST COMMANDS

Not all BBSs will allow you to use control characters on their Service. Regardless the capability to send any Control Character exists from your Host Commands, just use the form on the left to represent them. The most commonly used Control Characters are:

^@	Null (ASCII 0)	^[A	Up Arrow
^G	Beep	^[A	Up Arrow
^L	Clear Screen (Top of Form)	^[C	Right Arrow
^M	Carriage Return	^[D	Left Arrow
^C	Break (sometimes)	^[H	Home Key
^H	Backspace	^[K	End Key
^[	Escape character	^[L	Control Home
^S	Pause data transmission		
^Q	Resume data transmission		

Some hosts use the ^ (caret) for their own purposes. In these cases, you can use the ` (back quote) character instead of the caret. Some systems allow you to specify the caret symbol as two carets (^^). Consult your Host Software documentation to determine the best method for your needs.

Note: Galacticom Major and World Group BBS's require you to use the ` (back quote) character instead of the caret. You can usually find the back quote character on the bottom of the key with the ~ (Tilde)

POP-UP PICK LISTS

Any place that you can use a Text Variable (Queries, Button and Mouse Field return strings, and Keystroke Macros), you can take advantage of a unique feature of RIP *scrip* - Pop-Up Pick Lists. A Pop-Up Pick List is host command that tells RIPterm to pop up a dialog allowing you to choose from one of several available values. Whichever entry in the list you choose will replace the Pop-Up List in the Host Command.

A list is created by putting the special list instructions inside two sets of parenthesis like this: ((and)). The list may have an optional question followed by two colons (::), followed by one or more list entries. For example, ((Send Email to?::Sysop,Cosysop,Joe)) pops up a dialog that asks the user "Send Email to?", giving him/her the choices of "Sysop", "Cosysop", and "Joe".

If you do not specify a question, then the following default question will be used:

Choose one of the following:

If the user hits ESC instead of picking an entry in the list, then nothing will be inserted into the text of your Command. You can indicate that the user must pick an entry by putting an asterisk (*) at the beginning of the question. For example, ((*Send Mail to?::Sysop,Joe)). This would make it so that the user must choose either Sysop or Joe.

In the previous examples, Sysop and Joe are the text responses that are inserted into your Host Command. These commands are also the same things that are displayed in the listing. If you want to use something else in the listing instead of the return text, you can. When you make the list entry, add an @ description to the end of it. For example:

```
((Send Mail To?::Sysop@Head Honcho,Cosysop,Joe))
```

If the user selected Head Honcho, Sysop would be placed in the Host Command instead of the displayed choice of Head Honcho.

You may specify up to 64 entries for any one list. In RIPterm version 1.52 and earlier, the total length of a pick list was 256 bytes. In version 1.53 and later, this limit has been increased to 1024 bytes. In earlier revisions of RIP *scrip*, a maximum number of 20 entries in a pick list were allowed. This has been expanded to 64 for version 2.0.

Examples:

```
((Send E-Mail to?::Sysop,Joe,Mike))  
((*Send E-Mail to?::Sysop@The Head Honcho,Joe,  
Mike@My Brother))
```

USING HOT KEYS WITH POP-UP PICK LISTS

One feature of Pop-Up Pick Lists allows you to specify a hot key for each entry in the list. For example, if you wanted the first character of each entry to be highlighted (thus allowing you to select that character to activate the entry), simply put a tilde (~) or an underline (_) before and after the keystroke. For example "_S_ysop" would highlight the "S" in "Sysop".

You can highlight more than one character, but only the first one will be the active hot key. If you omit the second tilde or underline, then the remainder of the description will be highlighted.

NOTE: If you use a tilde or an underline in the Text Response (not the description), then those characters are inserted into your Host Command when it is transmitted to the host. You probably don't want to do this. Recommendation: only use hot key features on list entries where you specify a description!

Examples:

```
((::Sysop@_T_he Head Honcho,Joe,Mike@My _B_rother))
```

SPECIAL CHARACTERS IN POP-UP PICK LISTS

Some characters have special significance in the RIP *scrip* language. These characters are ! (exclamation mark, or for you UNIX-heads, "bang"), \ (backslash), and | (vertical bar). To use these characters in a Text Response, they must be preceded by a backslash (! becomes \!, \ becomes \\, and | becomes \|). RIPaint automatically adds these when creating Text Responses. You need to be aware of this only if you are editing RIP *scrip* files with a text editor. The _ (underline) and ~ (tilde) characters used to indicate the hot key in a Text Response are not able to be preceded by a backslash to be used by themselves. They will be returned to the host if they exist in a Text Response (not in the description), however everything after the underline or tilde will be underlined, and the first character will be considered the hot key.

POSITIONING A POP-UP LIST BOX

You may specify an X/Y location for the upper-left corner of your pop-up pick-list dialog box. To do so, specify the location (in world coordinates) in the following manner:

`((x,y:Question::option1))`

Notice the "x,y:" sequence immediately before the question text. If you omit this sequence then the dialog should be centered both horizontally and vertically on the screen. You may omit either one of these values to indicate that the dialog should be centered in that direction, but you must specify the comma (,) as in the following examples:

<code>((20,30:Question::option1))</code>	Place dialog at (20,30)
<code>((,30:Question::option1))</code>	Center horizontally, 30 vertical
<code>((20,:Question::option1))</code>	Center vertically, 20 horizontal
<code>((,:Question::option1))</code>	Center vertically and horizontal

It should be noted that if the "response required" directive is present (an asterisk), then it goes before any X/Y location data as in the following example:

`((*20,30:Question::option1,option2,option3))`

CHAPTER 9

Templates

RIPaint 2

TeleGrafix Communications, Inc.

Last updated on 10/20/95

TEMPLATES

A template is a special variety of a host command that is used to construct other host commands. They are only used with the RIP_BUTTON command, not with RIP_MOUSE or RIP_QUERY. Their use is solely dedicated to the Button command. In fact, their usefulness is predominantly designed for Radio Buttons and Check Box Buttons but this doesn't mean that you cannot use them for other purposes.

A template, like a normal "raw" Host Command, is stored in the Host Command field of the RIP_BUTTON command. Unlike Raw Host Commands that get sent to the Host immediately, templates do not transmit immediately. In fact, it's possible for a Template to never get transmitted to the HOST at all. Templates are not normally sent directly to the HOST - they are almost always used in conjunction with some other button's host command.

With normal Mouse Host Commands, you can send any piece of text you want when the user clicks on that button. You could send the word "HELLO" to the HOST, for example, if they click on a certain mouse button. This is an example of a "Direct Host Command".

There are three types of Host Commands. There are:

1. DIRECT HOST COMMANDS - Sends a string of text to the host immediately after the associated button is clicked.
2. TEMPLATE DEFINITIONS - Defines a template to be used by other buttons (if ever). See below for further details about how to define templates.
3. TEMPLATE EXECUTION - Allows you to plug a piece of string data into one or more templates (defined previously). The resulting string is then acted upon like a Direct Host Command and transmitted to the HOST immediately.

The RIP_BUTTON command "segments" its Text Parameter Block into three portions - the Icon File, Text Label followed by the Host Command

block. Each of these segments is separated by the two character delimiter "<>" like this:

```
ICONFILE.ICN<>Button Label<>HOST COMMAND
```

With the Button command, the Host Command segment can be subdivided into numerous smaller sub-segments, or Command Blocks. This is done with another two character delimiter "[]". So, technically, you could do this:

```
ICONFILE.ICN<>Button Label<>HELLO^m[WORLD^m
```

This command would show an Icon Button using the file ICONFILE.ICN as its Icon Image, labeling it with the phrase "Button Label", and defining an extended Host Command block with two segments. If the user clicks on this button, the following will be sent to the HOST:

```
HELLO<cr>
WORLD<cr>
```

Notice how the "[]" is not transmitted. This is because it is simply a delimiter separating two Command Blocks from each other. Now each of these two command blocks are DIRECT HOST COMMANDS, but they don't have to be. One of them could have been a Direct Host Command, and another could just as easily have been a template definition.

BASIC TEMPLATE MECHANICS

There can be up to 36 different templates defined simultaneously. Each template number corresponds directly to a Button Group Number. Templates are identified by a single MegaNum 0-9 and A-Z, leaving 36 distinctly separate groups. To define a template, you use a variation of the Command Block delimiter "[]" with the template identifier followed by a colon like this:

```
[5:]This is template #6's definition
[G:]This is template #16's definition
```

Defining a template is simple. Activating a template however, is another story. What if three buttons in the same group all define their own templates for the same group/template like this:

Button #1: [5:]This is button #1's template
Button #2: [5:]This is button #2's template
Button #3: [5:]This is button #3's template

Now, when these three buttons are received by the terminal, it [the terminal] knows the Host Commands for each button (it memorized each of them). Now, which of these three templates is the currently active one? None of them! A template definition doesn't become the active template until that button containing its definition is clicked (selected). What this means is, you can have a bunch of buttons all belonging to the same Button Group with their own respective template definitions, but only the template relating to the most recently clicked button will be the currently active template for that group.

There are two ways of activating a template:

1. Draw a button as "pre-selected" - in other words, the button is drawn pre-clicked immediately when it is received by the terminal. When this happens, the Host Command for that button is processed immediately and if a template definition exists in that host command, it is acted upon immediately thus making that template the currently active template for that Group.
2. The user clicks on a button containing a template definition. If a template is already active in that group, it is overwritten by the newly activated template.

In either case when templates are defined, nothing is actually transmitted to the HOST unless the Command Block contains some Direct Host Command sub-block(s) like this:

[5:]Template definition[]hello world^m

If this button were clicked, then Group number 5 would have the template activated with the template text "Template definition". Then the Direct Host Command "hello world^m" would be transmitted to the HOST. In this example, we see how a single host command can do multiple things - in this case, it defined a template in group #5 AND transmitted something to the HOST!

RADIO BUTTON TEMPLATES

Radio Buttons are a "type" of button group. Only one button in that group can be active (clicked) at any one time. If a button that is not active is clicked, any other buttons in that group that ARE clicked are de-selected and the one that is being clicked is selected. If that newly clicked button has a Host Command, it is processed. If it has a template definition, it too is processed, overwriting the currently defined template for the specified group. Since Radio Buttons can only have one currently active button in a single given Radio Group, similarly you can only have one template active for that group at any given moment.

Let's use a simple example to see step-by-step how templates are maintained internally.

Radio Button Templates

Let's say you have four buttons in Button Group #3, and that button group is defined as a Radio Button group. Here are the host command definitions for each of those buttons in this example:

Button #1:	[3:]ABCD
Button #2:	[3:]EFGH
Button #3:	[3:]IJKL
Button #4:	[3:]MNOP

Now, if none of the buttons are clicked, then template group #3 is blank. If button #1 is clicked, the template for group #3 would be defined as "ABCD". Now, if button #4 is clicked, what would the template definition for group #3 be? That's right, "MNOP". Notice how only one of the given templates is active at any given moment.

Check Box Templates

Check Box Buttons are another type of button group. Unlike Radio Buttons, Check Box buttons can have zero or more buttons active (clicked) at any one time. If a check box button group has ten buttons defined in it, zero, five or all ten of those buttons can be active simultaneously. What about each of their respective Host Commands? They too are all processed when the buttons are individually clicked. Now what about templates? Since you can only have one template defined in a group at any given moment, how do check box buttons accomplish this multiple-template concept? Whenever a check box button is clicked (or unclicked), the template for that group is re-calculated. Any buttons in that group that have template definitions are scanned, and any check box buttons in that group that are selected have their template definitions concatenated together (strung together) end on end. The result is one large template which is built up from the template definitions of each selected check box button.

As stated previously, check box templates can be strung along together to make a larger template - template construction of sorts. Let's say you have a Button Group #2 defined as a check box group with 7 buttons defined in it. Each of the buttons are initially drawn as "unselected", or unclicked (inactive). Here are the button host command definitions for each of the 7 buttons:

Button #1:	[2:]Apples^m
Button #2:	[2:]Oranges^m
Button #3:	[2:]Cherries^m
Button #4:	[2:]Grapes^m
Button #5:	[2:]Pears^m
Button #6:	[2:]Bananas^m
Button #7:	[2:]Lemons^m

Now, if all 7 buttons are not clicked, then template group #2 would be blank. Let's click on some buttons and see what the template will become as we change which buttons are clicked and which aren't:

Button #3 (on)	Template: Cherries^m
Button #5 (on)	Template: Cherries^mPears^m

Templates

Button #3 (off) Template: Pears^m
Button #5 (off) Template: <blank>
Button #2 (on) Template: Oranges^m
Button #1 (on) Template: Apples^mOranges^m
Button #4 (on) Template: Apples^mOranges^mGrapes^m

As you can see, the active template for a check box group is actually a combination of all selected buttons' templates, in ORDER OF DEFINITION, not in the order that they were clicked. Pay close attention to the end of the example where buttons were clicked in the order of 2, 1 then 4. If you notice the active template though, they are in 1, 2 then 4 order! They are in the order that the buttons were originally defined.

EMBEDDED TEMPLATES

Template embedding is a way of "inserting" a template inside another host command. What this means is that you can insert the contents of an active template inside a button's Direct Host Command. The direct host command is "expanded" around the inserted template and the contents of the specified template are made part of the direct host command.

Let's illustrate this with an example. Taking the check box example above which had a list of fruits as check box buttons, we can expand on this example to show how template embedding can be a useful tool. In this example, we will build a menu to take someone's order for fruit. Here is what the simple menu screen will look like:



As you can see, we have a set of 7 check box buttons on the left of the menu with the choices of the fruits for sale. On the right is a button to submit your order. To implement this example, we will use two separate button groups. Group #0 will contain one button, the "Submit Order" button. Button Group #2 will be a check box button group containing our seven choices of fruits as in the preceding example.

Here are the Host Command definitions for each of the eight buttons:

Group #0 (normal button - not a radio or check-box button)

Submit Order: I wish to order \$?2\$ right now^m

Group #2 (check box button group)

Apples: [2:]APPLES^m
Oranges: [2:]ORANGES^m
Cherries: [2:]CHERRIES^m
Grapes [2:]GRAPES^m
Pears: [2:]PEARS^m
Bananas: [2:]BANANAS^m
Lemons: [2:]LEMONS^m

Notice in the "Submit Order" button that there is a special code in the Direct Host Command "\$?2\$". This is a special variation of a text variable. This form of text variable is used only in Template Embedding. What it does is instructs the terminal to "insert template #2 here". The format of the template embedding code is:

\$?x\$

...where "x" is the template identifier (0-Z) that is to be inserted. Now back to the example. If the user clicked on "APPLES", "CHERRIES", "PEARS" and "LEMONS" as in the menu shown above, then clicks on "Submit Order", what would the Host Command look like when it gets transmitted to the HOST? Well, for starters, the Submit Order button's host command reads:

I wish to order \$?2\$ right now^m

After template #2 is inserted where the embedding code is, the host command would look like this:

```
I wish to order APPLES^mCHERRIES^mPEARS^mLEMONS^m
right now^m
```

And after the ^m's are converted to carriage returns the final host command would be like this:

```
I wish to order APPLES
CHERRIES
PEARS
LEMONS
right now
```

What if the user didn't click on any of the fruits, but did click on the Submit Order button? Well, since template #2 belongs to a group that is a Check Box group, which can have zero or more items selected simultaneously, the \$?2\$ code would be expanded to a null string, or nothing, so the final host command would be:

```
I wish to order right now
```

If template #2 was associated with a Radio group which has to have one button clicked at all times, and none of the buttons were active, then the terminal would highlight all the radio buttons in group #2 and instruct the user to choose one first. This is done automatically by the terminal you don't need to worry about Radio Buttons.

Just remember, Check Box buttons can legitimately have a blank template, but Radio Buttons cannot due to the very nature of the buttons.

The final host command after template embedding is limited to 4096 bytes of data. If a host command would grow beyond 4096 bytes due to embedding, it is truncated to exactly 4096 bytes.

TEMPLATE CHAINING

Now, on to the second form of using templates, Template Chaining. Template Chaining is another method of using templates. Unlike

Template Embedding, which inserts the contents of a template into a Host Command, template chaining feeds data into a template and then takes the result and transmits that to the HOST. In other words, Template Embedding inserts a template into a Host Command. In Template Chaining, a Host Command is inserted into a template (the reverse).

If you recall the Template Embedding discussion earlier, there was a code for inserting a template into a host command. The command was `$?x$` where "x" was the template number. Template Chaining uses a similar insertion code but with a subtle difference - there is no template identifier. The code is:

`$?$`

This is a "generic insertion code". It is used in the template definition itself, not in the Direct Host Command as the template embedding code was used. The actual data that replaces the `$?$` depends on the Direct Host Command that is "feeding" the template.

Before template chaining becomes crystal clear, we need to muddy the waters some more by introducing one more thing - the Chaining command. A template chaining operation is performed almost exactly as in defining a template with a subtle difference: the colon (:) is omitted from the template definition like this:

[5]This is template chaining
[5:]This is template definition

What's the difference? In the case of the [5:], a template for group #5 is defined. In the [5] example, the template chaining instruction is invoked on Template #5. The phrase "This is template chaining" will be fed into template number 5. If template number 5 has a generic insertion code `$?$` in it, then it will be replaced with the phrase "This is template chaining." The final result after the replacement will be a direct host command that will be transmitted to the HOST. Since template #5 doesn't have an insertion code, the phrase "This is template chaining" will be lost in the chaining process and the final host command would be "This is template definition".

Here are a couple examples of template chaining illustrating several ways that it may be used (we'll only show "active" template definitions):

```
[3:]This is a plain old template
[4:]This template inserts $?$ here!
[5:]This has two insertion codes $?$ and $?$
```

```
[3]This text is lost in the chaining process
[4]SOMETHING
[5]HERE
```

This would be the result of the three chaining operations:

```
Template 3: This is a plain old template
Template 4: This template inserts SOMETHING here!
Template 5: This has two insertion codes HERE and HERE
```

We skipped over the "user clicked on this button" operations and went directly to the end results to make the example as clear as possible. There are three distinctly different situations in this example. The first shows a regular template without an insertion code being used in a chaining operation. As you can see, the data that was fed into template #3 was lost because template #3 didn't have an insertion code. In the second example, a single insertion code is used and the word "SOMETHING" is inserted in place of the insertion code. The third example shows that an insertion code can be used more than once in a given template.

What if you tried to chain to a template that hasn't been activated yet (i.e., a blank template)? If the template in question belongs to a radio group, then the terminal would instruct the user to click on one of the radio buttons to activate the template (he doesn't know that templates are involved of course). If it was a check box template, then the final host command would be nothing and in effect, nothing would be transmitted to the host. If the template in question belonged to a generic button group, then also nothing would get transmitted to the HOST.

In the preceding Template Chaining examples, only one template was used (chained-to). In reality, you can chain to multiple templates with ease. The format of multiple Template Chaining is simple, just add the

template identifiers for the templates you want to chain to in the order you wish to chain to like this:

[1E3]This is a three-level chaining operation

Notice how the Template Chain command has three template identifiers in it, 1, E and 3. The Host Command would be fed into template #1 first. After any replacements, the final result of the template #1 chaining would be finished and that string of text would be fed into template #E. After any replacements/insertions are performed on template #E, then the final result is fed into template #3 and the final result of that chaining operation is sent to the HOST.

Here is an example of multiple template chaining calls:

```
[1:]red green $?$ blue  
[E:]LOUD QUIET $?$ YELL  
[3:]soft $?$ hard smooth gritty
```

[1E3](host command)

In this example, the phrase (host command) is fed into template #1, then the result into template #E then that result into template #3 then finally transmitted to the HOST. We can break this down conceptually into three separate chaining operations to illustrate what happens step-by-step:

[1E3](host command)

Results in:

[E3]red green (host command) blue

Resulting in:

[3]LOUD QUIET red green (host command) blue YELL

And then finally:

soft LOUD QUIET red green (host command) blue YELL hard
smooth gritty

This last phrase is then sent to the HOST verbatim. As you can see, things can get pretty complex when multiple template chains are used, however some dramatic things can be achieved with a little bit of effort and some well thought out planning of your template definitions. Template chaining is useful for controlling HOW particular data is transmitted to the HOST, while template embedding is useful for controlling WHAT data is transmitted to the HOST.

You can think of template chaining as a method of defining "commands" that will be sent to the HOST where the commands can be different depending on which buttons are clicked in that group. Template embedding on the other hand is often used for controlling the data parameters that are used with particular commands.

You are allowed up to 36 separate template chaining levels in one template chaining operation. After all chaining is completed, the final host command cannot exceed 4096 bytes. If a chaining operation would exceed that amount, it is truncated to exactly 4096 bytes.

CHAPTER 10

Advanced Templates

RIPaint 2

TeleGrafix Communications, Inc.

Last updated on 10/20/95

ADVANCED TEMPLATE CONCEPTS

You've already learned about Template Embedding and Template Chaining. These two methods of using templates do not need to be separate methods. You can combine these methods if you wish. This means that you can have embedded templates inside of template chains.

Here are a couple of examples of combinations:

Example 1:

```
[5:]EMBEDDED VALUE  
[3:]Here is a $$ and an $5$
```

```
[3]CHAINED HOST COMMAND
```

This would result in:

Here is a CHAINED HOST COMMAND and an EMBEDDED VALUE

Example 2:

```
[5:]EMBEDDED VALUE  
[3:]Here is a $$ and an $5$  
[4:]print("$")
```

```
[34]CHAINED HOST COMMAND
```

This would result in:

```
print("Here is a CHAINED HOST COMMAND and an EMBEDDED  
VALUE")
```

Example 3:

```
[4:](hello)  
[5:]ANOTHER $5$ EMBEDDED $4$ VALUE  
[3:]Here is a $$ and $5$
```

[3]CHAINED COMMAND

This would result in:

Here is a CHAINED COMMAND and ANOTHER ANOTHER \$?5\$
EMBEDDED \$?4\$ VALUE EMBEDDED (hello) VALUE

Taking this one step-by-step:

Step #1: Here is a CHAINED COMMAND and \$?5\$

Step #2: Here is a CHAINED COMMAND and ANOTHER \$?5\$
EMBEDDED \$?4\$ VALUE

Step #3: Here is a CHAINED COMMAND and ANOTHER
ANOTHER \$?5\$ EMBEDDED \$?4\$ VALUE EMBEDDED (hello)
VALUE

<end of processing>

Example 4:

```
[5:]HELLO  
[3:]$?5$ $?$ WORLD  
[4:]print("$?$")^m
```

```
[344](silly)
```

This would result in:

```
print("print("HELLO (silly) WORLD")^m")^m
```

Example #1 shows a situation with a single chaining operation and a single embedding operation done at the same time. You can see how the final host command is in relation to the data fed into the templates at different points.

Example #2 shows a more complex situation where multiple levels of template chaining are going on while embedding is also being used.

Example #3 is somewhat different though. Why weren't the embedded template codes expanded at the lowest level of template #5? If you look closely at template #5, it has an insertion code instructing the system to insert template #5 (itself) in the middle of the template. This is legal, but only because an embedded template inside an embedded template cannot have any more embedding performed on it. In other words, you are allowed up to two levels of embedded templates, but the lowest level (2nd) cannot have any insertion codes, text variables, control characters or pick-list definitions in it - if it does, they will be treated as "raw" text instead of host command directives.

Example #4 shows a more involved Template Chaining operation in which the same template is chained to more than once in a given operation. This is allowable for extra flexibility, although in the real world will probably not be used much.

HOST COMMAND LANGUAGE AND TEMPLATES

Now that we have thoroughly discussed templates, we come upon another subject - that of text variables, pop-up pick lists, and control characters. In any template definition or host command, you can have text variables, pick lists or control characters anywhere. This gives you the ability to do a great many things.

You can freely use text variables, control characters or pick lists anywhere in a Direct Host Command, in a template chain and embedded templates. There is nothing unusual about text variables, pop-up pick lists or control characters when they're used in template chains or Direct Host Commands. They do get a bit odd though in how they interact with Embedded Templates - but only if you have an embedded template within an embedded template.

Recall from a previous discussion about embedded templates within embedded templates. At the lowest level of the embedding, there is no processing done on the string of text. This is to prevent endless loops and combinatorial explosion of data and CPU time. Text Variables, pick lists and control characters WILL be processed at the first level of an embedded template, but not at the second level.

Here are some examples that better illustrate text variables:

Example 1:

[3:]The date is \$DATE\$... \$?\$

[3]HELLO

Results in:

The date is 07/18/93 ... HELLO

Example 2:

[3:]The date is \$DATE\$

HELLO THERE, \$?3\$

Results in:

HELLO THERE, The date is 07/18/93

Example 3:

[2:]The time is \$TIME\$^m

[3:]The date is \$DATE\$^m

HELLO THERE^m\$?3\$\$?2\$

Results in:

HELLO THERE

The date is 07/18/93

The time is 03:43:32

Example 4:

[1:]The day is \$DOW\$^m

[2:]The time is \$TIME\$^m

[3:]The date is \$DATE\$^m\$?1\$^m

HELLO THERE^m(\$?3\$)(\$?4\$)(\$?1\$)

Results in:

HELLO THERE

(The date is 07/18/93)

The day is \$DOW\$^m

(The time is 03:43:32)

(The day is Sunday)

Notice how the \$?1\$ embedded template used in template #3 does not get processed - it is just inserted raw.

PROCESSING OF TEMPLATES

As stated previously, template definitions may contain text variables, pick lists, control characters, and template insertion codes. When do these special "directives" get processed? The answer is when the template gets USED, not when it becomes ACTIVE! To better illustrate this, let's look at a simple example.

Let's say you have three radio buttons (group #2) on the screen and another button (group #0) which uses the template for the radio button group 3. Here's the host command definitions for each of these four buttons:

Group #2 - Radio Button Group

Button #1: [2:]It's a pretty day at \$TIME\$

Button #2: [2:]It's a rainy day at \$TIME\$

Button #3: [2:]It's a hazy day at \$TIME\$

Group #0 - Ordinary button group

Button #4: Today's forecast:^m\$?2\$

Now, let's say that none of the radio buttons are drawn as "selected" for starters. At 11:45:03 in the morning, the user clicks on button #2 indicating it's a rainy day. The active template for group #2 would be defined as:

It's a rainy day at \$TIME\$

At 11:46:37 he clicks on the Forecast button. What would be the host command sent to the BBS in this example? It would be:

Today's forecast:
It's a rainy day at 11:46:37

Notice that the time that is inserted in place of \$TIME\$ is the time that the user clicked on the Forecast button, NOT the time he clicked on the "rainy day" button which activated the proper template. This illustrates that text variables, pick lists, and control characters are not "processed" until they are referenced (used) by some other button or template in an active Host Command situation.

COMMAND BLOCK SEGMENTATION AND TEMPLATES

Near the beginning of our discussion of templates we spoke about Direct Host Commands, Template Definitions and Template Execution. You have seen how to define templates with a template definition command like [3:]HELLO, and how to execute templates by either Chaining (e.g., "[3]WORLD"), or by using Embedded Templates (e.g., "\$?3\$ WORLD").

Now, remember that we spoke about how a single Host Command can be segmented into multiple "Command Blocks" by separating them with the delimiter "[]". An example of this might be:

```
HELLO^m[ ]WORLD^m
```

This would transmit the following to the BBS:

```
HELLO<cr>  
WORLD<cr>
```

Realistically, you wouldn't use such an overly simple example like this, but would use the following instead:

```
HELLO^mWORLD^m
```

This would produce the same result. But it doesn't illustrate the purpose of Command Blocks. Here's a real-world example of a host command broken down into several command blocks:

[3:]Template Definition[4]Template chaining[]BBS TEXT

If you look carefully at the above host command, three distinct things are happening. First, template #3 is defined with the text "Template Definition". Secondly, the phrase "Template chaining" is chained (fed) into template #4 (whatever that one is) and the final result of the chaining operation is sent to the BBS. Finally, the last command block is processed which happens to be a Direct Host Command, so the text "BBS TEXT" will be transmitted to the Host as well.

If a host command doesn't have one of the command block delimiters like a template definition (e.g., "[3:]"), or a template chain directive (e.g., "[3]"), or a Direct Host Command Directive (e.g., "[]"), then the Host Command is, by default, considered to be a Direct Host Command. The following are all considered Direct Host Commands and they all do the exact same thing:

```
HELLO^mWORLD^m
HELLO^m[]WORLD^m
[]HELLO^mWORLD^m
[]HELLO^m[]WORLD^m
```

Here are some more examples of Command Blocks using the previous example as a foundation, but this time we throw in another command block. Again, all four of these examples do the exact same thing:

```
HELLO^mWORLD^m[3:]Template Definition
HELLO^m[]WORLD^m[3:]Template Definition
[]HELLO^mWORLD^m[3:]Template Definition
[]HELLO^m[]WORLD^m[3:]Template Definition
```

COMMAND BLOCKS, RADIO AND CHECK BOX TEMPLATES

If you recall from our earlier discussions about Radio Buttons and Check Box Buttons, the templates definitions are activated based upon a Button being activated. If you think about Command Blocks though, you might be inclined to think that figuring out which template definition

block of a command block to activate might get insane. It could! Look at the following set of three Radio Button definitions:

Button #1: [3:]Hello world[3:]This is Pluto
Button #2: [3:]Hello world[3:]This is Saturn
Button #3: [3:]Hello world[3:]This is Jupiter

What happens if button #2 is clicked, thus activating that template? What template command block is used to create the final, active template? The answer is the last one! The text that becomes the actual active template for template #3 would be this:

This is Saturn

If a Host Command references this template as in the following example, you will see that the secondary template definition is the one that is actually in use:

I'm a Martian singing in the rain^m\$?3\$

Resulting in:

I'm a Martian singing in the rain
This is Saturn

In short, Command Blocks are processed in all situations, and if a discrepancy exists where two or more template definitions in the same Host String correspond to the same template group, the last definition is the one that becomes active. This applies even if you are dealing with Check Box buttons. In the above example of the Martian, if Buttons #1, 2 and 3 were Check Box Buttons instead, the "Hello World" template definitions would still be lost and would NOT get concatenated together to create the final Host Command.

If buttons #1 and #3 are clicked, then template #3 would be:

This is PlutoThis is Jupiter

It would NOT be:

Hello worldThis is PlutoHello worldThis is Jupiter

CHAPTER 11

Host Commands - What Can Go Where

RIPaint 2

TeleGrafix Communications, Inc.

Last updated on 10/20/95

HOST COMMANDS DIRECTIVES - WHAT CAN GO WHERE?

With all this talk about Templates, Text Variables, Local RIP file playback, pop-up pick lists and control characters, you might be interested in knowing what can be used where in the RIP *scrip* language. It has already been stated that Templates can be used only in Button Host Commands. Text Variables, Local RIP file playback, pop-up lists and control characters can be used in several places though. The places that these commands can be used and not used are listed in the chart on the bottom of this page and the next.

The "Filenames" item in the above chart needs to be explained a bit. Whenever a RIP *scrip* command has a string parameter that can contain a filename, then they are parsed for pre-defined data text variables and also for user-defined data text variables. This allows for icons to be displayed based on the contents of some variable (for example). It should be noted that the only RIP *scrip* command that has a string parameter that doesn't support data text variable processing is the formatted text region command RIP_REGION_TEXT. Allowing data text variables in this command could adversely affect the justification and placement of formatted text.

Command/Area			
Button Host Command	Yes	Yes	Yes
Simple Mouse Fields	No	Yes	Yes
Query Command	Yes	Yes	Yes
Button Labels	No	Yes	No
Graphical Text	No	Yes	No
Text Window Text	No	Yes	No
Filenames	No	Yes	No
Refresh command	No	Yes	Yes
Columnar Text	No	No	No

Host Command Directives - What Can Go Where?

Command/Area			
Host Related Commands			
Button Host Commands	Yes	Yes	Yes
Simple Mouse Fields	Yes	Yes	Yes
Query Command	Yes	Yes	Yes
Text Output Related Commands			
Button Labels	No	No	No
Graphical Text	No	No	No
Text Window Text	No	No	No
Filenames	No	No	No
Refresh Command	Yes	Yes	Yes
Columnar Text	No	No	No

CHAPTER 12

RIPscrip Coordinate Systems

RIPaint 2

TeleGrafix Communications, Inc.

Last updated on 10/20/95

WORLD COORDINATE SYSTEMS

Starting with the v2.0 specification of RIP *scrip*, we have introduced a "world coordinate system". This means that you can alter the base coordinate system used to address the video's physical device coordinate system. This is the first step in achieving device independence. Each coordinate system is called a "Coordinate Frame". There are actually two distinct levels of coordinate systems. From the lowest (least abstract) to the highest level (the drawing board) you have the following coordinate systems:

1. **DEVICE COORDINATE FRAME** - the resolution of the actual display device. This is not determined as part of the RIP*scrip* language itself; it is actually determined by the terminal - based upon whatever type of video hardware is present (and what mode the user has chosen).
2. **WORLD COORDINATE FRAME** - This is the master coordinate system. This is the global set of coordinates. Ideally, it should be higher in dimensions than the Device Frame so that you do not lose resolution when "mapping" X/Y coordinates from the world coordinate system to the device coordinate system. The maximum dimensions of the world coordinate frame is 65535 x 65535.

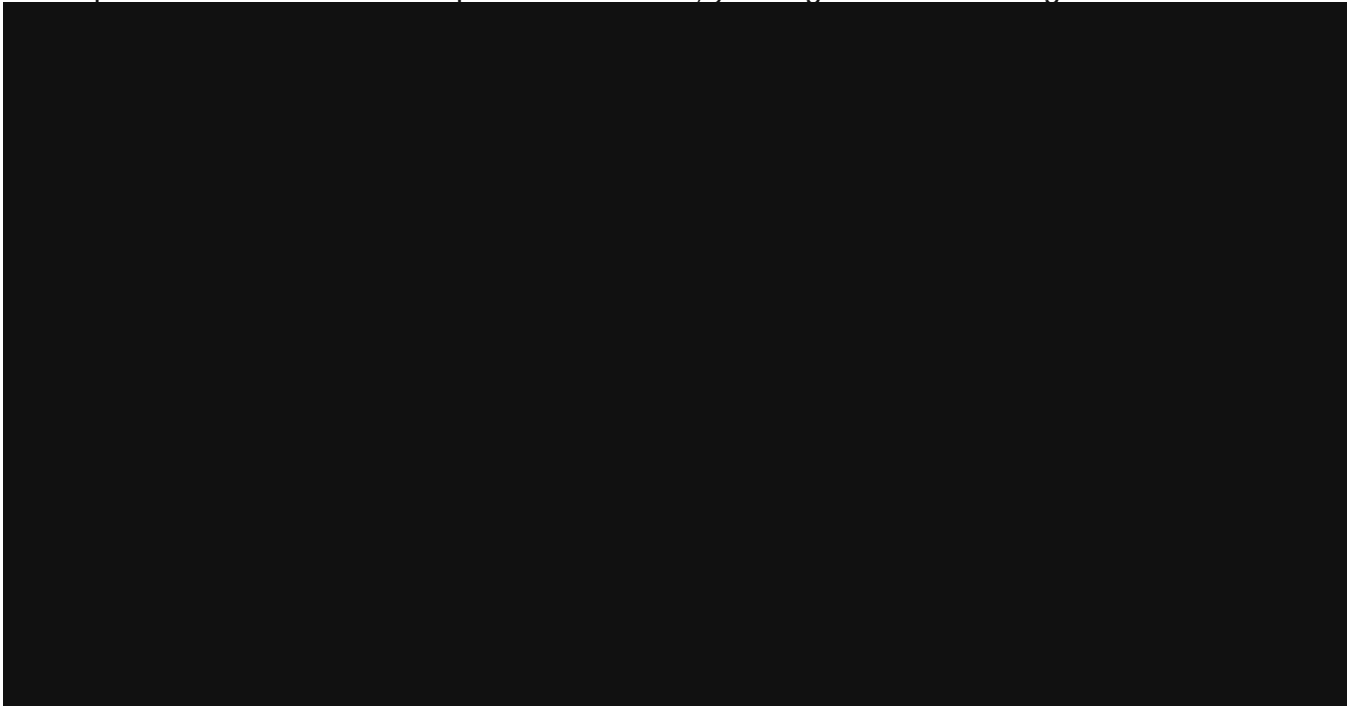
THE MATHEMATICS OF GRAPHICS AND COORDINATES

In graphics, we use the concept of coordinates and points to specify where a pixel is located. If graphics were like mathematics where a pixel was infinitely small, a graphics monitor wouldn't show anything other than black because a pixel would be infinitely small surrounded by blackness. But this is not the case. Graphics hardware has to have pixels defined as discrete areas of the screen - addressable areas of the screen that can be set to a particular color. This is how we see things on the screen, as individual pixels of data used together to represent some kind of image.

Relating these two distinctly different concepts to each other isn't easy. On the surface, a pixel's coordinate location appears to be the same thing as it is in mathematics. But, when you get into deeper issues of

graphics theory this isn't truly the case because of the difference in sizes of a point compared to a pixel. In math, points don't have to be on even integer boundaries - they can be fractional. A point can be at (2.53,1.295), whereas in graphics, a pixel cannot be at a fractional location - there's no such thing as a part of a pixel!

A graphics screen is designed so that every pixel location on the screen has a unique location specified (in human terms) as an X/Y coordinate pair. This is called the Cartesian Coordinate System. It is by far one of the easiest ways of representing a coordinate in a two-dimensional world like a monitor. If you zoom in closely on a monitor and look at the layout of pixels and their relationships to coordinates, you might see something



If you look closely at the locations of the coordinates in the previous diagram, the X/Y coordinates address the physical pixel cells themselves. Each pixel cell is separated by infinitely thin lines - the pixel cells' borders so to speak. Now, each pixel is physically adjacent to the ones next to it and there are no gaps between them. This is the way video cards work.

Now that we've described the way graphics hardware addresses pixel locations, let's look at why it's not really the best way of describing

graphics mathematically. For the hardware, this is the best way of handling things, but mathematically, it's not.

The world of graphics hardware has many different facets. One graphics display device can often times display graphics at different resolutions. A typical video card on the IBM-PC can display graphics at 320x200, 640x350, 640x400, 640x480, 800x600, 1024x768 and even as high as 1280x1024 and higher. That's quite a bit of difference in pixel grids (like the one shown earlier). Not only can the number of pixels change horizontally and vertically, but when the resolutions get higher, the pixels get smaller and harder to see.

What would happen if you had a line drawn on a graphics screen from coordinates (0,0) to (639,349) at 640x350 resolution? You would have a nice little line on the screen that stretches from the upper-left corner to the lower-right corner. Now, if you drew the same line at 800x600 resolution? Your line would no longer go all the way to the bottom of the screen or to the far right of the screen. It would stop about two thirds of the way across.

Problems emerge from the interface of mathematics and graphical representations when we try to make something look the same at one resolution as it does in another (this is called resolution independence). You can translate a coordinate in one resolution to some other coordinate in a different resolution. All you need to know are the dimensions of each resolution and a bit of algebra. But pixels aren't the same thickness! The line described previously could still be drawn properly to the lower-right of the screen, but it would appear a lot thinner. You could try drawing two lines offset by one pixel location and you would come close to the original thickness of the line at 640x350 resolution, but it wouldn't be a perfect match. A perfect match is a rare thing when dealing with different resolutions.

Now, on to the real issue at hand here: translation of coordinates. We won't worry about the size of pixels, but will concentrate on getting the locations of points correct at different resolutions. Let's say you have a screen at 640x350 resolution with two filled rectangles drawn on it. The first rectangle is drawn from (2,1) to (4,3) and the second one is drawn from (5,1) to (7,3) like this:

1st: (2,1) - (4,3)
 2nd: (5,1) - (7,3)

You notice that both rectangles have no gaps between them. Now, if we plot these rectangles on a graphics screen, we tried to make it appear in the external monitor, you would have to translate the new resolution and re-plot the rectangles. 350x2 is 700 so our new resolution is 700 in X and Y directions. (This is RARE) We translate our original coordinates to this new resolution by 2. Pretty simple. Our previous state and the newly translated coordinates are shown below:

	Untranslated	Translated
Rectangle 1	(2,1) - (4,3)	(4,2) - (8,6)
Rectangle 2	(5,1) - (7,3)	(10,2) - (14,6)

Let's plot these two rectangles on our new graphics screen at the newly translated coordinates:

```

      1 1 1 1 1 1 1 1
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8
0
1
2
3
4
5
6
7
8

```

What happened? We now have a black line in between the two rectangles. That's definitely not what we wanted to happen. Why did it? It's partly because of the nature of graphics hardware in the way it addresses pixels, and partly the way we humans think of pixels mathematically. What we need is a better mathematical representation

of pixels and graphics coordinates so that when we translate coordinates these things don't happen.

As we mentioned earlier, the lines on the above diagram represent the borders around each pixel. Each of these lines, like their mathematical counterparts, are infinitely thin. The only thing in the above diagram that has any size (or area) physically are the pixels themselves. Since the boundary lines around the pixels are defining the area that the pixel will occupy, it makes sense that we should think of those lines as the actual coordinates. If we think of the areas in between the pixels as the actual coordinates, we can think of pixels as the areas in between coordinates that are filled in with color. So, if we set out to draw a filled in rectangle like we did earlier, we would actually be responding and requesting "fill in this rectangle's interior". Just as when coloring books as children, you have to stay inside the lines. Think about pixels as spots between the lines that get filled in with color just like a coloring book - the only difference is, a pixel cannot go outside the lines .

Using this new way of thinking of graphics is not very difficult, it just seems a bit odd -especially if you've been working with graphics for awhile. To draw the previous two rectangles properly, their coordinates would have to be changed somewhat - like this:

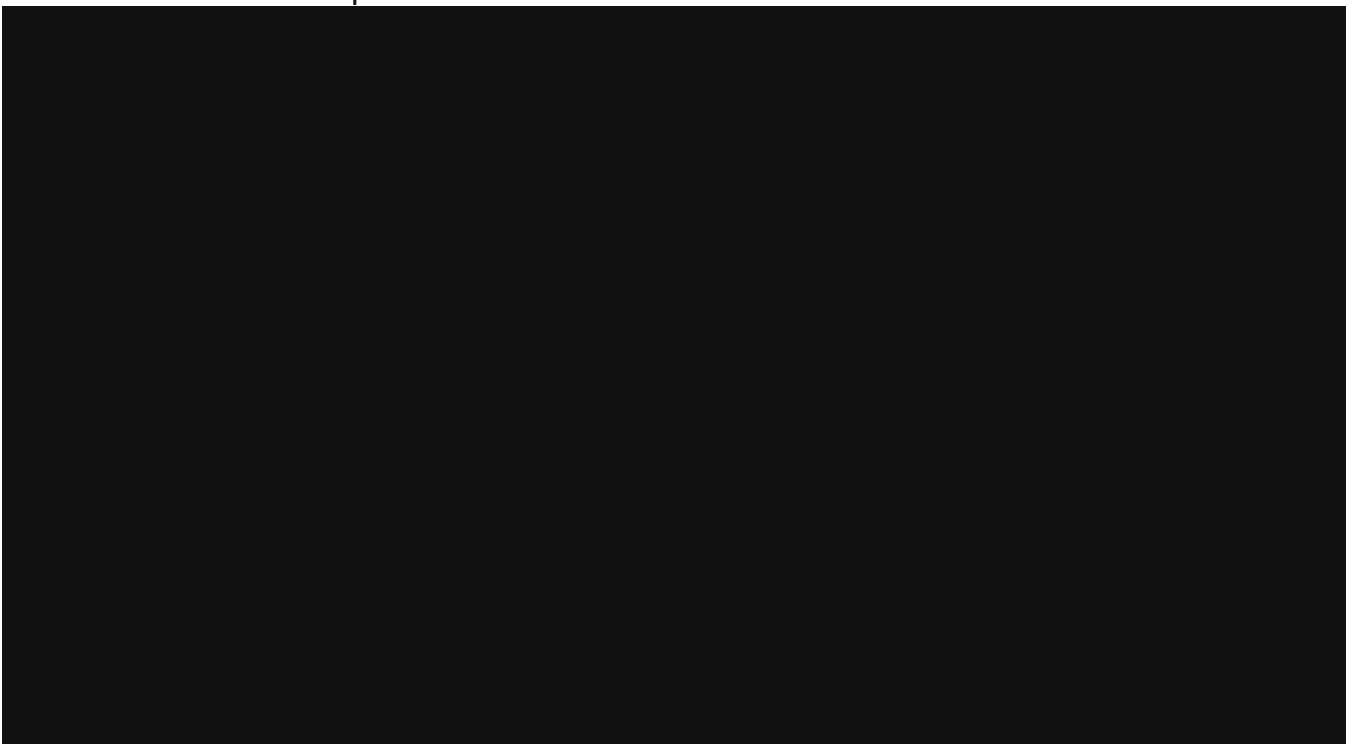
	(old method)	
	Untranslated	
Rectangle 1	(2,1) - (4,3)	(4,2) - (8,6)
Rectangle 2	(5,1) - (7,3)	(10,2) - (14,6)
	(new method)	
	Untranslated	
Rectangle 1	(2,1) - (5,4)	(4,2) - (10,8)
Rectangle 2	(5,1) - (8,4)	(10,2) - (16,8)

Notice that the untranslated rectangles' coordinates are almost the same. In fact, the upper-left coordinates haven't changed at all. The only coordinates that have changed are the lower-right coordinates, and only by one pixel location. Now, if we draw these two rectangles using our new mathematical model for graphics, we would get the following (in both resolutions) examples:



coordinate 10. But they don't overlap each other. That's the part about this new approach that challenges everyone's thinking.

Let's look at another example - one which draws the same two rectangles but this time, let's not fill them in. The diagram below shows the two at both example resolutions:



on the X axis and 2 and 3 on the Y axis. Look at the upper right diagram to convince yourself this is the case. Filled-in areas on the other hand, do not round up to the next highest pixel locations. They always fill in an area in-between the boundary lines. This means that in a filled-in rectangle, the right and bottom edges are "off by one" so to speak.

To better illustrate these two different situations, we will draw a filled-in rectangle and a non-filled rectangle both using the same dimensions at the same resolution, then we will superimpose them on top of each other to better show the differences:

Ç



graphics in two different ways. The way to think about them depends on the type of graphics operation being performed. If you are dealing with a fill operation of some kind, then you think about the drawing surface with coordinate lines "in between" the physical pixel cells. If you are dealing with a line drawing situation (or drawing points, circles, etc.), then you can think of coordinates directly addressing the pixel cells themselves. When dealing with line drawing operations, there is no difference in mathematical models - there's only a difference when you get into filling in areas. Whichever method of thinking of things works best for you, is fine.

Other filled objects work the same way as do filled rectangles. For example, consider an unfilled polygon with the following vertices:

		Vertices							
		0	1	2	3	4	5	6	7
X	Pos	0	8	4	4	2	4	0	0
Y	Pos	0	0	3	4	6	6	2	0

With this polygon we would have a polygon that looks like the left diagram below. The right diagram is what it would look like if it were filled and had a border drawn at the same time (the different shaded blocks indicates which is border, fill-color, or both):





CHAPTER 13

Graphical Viewports

RIPaint 2

TeleGrafix Communications, Inc.

Last updated on 10/20/95

DRAWING PORTS - WHAT ARE THEY?

A drawing port is an area where graphics can be drawn. This is much like having a tablet of paper, where each page can be drawn to individually. However, just like a tablet of paper, only the piece of paper that is on the top of the tablet can be drawn to (the current piece of paper). The same concept applies to drawing ports. At any one moment in time, there is a "current drawing port". This current port is the one that will display graphical drawing operations when they are received.

There are two types of drawing ports:

1. A screen (visual) drawing port, and
2. An offscreen bitmap drawing port (also known as a "Clipboard Port").

SCREEN DRAWING PORTS

A screen drawing port is the most commonly used form of drawing port. A screen drawing port is a port that is somewhere on top of the screen. What this means is that when a graphical drawing operation occurs, it is displayed in that port and subsequently to the user's screen. A screen drawing port is a way of dividing up the user's screen into separate regions, of which each can be thought of as a completely separate drawing area. You can switch between these areas pretty much at will to draw your graphics.

Here is a typical example of one screen and how it can be divided up with multiple drawing ports:



number 0 is defined as the actual screen and cannot be redefined (you can alter its viewport - see below for more details about viewports). Port 0 is always the full size of the screen. You can create your own screen ports, but port 0 is one of which you cannot redefine the boundary. This gives you the ability to always switch to port 0 to address the entire screen, but retain the luxury of other ports when you only want to work with only a portion of the screen.

OFFSCREEN BITMAP DRAWING PORTS OR, CLIPBOARD PORTS

An offscreen bitmap port, otherwise known as a "clipboard port", is very much like a screen port, but it isn't actually a part of the screen. It is more like another screen that you cannot see, but one that you can still draw upon. You might be asking yourself why something like this is part of *RIPscrip*? The answer is quite simple - they are extremely powerful! An offscreen port can be used for placing a piece of graphical data temporarily while you are showing a dialog box on the screen, only to be restore the original graphics when the user clicks the "OK" button on the dialog box. The graphical data that was overwritten by the dialog box isn't deleted - it's been saved temporarily on a screen that you cannot see, but is restored to the screen when necessary. Another use of this might be if you had a very complicated scene to display, but you didn't want the user to see each little graphics operation until the entire scene was complete; switch to an offscreen bitmap port, draw your scene, then

switch back to the screen and copy the offscreen port's graphics to your screen. Voila, the user sees the scene appear on his screen complete - not piecemeal!

There are numerous reasons why you would use offscreen ports; things like storing a bitmap that you just loaded from the hard disk onto an offscreen port so that when you paste it to the screen a large number of times, there isn't a large amount of disk activity on the user's machine. Another frequent use is for an on-line game where the screen is some kind of map, and one or more offscreen ports are used to hold small "icons", or bitmaps of game pieces. Simply copy the images from the offscreen port(s) to appropriate locations on the map and you've accomplished a rather complicated maneuver without adversely affecting the user's system with intense hard drive activity due to constantly loading bitmaps off the disk. As a side effect, the game moves much more swiftly and with a lot less "jerkiness" from the hard disk accesses.

Just as with screen drawing ports, you can select an offscreen bitmap port as the one which will receive graphical drawing operations. Selecting an offscreen port as the current port let's you draw simple graphics objects, photos and other such things to it. But remember, the graphics you draw to an offscreen port are of little value unless at some point you actually copy it to the screen so that the user can see it.

Some things cannot be done to offscreen ports. For example, you cannot place a mouse field or a clickable button on an offscreen port. The reason for this is that the mouse only has any meaning to the screen - the environment with which the user interacts. How can the user click on a button that he can't see? He can't. You also cannot place text windows or assign resident query expressions to offscreen ports. (These topics are described more fully in other chapters).

An offscreen port is most easily thought of as a graphics screen that you can't see. You can work with it to your heart's content, but the user can't see the contents of it until you copy the data to the screen. As long as you keep this one fundamental concept in mind, there should be no confusion about ports.

PORTS AND COORDINATES

A port is a specific drawing area. No matter whether the port is a screen port or an offscreen bitmap port, it is still a drawing area, and as such, there are only a limited number of horizontal and vertical pixels that can be drawn to. Every graphical drawing operation uses graphical coordinates with which it is drawn on the current port. For example, a circle needs to know the center point, and the radius of the circle in order to draw itself.

Each port has what can be thought of as its own set of coordinates. Just like the screen where (0,0) is the upper-left corner of the screen, when a port is active (i.e., current), (0,0) is the coordinate of the upper-left corner of the drawing port. If a port had a width of W and a height of H , then the lower-right coordinate would be $(W-1, H-1)$. This holds true for all drawing ports, even offscreen bitmap ports.

For example, if you draw a circle at (50,50) with a radius of (25,25) to

t
f

I
t
y

actually happens on the screen is that the circle is physically drawn at (50+10,50+10) to the screen, or centered at the "absolute" coordinate of (60,60) in relation to the actual screen.

As you can see, each port can be thought of as its own little drawing universe, with its own set of coordinates. This makes drawing things like graphs, or showing game map windows easy - without having to make extensive calculations in the host software to figure out where things need to be placed. This is considered "port relative" coordinates.

Screen ports are defined as some location on the actual screen from the upper-left corner to the lower-right corner. From this information, the width and height of the port can be easily calculated.

Offscreen ports are a bit different though. Since they're not part of the user's actual screen, they don't have this upper-left location on the screen. In this manner, an offscreen port is always thought of as having an upper-left corner of (0,0) and its lower-right corner as (W-1,H-1).

PORTS AND VIEWPORTS

Each drawing port has an associated viewport, or clipping rectangle. A viewport is a rectangle inside the port that defines the drawing limits inside that port. It is similar in concept to a coloring book with invisible lines. You can draw all you want, but you can't go outside the lines. If any operation would extend beyond the edge of this clipping rectangle, it will be truncated. For example, if you have a drawing port that is 100x100 pixels in size, and you define the viewport to be from (25,25) to (74,74), you would have a 50x50 drawing area right in the center of the drawing port. If you then draw a circle in the exact center of the drawing port with a radius of 60 pixels, you would have pieces of the circle that extend beyond the top, left, bottom and right borders of the viewport. What you would see would be four arcs in each corner of the viewport as in the following example:

If you
the
would
draw

You
draw
draw

viewport go outside the actual port, it will be adjusted to fit so that it is completely within the port. If an attempt is made to define a viewport that is completely outside the boundary of its port, then the definition is ignored as an error condition. If the lower-right corner is outside the boundary of the port, then the lower-right corner of the newly defined viewport is set to the lower-right corner of the port itself.

Just like drawing ports, a viewport also "adjusts" drawing coordinates. When you alter the location of a viewport inside of a drawing port, the origin (0,0) for drawing operations relates to the upper-left corner of the viewport itself, not the actual underlying drawing port. All drawing operations pertain to a port's viewport, not the port itself. The port is considered the maximum limit for the viewport inside - just like the screen, you can only draw to areas inside the screen; the same applies to viewports and ports.

As an example, let's say you have a screen port defined and you alter the viewport (remember, by default it is the full size of the port when the port is defined until you re-define it). The following diagram will give you an idea of our example:



(170,170). This makes our viewport 131 pixels wide and 131 pixels tall. The upper-left coordinate of the viewport would map to the screen coordinates (25+40,25+40) or (65,65), and the lower-right corner would map to (25+40+130, 25+40+ 130) or (195,195).

As you can see, mapping a screen port's coordinates to those actually used on the screen can get quite involved. But when you get down to the benefits of coordinate systems, this approach provides quite a bit of flexibility in moving things around without having to change coordinate systems all the time.

PORTS, VIEWPORTS AND GRAPHICAL OPERATIONS

We've already discussed how a viewport "truncates" a graphical operation if it extends beyond the border of the port's viewport.

What if you're copying graphical data from one port to another? Graphics data can only be copied from one port to another in rectangular portions, and just like all other graphical operations, this kind of situation adheres to the viewports of both the source and the destination ports' viewports! Let's take an example where you are copying a rectangle of graphics from port 1 to port 2. For sake of clarity, we'll copy the entire viewport over (not a sub-area of the viewport). The following diagram shows what would happen if the two viewports aren't the exact same size (specifically, the source viewport is larger than the destination viewport):



allowed in the *RIP - script* specification, some elaboration needs to be made on what happens if any overlap each other. Very simple, they do what they have always done - draw text, or draw graphics. For example, if a viewport overlaps a text window and you draw some graphics (say a circle) over the top of some text in a text window, and the text window subsequently scrolls, all or part of the circle could scroll with it! Now, of course, from the viewport's standpoint, the graphics are no longer what you originally sent to the viewport, but that doesn't matter - you don't preserve any of the commands you used to create the graphics - you

simply draw the graphics. So with this in mind, even if multiple viewports overlap each other it doesn't matter because the final result is what's on the screen.

If a text window overlaps another text window, the same thing happens. To a RIP *scrip* terminal, text is simply just a piece of graphical information that is being placed on the screen in a formatted fashion.

MISCELLANEOUS NOTES AND INFORMATION

Later in this document, references are made to Mouse Fields and Mouse Buttons. Specifically, it is noted that up to 128 of these types of commands may exist simultaneously on-screen. This means that you can have 128 mouse fields, 128 mouse buttons, or any combination of the above, but combined, their total number cannot exceed 128.

When the user clicks his/her mouse on the screen, all mouse regions (whether mouse fields or mouse buttons) are scanned from most recent to the least recent. This means that if a mouse region is received that overlaps another (previously received) mouse region, the newest one would be selected if the user clicked in that region.

COPYING DATA FROM ONE PORT TO ANOTHER

When you copy graphical data from one port to another (or to another location on the same port for that matter), you are duplicating the graphical contents of the source port onto the destination port. Whether the result is an exact replica of the original image is another matter. Whenever you copy a rectangle of data from one port to another, you need to specify a rectangle in the source port, and another one in the destination port. These two rectangles do not need to be the same pixel size. If they are different either in the width or height dimensions, then scaling of the source image will occur! For example, if the source image's rectangle doesn't have the same dimensions as the rectangle in the destination port, you could have a situation similar to the following:

Drawing Ports



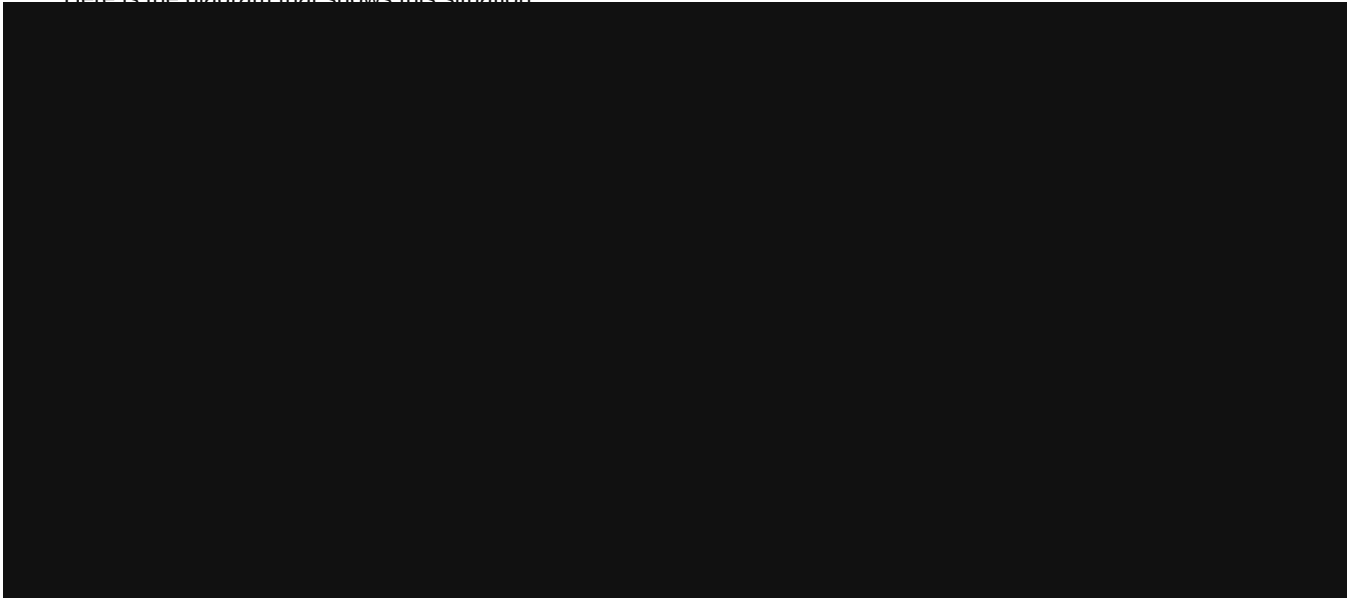
have to be careful about viewports. If either the source or the destination rectangle are completely outside that port's viewport, then the copy operation isn't performed - nothing would be visible if it did happen, because either the source or the destination image wouldn't be visible inside the viewport.

Here is another example showing a straight copy operation without any scaling being performed. We are assuming that the source and destination rectangles are the same size. The port's boundary and its



Let's take another example but this time we will show scaling because this situation may not be intuitive. Let's assume that our source rectangle is twice as large as our destination rectangle both in the width and height dimensions. This means that our image will be reduced by 1/2. But what happens if the source image needs to be "truncated" to fit in the viewport? The answer is simple - only the data that remains after truncation is scaled into the destination rectangle. If the destination rectangle also had to be truncated, then the scaling is still performed. Under no circumstances will a vertical or horizontal "blank" zone be created during scaling. When you say "scale to fit", it does exactly what you told it to - it makes the graphics fit in the given area.

Here is the diagram that shows this situation:



CHAPTER 14

Color Under RIP *scrip*

RIPaint 2

TeleGrafix Communications, Inc.

Last updated on 10/20/95

COLOR PALETTES - HOW COLOR IS HANDLED IN RIP*scrip*

The following sections describe various issues relating to color palettes and color translation as used in the RIP*scrip* language. Color translation can be a tricky subject, so we take it in pieces. We will describe how RIP*scrip* uses color, translates it and represents it numerically in the language.

COLOR PALETTES AND HARDWARE - COLOR TRANSLATION

With versions of RIP *scrip* prior to v2.0, the color palette was limited to 16 colors maximum out of a total of 64 colors in the Master Color Palette. This was due to the hardware origins of the language in its initial releases. With the popularity of higher color video sub-systems, RIP*scrip* needed to grow. In addition, it needed to be open-ended enough to accommodate yet unknown hardware environments that might provide for even higher color resolution.

The 1.xx RIP *scrip* specification provided for 16 colors out of a palette of 64 colors. This 64 color palette corresponded to 2-bits of Red, Green and Blue information each, giving you four levels of color saturation in each of the three categories. Unlike typical RGB implementations the color palette entry numbers are not sequentially ordered based on the R, G and B components. The base 16 color palette in RIP *scrip* 1.54 and earlier was modified by the RIP_SET_PALETTE and RIP_ONE_PALETTE commands. These commands are limited to a maximum of 16 colors and only accommodate 2-bits for each color component. These commands are here on out referred to as the "desktop palette" alteration commands and are only designed to make "simple" changes to the lowest 16 colors with a limited set of color saturation in each of the red, green and blue components.

Under RIP*scrip* 2.0 and later, we deal with the concept of a 256-entry color lookup table (otherwise known as the drawing palette). This palette of RGB colors is used to cross-reference a particular color number to some arbitrary RGB color combination much like the earlier 1.xx palette counterparts, but with some more important differences.

The newer set palette commands allow you to specify the number of bits for each red, green and blue component, and in addition, the internal layout of each RGB value isn't encoded in a seemingly haphazard fashion. A color value is a raw binary number without any frills or complications. If the bit value were 00100100 then the actual color saturation level would be 36, not encoded in a strange bit swapped fashion as in the older color palette commands. It is prophesied that the older color palette commands will not be used much (if at all) in the newer 2.0 environments due to their complexity - about the only way that the older commands will be used is via software packages that only output colors in the older syntax.

Since RIP *scrip* 1.54 and earlier didn't know about the 256-entry color lookup table, we have introduced two new color palette commands:

RIP_ONE_DRAWING_PALETTE	Set one color in the drawing palette to an arbitrary RGB color combination.
RIP_SET_DRAWING_PALETTE	Set a block of colors in the drawing palette to some set of arbitrary RGB color values.

These commands supersede the older 1.54 commands and are much more flexible. They not only allow you to access the entire 256-entry color lookup table, but they allow you to specify how many bits of precision of RGB component data are used in the RGB encoded data.

COLOR PALETTES - PALETTE MAPPING AND DIRECT RGB MODE

Under RIP *scrip* 2.0, you have two methods of specifying color values. You have already seen how colors can be specified by a color index number into the color lookup table (from 0-255). What RGB color that actually maps to is based on the contents of that entry in the actual color lookup table. This is called "Palette mapping mode." In palette mapping mode, most locations in RIP *scrip* commands that allow for color values take a color number (from 0-255) which references some RGB value in the color lookup table.

We also allow for another mode in RIP *scrip*, called "Direct RGB Encoding". This mode allows you to specify actual "raw RGB" values instead of color lookup table indices. When a direct RGB value is found, it is decoded (i.e., breaking up the red, green and blue components) and is used for the actual color. What happens at this particular moment depends on the environment under which the RIP *scrip* compatible software is running.

If the software is running in an environment with a color palette, where the hardware video palette is set to some combination of the colors used in the color lookup table, then the RGB encoded data is matched to the "closest" color in the actual target hardware video palette and that color number is actually used. Under many situations, this color value will be the same one as used in the actual color lookup table (for situations where the software is running on a 256-color palette environment with complete control over the hardware palette). Under modes that are more limited, but in which a hardware palette is still being used, the same algorithm still can be applied - but there won't be a one-to-one mapping of color lookup table values to the hardware palette. This is unavoidable and, in the case of direct RGB encoding, irrelevant.

If the software is running in an environment like 24-bit mode, then life couldn't be simpler. Simply take the color value and use its RGB components (possibly with some color component remapping), and activate that color in the environment.

Direct RGB encoding mode, unlike palette mapping mode, allows you to specify the bit-precision of RGB color codes used subsequently in RIP *scrip* code (where direct RGB colors are permitted - not all RIP *scrip* commands permit direct RGB encoded color codes). This allows you to have color values with the same flexibility as "set color palette" commands, where you can specify arbitrary precision of RGB data to potentially accommodate high-end color systems like 24-bit, 16-bit, 15-bit, etc.).

By default, RIP *scrip* operates in palette mapping mode. Whenever a reset operation occurs (any type of reset), palette mapping mode is once again re-enabled. In order to go to RGB encoding mode, you must explicitly specify a RIP_COLOR_MODE command, or a \$COLORMODE()\$ text variable to activate direct RGB encoding mode. When you specify direct RGB encoding mode, you indicate how many bits of precision to allow for each red, green and blue component. This

value is used for all three components. For example, if you specify 8 bits of precision for each component, then combined they will amount to 24 bits of RGB data (e.g., 24-bit mode). This is unlike some unusual environments on the IBM-PC where 16-bit modes (5 red, 6 green and 5 blue bits) are allowed on some VGA environments.

Valid values for *bits* are 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32.

When a direct color value is specified, the value is used uniformly for all three components. If *bits* is N bits of precision, then the value is used for all three components. For example, if you specify 8 bits of precision, then the value is used for all three components. If you specify 16 bits of precision, then the value is used for all three components. If you specify 24 bits of precision, then the value is used for all three components. If you specify 32 bits of precision, then the value is used for all three components.

THE DEFAULT COLOR PALETTE

The first 16 entries in the above color table correspond to the default color palette used in version 1.xx of RIP *scrip* and correspond directly to the color palette used for 16 color ANSI text. The next 16 colors in the color table are a 16 level gray-scale used for gray-scale image output and to provide a basic spectrum of grays.

The remaining 224 entries constitute a "uniform distribution" of RGB colors. The actual organization of the colors is mathematically based such that an arbitrary RGB color value can be mapped to a color value in the default palette with a simple calculation (instead of searching through the entire palette for the closest entry). This block tries to cover as many colors in the color spectrum as possible.

There are four levels of blue, eight levels of green and seven levels of red. Each block is organized with blue the color that changes every

entry, green the next most changing, and red the least frequently changing. If you have a RED value from 0-6, a GREEN value from 0-7 and a BLUE value from 0-3, you can easily calculate the correct palette index entry with the following equation:

$$\text{INDEX} = 32 + \text{BLUE} + \text{GREEN} * 4 + \text{RED} * 32$$

Or in C with bit-shifting, you could do it like this:

$$\text{INDEX} = 32 + \text{BLUE} + \text{GREEN} << 2 + \text{RED} << 5$$

The main reason for choosing this color palette was to facilitate the ability to display many color images on the screen at the same time in 256 color mode without having to alter the color palette for each image. The color representation may not be 100% exact, but it would be close enough with dithering to accurately represent the original image.

The exact determination of this color palette was very carefully chosen. The lowest 16 colors, which happen to default to the basic colors of ANSI color codes (universally used throughout the computer world), are a good sub-set of colors for basic 16-color operations. The next 16 colors are defined as a dedicated gray-scale color palette (only of use in 256 color modes), and gives a very good breakdown of the gray-scale monotone color palette. The remaining 224 entries are broken down mathematically (as you've seen above) with 7 levels of red, 8 levels of green and 4 levels of blue. This seemingly odd configuration of color distribution was very scientifically chosen.

Extensive experimental research has determined that the human eye is more susceptible to particular "hues" of light than others. Specifically, the human eye is most susceptible to subtle changes in the shade of green than any other color. Next comes red, then finally blue at the very lowest end of the perceptivity scale. If you recall, red is at the lower end of the light spectrum (e.g., near infrared), and blue is at the upper end of the spectrum (near ultra- violet). Green is in the middle. This tells us that our eye is most responsive to light in the middle-to-lower bands of the color spectrum. After scientific analysis, the eye was determined to be responsive to the following levels of each color band (comparatively based on an 8 level scale, 1 being lowest):

Bits	
8	Green
7	Red
4	Blue

Taking this information into account, we constructed a default palette which reflects the best distribution of color that the human eye can perceive. By no means is it perfectly suited for all environments, but it is a "best case" situation for most images and environments. As you can undoubtedly guess, a photograph with a lot of blues and violets would be significantly degraded in quality in this color environment, but many typical images where a broad range of colors is used works very well, and when dithering is used, the situation improves even more.

CHAPTER 15

Data Tables

RIPaint 2

TeleGrafix Communications, Inc.

Last updated on 10/20/95

DATA TABLES - A FUNDAMENTAL 2.0 ADVANCE

RIP*scrip* 2.0 is different from its previous versions architecturally on a number of levels. One of the key fundamental differences is the introduction of data tables.

A data table is a table of information used in RIP *scrip*. Data tables include the color palette table (where each table entry is a complete color palette definition), graphical style tables, button style tables, drawing port tables, text window definition tables, mouse field tables, or environment data tables. Some data tables have an entry that is "currently active." This might be the currently active text window in the text window data table, or the currently active color palette in the palette data table. Every data table has one or more entries in it, in which each entry is a specific piece of data. Data table entries can be "in use" and/or protected (see below).

Data can be moved from one table entry to another (in effect, making a copy of one table entry to another) this doesn't apply to all data tables (e.g., mouse field data table). Also, most tables' entries can be protected so that the contents of a particular entry cannot be modified or deleted until either an unprotect entry function is executed, or a hard reset is performed. (A hard reset completely resets the entire RIP *scrip* environment to ground zero). Again, the mouse field data table cannot have individual entries protected.

TYPES OF DATA TABLES

There are six different data tables for general data storage. They are: the drawing port table, color palette table, graphical style table, text window table, the button style table and the RIP *scrip* environment data table. There are backup areas for these data tables so you can make complete backup copies of these data tables for re-use later after some operation is performed (e.g., executing a dialog box, running a door, etc.). There are actually more data backup areas than there are data tables, but this will be described in a later section entitled "DATA BACKUP AREAS".

DRAWING PORT TABLE

One of the most fundamental aspects of RIP *scrip* 2.0 is the concept of a drawing port. Any drawing of graphics occurs inside a drawing port. There may be up to 36 ports defined simultaneously. Port number 0 is always defined as the screen. Its dimensions are those of the screen. The remaining 35 ports may be sub-sections of the screen or may be offscreen memory drawing ports (i.e., clipboard ports). At any one moment in time, only one port may be the current port - the port that will receive graphical drawing operations. If the current port is an offscreen clipboard port, then any drawing operations will not appear on the screen. The only way that those graphics can be viewed by the user is if the contents of the port are copied to a screen port.

All ports, when defined, are defined using the current world coordinate system. From those coordinates, the port's actual physical pixel dimensions are determined and those dimensions are remembered by the port. Any graphical operation that bypasses the viewport system (RIP_SCROLL_REGION, etc.) will pay close attention to coordinates and the boundaries of the drawing port. If any coordinate would go outside of the port, it is adjusted to fit inside.

A drawing port remembers a number of pieces of information which characterize that drawing port. Each port maintains its own list of the following pieces of data:

- Current X/Y location (used with RIP_TEXT commands)
- Current image style settings (see RIP_IMAGE_STYLE)
- Current viewport location and dimensions
- Current fixed viewport "resident query" expression
- The current "source X/Y location" for graphically stored data used with the PORT_COPY, RIP_GET_IMAGE, RIP_PUT_IMAGE, and \$PCB\$ commands (see these commands for more details).

The entire drawing port data table keeps track of the "current viewport" query expression (otherwise known as a floating viewport query). See the RIP_QUERY command for more details.

COLOR PALETTE TABLE

In RIP *scrip* there is a graphical color palette. The color palette is a group of 256 separate color RGB values, also known as a color lookup table. This color palette is used to map a particular color number to some combination of red, green and blue color components. For example, color 0 is usually black (no red, green or blue), and color 255 is typically white (full intensity red, green and blue). This color palette is normally written directly into the video hardware of your computer to activate the colors. So when you try to draw in color 255, you see a white line appear on the screen.

RIP*scrip* 2.0 allows you to have up to 36 separate color palettes. Combined, these make up a color palette table. Only one entry in the color palette table may be current at any one moment in time. An entry in the color palette table is a complete 256 color palette. When you switch from one color palette entry to another, the colors on the screen typically will change instantly (see below about 24 bit modes). This gives you the ability to quickly and easily switch from one color scheme to another without sending a large amount of information to the terminal every time you wish to switch color schemes.

RIP*scrip* 2.0 allows for two forms of color modes, color palette mapping mode (the default) and direct RGB color encoding. Color palette mapping mode refers to a method of selecting colors where colors are specified as numbers from 0-255, which are indices into the current color palette. This determines the actual color that will be rendered to the screen. In direct RGB color mode though, colors are specified as raw red, green and blue components (see the sections below on "COLOR MAPPING VS. DIRECT RGB ENCODING").

Under 24 bit configurations of a RIP *scrip* software package, a real color palette is not possible because there are no color registers on the video hardware. In those cases, a color palette is merely used as an RGB color lookup table. In other words, if you said to draw a line in color 255, you would be saying lookup color 255 in the color palette, take that RGB configuration and draw the line based on that color combination. In sense, it is the same as what the video hardware does, but the software has to do the job in that situation. When the system is in 24 bit color modes, where color palettes are not truly possible, switching from one

color palette to another will not make any changes to the colors on the screen - it only affects subsequent drawing operations.

When a normal reset operation is performed, all unprotected data table entries are reset to boot up default color palette values. A "hard reset" will reset the entire color palette data table to default values.

GRAPHICAL STYLE TABLE

There are 36 separate graphical style table entries. Initially, number 0 is current and is initialized to default values. A graphical style entry defines a number of currently active graphical attributes. A graphical style entry defines the following values (each entry has all of these attributes):

- Current drawing color
- Current background drawing color
- Current fill pattern number (or user-defined fill pattern)
- Current fill color
- Current line pattern number (or user-defined line pattern)
- Line pattern odd drawing rule (see RIP_LINE_STYLE)
- Current mouse cursor style number
- Current font number (or font name for extended fonts)
- Current font size, orientation and horizontal/vertical alignment
- Current write mode (raster/transfer operation)
- Current color mode (palette or direct RGB)

With the ability of having multiple graphical style entries defined simultaneously, you can setup an entire graphical environment, then with a single command, switch to a completely different configuration in only a couple of bytes of RIP *scrip* code. This can greatly reduce the amount of RIP*scrip* code that needs to be transmitted after the style data table entries are initially setup.

See below under Protected Data Table Entries for even more powerful aspects of the data table systems.

BUTTON STYLE TABLE

Button style data table entries contain a complete button style definition as defined by a `RIP_BUTTON_STYLE` command.

Having multiple button style slots defined at the same time allows you to have many commonly used button styles defined at once, so that you can simply reference an already defined button style rather than having to transmit a rather lengthy button style command each and every time you wish to define a button.

TEXT WINDOW TABLE

A text window is simply a region on the screen where raw, non-RIP *scrip* text is routed once it is received from the host. Initially, the text window is full screen, using the MicroANSI font #2 (80x25). This is different than previous versions of RIP *scrip* in that the initial setting was full screen using font #0 which corresponds to an 80x43 text configuration. 80x25 is more aesthetically pleasing to most people so it was made the default font configuration.

A text window typically supports ANSI and VT-102 screen control codes to support foreground and background colors, and other formatting options. This information is all part of a text window. Each text window table entry stores a number of pieces of information that define what a text window is. The information stored in each text window entry is as follows:

- The upper-left and lower-right corner definitions for the text window (coordinates are in text coordinates, not graphical screen coordinates).
 - The MicroANSI screen font number for the given window.
 - The current ANSI color attributes
 - The current cursor X/Y location
 - The status of the cursor (is it on or off)
 - The current vertical scrolling margins (ANSI formatting)
 - Whether the window is activated or deactivated.

A text window also can be deactivated. If the current text window slot is deactivated then any raw text that is received by the host will be discarded (i.e., not shown on screen at all).

Having multiple defined text windows allows you to quickly switch from one text window configuration to another without having to lose any previously used information (color, location, cursor, etc.). Note that the text contained inside the current text window is not stored as part of the text window definition - only the window's raw definition is preserved. This can easily be used for multiple data entry fields, multiple chat windows, or other such variations.

ENVIRONMENT TABLE

The environment data table is a central table for the maintenance of critical values that define the characteristics of the RIP *scrip* environment. This data table can contain up to 36 separate environments. The following pieces of information are stored in the environment data table:

- Current graphics style entry number
- Current button style entry number
- Current drawing port entry number
- Current text window entry number
- Current color palette entry number
- Current World coordinate dimensions (X and Y)
- Current base math settings (36 or 64)
- Current coordinate size (2 through 5)
- Current color mode (color palette mode or direct RGB mode)
- Current mouse pointer number

When you switch from one environment to another, you are effectively performing an entire "context swapping" operation. This might best be taken with an example:

Let's say you have set up all of your data tables, mouse pointer, palette entry etc, to work with your main screen. You save all these settings to the enviroment data table. Now when you come back to the main

screen you can load all the settings with one quick command instead of loading each one separately.

You could easily switch between one drawing environment to another simply by switching environments, in effect completely redefining your entire drawing world. Nothing is destroyed in this switching process, all you are doing is simply switching to another environment. You would still be free to return to the previous environment if you so desire. (Note: that if you perform any reset operations though, make sure that you protect things or they will disappear when you reset).

Note that there isn't a "current mouse field data table entry". This is because there's no such thing as a current mouse field in the mouse field data table. So, switching from one environment to another has no effect on the current mouse fields (unlike it would with text windows, ports, graphical styles, palettes, etc.)

MOUSE FIELD TABLE

The mouse field data table is unlike the other data tables. It can hold up to 128 separate mouse field definitions and does not have a "current mouse field" entry number. Each mouse field that is defined gets added to the mouse field data table at the very end until 128 fields are defined, or the table is deleted and things start over. Mouse fields only pertain to drawing port #0, so they can only be on the screen, not on an offscreen drawing port. See the RIP_MOUSE, RIP_BUTTON_STYLE and RIP_BUTTON command for more details on this special data table.

PROTECTED DATA TABLE ENTRIES

Data table entries by themselves have quite a bit of usefulness in short-term situations where several different configurations need to be switched among fairly often. Their usefulness becomes even more powerful with the concept of protected data table entries. A protected entry is one that cannot be cleared by a normal reset operation or a clear table entry operation until the entry is unprotected, or until a "hard reset" operation is performed. In addition, they cannot be modified while they are protected (see below).

A "hard reset" is a complete reset of the RIP *scrip* environment. Typically, a hard reset should only be performed immediately after

connection to a RIP *scrip* compatible host system to clear the slate, so to speak. Other uses for a hard reset are when the RIP *scrip* environment is hopelessly corrupted due to line noise, or when a complete re-synchronization of the environment is about to commence.

Protected entries become quite powerful when you look at a larger host system. For example, let's say that throughout your system you use five different graphical style configurations, all of which require quite a bit of transmitted RIP *scrip* code. If you define each of these styles as separate entries and then protect them, you don't need to re-transmit those blocks of data again unless a hard reset operation is performed. This can mean substantial savings in data transmission, and consequently faster transmission times. The same concept applies to text windows, ports, button styles, graphics style, palette and environment tables. When taken from a system-wide perspective, where many sections of the system may be made by other manufacturers (e.g., doors, BBS modules, etc.) The system operator may not be able to count on whether a section of the system uses one style or another, or does a certain kind of reset. Having protected entries gives you the ability to protect your data configurations so that other sub-sections will not destroy your data. When you return from the sub-section, simply re-activate your table configuration(s) and you can continue with all your data styles intact.

We just described that a protected data table entry cannot be deleted. We also mentioned that it cannot be modified. Let's clarify what we mean by not modifiable. A data table entry that is protected cannot be modified by any of the normal RIP *scrip* commands or text variables that alter their basic configuration (e.g., you cannot alter the foreground drawing color in a protected graphical style data table entry, etc.) It should be noted however, that normal drawing operations like rectangles, circles, etc., can still be performed to a port that is protected. In this case, the port is protected from having its "configuration" modified (e.g., the viewport cannot be modified nor can the actual port's orientation). The same applies to text windows (e.g., you can output text to a text window, perform ANSI/VT-102 operations to it, erase it, etc., but you cannot redefine it or delete it). One tiny exception rests with text windows that support VT-102. VT-102 has an ANSI escape sequence that allows you to alter the wrap/chop setting of text windows. Since ANSI operations are allowed on protected text windows, this is permissible even though it seemingly violates the nature of modifying a text window's protected status.

Terminal emulation modes also are unaffected by a text window's protection status. If a text window is enabled, VT-102 mode can be turned on or off because this is a "global" setting, not a text window specific setting. Also, as noted later, RIP *scrip* processing can be disabled via a special ANSI escape sequence. This does not affect protection of text windows or drawing ports either.

With these things in mind, performing operations like "copying a port's data to another port" where one or both are protected is allowable because you are not altering the basic configuration of the ports in question, you are simply altering their data contents. This applies to "clipboard" related commands too. Such commands might dynamically pick and choose ports to define. A clipboard related command will auto-define a new port. That is if one doesn't already exist, it will pick a currently undefined drawing port and define it. (If it's not defined, then it cannot be protected)

CHAPTER 16

Data Save Tables

RIPaint 2

TeleGrafix Communications, Inc.

Last updated on 10/20/95

DATA SAVE AREAS

A data save area is a backup area that composes one base save area and ten data save regions for specific data table types. The data save regions are broken up collectively as a set of individually addressable data save slots (like an array in most programming languages), and a data save stack. Each data backup area can contain up to eleven separate, complete data tables of one particular type - one in the base save area, and up to ten other data tables in the data save slots or the data save stack, or in some combination of the two (save slots or stack).

Each data backup area can store only one particular type of data table. For example, the color palette data backup area can only store color palette data tables. (Remember a color palette data table consists of 36 separate color palette definitions).

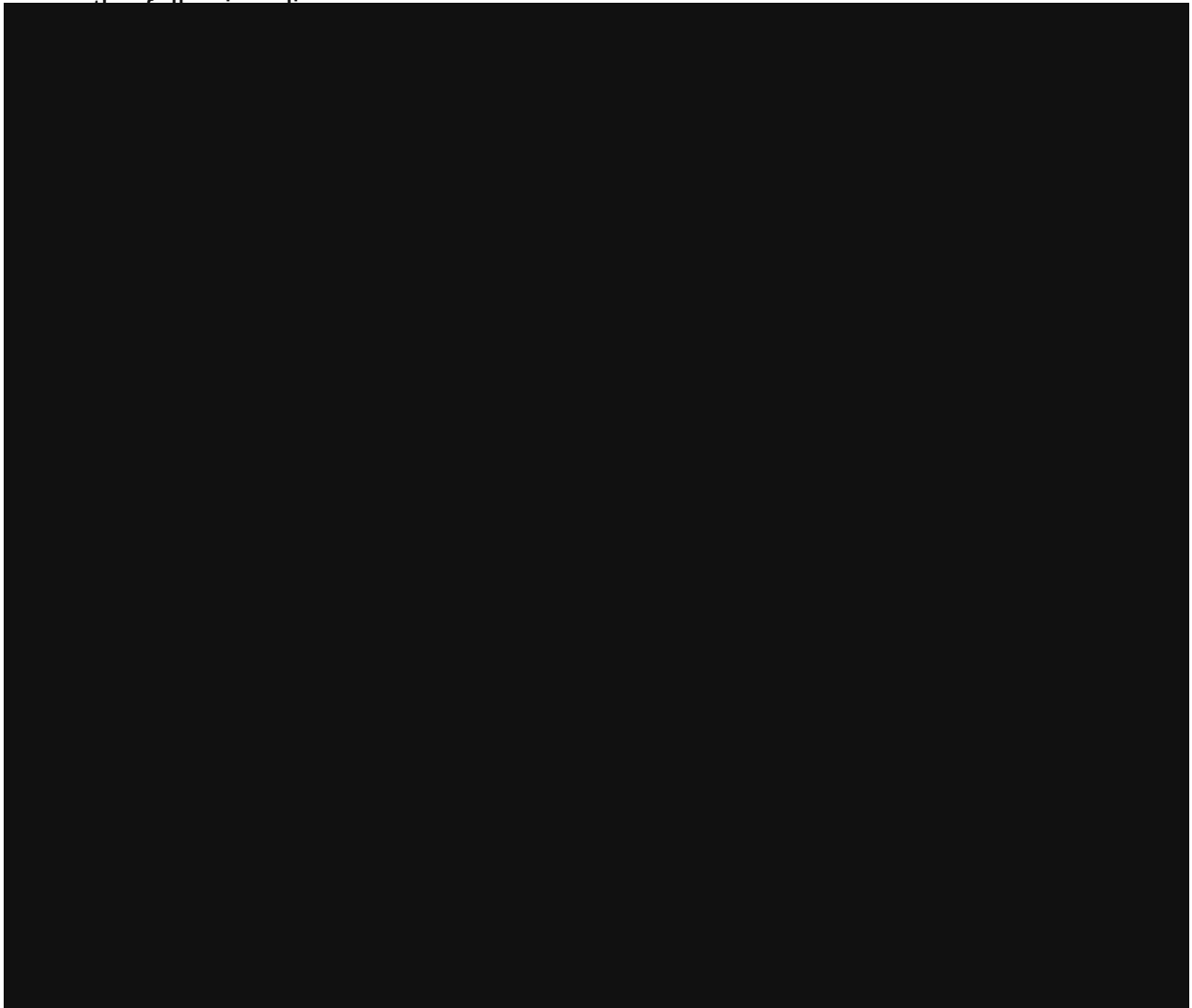
A data save slot is a backup area that is large enough to store an entire specific data table, or other object. (graphics screen, mouse fields, etc.) Each data backup area's save slots can hold one entire data table of the type designated for the data backup area. Each data save slot has a specific slot number associated with it (0-9). When the data in a data save slot is restored back to the actual RIP *scrip* drawing environment, that slot is deleted (cleared), providing that it isn't protected (see below).

A base save area is very similar to a data save slot, except that a base save area does not have a specific "slot number" associated with it. Unlike data save slots, when you restore a data table object from a base save area, the base save area is not cleared (i.e., deleted) so it can be restored multiple times without losing the saved data (the base save area cannot be protected like data save slots can be; - see below for more details).

Finally, we have stack save areas. Stack save areas are like a stack of plates - you put plates on top of the stack, and when you want one, you take one off the top of the stack. The plate at the bottom of the stack was the first plate placed on it, and to get to it, you need to take off each of the plates above it to get to it. When you place a data table onto a save stack, it is like placing a plate on top of our stack of plates. This is extremely useful (in fact, critical) to creating dialog boxes that overlap menus, and when you close the dialog box, you restore the menu's environment. The act of placing a data table onto a data save stack is

called "pushing" a table onto the stack. The act of removing a table from the top of the stack is called "popping" a table from the stack. In order to remember where the "top" of the stack is (i.e., how many layers are on the stack), something called a "stack pointer" is maintained internally which is simply a count of the number of items on the stack. Each time you add a new data table onto the stack, the pointer is increased by one (we have one more item on the stack).

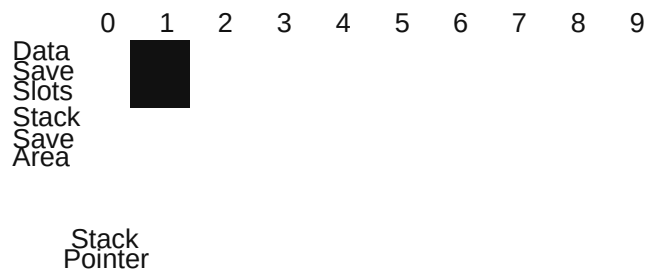
You can think of data tables and the data backup area conceptually in the following manner:



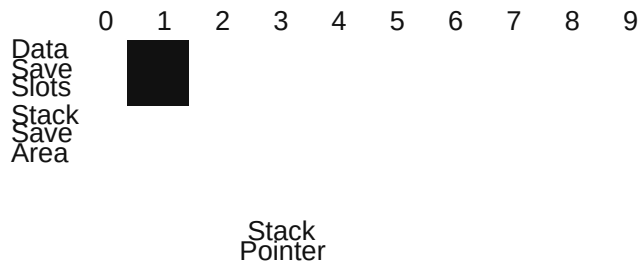
the save stack. (The slotstack system is effectively "full.") The diagram above shows a completely empty data backup area.

Let's take an example. Let's say we have tables saved to data save slots 1, 3, 4, 6, 8 and 9 (six tables total). We could envision our data

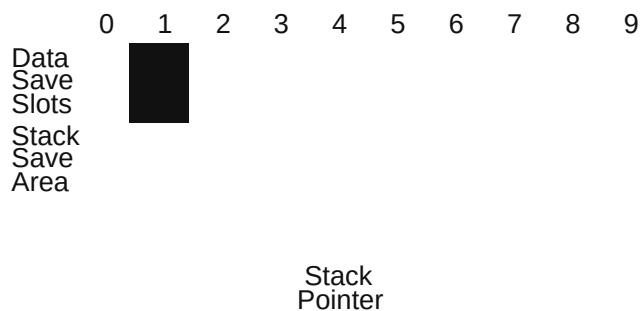
slot/stack areas like this (we have nothing on the stack yet in this example, so our stack pointer is 0, and points to the first stack entry which is open for filling):



If we now push three data tables onto our stack, we would have the following diagram showing the data slot/stack areas:



We now have nine data tables stored in our data slot/stack system. We can only hold one more. If one more is pushed onto the stack, we would have the following:



Our stack system is now full. We have four items on our stack and six in our data save slots. If we tried to push another data table onto our stack, the operation would be ignored (an error because our slot/stack system is full). If we tried to store a data table to data save slot #2 (which is currently open), it would also be ignored because the slot/stack system is full. If however, we tried to save a data table onto data save slot #3 (which is occupied), the operation would be performed, provided that data save slot #3 is not protected (see below).

A stack restore (also known as a "pop") is a method of restoring a data object from the data save stack into a data table. When you perform a stack pop, the data table at the top of the stack (the entry just below the stack pointer) is removed from the stack and placed in the appropriate data table (which one is based on the type of data that is stored in the data backup area). Once the restore operation is complete, the entry in that location of the stack is deleted (cleared), and the stack pointer is decremented by one, thus making room for one more entry in the data save slot/stack system to hold a new data table.

Each data backup area maintains its own stack pointer. Each backup area's stack pointer is unique and completely unrelated to the stack pointers of other backup areas.

The base data save area is provided as a "temporary" storage location for data tables. The slot/stack system is designed for more permanent storage of tables.

All totaled, a data backup area can store eleven separate data tables, and each data table in a data backup area contains the same type of information (e.g., text window, color palette, etc.). If a data table has 36 separate data table entries in it, then a data backup area for that data table would hold 396 separate data table entries, and would make for a lot of backup storage. You cannot individually access sub-elements of a data table inside of a data backup area; you can only save/restore "entire" data tables at any one moment.

Push and pop operations are vital to the concept of multiple overlapping windows and dialog boxes in RIP *scrip*. When you close a window, you need to restore the background graphics, mouse regions, text windows and any other data tables that were in use at the time the dialog/window was created. What if that restored background is itself a window which resides some background menu? That window should also have pushed

the data tables when it began. When that window closes, it will pop the information and restore the background environment's configuration.

COPYING DATA SAVE OBJECTS



7. Copy an entry in a data table over another entry in the same data table
8. Copy a data save slot to the base save area
9. Copy the base save area to a data save slot
10. Push the base save area onto the data save stack
11. Pop a data table from the stack into the base save area
12. Push a data save slot table onto the stack
13. Pop a data table from the stack into a data save slot
14. Copy a data save slot to another data save slot

DATA BACKUP AREAS - PROTECTION AND RESTORATION

The way that a stack operates has been clearly outlined in the previous section. When you store a data table onto the stack, it grows by one table. When you pop a data table off of the stack, you reduce the contents of the stack by one data table (i.e., that data table is deleted from the stack).

The base save area has already been described as a temporary storage location. A data table in the base save area remains in the base save area until it is either explicitly deleted, or until a hard reset of the *RIPscrip* environment is performed. In this manner, the base save area can be used as a source of a data table copy operation many times and still read the same data table.

When you restore a data table from a data save slot back into the actual data table on the other hand, that data save slot is deleted (cleared) unless it is protected from deletion. In other words, just as with individual data table entries, you can individually protect data save slots from deletion or overwriting. Only a data save slot that is actually "in use" can be protected - it is pointless to protect a data save slot that isn't holding a data table because what would you do with it? You couldn't store anything in it and you couldn't read anything from it, so this situation is not allowed.

There are only three ways of deleting or modifying a protected data save slot (two of which involve manually unprotecting the slot):

1. Explicitly unprotect the slot and then overwrite or delete it.
2. Explicitly unprotect the slot then copy the data table from it to the actual data table designed to work with that data. This deletes the contents of the slot once the copy operation is complete.
3. Perform a hard reset of the *RIPscrip* environment.

The base save area cannot be protected and neither can the data save stack. Protection goes against the very nature of these two backup areas.

It should be noted that when a data table is stored in the data backup area, any of the entries in that data table "retain" their protection status. This means that if you restore a data table from the backup area and put it back into action, the protection status of each entry in the table is restored as well.

You cannot protect an entire data table. This means that if you have entries in a data table that are protected and you perform a restore operation from a backup area, this will override any individual entries in the data table with those stored in the actual data table in the backup area. This is the only way that a data table entry's protection status can be bypassed.

INDIVIDUAL DATA BACKUP AREAS

There are data backup areas for the following types of information (not all of them are normal data tables per se):

BUTTON STYLE TABLE SAVE AREA

A backup area for an entire button style data table (all 36 button styles that may be defined).

This backup area stores the following information:

- The current button style data table entry number
- The actual button style data table (all 36 entries)

GRAPHICAL STYLE TABLE SAVE AREA

A backup area for an entire graphical style data table (all 36 graphical styles that may be defined).

This backup area stores the following information:

- The current graphical style data table entry number
- The actual graphical style data table (all 36 entries)

DRAWING PORT TABLE SAVE AREA

A backup area that stores all 36 separately defined graphical ports in the Port Table. Video Ports save only the specific definition of the ports. Clipboard Ports store not only the port definition, but also the graphical data that is saved inside the Clipboard Ports.

This backup area stores the following information:

- The current drawing port data table entry number
- The actual drawing port data table (all 36 entries)
- All offscreen/clipboard ports' actual graphical contents
- The current "floating" viewport query expression
- The current "clipboard port pointer" (see RIP_GET_IMAGE for more details).

TEXT WINDOW TABLE SAVE AREA

A backup area that stores the entire text window data table. This stores all 36 of the possibly defined text window configurations. Only the text window definitions are saved in these areas, not the contents of the text window(s).

This backup area stores the following information:

- The current text window data table entry number
- The actual text window data table (all 36 entries)
- The current "floating" text window query expression

COLOR PALETTE TABLE SAVE AREA

A backup area that stores all 36 currently defined color palettes in the color palette data table.

This backup area stores the following information:

- The current color palette data table entry number
- The actual color palette data table (all 36 entries)

MOUSE FIELD TABLE SAVE AREA

A backup area that stores all currently defined simple mouse fields and mouse button fields currently active on the screen. The mouse field boundaries and any mouse button configuration data is stored. Mouse fields do not have a directly accessible data table. In other words, you cannot directly protect a specific entry in the mouse field data table. The mouse field data table is different than normal data tables in that they can have up to 128 separate table entries and are not directly accessible like other simpler data tables like graphical style data tables, etc. When a mouse button definition is restored, the graphics displayed for a button are not restored - only the internal mouse button field definitions are restored.

This backup area stores the following information:

- The total number of mouse fields defined
- The actual mouse field data table (up to 128 entries)

SCREEN SAVE AREA

The screen save area is a special data backup area. The screen backup area has no real data table related to it, although you could think of it as a data table of video scan lines where each entry in the data table is one scan line of graphical data. The data save slots and the base save area of the screen backup area each contain a complete bitmap representation of a graphical screen, and its associated color palette. This backup area embodies some of the concepts of the drawing port backup area when it stores a Clipboard Port and also parts of the color palette backup area. This backup area allows you to have up to eleven separately saved graphical screens.

This backup area stores the following information:

- The current color palette actually in use on the video hardware (if any)
- The graphical screen contents of the data screen
- The on/off status of the status bar.

ENVIRONMENT TABLE SAVE AREA

The environment save area is a backup area to store the entire environment data table (all 36 entries). It is used for the purpose of backing up the entire environment table for later restoration.

This backup area stores the following information:

- The current environment data table entry number
- The actual environment data table (all 36 entries)

TEXT VARIABLES REFERENCE

This section details all of the pre-defined text variables in the RIP *scrip* language. Each variable is described thoroughly, and where applicable, simple ANSI-C source code extracts are provided to show how to implement the variable under the C programming language.

TEXT VARIABLE SYNTAX DESCRIPTIONS

Each text variable's description details the exact syntax of that variable. The syntax is described in concise detail so that you can easily spot at the glance of an eye exactly what parameters are allowed, and if omitted, what the default values are for those parameters. You can also determine what text variables have optional parameters, and what parameters are required.

There are two basic text variables - those that take parameters, and those that don't. Those that don't use any parameters are the simplest of all to describe syntactically. If a text variable does not require any parameters, then its syntax description would be nearly identical to the following:

Syntax: \$TEXTVAR\$

This simply states that in order use this text variable in an expression, you insert the text \$TEXTVAR\$ in your host command.

For text variables that require parameters though, more detailed descriptions are necessary. Here is an example listing of a text variable requiring one parameter and having an optional second parameter:

Syntax: \$TEXTVAR(*req:PARAM1* , *opt:PARAM2*)\$

You should notice the text *req:* and *opt:*. The *req:PARAM1* text indicates that the parameter named *PARAM1* is required. The second piece of text is *opt:PARAM2* , which means that the parameter named *PARAM2* is optional.

But what values can *PARAM1* or *PARAM2* be? It's not mentioned at all what values they can obtain. This is where the parameter value notation

comes in. Suppose that *PARAM1* can be set to TRUE or FALSE, and that *PARAM2* can be set to BLUE and RED.

To describe these values our new syntax description would be:

Syntax: \$TEXTVAR(*req:PARAM1*, *opt:PARAM2*)\$

PARAM1	TRUE	Use borders
	FALSE	Don't use borders
PARAM2	BLUE	Sets drawing color to blue
	RED	Sets drawing color to red

From the above description, it is quite simple to determine what parameters are required, which ones are optional, and which ones can be set to what values. Also notice the descriptions of the different settings on the right hand side.

Now, let's add a new color to *PARAM2*, called GREEN, and let's say that it can only be used as *PARAM2* when *PARAM1* is equal to TRUE. We denote this like the following:

Syntax: \$TEXTVAR(*req:PARAM1*, *opt:PARAM2*)\$

PARAM1	TRUE	Always use borders
	FALSE	Don't use borders
	MAYBE	Use borders if needed
PARAM2	BLUE	Sets drawing color to blue
	RED	Sets drawing color to red
	*GREEN	Sets drawing color to green

* Only valid if PARAM1 = TRUE or PARAM1 = MAYBE

This same situation could be written with the following:

Syntax: \$TEXTVAR(*req:PARAM1*, *opt:PARAM2*)\$

PARAM1	TRUE	Always use borders
	FALSE	Don't use borders
	MAYBE	Use borders if needed

Text Variable Reference

PARAM2	BLUE	Sets drawing color to blue
	RED	Sets drawing color to red
	*GREEN	Sets drawing color to green

* Not valid if PARAM1 = TRUE

This means that *PARAM2* can be set to GREEN only if *PARAM1* is not equal to FALSE.

Optional parameters do not need to be specified. If they are not, then some "suitable" default value will be used for that parameter. Let's take a variation of our previous example:

Syntax: \$TEXTVAR(*req:PARAM1*, *opt:PARAM2*)\$

PARAM1	TRUE	Always use borders
	FALSE	Don't use borders
	MAYBE	Use borders if needed
PARAM2	BLUE	Sets drawing color to blue
	RED	Sets drawing color to red
	*GREEN	Sets drawing color to green
	default = BLUE	

This says that if *PARAM2* is omitted, that BLUE will be the default. What if BLUE is only the default when *PARAM1* is set to TRUE, and RED is default all the other times? Then you would have the following syntax description:

Syntax: \$TEXTVAR(*req:PARAM1*, *opt:PARAM2*)\$

<i>PARAM1</i>	<i>TRUE</i>	Always use borders
	<i>FALSE</i>	Don't use borders
	<i>MAYBE</i>	Use borders if needed
PARAM2	BLUE	Sets drawing color to blue
	RED	Sets drawing color to red
	GREEN	Sets drawing color to green
	default = *RED/BLUE	

* If *PARAM1* = TRUE default = BLUE

Lastly, a number of text variables have a variable number of parameters. This means that one of the parameters can be repeated more than once, but it must be specified at least once. This is represented in the actual text variable "short description" with an ellipse "..." shown in the parameter list like the following:

`$TEXTVAR(param1,param2,...)$` ... Perform some kind of operation

In the syntax description, it is also shown as an ellipse "...". The parameter immediately preceding the ellipse is the one that is repeated and all subsequent instances of that parameter use the same syntax as the first one repeated. Here is an example if we allowed *PARAM2* to be repeated multiple times:

Syntax: `$TEXTVAR(req:PARAM1, req:PARAM2, ...)$`

PARAM1	TRUE	Always use borders
	FALSE	Don't use borders
	MAYBE	Use borders if needed
PARAM2	BLUE	Sets drawing color to blue
	RED	Sets drawing color to red
	GREEN	Sets drawing color to green
	default = BLUE	

Notice how *PARAM2* is set to *req:* . This means that *PARAM2* must be specified at least once, but can be repeated. It is also possible that the repeated parameter itself may be optional (e.g., *opt:*). If this is the case then there can be zero or more occurrences of that parameter.

If no specific default value is allowed for an omitted parameter (i.e., it has special significance if it is omitted), then the default value would be listed as `<none>` .

TEXT VARIABLE DESCRIPTIONS

The following sub-sections detail all of the pre-defined text variables in the RIP *scrip* language. They are organized alphabetically for your convenience.

\$ADOW\$... Abbreviated Day of Week

Format: \$ADOW\$

Syntax: \$ADOW\$

This Text Variable returns the current day of the week in abbreviated form. Possible values are: Sun, Mon, Tue, Wed, Thu, Fri and Sat.

Example: \$ADOW\$

Returns: Mon

\$ALARM\$.. Warning! This sound indicates failure!

Format: \$ALARM(*count*)\$

Syntax: \$ALARM(*opt:COUNT*)\$

COUNT	1-65535	Number of times to repeat
	default = 3	

This Active Text Variable produces a warning sound, indicating failure of an action. This sound is used for aborted downloads.

This command doesn't require any parameters. If none are specified, then the count is assumed to be 3. The count parameter is the number of times that the warning sound is repeated.

\$AMPM\$... Returns AM or PM depending on time

Format: \$AMPM\$

Syntax: \$AMPM\$

This Text Variable returns a two-character value of either AM or PM depending on what time it is.

Example: \$AMPM\$

Returns: PM

\$APP\$... External Application Call

Format: \$APP(*appno*,*argument*)\$

Syntax: \$APP(*opt:APPNO*, *opt:ARGUMENT*)\$

APPNO	0-9 default = 0	Application number to execute
-------	--------------------	-------------------------------

ARGUMENT	Text to be added to the command line argument of the application when it is run.
----------	--

This text variable instructs the terminal to execute an external application. If you do not specify any parameters, then application number zero will be executed. If you do specify any parameters, you may specify one or both of them. If you supply the *ARGUMENT* parameter, you must specify the *APPNO* option. *APPNO* is a number from 0-9 indicating which application should be executed. If used, the *ARGUMENT* parameter is an arbitrary command line argument that will be added to the end of the application's command line string. This gives the ability to allow the host to control portions of the external application's operation.

Example: \$APP(0,FILENAME.TXT)

Returns: *nothing*

\$APPx\$... External Application Call (x=0-9)

Format: \$APP0\$...\$APP9\$

Syntax: \$APP0\$... \$APP9\$

This active text variable instructs the terminal to execute an external application. By recommendation, \$APP0\$ is the user's text editor. There are ten external application slots available, numbered 0 - 9 . These are defined in the External menu in RIPterm.

Text Variable Reference

This command is obsolete. You should be using the `$APP(appno)$` command as it is more general in nature. Other than that, the new APP command performs the exact same operation. If you need to pass custom arguments to the external application, then use the more generic `$APP(appno)$` command instead.

Example: `$APP1$`

Returns: *nothing*

`$ATW$` ... Activates a text window definition

Format: `$ATW(window1,window2,... )$`

Syntax: `$ATW(opt:WINDOW1, ...)$`

WINDOW1	ALL	Activate all Text Windows
	CUR	Activate the Current Text Window
	0-35	Activate Text Window number 0-35
	default = CUR	

This command does the exact opposite of the `$DTW$` text variable which deactivates a text window. When a window is activated, it no longer discards raw ANSI text that is sent to it. This does not make any visual changes on the screen immediately unless the current text window is the one activated, whereby the cursor might appear all of a sudden.

You may specify one or more parameters to indicate that you wish to activate more than one text window in the same `$ATW$` statement. If any one of them fails to match the proper parameters, then the entire command is considered a syntax error and none of the parameters are processed.

If the text window being activated is protected, then it is not changed (i.e., activated).

Example: `$ATW(CUR)$`

Returns: *nothing*

`$AVP$` ... Activates a viewport definition

Format: `$AVP(port1,port2,... )$`

Syntax: \$AVP(*opt:PORT1, ...*)\$

PORT1	ALL	Activate all Viewports
	CUR	Activate current Viewport
	0-35	Activate Viewport number 0-35
	default = CUR	

This command does the exact opposite of the \$DVP\$ text variable which deactivates a viewport. When a viewport is activated, it no longer discards graphical commands that are sent to it. This does not make any visual changes on the screen immediately - they only affect subsequent graphical RIP *scrip* operations that are received when that viewport is selected as the current viewport. You may activate more than one viewport by specifying multiple viewport slot numbers.

You may specify one or more parameters to indicate that you wish to activate more than one viewport in the same \$AVP\$ statement. If any one of them fails to match the proper parameters, then the entire command is considered a syntax error and none of the parameters are processed.

If the port you are trying to activate is protected, then it is not changed (i.e., activated). You may activate more than one viewport.

Example: \$AVP(CUR)\$
Returns: *nothing*

\$BACKSTAT\$... Return backup area status information

Format: \$BACKSTAT(*type*)\$

Syntax: \$BACKSTAT(*req:TYPE, opt:MODE*)\$

<i>TYPE</i>	TW	Text Window Backup Area
	BUT	Button Style Backup Area
	STYLE	Graphical Style Backup Area
	PORT	Drawing Port Backup Area
	MOUSE	Mouse Field Backup Area
	PAL	Color Palette Backup Area
	ENV	Environment Backup Area

Text Variable Reference

	SCREEN	Screen Backup Area
MODE	USE	Reports what areas are in use
	PROT	Reports what areas are protected
	default = USE	

This function returns status information on the specified data backup area. The information returned provides a detailed breakdown of the specified data backup area.

The *MODE* parameter defines what kind of backup area status information you are requesting. If it is omitted, or specified as *USE*, then the data returned is a composite string of values detailing which areas of the backup area are "in use". The format of this string of text is:

base:stack:slots:free:s0:s1:s2:s3:s4:s5:s6:s7:s8:s9

The *base* field is set to 0 if the base save area is not in use, or 1 if it is in use. The *stack* field is set to the number of entries that are currently saved on the stack (0 if its empty). The *slots* field states how many entries are saved in the data save slots. The *free* field determines how many stack/slot areas are not in use (i.e., how many more can hold data). The final ten entries *s0* through *s9* are set to 0 or 1 indicating if that specific data save slot is currently in use or not. If you add up the *stack* and *slots* value, you will get a total value of entries stored in the stack/slot system. If you add this total value up with the value of *free*, then you get the maximum number of entries allowed to be saved in the stack/slot system (currently this should be set not exceed 10).

If the *MODE* parameter is set to *PROT*, then you are requesting which data save slots in the data backup area are protected or not. The format returned to the host is:

s0:s1:s2:s3:s4:s5:s6:s7:s8:s9

The contents of the *s0* through *s9* fields will be set to 0 to indicate that the data save slot is not protected, or 1 if it is protected.

Example: \$BACKSTAT(TW, USE)\$
Returns: 0:2:3:5:0:0:0:1:0:0:1:1:0:0

Example: \$BACKSTAT(TW, PROT)\$

Returns: 0:1:0:0:1:0:0:1:0:0

\$BASEMATH\$.. Set/query base math for RIP *scrip*

Format: \$BASEMATH(*env_no*, *setting*)\$

Syntax: \$BASEMATH(*opt:ENV_NO*, *opt:SETTING*)\$

ENV_NO	CUR	Current Enviroment setting
	0-35	Enviroment setting 0-35
	default = CUR	
SETTING	36	Use base 36 numbers
	64	Use base 64 numbers
	default = 36	

This command allows you to set or query the setting of the RIP *scrip* Base Math configuration. Valid base math settings are 36 (MegaNums) or 64 (UltraNums).

If you specify no parameters, then the base math of the current environment is returned (36 or 64).

If you specify one parameter, then you are querying the base math of a specific environment. You must specify the environment as a number from 0-35, or the value CUR to indicate the current environment. If the environment isn't in use, then a value of NONE is returned.

If you specify two parameters, then the first parameter must indicate which environment you're about to set (see above), and the second parameter must be the value 36 or 64. If the environment isn't in use then a syntax error is generated. When setting the base math, you are accomplishing the same thing as if you had used the RIP_SET_BASE_MATH command.

Example: \$BASEMATH(CUR, 36)\$

Returns: *nothing*

\$BAUDEMUL\$. Set/return baud rate emulation

Format: \$BAUDEMUL(*baud_rate*)\$

Syntax: \$BAUDEMUL(*opt:BAUD_RATE*)\$

BAUD_RATE	0-115200	Set playback rate to 0-115200 baud emulation
	default = 0	Full speed playback

This variable allows the host to determine the current baud rate emulation setting for use with local RIP file playback.

Without any parameters, this variable returns the current the baud rate setting. BAUD_RATE, if specified, actually changes the current baud rate emulation setting.

A return value of 0 means that playback will occur at full speed. Typical return values can be 0, 300, 1200, 2400, 4800, 9600, 14400, 16800, 19200, 28800, 38400, 57600 and 115200. These are not the only values that can be returned. If you want to set a baud rate emulation value of 12345 baud you can.

Example: \$BAUDEMUL\$

Returns: 9600

Example: \$BAUDEMUL(9600)\$

Returns: *nothing* - sets the current speed to 9600 baud

\$BEEP\$... Beep Sound (a.k.a. Ctrl-G)

Format: \$BEEP(*frequency,length*)\$

Syntax: \$BEEP(*opt:FREQUENCY, opt:LENGTH*)\$

FREQUENCY	1-65535	Frequency of note to play
	default = 1000	

LENGTH	1-65535	Time in milliseconds to play
	default = 75	

This Active Text Variable beeps the terminal, producing a Ctrl-G sound. No parameters are required. If none are provided then the frequency is assumed to be 1000 Hertz and the length of time that it should play is 75 milliseconds.

This command allows you to specify no parameters (default settings), only one parameter (the frequency), or both parameters (frequency and length/duration). Under no circumstances will values for any of the two parameters above 65535 be permitted. If values above these limits are encountered then the variable is not processed.

Note that this text variable has a 75 millisecond delay after the beep is complete where no sound is playing. Stringing multiple beeps together will have a noticeable gap between the sounds. To play continuous tones at different frequencies, use multiple `$T$` variables.

`$BLIP$` . Blipping Sound (like a hitting a barrier)

Format: `$BLIP(freq,length )$`

Syntax: `$BLIP(opt:FREQUENCY, opt:LENGTH)$`

FREQUENCY	1-65535	Frequency of note to play
	default = 50	

LENGTH	1-65535	Number of milliseconds to play
	default = 25	

This Active Text Variable is like `$BEEP$`, except the sound is different. It produces a barrier sound; like you're running into a wall.

This command allows you to specify no parameters (default settings), only one parameter (the frequency), or both parameters (frequency and length/duration). Under no circumstances will values for any of the two parameters above 65535 be permitted. If values above these limits are encountered then the variable is not processed.

`$CLS$` ... Clears the screen to background color (no reset)

Text Variable Reference

Format: \$CLS\$

Syntax: \$CLS\$

This command physically erases the entire screen to the current background color (color #0). No resetting of anything is performed. All this does is simply clear the screen.

Example: \$CLS\$

Returns: *nothing*

\$COFF\$... Disable the Text Cursor

Format: \$COFF(*window1*,*window2*,...)\$

Syntax: \$COFF(*opt*:WINDOW1, ...)\$

WINDOW1	ALL	In all Text Windows
	CUR	In the current Text Window
	0-35	In Text Window number 0-35
	default = CUR	

This active text variable turns off the text cursor in the specified text window(s). If that text window(s) is undefined or disabled, this command does nothing for that parameter. If the window parameters are omitted, then the cursor in the current text window is displayed. If the window parameter(s) are specified, then they can be a value from 0-35 to indicate a specific text window data table entry, the value CUR to indicate the current text window, or the value of ALL to indicate that you wish to turn off the cursor in all text windows simultaneously.

If you attempt to turn off a cursor in a text window that isn't defined, or that is deactivated, then that parameter does nothing. If any one of the parameters is invalid, then the entire command is considered a syntax error and none of the parameters are processed.

If you turn off the cursor in a text window other than the current text window, then only the internal cursor status of that text window is altered. Since the cursor wouldn't be visible in that text window anyway, you wouldn't expect anything to visually happen on the screen. If you switch to that text window though, the cursor would not be displayed upon switching to it (normally it would).

Example: \$COFF\$

Returns: *nothing*

\$COLORMODE\$... Query/alter the color mode

Format: \$COLORMODE(*env_no*, *mode*, *bits*)\$

Syntax: \$COLORMODE(*opt:ENV_NO*, *opt:MODE*, *opt:BITS*)\$

ENV_NO	CUR	In the current Enviroment
	0-35	In Enviroment number 0-35
	default = CUR	
MODE	PAL	Set to Palette mapping
	RGB	Set to RGB encoding
	default = PAL	
*BITS	1-8	Set RGB encoding to 1-8 bits

* Only used if MODE = RGB

If this variable has no parameters, then it queries the color mode of the current environment. It responds with a number that indicates whether or not the terminal is in direct RGB encoding mode or if it is in color palette mapping mode. If the value is 0, then it is palette mapping mode. If it a value from 1-8, then it is in direct RGB encoding mode and the value indicates how many bits of precision are used for the red, green and blue color components separately in RGB encoded data (a value of 8 indicates that 3x8 bits are used, or that we will be working with 24-bit color numbers).

If only one parameter is supplied, then it must be the environment number to query (0-35), or the value CUR to query the current environment. In either case, if the environment is in use, then the value returned is the same as if there were no parameters, based on the settings of that particular environment. If that environment isn't in use, then a value of -1 is returned.

If you supply two or more parameters, then you are indicating that you wish to change the color mode setting. To set the color mode to color palette mapping mode, supply the *MODE* parameter as the value PAL .

Text Variable Reference

No other parameters are needed in this case. To set RGB encoding mode, specify the *MODE* keyword of RGB and then you must supply a third parameter which indicates the number of bits of precision from 1-8 (see above). If the specified environment isn't in use, then a syntax error will be generated.

Example: \$COLORMODE\$

Returns: 0 ... *Get setting of current environment. Palette mode in this example.*

Example: \$COLORMODE(CUR, PAL)\$

Returns: *nothing ... Switch current environment to palette mode*

Example: \$COLORMODE(5, RGB, 8)\$

Returns: *nothing ... Enabled direct RGB color mode (bits=8) in environment #5*

Example: \$COLORMODE(6)\$

Returns: NONE ... *Returned if environment #6 isn't in use.*

\$COLORS\$... Total number of colors of current video device

Format: \$COLORS\$

Syntax: \$COLORS\$

This variable returns the total number of colors available on the destination video hardware device. Typical results are 2, 16, 256, 32768, 65536 and 16777216 (24 bit color). These aren't the only possible values, but are typical ones.

Example: \$COLORS\$

Returns: 256

\$COMPAT\$... Sets environment to RIP settings *scrip 1.54*

Format: \$COMPAT(*env_no*)\$

Syntax: \$COMPAT(*opt:ENV_NO*)\$

ENV_NO	CUR	Sets current Enviroment
	0-35	Sets Enviroment number 0-35
	default = CUR	

This text variable is designed to set an environment to older RIP 1.54 settings for "backward compatibility". If no parameters are specified, then the current environment will be modified to these settings. *scrip*

If you specify a parameter, then you can only set one and it must be an environment number 0-35, or the value CUR for the current environment. If the destination environment isn't in use, then a syntax error will be generated. The following environment settings are altered:

- World Coordinate Frame is set to 640x350
- Color mode is set to color palette mapping mode
- Coordinate sizes are set to 2 bytes
- Base math is set to base-36 numbers (MegaNums)
- Baud rate emulation is set to full speed (0)

Example: \$COMPAT(5)\$

Returns: *nothing*

\$CON\$... Enable the Text Cursor

Format: \$CON(*window1*,*window2*,...)\$

Syntax: \$CON(*opt*:WINDOW1, ...)\$

WINDOW1	ALL	In all Text Windows
	CUR	In the current Text Window
	0-35	In Text Window number 0-35
	default = CUR	

This Active Text Variable turns on the text cursor in the specified text window(s). If that text window is undefined or deactivated, this command does nothing. If the window parameter(s) are omitted, then the cursor in the current text window is displayed. If the window parameter is specified, then it can be a value from 0-35 to indicate a specific text window data table entry, the value CUR to indicate the

Text Variable Reference

current text window, or the value `ALL` to indicate all defined text windows (i.e., in use).

If you attempt to turn on a cursor in a text window that isn't defined, or that is deactivated, then that parameter does nothing.

If any one of the parameters is invalid, then the entire command is considered a syntax error and none of the parameters are processed.

If you turn on the cursor in a text window other than the current text window, then only the internal cursor status of that text window is altered - it will not be turned on until you switch to that text window. In this respect, only altering the current text window's cursor will actually make any immediate visible effect on the screen.

Example: `$CON$`

Returns: *nothing*

`$COORDSIZE$` ... Set byte-size of X/Y coordinates

Format: `$COORDSIZE(env_no, size )$`

Syntax: `$COORDSIZE(opt:ENV_NO, opt:SIZE)$`

ENV_NO	CUR	In current Enviroment
	0-35	In Enviroment number 0-35
	default = CUR	
SIZE	2-5	Set X/Y coordinate size to 2-5
	default = 2	

This command sets or queries the setting of the byte-width of RIP *scrip* coordinate parameters in raw RIP *scrip* code. If no parameters are specified, then this command returns the setting of the coordinate size in the current environment. The valid return values are 2 through 5.

If you specify only one parameter, then it must be an environment number from 0-35, or it must be `CUR` to indicate the current environment. If the environment isn't in use when querying it's coordinate size, then the value -1 is returned.

If you intend to set the environment's coordinate size, then you must specify two parameters. The first one is the environment number (see above), and the second one must be the coordinate size itself (2-5). If you specify an environment that isn't in use, or an invalid coordinate size value then a syntax error is generated.

Example: \$COORDSIZE(CUR, 2)\$

Returns: *nothing*

Example: \$COORDSIZE(5)\$

Returns: 2

\$COPY\$... Copy object to 1/more locations

Format: \$COPY(*type,source,dest1,...*)\$

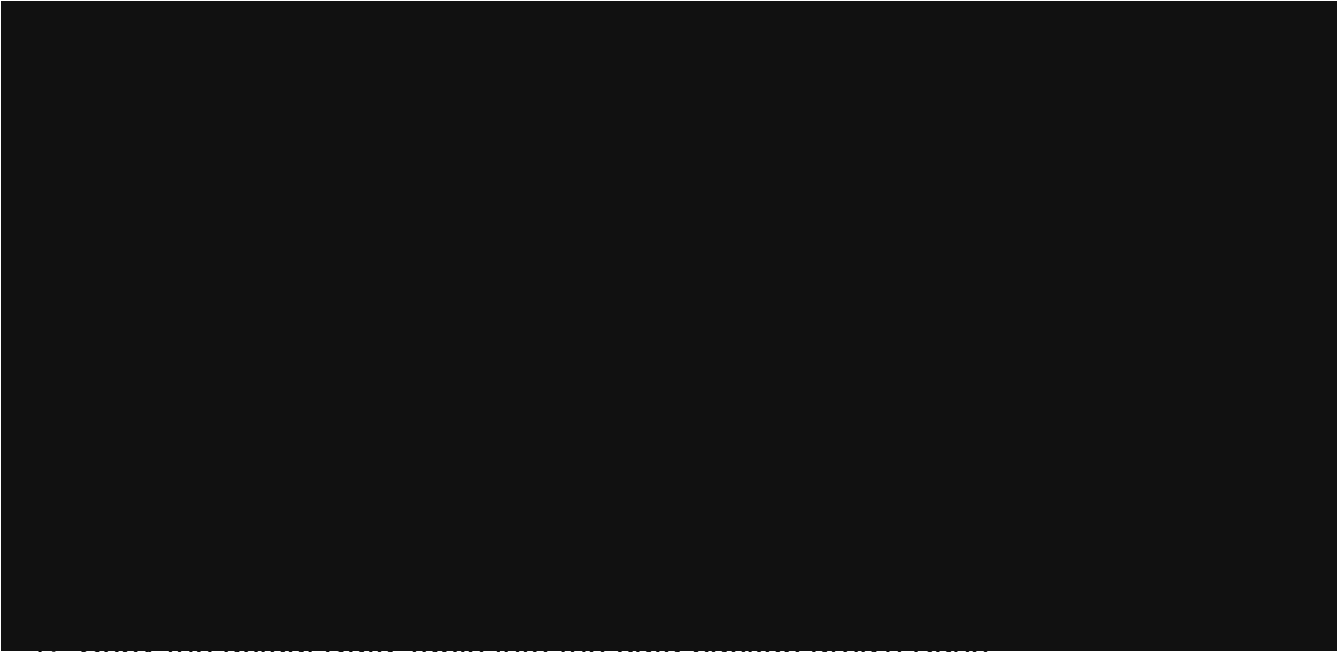
Syntax: \$COPY(req:TYPE, req:SOURCE, req:DEST1,...)\$

TYPE	TW	Text Window data
	BUT	Button Style data
	STYLE	Graphics Style data
	PAL	Color Palette data
	PORT	Drawing Port data
	MOUSE	Mouse Field data
	ENV	Environment data
SOURCE	SCREEN	Graphical Screen data
	*CUR	Copy data from current entry
	*0-35	Copy data table entry number 0-35
	TBL	Copy entire data table
	BASE	Copy from Base save area
	S0-S9	Copy from Slot number 0-9
	POP	Copy from top of data save stack
DEST1	*CUR	Copy onto current entry
	*0-35	Copy onto data table entry number 0-35
	TBL	Copy onto entire data table
	BASE	Copy onto Base save area
	S0-S9	Copy onto Slot number 0-9
	PUSH	Push copy onto top of data save stack

* Not valid if TYPE = MOUSE, SCREEN or PORT

Text Variable Reference

This complex command embodies many different "data copy" operations. The *TYPE* parameter defines what type of data object is to be copied from one location to another. The basic "methods" of copying data objects around can be described visually as follows:

- 
1. Copy the actual Data Table into the data backup area's Base Data Save Area.
 2. Copy the actual Data Table into a specific backup area's Data Save Slot (via a slot index number from 0-9).
 3. Copy the Base Data Save Area to the actual Data Table
 4. Copy a Data Save Slot (specified with an index number) directly into the actual Data Table.
 5. Push an actual Data Table onto the Data Save Slot "stack" via the stack pointer.
 6. Pop an actual Data Table off of the Data Save Slot "stack" via the stack pointer.
 7. Copy one Data Save Slot to another Data Save Slot (both slots are specified with slot index numbers).
 8. Copy a Data Save Slot (specified with an index number) into the Base Data Save Area.
 9. Copy the Base Data Save Area into a specific Data Save Slot (specified by a slot index number from 0-9).
 10. Push the Base Save Area onto the "stack" via the stack pointer.
 11. Pop an actual Data Table off of the "stack" and place it into the Base Save Area.

12. Push a Data Save Slot onto the "stack" via the stack pointer
13. Pop an actual Data Table off of the "stack" and place it into a specific Save slot.

Copying Data Table Entries around (intra-table copying)

14. Copy one entry in an actual Data to one or more other entries inside the same Data Table

When you are referring to a copy operation that involves the entire data table, you use the keyword `TBL`. When referring to the base data save area, the keyword `BASE` is used. A data save slot number is specified with the letter `S` followed by the slot index number (e.g., `S0`, `S3`, `S5`, etc.). An actual data table entry number is specified simply as an actual number from 0 to the total number of entries in the data table minus one (e.g., 35).

Destinations for entire data tables can be the `BASE` save area, a data save slot (`S0-S9`) or you can `PUSH` it onto the stack.

You can also specify the source as `POP` to indicate that you are copying (popping) data from the stack to some other destination (`TBL`, `BASE` or `S0-S9`). If you specify only one destination location, then the contents of the stack are copied to that location, then the stack is "popped", which basically deletes the top item on the stack (the one that was just copied). If you have multiple destinations, then the stack item is copied to each of those locations, then it is finally deleted from the stack after all of the copy operations are complete. If the source refers to an entry in a data table (not the entire data table), then the destination(s) must also be individual data table entry numbers.

Some data tables allow you to directly access individual data entries inside the data table (i.e., text window tables, graphical style tables, color palette tables, button style tables and drawing port tables). You cannot directly access data table entries for the Mouse Field data table, and you cannot directly access the data table entries for the screen data table. In these two cases, copying one data table entry over another entry in the same data table entry is not allowed.

Some data tables allow you to have a specific entry selected as the current table entry. For example, the text window data table has one

Text Variable Reference

data table entry selected as the current text window at any one time. When a data table allows an entry to be the current entry, then the keyword CUR may be used to specify the current table entry when performing copy operations from one data table entry to another entry. You cannot copy a data table over itself, and you cannot copy a data table entry on top of itself. All destination parameters are checked for validity before any actual copy operations are performed. If any of them fail a syntax check then the entire \$COPY\$ expression is rejected as a syntax error.

Our of all six data tables that have 36 separate entries, one of them doesn't permit direct copying from one entry to another within the same data table. That table is the drawing port table. The reason for this is because of how complicated the subject of performing port copying from one entry to another. Do you copy over just the graphical data, and if so, do you stretch it to fit in the destination port if the dimensions of the ports don't match? On the other hand do you duplicate the port definition entirely, potentially copying over the bitmap data? If so, how do you handle copying to port 0 which cannot be deleted or re-defined? With all of these complicated issues involved, it was decided not to allow for ports to be copied from one entry to another. Perhaps in the future we will allow for this kind of operation when more research on the issues can be performed.

Note that copying individual text window table entries, or individual drawing port entries around also copies around the resident query associated with that particular text window or port!

The following sections describe the various combinations of copy parameters:

Copy actual Data Table into the data backup area's Base Save Area

\$COPY(TW, TBL, BASE)\$

Copy actual Data Table into a specific backup area's Data Save Slot

\$COPY(TW, TBL, S3)\$

Copy the Base Data Save Area to the actual Data Table

\$COPY(TW, BASE, TBL)\$

Copy a Data Save Slot directly into the actual Data Table

`$COPY(TW, S3, TBL)$`

Copy a Data Save Slot into the Base Data Save Area

`$COPY(TW, S3, BASE)$`

Copy the Base Data Save Area into a specific Data Save Slot

`$COPY(TW, BASE, S3)$`

Copy one Data Save Slot to another Data Save Slot

`$COPY(TW, S3, S5)$`

Push an actual Data Table onto the Data Save Slot "stack"

`$COPY(TW, TBL, PUSH)$`

Pop an actual Data Table off of the Data Save Slot "stack"

`$COPY(TW, POP, TBL)$`

Copy one entry in an actual Data Table to one or more other entries

`$COPY(TW, 3, 5)$`

You can also combine various copy operations into one copy command like this:

`$COPY(TW, TBL, S3, BASE, S5)$`

Copies the entire data table into data save slot #3, #5 and into the base data save area.

`$COPY(TW, CUR, 5, 7, 9)$`

Text Variable Reference

Copies the current data table entry into data table entry numbers 5, 7 and 9 (overwriting those entries with the data inside the current data table entry).

\$CUR\$... Select/query current data table entry

Format: \$CUR(*type,which*)\$

Syntax: \$CUR(*req:TYPE, opt:WHICH*)\$

<i>TYPE</i>	<i>TW</i>	Text Window data table
	<i>PORT</i>	Drawing Port data table
	<i>STYLE</i>	Graphical Style data table
	<i>PAL</i>	Color Palette data table
	<i>BUT</i>	Button Style data table
	<i>ENV</i>	Environment data table
<i>WHICH</i>	0-35	Select data table entry 0-35

This variable sets or inquires about the current data table entry number associated with a specific data table indicating by the *TYPE* parameter (which must be specified).

The *WHICH* parameter is only used when setting the current entry for the given data table. If you omit the *WHICH* parameter, then you are asking the terminal "which entry in the specified data table is the current one?". In situations like this, this text variable will return a value from 0-35.

If the *WHICH* parameter is specified, then you are not inquiring about the current entry in that data table, you are setting it. Possible values for *WHICH* are numbers from 0-35 to indicate which data table entry you wish to make the current one.

If the entry you specify isn't defined, then it is set to some suitable defaults based on the following table:

Data Table	
TW	The text window is made full screen in the user's default font.
PORT	The port is defined as a screen port occupying the entire screen. The viewport is made the full size of the port.
STYLE	The basic graphics style used upon a \$RESET\$ operation is activated.
PAL	The standard 256 color lookup table is established and activated.
BUT	The basic button style used upon a \$RESET\$ operation is activated.
ENV	A basic environment is established. 640x350 world coordinates, 2-byte coordinates, color palette mapping mode, mouse cursor 0, etc.

Example: \$CUR(TW, 5)\$

Returns: *nothing*

Example: \$CUR(TW)\$

Returns: 5

\$CURSOR\$... Text Cursor Status

Format: \$CURSOR(*window*)\$

Syntax: \$CURSOR(opt:WINDOW)\$

WINDOW	CUR	Current Text Window
	0-35	Text Window number 0-35
	default = CUR	

This text variable returns YES if the Text Cursor is enabled in the specified text window, and NO if the Text Cursor is deactivated. If the specified text window doesn't exist, or is currently deactivated, then this command returns a value of NO.

If you do not specify a window parameter, then this command refers to the current text window. If you specify the window number parameter,

Text Variable Reference

then you can specify a value from 0-35 to indicate a specific text window data table entry, or a value of CUR to indicate the current text window.

Example: \$CURSOR\$

Returns: YES

\$CURX\$... Text Cursor X Coordinate

Format: \$CURX(*window*)\$

Syntax: \$CURX(*opt:WINDOW*)\$

WINDOW	CUR	Current Text Window
	0-35	Text Window number 0-35
	default = CUR	

This text variable returns the X coordinate of the text cursor in the specified text window, relative to the upper left of the Text Window. The location is "one based", so the first column of data is a value of 1, not 0. Typical return values for valid cursor locations could easily range from 1-91. If the specified text window is not defined, or is currently deactivated (via the \$DTW\$ command for example), then this text variable returns a value of 0 to indicate that the information isn't available.

If you do not specify a window parameter, then this command refers to the current text window. If you specify the window number parameter, then you can specify a value from 0-35 to indicate a specific text window data table entry, or a value of CUR to indicate the current text window.

Example: \$CURX(4)\$

Returns: 2

\$CURY\$... Text Cursor Y Coordinate

Format: \$CURY(*window*)\$

Syntax: \$CURY(*opt:WINDOW*)\$

WINDOW	CUR	Current Text Window
	0-35	Text Window number 0-35

default = CUR

This Text Variable returns the Y coordinate of the text cursor in the specified text window, relative to the upper left of the Text Window. The location is "one based", so the first column of data is a value of 1, not 0. Typical return values for valid cursor locations could easily range from 1-43. If the specified text window is not defined, or is currently deactivated (via the `$DTW$` command for example), then this text variable returns a value of 0 to indicate that the information isn't available.

If you do not specify a window parameter, then this command refers to the current text window. If you specify the window number parameter, then you can specify a value from 0-35 to indicate a specific text window data table entry, or a value of CUR to indicate the current text window.

Example: `$CURY(4)$`

Returns: 5

`$D$ ... Delay for a number of milliseconds`

Format: `$D(duration)$`

Syntax: `$D(req:DURATION)$`

DURATION	1-65535	Time of delay in 60ths of a second
----------	---------	------------------------------------

This command causes a delay to occur. During this time, the terminal program "stops" everything except for any sound related activity. The *DURATION* parameter must be specified. Its value is specified in 60ths of a second.

This command is useful to pause for a short time for things like prompts to go by or whatever.

Example: `$D(60)$`

Returns: *nothing*

`$DATE$ ... Date in short format`

Text Variable Reference

Format: \$DATE\$

Syntax: \$DATE\$

This Text Variable returns the current date. in the format *MM/DD/YY*

Example: \$DATE\$

Returns: 12/19/93

\$DATETIME\$... Date and Time

Format: \$DATETIME\$

Syntax: \$DATETIME\$

This Text Variable returns a combination date and time. The format is somewhat different than standard time/date notation. It is:

DAY-OF-WEEK MONTH DAY-OF-MONTH HH:MM:SS YEAR

Example: \$DATETIME\$

Returns: Sat Dec 19 14:38:50 1993

NOTE: *This is the standard UNIX date/time notation.*

\$DAY\$... Day of Month Number

Format: \$DAY\$

Syntax: \$DAY\$

This Text Variable returns the current day of the month. Possible values for this Variable are from 01-31.

Example: \$DAY\$

Returns: 05

\$DOW\$... Day of week fully spelled out

Format: \$DOW\$

Syntax: \$DOW\$

This Text Variable returns the current day of the week. The name is fully spelled out. Possible values are: Sunday , Monday , Tuesday , Wednesday , Thursday , Friday and Saturday .

Example: \$DOW\$

Returns: Saturday

\$DOY\$... Day of year

Format: \$DOY\$

Syntax: \$DOY\$

This Text Variable returns the number of days so far in the year. A year has 365 days (except leap years which have 366). \$DOY\$ can return 001 - 366.

Example: \$DOY\$

Returns: 214

\$DTW\$... Deactivate Text Window

Format: \$DTW(*window1*,*window2*,...)\$

Syntax: \$DTW(*opt*:WINDOW1, ...)\$

WINDOW1	ALL	All Text Windows
	CUR	Current Text Window
	0-35	Text Window number 0-35
	default = CUR	

If no slot parameter is specified then the current text window is deactivated, preventing any received raw text from being displayed in that window. Switching to another text window data table entry can re-enable text displaying if that other window is "activated".

You may specify one or more parameters to indicate that you wish to disable more than one text window in the same \$DTW\$ statement. If any one of them fails to match the proper parameters, then the entire command is considered a syntax error and none of the parameters are processed.

Text Variable Reference

If a slot (0-35) is specified then that text window slot is deactivated. This may or may not affect the current status of received text. If the slot specified just happens to be the current slot then all received text is deactivated. If the text window slot isn't the current text window then that text window is deactivated, but what is done with received raw text depends on the status of the current text window's status.

You may specify CUR to explicitly deactivate the current text window.

You may also specify a slot of ALL to deactivate all defined text window slots. Text window data table entries that are not defined, or that are protected are not affected by this operation. This would also deactivate any raw text that is received because the act of deactivating all windows will consequently deactivate the current one too.

If the text window being deactivated is protected, then it is not changed (i.e., deactivated).

This command is useful in Host Commands when you click on a Mouse Field, it would halt any further output to the text window.

Example: \$DTW\$

Returns: *nothing*

\$DVP\$... Deactivate a viewport definition

Format: \$DVP(*port1,port2,...*)\$

Syntax: \$DVP(*opt:PORT1, ...*)\$

PORT1	ALL	Disable all viewports
	CUR	Disable Current viewport
	0-35	Disable Viewport number 0-35
	default = CUR	

This command deactivates a viewport, making no more RIP *scrip* graphics commands displayable in that viewport. When a viewport is deactivated, it no longer accepts graphical commands that are sent to it. This does not make any visual changes on the screen immediately; they only affect subsequent graphical RIP *scrip* operations that are received when that viewport is selected as the current viewport. You

may deactivate more than one viewport by specifying multiple viewport slot numbers.

You may specify one or more parameters to indicate that you wish to deactivate more than one viewport in the same `$DVP$` statement. If any one of them fails to match the proper parameters, then the entire command is considered a syntax error and none of the parameters are processed.

If the port you are trying to deactivate is protected, then it is not changed (i.e., deactivated).

You may deactivate more than one viewport by specifying more than one.

Example: `$DVP(CUR)$`

Returns: *nothing*

`$DWAYOFF$` ... Turn Doorway Mode OFF

Format: `$DWAYOFF$`

Syntax: `$DWAYOFF$`

This Active Text Variable disables the Doorway keyboard mode. This will return the keyboard to normal operation.

Example: `$DWAYOFF$`

Returns: *nothing*

`$DWAYON$` ... Turn Doorway Mode ON

Format: `$DWAYON$`

Syntax: `$DWAYON$`

This Active Text Variable enables Doorway Mode. This is intended to be used by a Host system that wishes to take advantage of the Doorway mode available in Marshall Dudley's Doorway™ software package.

Text Variable Reference

Example: \$DWAYON\$

Returns: *nothing*

\$EGW\$... Erase Graphics viewport

Format: \$EGW(*port1,port2,...*)\$

Syntax: \$EGW(*opt:PORTNO, ...*)\$

PORTNO	ALL	Erase all viewports
	CUR	Erase Current viewport
	0-35	Erase Viewport number 0-35
	default = CUR	

This active text variable erases the graphics viewport (much like a Reset Windows command does). This command is useful in Host

Commands. When you click on a Mouse Field, it could erase the viewport window then transmit the remainder of the return string (if any) to the host.

Remember, this may not clear the entire screen (although it will quite often since the Graphical Viewport is often full-screen).

This command does not require the slot parameter. If it is omitted, then the current viewport is cleared. If you specify a number (0-35) then the corresponding viewport slot number is erased. A special value of ALL may be used to erase all currently defined viewports.

If any viewports specified belong to a port that isn't defined, or if the viewport is deactivated, then this command does nothing for that viewport parameter.

You can specify multiple viewport/port slots to erase to erase more than one within one command if you wish.

Example: \$EGW(ALL)\$

Erase all viewports

Returns: *nothing*

Example: \$EGW\$

Erase current viewport

Returns: *nothing*

Example: \$EGW(5)\$

Erase viewport slot #5

Returns: *nothing*

Example: \$EGW(CUR)\$

Erase current viewport

Returns: *nothing*

\$ETW\$... Erase Text Window

Format: \$ETW(*window1*,*window2*,...)\$

Syntax: \$ETW(opt:WINDOW1, ...)\$

WINDOW1	ALL	All Text Windows
	CUR	Current Text Window
	0-35	Text Window number 0-35
	default = CUR	

If no parameter(s) are specified then the current text window in use is cleared. If you specify a slot from 0-35 then the corresponding text window slot is erased. You may also specify ALL to erase all currently defined text window slots. Finally, you may specify the value of CUR to also indicate the current text window.

It should be noted that the entire text window's bounding rectangle is erased, not just the display region inside the bounding rectangle.

You may specify one or more parameters to indicate that you wish to erase more than one text window in the same \$ETW\$ statement. If any one of them fails to match the proper parameters, then the entire command is considered a syntax error and none of the parameters are processed.

If any text window specified aren't defined, or if the viewport is deactivated, then this command does nothing for that viewport parameter.

This command is useful in Host Commands when you click on a Mouse Field, it could erase the text window then transmit the remainder of the Host Command (if any).

Example: \$ETW(ALL)\$

Returns: *nothing*

\$FIELDDID\$... Returns the current mouse field's ID value

Format: \$FIELDDID\$

Syntax: \$FIELDDID\$

This text variable returns the ID value for the current mouse field or mouse-field button. This text variable is only useful in mouse field entry/exit query expressions, or in host commands used with the RIP_MOUSE and RIP_BUTTON commands. If the mouse field associated with the action has an ID value associated with it, then that ID value is inserted in place of the text variable. If the mouse field does not have an ID value, then the value " -1" is inserted in place of the text variable. This is most useful in mouse field entry/exit resident queries so that the host system can identify which mouse field was entered, but without having to send the entire host command back to the system.

Example: \$FIELDDID\$

Returns: 23

Example: \$FIELDDID\$

Returns: -1

\$FILEDEL\$... Delete one or more host files

Format: \$FILEDEL(*filename*,...)\$

Syntax: \$FILEDEL(*req:FILENAME*, ...)\$

FILENAME Name of file to delete.

This text variable exists so that a host system can clean up after itself, with the ability to delete files that it created, but no longer needs. At least one parameter must be specified, and it must be a filename that the host created, or downloaded to the terminal via RIP_ENTER_BLOCK_MODE or by some other means. No form of wildcard information is processed, nor is path information. If present, it will be ignored to the best of the terminal's ability. You may specify more than one filename parameter with this command to delete multiple files with one "shorter" command.

Example: \$FILEDEL(EMAIL.BMP)\$

Returns: *nothing*

Example: \$FILEDEL(EMAIL.BMP,FILES.BMP,NEWS.RIP)\$

Returns: *nothing*

\$FYEAR\$... 4 digit year

Format: \$FYEAR\$

Syntax: \$FYEAR\$

This Text Variable returns the four-digit number of the current year.

Example: \$FYEAR\$

Returns: 1993

\$HKEYOFF\$... Disable Button Hotkeys

Format: \$HKEYOFF\$

Syntax: \$HKEYOFF\$

This Active Text Variable turns off Button Hotkeys. This should be done when entering a full-screen editor, or any part of the system where the user is entering a string of text. This is to prevent the user from accidentally selecting a button when typing in text.

Example: \$HKEYOFF\$

Returns: *nothing*

\$HKEYON\$... Enable Button Hotkeys

Format: \$HKEYON\$

Syntax: \$HKEYON\$

This Active Text Variable turns on use of Button Hotkeys. When enabled, if the user presses a key associated with a button, it is selected

Text Variable Reference

just as if it were clicked. The Scroll Lock light on the keyboard is turned on.

Example: \$HKEYON\$

Returns: *nothing*

\$HOUR\$... Hour (format HH) - normal style

Format: \$HOUR\$

Syntax: \$HOUR\$

This Text Variable returns the two digit number of the current hour. This variable range from 01 - 12. This does not use military format.

Example: \$HOUR\$

Returns: 11

\$IFS\$... Is Feature Supported

Format: `$IFS(keyword,function )$`

Syntax: `$IFS(req:KEYWORD, opt:CATEGORY)$`

KEYWORD	CATAGORY	Returns
LIST	ALL	List of all supported features
LIST	_AUDIO	List all supported Audio formats
LIST	_EMULATIONS	List of all supported terminal emulations
LIST	_IMAGE	List of all supported Image formats
LIST	_LANGUAGES	List of all supported languages
LIST	_MISC	List of all supported Misc. features
LIST	_PROTOCOLS	List of all supported File Tranfer Protocols
LIST		List of all supported categories
ANSI		*ANSI terminal emulation
BMP		*BMP image format
CISQUICKB		*CompuServe QuickB file transfers
DOORWAY		*Doorway terminal emulation
ENG		*English language
EXTAPPS		*External applications
JPEG		*JPEG image format
KERMIT		*Kermit file transfers
RIPSCRIP		*RIPscrip terminal emulation
SKERMIT		*Super Kermit file transfers
VT102		*VT-102 terminal emulation
WAV		*WAV audio format
XMODEM		*X-Modem file transfers
XMODEMCRC		*X-Modem CRC file transfers
XMODEM1K		*X-Modem 1k block file transfers
XMODEM1KG		*X-Modem G 1k block file transfers
YMODEM		*Y-Modem file transfers
YMODEMG		*Y-Modem G file transfers
ZMODEM		*Z-Modem file transfers
ZMODEMCR		*Z-Modem crash recovery file transfers

* Returns a '1' if supported or a '0' if not supported

This is a unique and powerful text variable. This is the mechanism by which the host can find out what capabilities are supported in the remote

Text Variable Reference

terminal. For example, if the host needs to know if the terminal supports JPEG files, the host could send a query:

```
!|1_0000$IFS(JPEG)$
```

Note: The character "_" is actually an escape character (ASCII value 27).

The terminal would respond with `1' if it has JPEG display ability, or "0" if it doesn't.

Because the number of features supported by a software package can get quite extensive, they are categorized for the purposes of this command. This allows the host to be able to query particular sub-sets of information (e.g., what file transfer protocols are supported, etc.). These categories are simple keyword names just like the keyword names used to identify particular features, but with one slight difference. Categories begin with an underscore character (_) to differentiate them from actual feature keywords. These categories are used in conjunction with a special \$IFS\$ directive called LIST. This will list out all keywords for a specific category, or if no category is specified, it will list out all categories.

If you specify the LIST directive all by itself as in the text variable \$IFS(LIST)\$, then the categories will be returned in an alphabetical, comma-delimited list like this:

Example: \$IFS(LIST)\$

Returns: AUDIO,_EMULATIONS,_IMAGE,_LANGUAGES,
_MISC,_PROTOCOLS

You may list out a particular category by issuing an \$IFS\$ variable with the LIST keyword and the category as the second parameter as in the following example:

Example: \$IFS(LIST, _EMULATIONS)\$

Returns: ANSI,DOORWAY,RIPSCRIP,VT102

One final LIST directive is the ALL directive. This returns a list of all feature keywords (omitting category keywords) in one very long alphabetical, comma-delimited list. This would be like asking for each category's listing separately then sorting the list of keywords and

stringing them all together. Here's an example of the above keywords being queried in ALL mode.

Example: `$IFS(LIST, ALL)$`

Returns: ANSI,BMP,CISQUICKB,DOORWAY,ENG,
EXTAPPS,JPEG,KERMIT,RIPSCRIP,SUPERKERMIT,VT102,WAV,
XMODEM,XMODEM1K,XMODEM1KG,XMODEMCRC,YMODEM,Y
MODEMG,ZMODEM, ZMODEMCR

If you omit all parameters from the `$IFS$` variable, then it should be considered a text variable syntax error.

In the future, more categories will probably be added for different purposes like hardcopy support, network support and many other things. That is why this command has been designed with such flexibility in mind.

Note that the comma-delimited list of keywords returned from a `LIST` directive are alphabetically sorted and have no spaces in them. In addition, there are no carriage returns or any other form of delimiter after the last keyword returned. If you wish to have a carriage return after the list, place a `^M` control character directive in the query command that you used to work with this text variable.

Note, a category must have at least one keyword defined underneath it in order for it to be considered "defined".

If a `LIST` directive is specified on a category that doesn't exist, then nothing is returned to the host (a null string). If a specific keyword is inquired about and it doesn't exist, a `0` is returned to indicate that the feature isn't supported.

`$IMGSTYLE$` ... Set/query image style settings

Format: `$IMGSTYLE (port,x0,y0,x1,y1,mode,... )$`

Syntax: `$IMGSTYLE(opt:PORT, opt:X0, opt:Y0, opt:X1, opt:Y1,
opt:mode, ...)$`

PORT	CUR	Current viewport
	0-35	Viewport number 0-35

Text Variable Reference

default = CUR

X0,Y0,X1,Y1 Coordinates of display area

MODE	ASPECT	Maintain aspect ratio
	DELETE	Delete image file after display
	NOCLEAR	Do not clear display area first
	USEPAL	Use image's palette
	WALLPAP	Wallpaper image
	*STAGGER	Stagger the wallpaper

*WALLPAP must also be set for this to work

This command performs two separate functions. In one mode, it returns information about a specific drawing port's image style. In the other mode it actually modifies the image style setting of a particular drawing port.

If this command is set with zero or one parameters then it is requesting the image style settings of a specific drawing port. With no parameters, it is asking about the current drawing port's image style settings. If you specify a single parameter then it can be set to CUR to indicate the current drawing port or a number from 0-35 to request the image style settings of a particular drawing port. If the port doesn't exist or it is currently deactivated then a value of -1 will be sent to the host. If it is defined and activated then it will return a sequence in the following syntax:

x0:y0:x1:y1:flags

The (X0,Y0) values are the upper-left corner of the image style. The (X1,Y1) values are the lower-right corner of the image style. If the all four of these values are set to 0 then the image style is currently disabled. The flags value is identical to the flags parameter of RIP_IMAGE_STYLE command. The flags are returned in decimal notation, not hexadecimal. If the image style hasn't been defined yet then it will return a value of "-1:-1:-1:-1:0".

If you specify more than one parameter then you are altering a specific drawing port's image style. If that port doesn't exist, is deactivated or is protected, then this text variable is ignored. When altering an image

style, you must specify at least 5 parameters - the first parameter is CUR or 0-35 to indicate the proper drawing port to modify. The remaining four parameters are (X0,Y0), the upper-left corner of the image rectangle followed by (X1,Y1), the upper-right corner of the image rectangle. The values for these parameters are assumed to be in the current environment's world coordinate system. The valid ranges for these parameters are identical to those of the RIP_IMAGE_STYLE command (see that command for more details).

After the image rectangle parameters, you may specify zero or more mode parameters which basically activate special modes of an image style. Each mode parameter can be set to the parameters:

Mode	
ASPECT	Maintain image aspect ratio inside the image's rectangle. If this mode is not used then the image will fill the entire image rectangle regardless of the original dimensions of the image.
DELETE	This will delete the image after it has been displayed. This is only valid with RIP_ENTER_BLOCK_MODE where the image to be displayed has been downloaded from the host.
NOCLEAR	Do not clear the image area's background before drawing the image. By default the image's rectangle is erased to color 0 (typically black) before the image is drawn. Use this mode to prevent this from happening.
USEPAL	When this mode is used, any color palette data in the image file is activated in the current video system (and the current color palette table entry). This is currently only useful with GIF files that have an internal color palette - JPEG images do not have an internal color palette. If this mode is not used then the image will be displayed in the current color palette - mapping colors in the image to the closest colors currently in use.
WALLPAP	The image that is drawn to the screen is used to wallpaper the currently defined viewport with. If aspect ratio is maintained, then the image displayed after aspect ratio calculations is used to wallpaper the viewport. See the RIP_IMAGE_STYLE and RIP_LOAD_BITMAP commands for more details about wallpapering.
STAGGER	The image that is drawn to the screen is wallpapered to the viewport in a staggered manner. This keyword is useless without the WALLPAP keyword being specified. See the RIP_IMAGE_STYLE and RIP_LOAD_BITMAP command for more details about wallpapering.

Text Variable Reference

Example: MGSTYLE\$

Returns: 0:640:350:15

Example: IMGSTYLE(CUR,0,0,640,350,ASPECT, USEPAL)\$

Returns: *nothing*

\$INUSE\$... Is a data object in use?

Format: \$INUSE(*data_object*,*element*)\$

Syntax: \$INUSE(*req:OBJECT*, *req:ELEMENT*)\$

OBJECT	Element	Returns '1' if true '0' if false
TV	<name>	Text variable <name> is defined
TW	ALL	All text windows 1-35 are in use
TW	ANY	Any of text windows 1-35 are in use
TW	BASE	Text window base slot is used
TW	S0-S9	Text window save 0-9 is in use
TW	ALLSLOTS	All text window save slots are in use
TW	ANYSLOTS	Any of text window slots are in use
TW	STACK	Text window stack is in use
PORT	*ALL	All Drawing ports 1-35 in use
STYLE	*ALL	All Graphics style entries 1-35 in use
BUT	*ALL	All Button style entries 1-35 in use
PAL	*ALL	All Color palettes entries 1-35 in use
ENV	*ALL	All Enviroments entries 1-35 in use
MOUSE	**ALL	All Mouse Field entries 1-35 in use
SCREEN	***BASE	All Graphics Screen entries 1-35 in use

* ANY, BASE, S0-S9 ALLSLOTS, ANYSLOTS and STACK are also allowed

** ANY, BASE, S0-S9 ALLSLOTS, ANYSLOTS and STACK are allowed

*** S0-S9 SLOTS, ALLSLOTS and STACK are also allowed.

This command determines if a specific data object is currently in use or not (i.e., defined).

Example: \$INUSE(TW, BASE)\$

Returns: 0

\$ISEXTWIN\$... Is text window an extended text window?

Text Variables Reference

Format: `$ISEXTWIN(windowno )$`

Syntax: `$ISEXTWIN(opt:WINDOWNO)$`

WINDOWNO

Returns '1' if true '0' if not

CUR	Current text window is an extended text window
0-35	Text window entry 0-35 is an extended text window
default = CUR	

If the specified text window is not defined, or deactivated, then this text variable returns a value of -1.

Example: `$ISEXTWIN(5)$`

Returns: 1

`$ISPALETTE$` ... Reports if a color palette exists or not

Format: `$ISPALETTE$`

Syntax: `$ISPALETTE$`

If the destination terminal is operating with a video device that has an actual color palette then this variable returns 1. If however, the terminal is running on a device that's in 24-bit color mode where there is no color palette then this variable returns 0.

Example: `$ISPALETTE$`

Returns: 1

\$ISPROT\$... Is a data object protected?Format: \$ISPROT(*data_object,element*)\$Syntax: \$ISPROT(*req:DATA OBJECT, req:ELEMENT*)\$

OBJECT	ELEMENT	Returns '1' if true or '0' if false
SCREEN	S0-S9	Is screen entry 0-9 protected
SCREEN	ALLSLOTS	All screen slots 0-9 are protected
SCREEN	ANYSLOTS	Any screen slots 0-9 is protected
MOUSE	S0-S9	Is mouse slot 0-9 protected
MOUSE	ALLSLOTS	Are all mouse slots 0-9 protected
MOUSE	ANYSLOTS	Any mouse slots 0-9 protected
TW	CUR	Is current text window protected
TW	1-35	Text window entry 1-35 is protected
TW	ALL	All text window entries 1-35 are protected
TW	ANY	Any text window entry 1-35 is protected
TW	S0-S9	Text window slot 0-9 is protected
TW	ALLSLOTS	All text window slots 0-9 are protected
TW	ANYSLOTS	Any text window slots 0-9 is protected
PORT	*CUR	Current graphic port is protected
BUT	*CUR	Current button entry is protected
STYLE	*CUR	Current graphic style is protected
PAL	*CUR	Current palette entry protected
ENV	*CUR	Current enviroment entry protected

* 1-35, ALL, ANY, S0-S9 ALLSLOTS\$ andANYSLOTS\$are also available

This command returns a value indicating if the specified element in the desired data object is protected or not. If it is protected, then it returns a 1. 0 is returned if it is not protected. If the specified data object element is not in use (i.e., not defined), then this command returns -1.

Example: \$UNPROT(TW, S7)\$

Returns: *nothing***\$M\$... Mouse Button Status: LMR**

Format: \$M\$

Syntax: \$M\$

This text variable returns a 3-character code representing the status of each mouse button. This variable works with two button and three button mice. The format of the code is *LMR* where *L*=Left, *M*=Middle (if any), and *R*=Right. If any button is clicked, the code for that button is 1. If the button is not depressed, it is 0. A return value of 100 would mean the left mouse button is depressed, but none of the others are.

Example: \$M\$

Returns: 010

\$MCURSOR\$... Set the mouse cursor style number

Format: \$MCURSOR(*cursor_no*)\$

Syntax: \$MCURSOR(*opt*:CURSOR_NO)\$

CURSOR_NO	0	Arrow cursor
	1	Wristwatch cursor
	2	Crosshair cursor
	3	I-bar cursor (Text edit)
	4	Pointing finger cursor
	5	Hand held up cursor
	6	Hourglass cursor
	default = 0	Arrow cursor

This command changes the current mouse cursor style to one of the pre-defined mouse cursor styles. The *CURSOR_NO* parameter is identical to the cursor style numbers permitted for the RIP *scrip* command to alter the cursor.

\$M HOUR\$... Hour (format HH) - military style

Format: \$M HOUR\$

Syntax: \$M HOUR\$

This Text Variable returns a two-digit number of the current hour in military format. This variable may range from 00 - 23.

Example: \$M HOUR\$

Returns: 17

\$MIN\$... Minutes

Format: \$MIN\$

Syntax: \$MIN\$

This Text Variable returns the two-digit number representing the current minutes in the hour. Possible values for this variable are 00-59.

Example: \$MIN\$

Returns: 45

\$MKILL\$... Kill Mouse Fields

Format: \$MKILL(*x0,y0,x1,y1,inout*)\$

Syntax: \$MKILL(*opt:X0, opt:Y0, opt:X1, opt:Y1,opt:INOUT*)\$

X0, Y0, X1, Y1 Coordinates area to be affected.
default = full screen

INOUT	IN	Delete mouse fields in area
	OUT	Delete mouse fields not in area
	default = IN	

If no parameters are specified then all mouse fields currently defined are deleted (just like RIP_KILL_MOUSE_FIELDS does).

If you specify the five parameters, you are defining a box on the screen that should have the mouse fields inside or outside of it deleted. Whether mouse fields inside the box or outside the box are destroyed depends on the INOUT parameter. If this parameter is specified as IN then all mouse fields inside the box are deleted. If the parameter is set to OUT then all mouse fields outside the box are deleted. Coordinates are specified in World coordinates.

This Active Text Variable deletes all defined Mouse Fields exactly like RIP_KILL_MOUSE_FIELDS does. The benefit is when the user clicks on a Mouse Fields or Button, the Mouse Fields are removed, but the graphics remain on the screen. The fields could be subsequently re-

defined quickly and easily without having to re-transmit an identical menu over again.

Example: \$MKILL\$ Kill all mouse fields defined
Returns: nothing

Example: \$MKILL(0,0,639,100,IN)\$ Kill all mouse fields inside
the box (0,0) to (639,100).
Returns: nothing

\$MONTH\$... Month Name

Format: \$MONTH\$
Syntax: \$MONTH\$

This Text Variable returns the full name of the current month. It is not abbreviated (e.g., November instead of Nov)

Example: \$MONTH\$
Returns: December

\$MONTHNUM\$... Month Number

Format: \$MONTHNUM\$
Syntax: \$MONTHNUM\$

This Text Variable returns the number of the current month. January= 01 and December=12.

Example: \$MONTHNUM\$
Returns: 12

\$MSTAT\$... Mouse Status

Format: \$MSTAT\$
Syntax: \$MSTAT\$

This text variable returns a YES if there is a mouse installed on the RIPTerm computer. If no mouse is installed, this variable returns NO.

Example: \$MSTAT\$

Returns: YES

\$MTW\$... Maximize text window to full size

Format: \$MTW(*window1*,*window2*,...)\$

Syntax: \$MTW(*opt*:WINDOW1, ...)\$

WINDOW1	ALL	Maximize all text windows
	CUR	Maximize current text window
	0-35	Maximize text window number 0-35
	default = CUR	

This command maximizes a particular text window to full screen using whatever text window font is associated with that text window. This does not affect any text that is already on the screen, only the internal definition of the text window (any scroll margins are reset). The cursor is moved to the home position, but is not re-enabled if it was previously disabled (use \$CON\$ for this). ANSI color attributes are not changed. You may specify one or more parameters to indicate that you wish to maximize more than one text window in the same \$MTW\$ statement. If any one of them fails to match the proper parameters, then the entire command is considered a syntax error and none of the parameters are processed.

It should be noted that if a text window being maximized is a standard text window created with the RIP_TEXT_WINDOW command, or if you just switched to a previously "unused" text window without explicitly defining it with a RIP_EXTENDED_TEXT_WINDOW command, then this command will treat the text window still as a "standard" text window. The window is moved to the very upper-left corner of the display. If it was an extended text window, then it is still moved to the upper-left corner of the screen, and the bounding box is made to exactly fit the actual text window display region. Remember, extended text windows don't know anything about text coordinate base locations of the upper-left corner, but standard text windows do. This command maintains this nature of the text window being maximized.

If a text window being maximized is currently protected, this command also does nothing to that text window.

Example: \$MTW(ALL)\$

Returns: *nothing*

\$MUSIC\$... Musical (cheerful) sound

Format: \$MUSIC(*count*)\$

Syntax: \$MUSIC(*opt:COUNT*)\$

COUNT	1-65535 default = 4	Number of times the sound repeats
-------	------------------------	-----------------------------------

This Active Text Variable produces a cheerful sound, indicating success of an action. This sound is used for successful downloads and dialed connections.

The count parameter determines how many times the musical sound is repeated.

\$MVP\$... Maximizes viewport to full screen

Format: \$MVP(*port1,port2,...*)\$

Syntax: \$MVP(*opt:PORT1, ...*)\$

PORT1	ALL CUR 0-35 default = CUR	Maximize all viewports Maximize current viewport Maximize viewport number 0-35
-------	-------------------------------------	--

This command will take the specified viewport and make it full-screen regardless of its current settings. This does not affect any current graphics on the screen, only the internal viewport definition.

To maximize more than one slot at the same time without maximizing all slots, simply specify more than one slot number parameter.

If any one of the parameters fails to match the proper parameters, then the entire command is considered a syntax error and none of the parameters are processed. If the underlying port of any viewport

specified is not in use (i.e., defined), deactivated or protected, then this command does nothing for that viewport.

Example: \$MVP(5)\$

Returns: *nothing*

\$NOREFRESH\$... Disables screen refresh expression

Format: \$NOREFRESH\$

Syntax: \$NOREFRESH\$

This command disables a host defined refresh expression. When you do this, the refresh option for the terminal is disabled and cannot be selected (or if it can be selected, does nothing). This is equivalent to issuing a RIP_SET_REFRESH with a \$OFF\$ parameter to disable refreshing.

Example: \$NOREFRESH\$

Returns: *nothing*

\$NULL\$... A null text variable (returns nothing)

Format: \$NULL\$

Syntax: \$NULL\$

This text variable is a special variable. It always returns nothing to the host system. It doesn't prompt the user or any information and it doesn't set anything. It is intended to be a place-holder for commands that require a text parameter (RIP_MOUSE, RIP_BUTTON and RIP_QUERY). When you have this text variable all by itself in a host command, it makes the host command do absolutely nothing, but has something defined for the host command to satisfy the RIP *scrip* interpreter (which expects something to be defined in host command text parameters).

Example: \$NULL\$

Returns: *nothing*

\$OFFSCREEN\$... Get offscreen bitmap port pixel data

Format: \$OFFSCREEN(*mode*)\$

Syntax: \$OFFSCREEN(*opt:MODE*)\$

MODE	FREE	Returns total free available pixels
	USED	Returns total used pixels
	TOTAL	Returns total of free and used pixels
	CLIP	Returns the port number that clipboard is currently using.
	default = FREE	

This command determines offscreen bitmap port pixel data. As described earlier on in this document, the offscreen bitmap ports that you may defined cannot exceed the total number of device pixels used on your screen. In other words, if you have a screen that is 1000 pixels wide, by 500 tall, then you would have a total number of pixels available for offscreen bitmap ports of 1000x500 (or 500,000).

When a new port is defined as an offscreen bitmap port, it calculates the width and height of that port in hardware pixels and reduces the remaining offscreen pixels by that amount.

If you specify the *MODE* parameter of this command as *FREE*, then you are requesting the total number of unused pixels available for use in offscreen bitmap ports. If *MODE* is set to *USED*, then you are requesting the total number of offscreen pixels current "in use" by offscreen bitmap ports. Finally, if *MODE* is set to *TOTAL*, then you are requesting the total number of offscreen pixels for all bitmap ports at once (*FREE* plus *USED* should equal *TOTAL*). If you omit the *MODE* parameter, then *FREE* is assumed.

If the mode parameter is set to the keyword *CLIP*, then you are asking what the current clipboard pointer is set to. This will return the clipboard "port number" associated with the clipboard pointer as a number for 1-35 for valid ports. If the clipboard pointer is not defined (e.g., no clipboard operations have transpired yet), then this will return a value of -1.

Example: \$OFFSCREEN(TOTAL)\$

Returns: 500000

Text Variable Reference

Example: \$OFFSCREEN(USED)\$
Returns: 200000

Example: \$OFFSCREEN(FREE)\$
Returns: 300000

\$OPTION\$... Enable/disable a software option

Format: \$OPTION(*option_name*,*mode*)\$

Syntax: \$OPTION(*req:OPTION_NAME*, *opt:MODE*)\$

OPTION_NAME	LIST	Returns a list of software options that can be enabled/disabled
	HOTKEY	Turn on/off mouse field Hot Keys
	TAB	Turn on/off TAB selection of mouse fields
	DOORWAY	Turn on/off Doorway mode
	STATBAR	Turn on/off Statusbar
	VT102	Turn on/off VT-102 support
*MODE	ON	Enables software option.
	OFF	Disables software option
	QUERY	Returns '1' if enabled '0' if not
	default = QUERY	

* Not used if OPTION = LIST

This command allows you to turn on or turn off a specific system option in the *RIPscrip* application.

The LIST keyword is designed to list out all options in the software that can be enabled or disabled. This allows for extendibility for custom *RIPscrip* packages by other vendors - some vendors may have certain features, and others may not. For example, if a specific terminal program doesn't support Doorway Mode, then the DOORWAY option wouldn't be available (and wouldn't show up if you did a \$OPTION(LIST)\$ directive). The list is not carriage return delimited. The text returned to the host is not terminated with any carriage returns

or anything like that. It's up to you to provide that kind of information in a button's host string or in a query string. The `LIST` directive though returns a list of all recognized keywords for that terminal. For example, RIPterm Pro returns the following for the `$OPTION(LIST)$` expression:

DOORWAY,HOTKEY,LIST,STATBAR,TAB,VT102

Note, the list is comma (,) delimited between keywords, but not after the last keyword. In addition, the keywords are returned in alphabetical order, converted to all capitals.

You may ask the terminal to report the status of a particular option with this command as well. To do so, specify a `MODE` of `QUERY` or omit it altogether. If the option is enabled, a `1` is returned to the host. If it is disabled, then `0` is returned. Again, no carriage returns or other delimiters are returned to the host.

Example: `$OPTION(DOORWAY, ON)$`
Returns: *nothing - enabled doorway mode*

Example: `$OPTION(LIST)$`
Returns: DOORWAY,HOTKEY,LIST,STATBAR,TAB,VT102

Example: `$OPTION(DOORWAY, QUERY)$`
Returns: 0

Example: `$OPTION(DOORWAY)$`
Returns: 0

Example: `$OPTION(LIST, QUERY)$`
Returns: `$OPTION(LIST, QUERY)$ ... Syntax error`

`$PAENTRY$` ... Return RGB values of palette

Format: `$PAENTRY(palno,start,stop )$`
Syntax: `$PAENTRY(req:PALNO, req:START, opt:STOP)$`

<i>PALNO</i>	<i>CUR</i>	Return values from current palette
	0-35	Return values from palette 0-35
	default = CUR	

Text Variable Reference

START	0-255	Start list from color 0-255
	ALL	Start list from beginning
END	0-255 default = START	End list at color 0-255

This variable returns one or more RGB values stored in one of the drawing color palettes. The *PALNO* parameter determines which palette in the palette data table is to be inquired about, and its value can be from 0-35 to explicitly reference a specific color palette number, or it can be CUR to indicate the current color palette.

The *START* parameter must be specified, and indicates which palette entry in the palette is to be inquired about. If the *STOP* parameter is omitted, then only one entry will be queried, the *START* entry. If *STOP* is supplied, then it must be equal to or greater than *START* (not to exceed 255). So if *START* is 5 and *STOP* is 7 then three palette entries will be inquired about, starting with entry 5.

If the palette isn't in use at all, then this variable returns the value -1 to the host. If it is in use, then the string will be a formatted block of text. The format of the return string to the host is:

<bits> ; <pal-entry> [, <pal-entry> ...]

The *BITS* field is the total number of bits of precision for the red, green or blue fields. If the *BITS* is 8, then red values can range from 0-255, etc. The *PAL-ENTRY* is a segmented response in the format:

red:green:blue

where each of red, green and blue are decimal numbers from 0 up to the total number of values allowed based on the *BITS* parameter.

If more than one palette entry is requested, then each *PAL-ENTRY* field will be separated by a comma delimiter (,) as in the following example:

Example: \$PAENTRY(CUR, 5, 7)\$
Returns: 6;42:0:42,42:21:0,42:42:42

Notice how the *BITS* is specified as " 6;". RIPterm maintains its own internal palette with 6 bits of precision - this isn't the only way to do it, its just the way RIPterm does it currently. With 6 bits of precision, each red, green or blue component will never exceed 2 bytes of data (8 bits could occupy 3 bytes).

Example: \$PAENTRY(CUR, 15)\$
Returns: 8;255:255:255

Lastly, the *START* parameter could be specified as ALL. If this parameter is present, then *STOP* isn't required. If ALL is specified, then all palette entries from 0-255 are returned as in the following:

Example: \$PAENTRY(CUR, ALL)\$
Returns: 6;0:0:0,0:0:42, ... 63:63:42,63:63:63

\$PCB\$... Paste Clipboard at last location

Format: \$PCB(*portno*)\$
Syntax: \$PCB(*opt:PORTNO*)\$

PORTNO	CUR	Use current port for source
	0-35	Use port 0-35 for the source
	default = CUR	

This text variable copies a portion of the specified port onto the current drawing port (these ports may be the same). The area on the specified port that is considered to be the source is "remembered" from the last rectangle of data that was put on this port. What this means is when you perform a RIP_PORT_COPY command, you copy a rectangle of data from one port to another. The two rectangles of

information are stored in the destination port of the copy operation. If the original port were port 0 and the destination port were port 5, then both rectangles of data are stored in port 5's definition. If you switch to port 0 and then perform a \$PCB(5)\$ operation, then the rectangle of data on port 5 is copied back to its original location in the current port.

This variable is extremely useful for restoring graphics that were previously saved to an offscreen port. It is typically used in dialog boxes

Text Variable Reference

when the user clicks on the "OK" button, where the dialog box should be erased and the original graphical screen would be restored.

If no parameters are specified, then the port that will be pasted from will be whatever port is associated with the clipboard pointer. See the `RIP_GET_IMAGE` command for more details on the clipboard pointer.

If you do specify a parameter, then it can be `CUR` to indicate the current port, or a value `0-35` to specify a specific port number to retrieve the data from. Note that if the specified port doesn't exist, or if either the current port or the specified port are "deactivated", then it is ignored.

It should be noted that the rectangles of information stored in the specified port are "viewport relative". What this means is that the coordinates in those rectangles are based on viewports. If you copy data from say port 0 to port 5, then change port 0's viewport before you perform a `$PCB(5)$` operation, then when you actually perform the `$PCB(5)$` command, the data will be pasted in a new location on the screen - not where it came from. If you're going to be using this command, be careful how you manipulate your viewports.

Example: `$PCB(5)$ Paste port 5's data to the current port`
Returns: *nothing*

`$PHASER$ ... Fire phasers!`

Format: `$PHASER(start,stop,inc,time )$`

Syntax: `$PHASER(opt:START, opt:STOP, opt:INC, opt:TIME)$`

<code>START</code>	<code>1-65535</code> default = 2500	Starting Frequency in Hertz
--------------------	--	-----------------------------

<code>STOP</code>	<code>1-65535</code> default = 50	Ending Frequency in Hertz
-------------------	--------------------------------------	---------------------------

<code>INC</code>	<code>1-65535</code> default = 20	Increment value
------------------	--------------------------------------	-----------------

<code>TIME</code>	<code>1-65535</code> default = 2	Increment time delay in milliseconds
-------------------	-------------------------------------	--------------------------------------

This Active Text Variable produces a sound like firing your energy weapons in a game. Now you too can blast away with the best of them. Trivia question: What does phaser stand for? See `$REVPHASERS$` for the answers.

This command doesn't require any parameters. If none are specified then the *START* is assumed to be 2500 Hertz. *STOP* is assumed to be 50 Hertz, *INC* is assumed to be 20 Hertz increments and *TIME* is assumed to be 2 milliseconds. *START* must be greater than *STOP* and *INC* must be greater than zero. If none of these conditions are met then the defaults are used for the sound effect.

This command allows you to specify no parameters (default settings), only one parameter (the starting frequency), two parameters (start and end frequency), three parameters (start and end frequency as well as the increment frequency), or finally all four parameters which correspond to the start and stop frequencies, the increment frequency and lastly the increment time delay (in milliseconds). Under no circumstances will values for any of the four parameters above 65535 be permitted. If values above these limits are encountered then the variable is not processed.

`$PORTH$` ... Height of port

Format: `$PORTH(portno,type,domain )$`

Syntax: `$PORTH(opt:PORTNO, opt:TYPE, opt:DOMAIN)$`

PORTNO	CUR	Return height of current port
	0-35 default = CUR	Return height of port number 0-35
TYPE	PORT	Return height of the port
	VIEW default = PORT	Return height of the port's viewport
DOMAIN	WORLD	Return in World Coordinates
	DEVICE default = WORLD	Return in Device Coordinates

Text Variable Reference

This text variable is identical in nature to the \$PORTW\$ text variable, except that it returns height information about the port instead of width.

\$PORTH\$... Width of port

Format: \$PORTH(*portno,type,domain*)\$

Syntax: \$PORTH(*opt:PORTNO, opt:TYPE, opt:DOMAIN*)\$

PORTNO	CUR	Return width of current port
	0-35	Return width of port number 0-35
	default = CUR	

TYPE	PORT	Return width of the port
	VIEW	Return width of the port's viewport
	default = PORT	

DOMAIN	WORLD	Return in World Coordinates
	DEVICE	Return in Device Coordinates
	default = WORLD	

This text variable returns the a piece of width information about the specified port.

If no parameters are specified, then you a request is being made for the width of the current drawing port itself.

If you specify the *PORTNO* parameter, then you are indicating which port you are requesting information on. You may indicate *CUR* for the current port, or a value from 0-35 to indicate a specific port number.

The *TYPE* parameter defines what type of port information you are requesting. If it is omitted then you can only specify the *PORTNO* parameter and this will return the width of the port itself. If you specify the *TYPE* parameter, then you may set this to *PORT* to explicitly indicate that you want information specifically on the port, or a value of *VIEW* to return information on the viewport.

If the *TYPE* parameter is *VIEW*, then you are specifically requesting the width of the viewport itself.

The third and final parameter is the *DOMAIN* parameter. This specifies what type of coordinate information you are requesting. By default, if this parameter is omitted then you are requesting your coordinate information in world coordinate values. Possible domain

values are *WORLD* to explicitly request world coordinate values, or *DEVICE* for physical hardware device pixel coordinates.

If you request information on a port (or the viewport belonging to a port) where the port isn't defined (i.e., not in use), or it is deactivated, then this variable returns a value of 0 to indicate this error condition.

Example: \$PORTW\$
Returns: 100

Example: \$PORTW(CUR)\$
Returns: 100

Example: \$PORTW(CUR,PORT)\$
Returns: 100

Example: \$PORTW(CUR,PORT,WORLD)\$
Returns: 100

Example: \$PORTW(CUR,PORT,DEVICE)\$
Returns: 50

\$PORTX0\$... Port's upper left X coordinate

Format: \$PORTX0(*portno,type, domain*)\$
Syntax: \$PORTX0(*opt:PORTNO, opt:TYPE, opt:DOMAIN*)\$

<i>PORTNO</i>	<i>CUR</i> 0-35 default = CUR	Return value of current port Return value of port number 0-35
<i>TYPE</i>	<i>PORT</i> <i>VIEW</i> default = PORT	Return value of the port Return value of the port's viewport

Text Variable Reference

DOMAIN	WORLD	Return in World Coordinates
	DEVICE	Return in Device Coordinates
	default = WORLD	

This text variable returns the a piece of upper-left X coordinate information about the specified port.

If no parameters are specified, then you a request is being made for the upper-left X coordinate of the current drawing port itself related to the current screen. This is only meaningful for "screen ports" where this X coordinate specifies an offset from the left border of the screen in pixels. For offscreen bitmapped ports, there is no upper-left X coordinate (or more precisely, an offset from some screen), so for offscreen ports this variation returns a value of 0. Port number 0 would always yield a value of 0 because its port is set to the full dimensions of the screen so there's no offset information for the upper-left X coordinate.

If you specify the *PORTNO* parameter, then you are indicating which port you are requesting information on. You may indicate *CUR* for the current port, or a value from 0-35 to indicate a specific port number.

The *TYPE* parameter defines what type of port information you are requesting. If it is omitted then you can only specify the *PORTNO* parameter and this will return the upper-left X coordinate of the port itself. If you specify the *TYPE* parameter, then you may set this to *PORT* to explicitly indicate that you want information specifically on the port, or a value of *VIEW* to return information on the viewport.

If the *TYPE* parameter is *VIEW*, then you are specifically requesting the upper-left X coordinate of the viewport in relation to the port's actual origin (the upper-left corner of the port).

The third and final parameter is the *DOMAIN* parameter. This specifies what type of coordinate information you are requesting. By default, if this parameter is omitted then you are requesting your coordinate information in world coordinate values. Possible domain values are *WORLD* to explicitly request world coordinate values, or *DEVICE* for physical hardware device pixel coordinates.

If you request information on a port (or the viewport belonging to a port) where the port isn't defined (i.e., not in use), or it is deactivated, then this variable returns a value of -1 to indicate this error condition.

Example: \$PORTX0\$

Returns: 100

Example: \$PORTX0(CUR)\$

Returns: 100

Example: \$PORTX0(CUR,PORT)\$

Returns: 100

Example: \$PORTX0(CUR,PORT,WORLD)\$

Returns: 100

Example: \$PORTX0(CUR,PORT,DEVICE)\$

Returns: 50

\$PORTX1\$... Port's lower right X coordinate

Format: \$PORTX1(*portno,type,domain*)\$

Syntax: \$PORTX1(*opt:PORTNO, opt:TYPE, opt:DOMAIN*)\$

<i>PORTNO</i>	<i>CUR</i>	Return value of current port
	0-35 default = CUR	Return value of port number 0-35
<i>TYPE</i>	<i>PORT</i>	Return value of the port
	<i>VIEW</i> default = PORT	Return value of the port's viewport
<i>DOMAIN</i>	<i>WORLD</i>	Return in World Coordinates
	<i>DEVICE</i> default = WORLD	Return in Device Coordinates

This text variable returns the a piece of lower-right X coordinate information about the specified port.

Text Variable Reference

If no parameters are specified, then you a request is being made for the lower-right X coordinate of the current drawing port itself related to the current screen. This is only meaningful for "screen ports" where this X coordinate specifies an offset from the left border of the screen in pixels. For offscreen bitmapped ports, there is no upper-left X coordinate (or more precisely, an offset from some screen), so for offscreen ports this variation returns a value that is equivalent to the width of the port. Port number 0 would always yield a value that is the width of the actual screen because its port is set to the full dimensions of the screen so there's no offset information for the upper-left X coordinate.

If you specify the *PORTNO* parameter, then you are indicating which port you are requesting information on. You may indicate *CUR* for the current port, or a value from 0-35 to indicate a specific port number.

The *TYPE* parameter defines what type of port information you are requesting. If it is omitted then you can only specify the *PORTNO* parameter and this will return the lower-right X coordinate of the port itself. If you specify the *TYPE* parameter, then you may set this to *PORT* to explicitly indicate that you want information specifically on the port, or a value of *VIEW* to return information on the viewport.

If the *TYPE* parameter is *VIEW*, then you are specifically requesting the lower-right X coordinate of the viewport in relation to the port's actual origin (the lower-right corner of the port).

The third and final parameter is the *DOMAIN* parameter. This specifies what type of coordinate information you are requesting. By default, if this parameter is omitted then you are requesting your coordinate information in world coordinate values. Possible domain values are *WORLD* to explicitly request world coordinate values, or *DEVICE* for physical hardware device pixel coordinates.

If you request information on a port (or the viewport belonging to a port) where the port isn't defined (i.e., not in use), or it is deactivated, then this variable returns a value of -1 to indicate this error condition.

Note that the lower-right X coordinate is non-inclusive. This means that the lower-right X coordinate is not actually part of the port's drawing area. It works exactly like the rectangle defining a filled rectangle.

Example: \$PORTX1\$

Returns: 100

Example: \$PORTX1(CUR)\$

Returns: 100

Example: \$PORTX1(CUR,PORT)\$

Returns: 100

Example: \$PORTX1(CUR,PORT,WORLD)\$

Returns: 100

Example: \$PORTX1(CUR,PORT,DEVICE)\$

Returns: 50

\$PORTY0\$... Port's upper left Y coordinate

Format: \$PORTY0(*portno,type,domain*)\$

Syntax: \$PORTY0(*opt:PORTNO, opt:TYPE, opt:DOMAIN*)\$

<i>PORTNO</i>	<i>CUR</i>	Return value of current port
	0-35 default = CUR	Return value of port number 0-35
<i>TYPE</i>	<i>PORT</i>	Return value of the port
	<i>VIEW</i> default = PORT	Return value of the port's viewport
<i>DOMAIN</i>	<i>WORLD</i>	Return in World Coordinates
	<i>DEVICE</i> default = WORLD	Return in Device Coordinates

This command is identical in nature to the \$PORTX0\$ text variable, except that it returns upper-left Y coordinate information on the port.

\$PORTY1\$... Port's lower right Y coordinate

Format: \$PORTY1(*portno,type,domain*)\$

Text Variable Reference

Syntax: \$PORTY1(*opt:PORTNO*, *opt:TYPE*, *opt:DOMAIN*)\$

PORTNO	CUR	Return value of current port
	0-35	Return value of port number 0-35
	default = CUR	
TYPE	PORT	Return value of the port
	VIEW	Return value of the port's viewport
	default = PORT	
DOMAIN	WORLD	Return in World Coordinates
	DEVICE	Return in Device Coordinates
	default = WORLD	

This command is identical in nature to the \$PORTX1\$ text variable, except that it returns lower-right Y coordinate information on the port.

\$PROT\$... Protect data from deletion

Format: \$PROT(*data_object*, *element1*, ...)\$

Syntax: \$PROT(*req:OBJECT*, *req:ELEMENT1*, ...)\$

This command protects a specific element of a given data object. What is element of the data object that is to be protected is defined by the *ELEMENT* parameter. This parameter must be specified (you may specify more than one to protect multiple elements in one command). *ELEMENT* may be set to the following values:

If a data object element is attempted to be protected, but it is not in use then this command does nothing for that parameter.

Example: \$PROT(TW, S7)\$

Returns: *nothing*

\$RBS\$... Restore a button style from a backup area

Format: \$RBS(*source*)\$

Syntax: \$RBS(*opt:SOURCE*)\$

SOURCE	BASE	Restore from BASE save area
	0-9	Restore from save slot 0-9
	POP	Restore from stack
	default = BASE	

This text variable restores the button style data table from a data backup area. If the *SOURCE* parameter is omitted, then it is restored from the base save area of the backup data area. Possible sources are BASE to restore from the base save area, the value 0-9 to read from a specific data save slot, or the value POP to indicate that you wish to pop the button style data table from the button style data backup area's save stack.

Example: \$RBS(POP)\$

Returns: *nothing*

\$RCB\$... Restore Clipboard

Format: \$RCB\$(*source*)

Syntax: \$RCB(*opt*:SOURCE)\$

SOURCE	BASE	Restore from BASE save area
	0-9	Restore from save slot 0-9
	POP	Restore from stack
	default = BASE	

This Active Text Variable restores the Clipboard from a previously executed \$SCB\$ command. Not only are the clipboard contents saved, but so is the last clipboard location, so Paste Clipboard (\$PCB\$) restores the clipboard's contents AND location.

If you do not specify a *SOURCE* parameter then the clipboard is restored from the base save area of the port backup area. It is not deleted so you can do multiple identical clipboard restorations.

If you do specify a *SOURCE* parameter then you may restore the clipboard from any of ten different clipboard slots (0-9). Once restored, the clipboard slot file is deleted.

Text Variable Reference

You may specify a slot number of POP to perform a stack-based pop operation (e.g., \$RCB(POP)\$).

Example: \$RCB(4)\$

Returns: *nothing*

\$RCP\$... Restore a color palette from a backup area

Format: \$RCP(*source*)\$

Syntax: \$RCP(*opt:SOURCE*)\$

SOURCE	BASE	Restore from BASE save area
	0-9	Restore from save slot 0-9
	POP	Restore from stack
	default = BASE	

This text variable restores the color palette data table from a data backup area. If the *SOURCE* parameter is omitted, then it is restored from the base save area of the backup data area. Possible sources are BASE to restore from the base save area, the value 0-9 to read from a specific data save slot, or the value POP to indicate that you wish to pop the color palette data table from the color palette data backup area's save stack.

Example: \$RCP(POP)\$

Returns: *nothing*

\$RENV\$... Activates a previously snapshotted environment

Format: \$RENV(*source*)\$

Syntax: \$RENV(*opt:SOURCE*)\$

SOURCE	BASE	Restore from BASE save area
	0-9	Restore from save slot 0-9
	POP	Restore from stack
	default = BASE	

This command restores a previously saved environment data table back into the actual data table and puts it to use. If you omit the *SOURCE* parameter, then the environment is restored from the base save area of the environment data backup area.

If the *SOURCE* value is set to POP, then the data table is popped off of the environment data backup area's stack.

If the *SOURCE* value is set to BASE, then the data table is restored from the base save area of the environment data backup area.

If the *SOURCE* value is set to a number from 0-9, then the environment data table is restored from a data save slot number in the environment data backup area.

Example: \$RENV(POP)\$
Returns: *nothing*

\$REFRESH\$... Forces terminal to Transmit Refresh

Format: \$REFRESH\$
Syntax: \$REFRESH\$

This text variable instructs the terminal to transmit the refresh host command to the remote host system if the refresh expression is non-NULL. The host has the ability to set a refresh expression that will, when sent, redisplay the current screen.

Example: \$REFRESH\$
Returns: ^m

\$RESET\$... Perform a reset operation

Format: \$RESET(*option,element,sub_element*)\$
Syntax:\$RESET(*opt:OPTION, opt:ELEMENT, t:SUB_ELEMENT*)\$

<i>OPTION</i>	<i>ELEMENT</i>
<i>SOFT</i>	Performs a soft reset
<i>HARD</i>	Does a hard reset

Text Variable Reference

MCURSOR	Cursor becomes Arrow. Mouse input is re-enabled.
KEYBOARD	Keyboard input is re-enabled
SOUND	Stops playing current sound
TV	See following section
TW	See section below
STYLE	See section below
BUT	See section below
PAL	See section below
PORT	See section below
ENV	See section below
MOUSE	See section below
SCREEN	See following section
VIEW	See following section
QUERY	See following section

The reset text variable is a general purpose reset command. Without any parameters, the text variable adheres to the original 1.54 RIP *scrip* command which performs the exact same operations as a RIP_RESET_WINDOWS command. Some parameters used with the reset command require additional parameters to further specify what section is to be reset (e.g., if you specify TW to reset a text window, you need to specify what text window data table entry is to be reset). The possible reset "types" are as follows:

The following paragraphs describe the purpose of each parameter and how each parameter operations:

SOFT ... This makes the reset command perform a soft reset. This is identical in nature to the RIP_RESET_WINDOWS command. See that RIP*scrip* command for a complete, detailed description of what a soft reset does.

HARD ... This performs a hard reset command. This is as if you issued the RIP_HEADER command with the "hard reset" flag enabled. See the RIP_HEADER command for more details about a hard reset.

MCURSOR ... When this parameter is specified then the mouse cursor on the terminal is switched to its default, arrow shape. If the mouse input is currently disabled (via a RIP_HEADER command), then it is once again re-enabled as if a RIP_NO_MORE command were received.

KEYBOARD ... When this parameter is specified, then keyboard input is once again enabled. Keyboard input can only be disabled via the **RIP_HEADER** command, and typically is re-enabled by a **RIP_NO_MORE** sequence. This allows you to explicitly re-enable the keyboard without a **RIP_NO_MORE** command being specified.

SOUND ... This reset command stops any playing digitized sound (if any). If the sound is currently in the process of playing, it is aborted immediately to where there's no sound playing anymore.

TW ... This parameter allows you to reset certain aspects of the text the text window system. The exact operation depends on what text window is being acted upon. If text window #0 is being reset, then it is set to full screen (no window clearing is performed). If it refers to a text window other than text window #0, then that window is formally deleted. If the current text window is a window other than window #0 and it is reset, then the text window is automatically switched to text window #0. If the reset affects a data backup area, then the data backup area in question is deleted entirely. If you specify no parameters, then the current text window is reset. You may specify one or more parameters with this command to specifically alter particular aspects of the text window system.

Text Variable Reference

You may specify the following parameters:

Element	
CUR	Reset the current text window.
0-35	Reset a specific text window data table entry.
TBL	Reset all text window data table entries.
BASE	Reset (clear) the text window base save area
S0-S9	Reset (clear) a specific text window data save slot
SLOTS	Reset (clears) all text window data save slots
STACK	Reset (clears) the text window data save stack.
BACKUP	Reset all text window backup areas (base area, slots and stack.
ALL	Reset all text window data tables and backup areas

Example: `$RESET(TW, CUR, 30, BASE, S5)$`

Element	
STYLE	This parameter resets a graphical style back to its default settings. If you specify no parameters, then the current graphical style is reset to bootup defaults. You may specify one or more parameters with this command to specifically alter particular aspects of the graphical style
CUR	Reset the current graphical style to default values.
0-35	Reset a specific graphical style data table to defaults.
TBL	Reset all graphical style data table entries to defaults.
BASE	Reset (clear) the graphical style base save area
S0-S9	Reset (clear) a specific graphical style data save slot
SLOTS	Reset (clears) all graphical style data save slots.
STACK	Clears the graphical style data save stack.
BACKUP	Reset (clears) all graphical style backup areas (base area, slots and stack.
ALL	Reset all graphical style data tables and backup areas

Example: `$RESET(STYLE, CUR, 30, BASE, S5)$`

BUT ... This parameter resets a button style to basic button default values (see the `RIP_BUTTON_STYLE` command for more details). If

you specify no parameters, then the current button style is reset to bootup defaults. You may specify one or more parameters with this command to specifically alter particular aspects of the button style system. You may specify the following parameters:

Element	
CUR	Reset the current button style to default values.
0-35	Reset a specific button style data table to defaults.
TBL	Reset all button style data table entries to defaults.
BASE	Reset (clear) the button style base save area
S0-S9	Reset (clear) a specific button style data save slot
SLOTS	Reset (clears) all button style data save slots.
STACK	Clears the button style data save stack.
BACKUP	Reset (clears) all button style backup areas (base area, slots and stack.
ALL	Reset all button style data tables and backup areas

Example: \$RESET(BUT, CUR, 30, BASE, S5)\$

PAL ... Resets a color palette back to the default color palette. If you specify no parameters, then the current color palette is reset to bootup defaults. You may specify one or more parameters with this command to specifically alter particular aspects of the color palette system. You may specify the following parameters:

Element	
CUR	Reset the current color palette to default values.
0-35	Reset a specific color palette data table to defaults.
TBL	Reset all color palette data table entries to defaults.
BASE	Reset (clear) the color palette base save area.
S0-S9	Reset (clear) a specific color palette data save slot.
SLOTS	Reset (clears) all color palette data save slots.
STACK	Clears the color palette data save stack.
BACKUP	Reset (clears) all color palette backup areas (base area, slots and stack.
ALL	reset all color palette data tables and backup areas

Example: \$RESET(PAL, CUR, 30, BASE, S5)\$

Text Variable Reference

PORT ... This variation on the reset command resets one or more ports in some way. The exact way that one is reset varies depending on the type of port specified. If no additional parameters are specified, then the current port is reset. Otherwise, you may reset a specific port number in the port data table, or one (or all) of the port data backup areas. If you reset a specific port in the port data table, what happens varies depending on what kind of port it is. If it is port #0 (which cannot be deleted), then the only thing that happens is that any resident queries for that port/viewport are deleted, the viewport is made full screen and the viewport is erased to the background color (in that order). If the port number (from 1-35) represents a screen port, then any resident query attached to that port/viewport is deleted and the port itself is deleted. If it is an offscreen port, then that port is also deleted (offscreen ports cannot have resident queries). In any event, if the port being deleted happens to be the current port, then the port is automatically switched to port #0 (the screen's port). The following parameters are permitted with this reset port). Whenever the current port is reset, the port is automatically switched to port #0. Beware of this when chaining multiple port resets together at the same time. For example, if port #5 is current and you issue the following command `$RESET(PORT, 5, CUR)$`, then port number 5 will be deleted, then it would switch back to port #0 which is the screen port and cannot be deleted, so only its viewport will be reset. The following parameters are permitted with this reset command to control what type of reset operation is to be performed:

Element	
CUR	Reset the current environment to default values.
0-35	Reset a specific environment data table entry to defaults.
TBL	Reset all environment data table entries to defaults.
BASE	Reset (clear) the environment base save area
S0-S9	Reset (clear) a specific environment data save slot.
SLOTS	Reset (clears) all environment data save slots.
STACK	Clears the environment data save stack.
BACKUP	Reset (clears) all environment backup areas (base area, slots and stack).
ALL	Reset all environment data tables and backup areas

Example: `$RESET(PORT, CUR, 30, BASE, S5)$`

Text Variables Reference

ENV ... This command resets an environment to either default values or to a status of "erased". If you specify no parameters, then the current environment data table entry is reset to default bootup values. If you do specify any parameters, then the following ones may be used:

Element	
CUR	Reset (delete) the current port.
0-35	Reset (delete) a specifc Port data table entry.
TBL	Reset (delete) all port data table entries.
BASE	Reset (clear) the port base save area.
S0-S9	Reset (clears) a specific port data save slot.
SLOTS	Reset (clears) all port data save slots.
STACK	Reset (clears) all port data save stack.
BACKUP	Reset (clears) all port backup areas (base area, slots and stacks.)
ALL	Reset all port data tables and backup areas.

Example: \$RESET(ENV, CUR, 30, BASE, S5)\$

MOUSE ... Resets (deletes) all mouse fields defined in the specified destination. If no parameter is defined, then all current mouse definitions are reset. If you specify one or more parameters, then they may be any of the following:

Element	
TBL	Reset (clear) all existing mouse definitions in use
BASE	Reset (clear) the mouse field base save area
S0-S9	Reset a specific mouse field data save slot
0-32767	Reset mouse field with a specific ID value.
SLOTS	Reset all mouse field data save slots
STACK	Clears the data save stack pointer, but don't reset slots
BACKUP	Reset all mouse field backup areas
ALL	Reset all mouse field data tables and backup areas

Example: \$RESET(MOUSE, BASE, S5)\$

Element	
BASE	Reset (clear) the screen base save area

Text Variable Reference

S0-S9	Reset a specific screen data save slot area
SLOTS	Reset all screen data save slots
STACK	Clears the data save stack pointer, but don't reset slots
BACKUP	Reset all screen backup areas
ALL	Reset the screen (erase) and all backup areas

SCREEN ... When no parameters are specified, the entire video screen is cleared to color #0 (usually black). No viewports are modified, nor are any drawing ports, text windows, mouse fields or anything else. The following parameters are allowed, providing you with the ability to reset particular screen aspects:

VIEW ... When no parameters are specified, then the viewport of the current drawing port is reset to the full dimensions of the port itself, and activated. Other possible parameters allow you to modify other viewports of other ports. The possible parameters are:

Element	
CUR	Reset current viewport to full port size
0-35	Reset the viewport of specific port to full port size
TBL	Resets all viewports of all data table ports to full port size

QUERY ... If no parameters are specified or a single parameter of ALL is specified then all resident queries are reset (deleted). If you specify any parameters, you may specify more than one.

OPTION	ELEMENT	SUB ELEMENT	
QUERY	TW	ALL	Resets all text window based queries
QUERY	TW	CUR	Resets current text window queries
QUERY	TW	TBL	Reset all queries for entries 0-35
QUERY	TW	0-35	Reset queries for text window 0-35
QUERY	VIEW	ALL	Resets all graphical viewport queries
QUERY	VIEW	CUR	Resets current viewports queries
QUERY	VIEW	TBL	Resets all queries for entries 0-35
QUERY	VIEW	0-35	Resets viewport 0-35's queries
QUERY	ENTRY		Resets mouse field entry query
QUERY	EXIT		Resets mouse field exit query..

See the RIP_QUERY command for more detailed information about the various possible resident query types.

Example: \$RESET(QUERY,TW,CUR)\$... Reset current text window's query

Example: \$RESET(QUERY,TW,TBL)\$... Reset all queries for entries 0-35

Example: RESET(QUERY,TW,5)\$ Reset resident query for Text window #5

It should be noted that if you attempt to reset something that cannot be reset (e.g., drawing port #0, a protected data table entry, a stack that's empty, etc.), then this command does nothing and does not generate a syntax error. A syntax error can only be generated if an invalid parameter is encountered. If even so much as a single parameter is invalid then the entire command is discarded as a syntax error without any of the parameters being processed.

TV ... This option requires two full parameters - the *OPTION* parameter set to TV and the *ELEMENT* parameter set to a text variable name. No other parameters are allowed. When this form of a reset command is issued, the specified text variable is deleted. An example of this command might be:

Example: \$RESET(TV,VARIABLE_NAME)\$

\$RESTORE\$... Restore graphics screen

Format: \$RESTORE(*source*)

Syntax: \$RESTORE(*opt:SOURCE*)\$

SOURCE	BASE	Restore from BASE save area
	0-9	Restore from save slot 0-9
	POP	Restore from stack
	default = BASE	

If no *SOURCE* parameter is specified, then the graphical screen is restored from the screen backup area's base save area. The base save area is not erased of its contents after this restoration operation is complete.

Text Variable Reference

If you specify a *SOURCE* as a data backup slot number (0-9), then the screen restored will be restored from the specified graphics screen data save slot. After a data save slot's contents are restored to the screen, the contents of that backup slot are deleted (unless that slot is protected).

If you specify *POP* instead of a slot number then a stack-based pop operation will be performed (e.g., \$RESTORE(POP)\$).

If you specify *BASE* instead of a slot number than the screen is restored from the screen data backup area's base save area just as if no parameter was specified.

Only the graphics screen is restored - not the Drawing Ports, Mouse Fields, Button Styles, Color Palettes, RIP *scrip* Environments, Graphics Window, or Text Window settings.

When the graphics screen is restored, the Graphics Viewport settings that were in effect when the screen was saved will be restored as well. Also, the color palette that was active the moment the screen was saved will be restored.

To restore the entire context of the graphics environment
\$RESTOREALL\$.

Example: \$RESTORE(3)\$
Returns: *nothing*

\$RESTOREx\$... Restore graphics screen (x=0-9)

Format: \$RESTORE0\$... \$RESTORE9\$
Syntax: \$RESTORE0\$... \$RESTORE9\$

The \$RESTORE0\$ through \$RESTORE9\$ restore the graphics screen from one of the graphics screen data backup slots. The slot referenced is specified by the number 0-9. When the screen is restored, the contents of that data backup slot are deleted, providing that the slot isn't protected.

When the graphics screen is restored, the Graphics Viewport settings that were in effect when the screen was saved will be restored as well. Also, the color palette in use when the screen was saved is restored.

To restore the entire context of the graphics environment
\$RESTOREALL\$.

NOTE: *This method of restoring graphical screens is obsolete. Use the \$RESTORE\$ function (see above) with a parameter.*

Example: \$RESTORE3\$
Returns: *nothing*

\$RESTOREALL\$... Restore all screen attributes

Format: \$RESTOREALL(*source*)\$
Syntax: \$RESTOREALL(*opt*:SOURCE)\$

SOURCE	BASE	Restore from BASE save area
	0-9	Restore from save slot 0-9
	POP	Restore from stack
	default = BASE	

This Active Text Variable restores the Text Windows coordinates, restores the contents of the clipboard, restores all mouse fields, and restores the contents of the screen. It is equal to the following operations (in this order):

\$RTW(*source*)\$
\$RCB(*source*)\$
\$RMF(*source*)\$
\$RGS(*source*)\$
\$RBS(*source*)\$
\$RESTORE(*source*)\$
\$RCP(*source*)\$
\$RENV(*source*)\$

This command does not require any parameters. If none, or the *SOURCE* parameter is specified as the value BASE are specified then

Text Variable Reference

the restore is performed from the base save area and can be restored many times. The ability to specify a *SOURCE* gives you the ability to have restore one of many screen configurations. You are allowed up to ten separate slots (0-9).

In place of a slot number, you can provide a parameter of *POP* to perform a stack-based pop operation to restore the last pushed \$SAVEALL\$ (e.g., \$RESTOREALL(POP)\$).

Example: \$RESTOREALL(0)\$

Returns: *nothing*

\$RESX\$... Horizontal resolution of current video device

Format: \$RESX\$

Syntax: \$RESX\$

This variable returns the horizontal resolution of the video device in pixels. Typical results for this might be 640, 800, 1024 or 1280. But this might be different depending on the various possible video devices in existence.

Example: \$RESX\$

Returns: 640

\$RESY\$... Vertical resolution of current video device

Format: \$RESY\$

Syntax: \$RESY\$

This variable returns the vertical resolution of the video device in scan lines. Typical results for this might be 350, 480, 600, 768 or 1024. But this might be different depending on the various possible video devices in existence.

Example: \$RESY\$

Returns: 768

\$REVPHASER\$... Fire phasers!

Format: \$REVPHASER(*start,stop,inc,time*)\$

Syntax: \$REVPHASER(*opt:START, opt:STOP, opt:INC, opt:TIME*)\$

<i>START</i>	1-65535 default = 50	Starting Frequency in Hertz
<i>STOP</i>	1-65535 default = 2500	Ending Frequency in Hertz
<i>INC</i>	1-65535 default = 20	Increment value
<i>TIME</i>	1-65535 default = 2	Increment time delay in milliseconds

This Active Text Variable produces a sound like firing your energy weapons in a game. Like \$PHASER\$ makes an ascending tone, \$REVPHASER\$ makes a descending tone. Answer to trivia question in \$PHASER\$: Phaser stands for PHoton Amplification by Stimulated Emission of Radiation. Sound familiar? Laser is Light Amplification by Stimulated Emission of Radiation, and Maser is Microwave Amplification by Stimulated Emission of Radiation.

This command doesn't require any parameters. If none are specified then the *START* is assumed to be 50 Hertz. *STOP* is assumed to be 2500 Hertz, *INC* is assumed to be 20 Hertz increments and *TIME* is assumed to be 2 milliseconds. *START* must be greater than *STOP* and *INC* must be greater than zero. If none of these conditions are met then the defaults are used for the sound effect.

This command allows you to specify no parameters (default settings), only one parameter (the starting frequency), two parameters (start and end frequency), three parameters (start and end frequency as well as the increment frequency), or finally all four parameters which correspond to the start and stop frequencies, the increment frequency and lastly the increment time delay (in milliseconds). Under no circumstances will values for any of the four parameters above 65535 be permitted. If values above these limits are encountered then the variable is not processed.

\$RGS\$... Restore a graphics style from a backup areaFormat: \$RGS(*source*)\$Syntax: \$RGS(*opt*:SOURCE)\$

SOURCE	BASE	Restore from BASE save area
	0-9	Restore from save slot 0-9
	POP	Restore from stack
	default = BASE	

This text variable restores the graphics style data table from a data backup area. If the *SOURCE* parameter is omitted, then it is restored from the base save area of the backup data area. Possible sources are BASE to restore from the base save area, the value 0-9 to read from a specific data save slot, or the value POP to indicate that you wish to pop the color palette data table from the graphics style data backup area's save stack.

Example: \$RGS(POP)\$

Returns: *nothing***\$RIPVER\$... RIPscrip version (e.g., RIPSCRIP015300)**

Format: \$RIPVER\$

Syntax: \$RIPVER\$

This Text Variable returns a phrase which will identify a RIP *scrip*-compatible software package. It is designed to be used by a host to detect what version of RIP *scrip* graphics your terminal can support as well as the type (brand) of RIP *scrip* terminal that is in use. When this Text Variable is used, it will respond back with RIPSCRIP followed by the Version Number (e.g., 01.54), followed by two digits identifying the Vendor of the terminal. The first digit of the Vendor ID field is the Vendor Code (1=RIPterm). The second digit is the Vendor's sub-version code identifying sub-versions of the software that still support the same RIP*scrip* software version. Valid Vendor Codes are:

Code	
0	Generic RIPscrip terminal (vendor unknown)
1	RIPterm (from TeleGrafix Communications)

See the section earlier in this document on ANSI sequences for a more robust description of the Vendor Codes and Auto-Sensing.

Example: \$RIPVER\$

Returns: RIPSCRIP015400

\$RMF\$... Restore Mouse Fields

Format: \$RMF\$(*source*)

Syntax: \$RMF(*opt*:SOURCE)\$

SOURCE	BASE	Restore from BASE save area
	0-9	Restore from save slot 0-9
	POP	Restore from stack
	default = BASE	

This Active Text Variable restores any Mouse Fields saved with \$SMF\$. You may have only one set of mouse fields saved at once. If no mouse fields were saved, or if the number of fields saved is 0, then no mouse fields are active.

If no *SOURCE* parameter is specified, then the mouse fields are restored from the mouse backup area's base save area. If a *SOURCE* parameter is specified then the mouse fields saved with \$SMF(*source*)\$ are restored then the data backup slot is deleted.

If you specify a *SOURCE* number of POP then you will be performing a stack-based pop operation (e.g., \$RMF(POP)\$).

If you specify BASE instead of a slot number then you will be restoring the mouse fields from the mouse field data backup area's base save area.

NOTE: *You may restore Mouse Fields from the base save area more than once is you wish. In other words, if you do a \$SMF\$ or an \$SMF(BASE)\$ command, you may execute \$RMF\$ or \$RMF(BASE)\$ one or more times. But if you do a \$SMF(1)\$ you may only do a \$RMF(1)\$ once. Restoring from a slot based save*

area deletes that slot immediately after the restoration (unless the slot is protected).

Example: \$RMF(4)\$

Returns: *nothing*

\$RTW\$... Restore Text Window information

Format: \$RTW(*source*)\$

Syntax: \$RTW(*opt*:SOURCE)\$

SOURCE	BASE	Restore from BASE save area
	0-9	Restore from save slot 0-9
	POP	Restore from stack
	default = BASE	

If no slot parameter is specified then this command restores all text window definitions from the slot-less definition file saved with a \$STW\$ command (with no parameters). The file is not deleted and you can restore many times.

If you specify a slot parameter then that identifies which of the ten different slots you wish to restore from (0-9). Once the slot file is read it is deleted.

If you specify a slot number of "POP" then you are performing a stack-based pop operation (e.g., \$RTW(POP)\$).

If you specify BASE instead of a slot number then the text window table will be restored from the text window backup area's base save area.

In any case, the text window settings active when a \$STW\$ (Save Text Window) was executed are saved. The current cursor location, window location, ANSI attributes, cursor ON/OFF status, vertical scrolling margins, and the System Font are restored.

NOTE: *The text contents of the window are not restored.*

Example: \$RTW(5)\$

Returns: *nothing*

\$SAVE\$... Save graphics screen

Format: \$SAVE (*destination*)

Syntax: \$SAVE (*opt:DESTINATION*)\$

DESTINATION	BASE	Save to base save area
	0-9	Save to save slot 0-9
	PUSH	Save on stack
	default = BASE	

If no *DESTINATION* parameter is specified, then this command will save the contents of the entire graphics screen to the graphics screen backup area's base save area. No mouse fields, text windows, drawing ports, graphics styles, color palettes, or button styles are saved - just

the graphics screen. A graphics screen saved to the base save area can be restored multiple times without the screen being deleted from the backup area.

If you specify the *DESTINATION* parameter as a number from 0-9, then that identifies a specific graphics screen data backup slot. Once a graphics screen is stored in a data backup slot, it may be restored again with the \$RESTORE\$ command, but when a data backup slot is restored, the contents of the slot are immediately deleted (unless it is protected).

If you specify PUSH instead of a slot number then a stack-based save operation will be performed (e.g., \$SAVE(PUSH)\$).

If you specify BASE instead of a slot number then you the screen will be stored in the base save area of the screen data backup area just as if no parameter was specified.

In addition to the Graphical data that is currently on-screen, the current Graphical Viewport settings and the currently active color palette are saved as well so that when a restore is done, the viewport and colors will be properly restored.

Text Variable Reference

If you wish to save the entire state of the RIPTerm system, use \$SAVEALL\$.

Example: \$SAVE(7)\$

Returns: *nothing*

\$SAVEx\$... Save graphics screen (x=0-9)

Format: \$SAVE0\$... \$SAVE9\$

Syntax: \$SAVE0\$... \$SAVE9\$

If you choose the \$SAVE0\$ through \$SAVE9\$, the screen is saved to a specified graphics screen data backup slot (from 0-9). When a screen stored in a data backup slot is restored (via the \$RESTORE\$ command), that data backup slot is deleted (unless it is protected).

In addition to the Graphical data that is currently on-screen, the current Graphical Viewport settings and the currently active color palette is saved as well so that when a restore is done, the viewport and colors will be properly restored.

If you wish to save the entire state of the RIPTerm system, use \$SAVEALL\$.

NOTE: *This method of saving graphical screens is obsolete. Use the \$SAVE\$ function (see above) with a parameter.*

Example: \$SAVE7\$

Returns: *nothing*

\$SAVEALL\$... Save all screen attributes

Format: \$SAVEALL(*destination*)\$

Syntax: \$SAVEALL(*opt:DESTINATION*)\$

DESTINATION	BASE	Save to base save area
	0-9	Save to save slot 0-9
	PUSH	Save on stack
	default = BASE	

This Active Text Variable saves the Text Window coordinates, the contents of the entire clipboard, all mouse fields, and the contents of the entire screen. It is the same as performing the following operations (in this order):

```
$STW(destination )$  
$SCB(destination )$  
$SMF(destination )$  
$SGS(destination )$  
$SBS(destination )$  
$SAVE(destination )$  
$SCP(destination )$  
$SENV(destination )$
```

This command does not require any parameters. If no parameter is specified, or if the *DESTINATION* parameter is set to *BASE*, then the saved information is saved to the base save area and can be restored many times. If you specify a slot number then that slot number is the slot that will be saved over. You are allowed up to ten separate slots (0-9).

In place of a slot number, you can provide a parameter of *PUSH* to perform a push-type stack saving operation (e.g., \$SAVEALL(PUSH)\$).

Example: \$SAVEALL(0)\$

Returns: *nothing*

\$SBAROFF\$... Turn OFF the Status Bar

Format: \$SBAROFF\$

Syntax: \$SBAROFF\$

This Active Text Variable turns OFF the Status Bar in the terminal.

Example: \$SBAROFF\$

Returns: *nothing*

\$SBARON\$... Turn ON the Status Bar

Text Variable Reference

Format: \$SBARON\$

Syntax: \$SBARON\$

This Active Text Variable turns ON the Status Bar in the terminal.

Example: \$SBARON\$

Returns: *nothing*

\$SBS\$... Save a button style to the backup area

Format: \$SBS(*destination*)\$

Syntax: \$SBS(*opt:DESTINATION*)\$

DESTINATION	BASE	Save to base save area
	0-9	Save to save slot 0-9
	PUSH	Save on stack
	default = BASE	

This text variable saves the button style data table to a data backup area. If the *DESTINATION* parameter is omitted, then it is saved to the base save area of the backup data area. Possible destinations are BASE to save over the base save area, the value 0-9 to overwrite a specific data save slot, or the value PUSH to indicate that you wish to push the button style data table onto the button style data backup area's save stack.

Example: \$SBS(PUSH)\$

Returns: *nothing*

\$SCB\$... Save Clipboard

Format: \$SCB(*destination*)\$

Syntax: \$SCB(*opt:DESTINATION*)\$

DESTINATION	BASE	Save to base save area
	0-9	Save to save slot 0-9
	PUSH	Save on stack
	default = BASE	

This Active Text Variable saves the Clipboard to disk for later retrieval by a Query or Host Command. If the clipboard is empty, the temporary file is deleted so Restore Clipboard knows there shouldn't be a clipboard active.

If you do not specify a *DESTINATION* parameter then the clipboard is saved to the drawing port backup area's base save area. When it is restored the file is not deleted so you can do multiple identical clipboard restorations.

If you do specify a *DESTINATION* parameter then you may save the clipboard to one of ten different clipboard slots (0-9). When you restore a clipboard slot, the clipboard file is deleted.

You may specify a slot number of *PUSH* to perform a stack-based push operation (e.g., \$SCB(PUSH)\$).

Example: \$SCB(4)\$

Returns: *nothing*

\$SCP\$... Save a color palette to the backup area

Format: \$SCP (*destination*)\$

Syntax: \$SCP (*opt:DESTINATION*)\$

DESTINATION	BASE	Save to base save area
	0-9	Save to save slot 0-9
	PUSH	Save on stack
	default = BASE	

This text variable saves the color palette data table to a data backup area. If the *DESTINATION* parameter is omitted, then it is saved to the base save area of the backup data area. Possible destinations are BASE to save over the base save area, the value 0-9 to overwrite a specific data save slot, or the value *PUSH* to indicate that you wish to push the color palette data table onto the color palette data backup area's save stack.

Text Variable Reference

Example: \$SCP(PUSH)\$

Returns: *nothing*

\$SEC\$... Seconds

Format: \$SEC\$

Syntax: \$SEC\$

This Text Variable returns a 2-digit number representing the current seconds of the minute. Possible values for this variable are 00-59.

Example: \$SEC\$

Returns: 59

\$SENV\$... Records environmental configuration

Format: \$SENV (*destination*)\$

Syntax: \$SENV (*opt:DESTINATION*)\$

DESTINATION	BASE	Save to base save area
	0-9	Save to save slot 0-9
	PUSH	Save on stack
	default = BASE	

This command takes a snapshot of the RIP *scrip* configuration over the one of the data backup area's particular save regions. This includes recording the values of:

- Current graphics style entry number
- Current button style entry number
- Current drawing port entry number
- Current text window entry number
- Current color palette entry number
- Current World coordinate dimensions (X and Y)
- Current base math settings (36 or 64)
- Current coordinate size (2 through 5)
- Current color mode (color palette mode or direct RGB mode)
- Current mouse pointer number

- Current baud rate emulation value
- Current environment data table entry currently active

This command saves an environment data table over an actual data backup area for later retrieval. If you omit the *DESTINATION* parameter, then the environment is stored into the base save area of the environment data backup area.

If the *DESTINATION* value is set to PUSH, then the data table is pushed onto the environment data backup area's stack.

If the *DESTINATION* value is set to BASE, then the data table is stored into the base save area of the environment data backup area.

If the *DESTINATION* value is set to a number from 0-9, then the environment data table is stored into a data save slot number in the environment data backup area.

Example: \$SENV(PUSH)\$

Returns: *nothing*

\$SGS\$... Save a graphics style to the backup area

Format: \$SGS (*destination*)\$

Syntax: \$SGS (*opt:DESTINATION*)\$

DESTINATION	BASE	Save to base save area
	0-9	Save to save slot 0-9
	PUSH	Save on stack
	default = BASE	

This text variable saves the graphics style data table to a data backup area. If the *DESTINATION* parameter is omitted, then it is saved to the base save area of the backup data area. Possible destinations are BASE to save over the base save area, the value 0-9 to overwrite a specific data save slot, or the value PUSH to indicate that you wish to push the graphics style data table onto the graphics style data backup area's save stack.

Example: \$SGS(PUSH)\$

Returns: *nothing*

\$SMF\$... Save Mouse Fields

Format: \$SMF\$(*destination*)

Syntax: \$SMF(*opt:DESTINATION*)\$

DESTINATION	BASE	Save to base save area
	0-9	Save to save slot 0-9
	PUSH	Save on stack
	default = BASE	

This Active Text Variable saves all defined Mouse Fields and Mouse Buttons to one of the mouse field data backup areas as specified by the *DESTINATION* parameter. This is designed especially for the graphical designer who wishes to pop-up a dialog box on the screen with one or more mouse fields, and when finished, to restore the screen and original mouse fields. This command is intended to be used with the Restore Mouse Fields text variable \$RMF\$.

If no *DESTINATION* parameter is specified then the mouse field definitions will be stored to the mouse field area's base save area. Mouse fields stored in the base save area can be restored multiple times. If you specify a the *DESTINATION* as a slot number from 0-9, then the fields are stored to specified data backup slot in the mouse field backup area. When mouse fields saved in a particular backup slot are restored, the corresponding data backup slot is erased (unless it is protected).

If you specify a *DESTINATION* of PUSH then you will be performing a stack-based push operation (e.g., \$SMF(PUSH)\$).

If you specify BASE instead of a slot number then you will be storing the mouse fields into the mouse field data backup area's base save area (just as if you had specified no parameters at all).

Example: \$SMF(4)\$

Returns: *nothing*

\$STATBAR\$... Status Bar Status

Format: \$STATBAR\$

Syntax: \$STATBAR\$

This Text Variable returns YES if the Status Bar is visible in the terminal. If the Status Bar is not visible, then NO is returned.

Example: \$STATBAR\$

Returns: YES

\$STW\$... Save Text Window information

Format: \$STW(*destination*)\$

Syntax: \$STW(*opt:DESTINATION*)\$

DESTINATION	BASE	Save to base save area
	0-9	Save to save slot 0-9
	PUSH	Save on stack
	default = BASE	

If no slot parameter is specified, then this command stores all currently defined text windows' settings base save area. The windows' X/Y dimensions are preserved, as are the current cursor location, ANSI attributes, cursor ON/OFF status and the vertical scrolling margins. Even the current System Fonts used for all windows are saved (if necessary).

If you specify a slot parameter, then you are specifying to save the text window definitions to one of up to ten different slots (0-9). These allow you to have up to ten different text window configurations saved at the same time. If you save to a slot then when you restore that slot the saved definitions are deleted from the disk. If you don't specify a slot parameter (a slot-less save) then you can restore that definition file multiple times without the file being deleted.

If you specify a slot number of PUSH then you are performing a stack-based save operation (e.g., \$STW(PUSH)\$).

If you specify BASE instead of a slot number than the text window table will be stored into the text window backup area's base save area.

Text Variable Reference

NOTE: *The contents of the Text Window are not saved.*

Example: \$STW(5)\$

Returns: *nothing*

\$T\$... Play a simple audio tone

Format: \$T(freq,length)\$

Syntax: \$T(opt:FREQUENCY, opt:LENGTH)\$

FREQUENCY	1-65535	Frequency of tone in Hertz
	default = 1000	

LENGTH	1-65535	Length to play in milliseconds
	default = 75	

This Active Text Variable produces an audible sound. Both parameters are required. The *FREQ* parameter determines the frequency in Hertz and the *LENGTH* parameter determines the duration of the tone in milliseconds. frequency is assumed to be 1000 Hertz and the length of time that it should play is 75 milliseconds.

Under no circumstances will values for any of the two parameters above 65535 be permitted. If values above these limits are encountered then the variable is not processed.

Unlike the \$BEEP\$ command, this variable has no pause after the sound stops playing, thus allowing you to string multiple \$T\$ variables together in rapid succession to produce musical notes.

\$TABOFF\$... Disable TAB key Mouse Field select

Format: \$TABOFF\$

Syntax: \$TABOFF\$

This Active Text Variable turns off the use of the TAB key to jump from one defined Mouse or Button Field to another. If this command is received when a field is highlighted, it is deselected. This should be done when entering a full-screen editor so that the user can use the TAB key as a TAB, not a Mouse Field selector.

Example: \$TABOFF\$

Returns: *nothing*

\$TABON\$... Enable TAB key Mouse Field select

Format: \$TABON\$

Syntax: \$TABON\$

This Active Text Variable turns on the use of the TAB key to jump from one defined Mouse or Button Field to another.

Example: \$TABON\$

Returns: *nothing*

\$TERMINFO\$... Returns vendor specific data

Format: \$TERMINFO(*keyword*)\$

Syntax: \$TERMINFO(*opt:KEYWORD*)\$

KEYWORD	NAME	Terminal software's name
	VENDER	Terminal software's maker
	VERSION	Software's version number
	LIST	List of available keywords
	default = NAME	

This text variable returns specific information about the RIP *scrip* software package in use by the terminal (remote) user. If no parameter is specified, for example, \$TERMINFO\$ or \$TERMINFO()\$, then the sequence returned to the host is the name of the terminal (see the NAME keyword below). Otherwise, you may specify a particular terminal information keyword to request information about. If the specific keyword is undefined (i.e., not used by the terminal), a value of NONE will be returned.

The text returned to the host is not terminated with any carriage returns or anything like that. It's up to you to provide that kind of information in a button's host string or in a query string. The LIST directive though returns a list of all recognized keywords for that terminal. For example,

Text Variable Reference

RIPterm Pro returns the following for the expression: `$TERMINFO(LIST)$`

`LIST,NAME,VENDOR,VERSION`

Note, the list is comma (,) delimited between keywords, but not after the last keyword. In addition, the keywords are returned in alphabetical order, converted to all capitals.

Example: `$TERMINFO$`
Returns: RIPterm Professional

Example: `$TERMINFO(NAME)$`
Returns: RIPterm Professional

Example: `$TERMINFO(VERSION)$`
Returns: 2.00.00

Example: `$TERMINFO(VENDOR)$`
Returns: TeleGrafix Communications, Inc.

Example: `$TERMINFO(LIST)$`
Returns: LIST,NAME,VENDOR,VERSION

Example: `$TERMINFO(GOOSE)$`
Returns: NONE

`$TEXTXY$` ... Obtain last X/Y graphical text location

Format: `$TEXTXY(port, domain )$`
Syntax: `$TEXTXY(opt:PORT, opt:DOMAIN)$`

<i>PORT</i>	<i>CUR</i>	Get location in current viewport
	0-35	Get location in viewport 0-35
	default = CUR	

<i>DOMAIN</i>	<i>WORLD</i>	Return in World coordinates
	<i>DEVICE</i>	Return in device coordinates
	default = WORLD	

This text variable returns the last X/Y graphical location that was updated with a `RIP_TEXT` or a `RIP_TEXT_XY` command. The data returned to the host is in the format `XXX:YYY`. If no parameters are specified then the last X/Y location of the current drawing port is returned to the host in the current environment's world coordinates. If you specify only one parameter, then you must specify either `CUR` for the current drawing port, or a numeric value from `0-35` to indicate a specific drawing port. When only the `PORT` parameter is specified then coordinates are returned in the current environment's world coordinates.

If you specify two parameters then the last one determines the *DOMAIN* of the coordinates returned to the host. Valid keywords for the *DOMAIN* parameter are `WORLD` and `DEVICE`. If it is set to `WORLD` then the coordinates are returned to the host in the current environment's world coordinate system. If it is `DEVICE` then the coordinates are returned in raw device pixel coordinates.

If the specified drawing port doesn't exist then the value `-1` is returned to the host.

If no `RIP_TEXT` or `RIP_TEXT_XY` commands have been issued then the coordinates returned will be `0:0`.

Example: `$TEXTXY(CUR,DEVICE)$`
Returns: `320:175`

`$TIME$` ... Time in standard format

Format: `$TIME$`
Syntax: `$TIME$`

This Text Variable returns the time in military format (hours from `00` - `23`). The format is hours, minutes, and seconds separated by colons:
HH:MM:SS

Example: `$TIME$`
Returns: `18:09:33`

`$TIMEZONE$` ... Time Zone or "NONE" if unknown

Text Variable Reference

Format: \$TIMEZONE\$

Syntax: \$TIMEZONE\$

This Text Variable returns a word/phrase that describes the time-zone the terminal is in. This may be returned as anything like PST for Pacific Standard Time, EST for Eastern Standard Time, etc. If the time zone is not set on your PC, this variable will respond with NONE

Example: \$TIMEZONE\$

Returns: PST

\$TWERASEEOL\$... Erase text window to line end

Format: \$TWERASEEOL(*window1*,*window2*,...)

Syntax \$TWERASEEOL(*opt:WINDOW1*, ...)\$

WINDOW1	CUR	Current Text Window
	0-35	Text Window number 0-35
	default = CUR	

This variable clears the current line in the specified text window from the cursor (inclusive) to the end of the line in the current ANSI color attributes. If the you specify no parameters, then the current text window is used for the destination window. If you do specify any parameters, they must be set to CUR for the current text window, ALL for all currently defined text windows, or 0-35 to indicate a specific text window slot number.

You may specify one or more parameters to indicate that you wish to erase more than one text window's current lines in the same \$TWERASEEOL\$ statement. If any one of them fails to match the proper parameters, then the entire command is considered a syntax error and none of the parameters are processed.

If the specified text window is deactivated or undefined then nothing happens for that parameter.

Unlike many other text window commands, this one can make an actual visual effect on the screen even if the specified text window isn't the current text window.

Example: \$TWERASEEOL(5)\$

Returns: *nothing*

\$TWFONT\$... Active Text Window Font

Format: \$TWFONT(*window*)\$

Syntax: \$TWFONT(*opt:WINDOW*)\$

WINDOW	CUR	Current Text Window
	0-35	Text Window number 0-35
	default = CUR	

This Text Variable returns which of the five Text Window Fonts is active, or 0 (zero) if specified text window is not defined, or is deactivated. The following values are returned:

Value	
0	No Text Window
1	80 x 43 font
2	91 x 43 MicroANSI font
3	80 x 25 font
4	91 x 25 MicroANSI font
5	40 x 25 font

Note that this command returns the actual text window font number plus one. So to match the returned value to actual RIP *scrip* text window font numbers, subtract one from this result (providing that it is greater than 0).

If you do not specify a window parameter, then this command refers to the current text window. If you do specify the window number parameter, then you may specify a value of 0-35 for a specific text window data table entry, or the value CUR to indicate the current text window (the default).

Example: \$TWFONT(4)\$

Returns: 1

\$TWGOTO\$... Move cursor to (X,Y) in Text Window

Format: \$TWGOTO(*winno*,*x_pos*,*y_pos*)\$

Syntax: \$TWGOTO(*req:WINNO*, *opt:X_POS*, *opt:Y_POS*)\$

<i>WINNO</i>	<i>CUR</i>	Current Text Window
	0-35	Text Window number 0-35
	default = CUR	

<i>X_POS</i>	X coordinate to move to
--------------	-------------------------

<i>Y_POS</i>	Y coordinate to move to
--------------	-------------------------

This variable allows you to move the cursor position in a given text window to a specified X/Y position. If no parameters are given or no X/Y parameters are given then nothing happens when this is executed.

If the specified text window doesn't exist, or is deactivated, then this command does nothing.

The *WINNO* parameter specifies a text window data table entry to move the cursor in. Valid settings for *WINNO* are 0-35 for a specific

text window data table entry, or *CUR* to indicate the current text window. If the specified window is not the current text window then the cursor in that window's definition is moved but no visible things occur on the screen. If the specified window is *CUR* or the window number happens to be the current text window, then the cursor that is (potentially) on the screen is moved to the new location.

The X/Y coordinates are text coordinates (1 based). Specifying both coordinates allows you to place the cursor anywhere in the chosen window. You may omit the *Y_POS* parameter entirely to move the cursor horizontally only. Either the *X_POS* or *Y_POS* parameter (or both) may be specified as *CUR* to indicate the row or column is not to change. In other words, if you wanted to move the cursor to line 2 in the text window 5, but leave its column unchanged, you would use \$TWGOTO(5,CUR,2)\$.

If the go to sequence would put the cursor outside the dimensions of the text window then the cursor is placed at the closest point to the given location at the windows border.

Text locations in this command are zero based (e.g., the first column is specified as 0). If the no `X_POS` or `Y_POS` parameters are specified, then this command does nothing because there is no X/Y information to move to.

Example: `$TWGOTO(CUR,10,2)$`

Returns: *nothing*

`$TWH$` ... Text Window Height

Format: `$TWH(window,type,domain )$`

Syntax: `$TWH(opt:WINDOW, opt:TYPE,opt:DOMAIN)$`

WINDOW	CUR	Current Text Window
	0-35	Text Window number 0-35
	default = CUR	
TYPE	BOUND	Return height of bounding box in pixels
	TEXT	Return the height of the display region
	CELL	Return the height of a text cell
DOMAIN	WORLD	Return value in World coordinates
	DEVICE	Return value in Device coordinates
	*ROWS	Return value in text lines
	default = WORLD	

* only valid if TYPE = TEXT

If no parameters are specified, then this command returns the height of the current text window in text cells (lines). You may specify a window number to indicate which text window you are inquiring about. The text window number parameters may be a value from 0-35 to reference a specific text window data table entry, or you may specify a keyword of CUR to indicate the current text window. If the specified text window doesn't exist, or is deactivated, then a value of 0 is returned.

Text Variable Reference

If you specify any parameters then they must be in a specific order. The first (already discussed) is the text window number, and if it is omitted, then the current text window is assumed. If you specify two parameters, then you must specify the window number parameter and the type parameter. The type parameter determines what type of information you are inquiring about. If you omit the type parameter then you are inquiring about the text window height itself, in actual text window cell sizes (e.g., the window might respond with 5 to indicate that it is 5 lines of text tall.

If you specify the TYPE parameter, then you are directing this command to respond with information about a specific text window piece of information.

Type	
BOUND	Return the height of the bounding box in pixels
TEXT	Return the height of the display region
CELL	Return the height of a text cell in the text window

The third and final parameter is only of use if the BOUND or CELL parameter is used, and it determines the *DOMAIN* of the pixel size returned by the text variable. The possible values of domain are:

Domain	
WORLD	Return the value in world coordinates
DEVICE	Return the vaule in raw device pixel coordinates
COLS	Return width of area in text columns

If the domain parameter is omitted, then it automatically defaults to WORLD (except for the TEXT type, which defaults to the height of the text window in lines of text). When a world coordinate value is returned, the actual "raw" dimensions are converted to the current environment's world coordinate system then returned to the host. If the domain is set to DEVICE then the value is returned in raw, device-level pixel coordinates.

A *TYPE* of BOUND refers to the text window's bounding rectangle. It is the absolute exterior that the text window may extend to. It can also be

thought of as a margin outside the actual text window where nothing can be displayed from the text window. See the `RIP_EXTENDED_TEXT_WINDOW` for more details about this bounding rectangle.

A *TYPE* of `CELL` indicates that you are inquiring about the height of a character cell used in the text window. The height of a cell is based on the font number and resolution currently in use on the actual destination *RIPscrip* software package. Under 640x350, this was typically 8x8 for font #0 (i.e., in *RIP scrip* 1.54). This may be different now at varying resolutions. So, if you are running at a resolution of 640x350 and you inquire about a text window's cell height for a text window using font #0, then it most likely will return a result of 8 pixels (in device coordinates). The actual value in world coordinates could vary well be different based on the size of the world coordinate resolution frame.

The `TEXT` type indicates that you are requesting information about the text window sub-section of the text window definition. If the domain parameter is specified, then you are requesting the actual height of the text window display area in "lines of text". This can also be retrieved if the domain parameter is `ROWS`. This particular domain parameter known as `ROWS` is only valid for a type parameter of `TEXT`. Using it in any other context is considered a syntax error. If the domain parameter is specified, and it is set to `WORLD`, then the pixel height of the display region is being requested in world coordinates. If domain is set to `DEVICE`, then you are requesting the height of the display region in actual hardware pixel coordinates.

It should be noted that the bounding box of a text window can be obtained regardless of whether it is an old-style 1.54 text window created with `RIP_TEXT_WINDOW` or a new "resolution independent" text window defined via the `RIP_EXTENDED_TEXT_WINDOW` command. This can be because under the older text window concept, the bounding box is exactly as large as the text window itself, while under the extended text window, the text window will typically be inside the bounding rectangle.

If you completely omit the type parameter, then you must omit the domain parameter. In this case, you are asking for the height of the text window in "lines of text", not in pixels.

Text Variable Reference

Example: \$TWH\$	Height of current text window in lines of text.
Returns: 5	
Example: \$TWH(CUR)\$	Height of current text window in lines of text.
Returns: 5	
Example: \$TWH(5,BOUND)\$	Height of text window 5's bounding box in world coordinates.
Returns: 800	
Example: \$TWH(5,BOUND,DEVICE)\$	Height of text window 5's bounding box in device pixel coordinates
Returns: 400	
Example: \$TWH(CUR,CELL)\$	Height of current text window's character cell in world coordinates.
Returns: 16	
Example: \$TWH(5,CELL,DEVICE)\$	Height of text window 5's character cell in device pixel coordinates.
Returns: 8	

\$TWHOME\$... Move cursor to home position in text window

Format: \$TWHOME(*winno*)\$
Syntax: \$TWHOME(*opt:WINNO*)\$

WINNO	CUR	Current Text Window
	0-35	Text Window number 0-35
	default = CUR	

This command moves the cursor in the specified text window data table entry to the upper-left corner of the window. *WINNO* may be any value from 0-35 to indicate the desired data table entry, or the value *CUR* to indicate the current text window. If the *WINNO* slot number happens to

correspond to the current text window or the `WINNO` parameter is specified as `CUR` then the cursor that is on the screen (potentially) will be visibly moved. If the `WINNO` parameter specifies a window that is not the current text window then the cursor position definition for that non-current window is altered only - no visible thing would happen on the screen.

If the specified window is deactivated or is undefined then nothing happens.

If the `WINNO` parameter is omitted then the current text window is assumed.

Example: `$TWHOME(5)$`

Returns: nothing

`$TWIN$` ... Text Window Status

Format: `$TWIN(window )$`

Syntax: `$TWIN(opt:WINDOW)$`

<code>WINDOW</code>	<code>CUR</code>	Current Text Window
	<code>0-35</code>	Text Window number 0-35
	default = <code>CUR</code>	

This text variable checks to see if the specified Text Window is activated, and returns `YES` if the specified text window is activated or returns `NO` if it is deactivated (e.g., with the `$DTW$` command for example). If the specified text window isn't defined, then this command returns a value of `NO`.

If you do not specify a window parameter, then this command refers to the current text window. If you do specify a window parameter then it can be a value from `0-35` to indicate a specific text window data table entry, or the value `CUR` to indicate the current text window.

Example: `$TWIN(4)$`

Returns: `YES`

\$TWW\$... Text Window Width

Format: `$TWW(window,type,domain )$`

Syntax: `$TWW(opt:WINDOW, opt:TYPE, opt:DOMAIN)$`

<i>WINDOW</i>	<i>CUR</i>	Current Text Window
	0-35 default = CUR	Text Window number 0-35
<i>TYPE</i>	<i>BOUND</i>	Return width of bounding box in pixels
	<i>TEXT</i>	Return the width of the display region
	<i>CELL</i>	Return the width of a text cell
<i>DOMAIN</i>	<i>WORLD</i>	Return value in World coordinates
	<i>DEVICE</i>	Return value in Device coordinates
	<i>*COLS</i>	Return value in text columns
	default = WORLD	

* only valid if TYPE = TEXT

If no parameters are specified, then this command returns the width of the current text window in text cells (columns). You may specify a window number to indicate which text window you are inquiring about. The text window number parameters may be a value from 0-35 to reference a specific text window data table entry, or you may specify a keyword of CUR to indicate the current text window. If the specified text window doesn't exist, or is deactivated, then a value of "0" is returned

If you specify any parameters then they must be in a specific order. The first (already discussed) is the text window number, and if it is omitted, then the current text window is assumed. If you specify two parameters, then you must specify the window number parameter and the type parameter. The type parameter determines what type of information you are inquiring about. If you omit the type parameter then you are inquiring about the text window width itself, in actual text window cell sizes (e.g., the window might respond with 5 to indicate that it is 5 columns wide.

If you specify the *TYPE* parameter, then you are directing this command to respond with information about a specific text window piece of information. The available type parameters are:

Text Variables Reference

Type	
BOUND	Return the width of the bounding box in pixels
TEXT	Return the width of the display region
CELL	Return the width of a text cell in the text window

The third and final parameter is only of use if the *TYPE* parameter is set to *BOUND* or *CELL*. It determines the *DOMAIN* of the pixel size returned by the text variable. The possible values of *DOMAIN* are:

Domian	
WORLD	Return the value in world coordinates
DEVICE	Return the value in raw device pixel coordinates
COLS	Return width of area in text columns (valid only for a type of "TEXT")

If the *DOMAIN* parameter is omitted, then it automatically defaults to *WORLD* (except for the *TEXT* type, which defaults to the height of the text window in lines of text). When a world coordinate value is returned, the actual "raw" dimensions are converted to the current environment's world coordinate system then returned to the host. If the *DOMAIN* is set to *DEVICE* then the value is returned in raw, device-level pixel coordinates.

If the *DOMAIN* parameter is omitted, then it automatically defaults to *WORLD* (except for the *TEXT* type, which defaults to the width of the text window in columns of text). When a world coordinate value is returned, the actual "raw" dimensions are converted to the current environment's world coordinate system then returned to the host. If the *DOMAIN* is set to *DEVICE* then the value is returned in raw, device-level pixel coordinates.

A *TYPE* of *BOUND* refers to the text window's bounding rectangle. It is the absolute exterior that the text window may extend to. It can also be thought of as a margin outside the actual text window where nothing can be displayed from the text window. See the *RIP_EXTENDED_TEXT_WINDOW* for more details about this bounding rectangle.

Text Variable Reference

A *TYPE* of CELL indicates that you are inquiring about the width of a character cell used in the text window. The width of a cell is based on the font number and resolution currently in use on the actual destination RIP*scrip* software package. Under 640x350, this was typically 8x8 for font #0 (i.e., in RIP *scrip* 1.54). This may be different now at varying resolutions. So, if you are running at a resolution of

640x350 and you inquire about a text window's cell width for a text window using font #0, then it most likely will return a result of 8 pixels (in device coordinates). The actual value in world coordinates could vary well be different based on the size of the world coordinate resolution frame.

The TEXT type indicates that you are requesting information about the text window sub-section of the text window definition. If the domain parameter is specified, then you are requesting the actual width of the text window display area in "columns of text". This can also be retrieved if the domain parameter is COLS. This particular domain parameter known as COLS is only valid for a type parameter

of TEXT. Using it in any other context is considered a syntax error. If the *DOMAIN* parameter is specified, and it is set to WORLD, then the pixel width of the display region is being requested in world coordinates. If domain is set to DEVICE, then you are requesting the width of the display region in actual hardware pixel coordinates.

It should be noted that the bounding box of a text window can be obtained regardless of whether it is an old-style 1.54 text window created with RIP_TEXT_WINDOW or a new "resolution independent" text window defined via the RIP_EXTENDED_TEXT_WINDOW command. This can be because under the older text window concept, the bounding box is exactly as large as the text window itself, while under the extended text window, the text window will typically be inside the bounding rectangle.

If you completely omit the *TYPE* parameter, then you must omit the *DOMAIN* parameter. In this case, you are asking for the width of the text window in "text columns", not in pixels.

Example: \$TWW\$Width of current text window in columns of text.

Returns: 5

Example: \$TWW(CUR)\$ Width of current text window in columns of text.

Returns: 5

Example: \$TWW(5,BOUND)\$ Width of text window 5's bounding box in world coordinates.

Returns: 800

Example: \$TWW(5,BOUND,DEVICE)\$ Width of text window 5's bounding box in device pixel coordinates

Returns: 400

Example: \$TWW(CUR,CELL)\$ Width of current text window's character cell in world coordinates.

Returns: 16

Example: \$TWW(5,CELL,DEVICE)\$ Width of text window 5's character cell in device pixel coordinates.

Returns: 8

\$TWX0\$... Text Win Upper Left X Coordinates

Format: \$TWX0(*window,type,domain*)\$

Syntax: \$TWX0(*opt:WINDOW*, *opt:TYPE*, *opt:DOMAIN*)\$

WINDOW	CUR	Current Text Window
	0-35 default = CUR	Text Window number 0-35
TYPE	BOUND	Upper left X of bounding rectangle
	TEXT	Upper left X of display rectangle
DOMAIN	WORLD	Return value in World coordinates
	DEVICE	Return value in Device coordinates
	default = WORLD	

Text Variable Reference

This variable returns upper-left X coordinate information about a text window. The exact type and nature of the returned information is determined by the parameters provided (if any). Without any

parameters, this command returns the upper-left X coordinate of the text window in text coordinates (see below for a more detailed explanation).

If the first parameter is provided, then it indicates which text window you requesting information on. It may be set to a numeric value from 0-35 to indicate a specific text window data table entry, or it may be CUR to indicate the current text window.

If the specified text window doesn't exist, or is deactivated, then this command returns a value of -1 to indicate failure.

NOTE: Under 1.54, this text variable used to return "0" to indicate that a text window was deactivated. This was inaccurately documented in previous RIPscrip releases because "0" could be a valid upper-left X coordinate.

If the second parameter, *TYPE*, is provided, then the window number parameter must be present. The *TYPE* parameter defines what type of information you are requesting about the text window. The available keyword values for the type parameter are as follows:

Type	
BOUND	Upper left X coordinate of the bounding rectangle
TEXT	Upper left X coordinate of text window display rectangle

A *TYPE* of BOUND indicates that you are inquiring about the text window's bounding rectangle's upper left X coordinate. The result is returned in either world coordinates or in physical device coordinates based on the domain parameter (see below).

A *TYPE* of TEXT indicates that you are inquiring about the actual text window's display area somewhere inside the bounding rectangle. The information returned is the upper-left X graphical coordinate of the text window display rectangle. It is returned either in world coordinates or in device pixel coordinates based on the domain parameter (see below).

The third parameter is only valid if the *TYPE* parameter is provided. It indicates the *DOMAIN* under which the *TYPE* parameter data is returned to the host. The possible values for *DOMAIN* are:

Domain	
WORLD	Return information in current world coordinates
DEVICE	Return information in physical device coordinates

The *DOMAIN* parameter determines what type of numbers are returned for the bounding box or the text window display rectangle. If this parameter is omitted, then the coordinates are returned in world coordinates. If the parameter is specified, then the result is in either world or device coordinates depending on the value of the parameter.

If no *TYPE* parameter is specified (as previously described), then the text window's upper-left X coordinate is returned in text coordinates. This is to maintain backward compatibility with older RIP *scrip* v1.54 related commands. In fact, this is not the best way of determining where the text window is on the screen - graphical coordinates are a much better method. If a text window is defined with the *RIP_TEXT_WINDOW* command where you specify the location of the text window solely on the basis of text coordinate X/Y data, then this form of this command will return a number indicating which X character cell the text window starts at. If the text window is defined using the *RIP_EXTENDED_TEXT_WINDOW* command, where the upper-left corner of the text window might not start on an even multiple of the window's cell size, then this command returns a value of -1 to indicate that the desired request cannot be processed because the text window isn't the right kind of text window. In this manner, the value -1 is used to indicate that an error has occurred with this command.

Example: \$TWX0\$ Upper left X coordinate of the current text window in text coordinates

Returns: 5

Example: \$TWX0(CUR)\$ Upper left X coordinate of the current text window in text coordinates

Returns: 5

Text Variable Reference

Example: \$TWX0(CUR,BOUND)\$ Upper left X coordinate of the current text window's bounding box in world coordinates.

Returns: 100

Example: \$TWX0(CUR,BOUND,WORLD)\$ Upper left X coordinate of the current text window's bounding box in world coordinates.

Returns: 100

Example: \$TWX0(CUR,BOUND,DEVICE)\$ Upper left X coordinate of the current text window's bounding box in device pixel coordinates.

Returns: 50

Example: \$TWX0(CUR,TEXT,DEVICE)\$ Upper left X coordinate of the text window's display box in device pixel coordinates.

Returns: 55

\$TWX1\$... Text Win Lower Right X Coordinate

Format: \$TWX1(*window,type,domain*)\$

Syntax: \$TWX1(*opt:WINDOW, opt:TYPE, opt:DOMAIN*)\$

WINDOW	CUR	Current Text Window
	0-35 default = CUR	Text Window number 0-35
TYPE	BOUND	Lower right X of bounding rectangle
	TEXT	Lower right X of display rectangle
DOMAIN	WORLD	Return value in World coordinates
	DEVICE	Return value in Device coordinates
	default = WORLD	

This variable returns lower-right X coordinate information about a text window. The exact type and nature of the returned information is determined by the parameters provided (if any). Without any

parameters, this command returns the lower-right X coordinate of the text window in text coordinates (see below for a more detailed explanation).

If the first parameter is provided, then it indicates which text window you requesting information on. It may be set to a numeric value from 0-35 to indicate a specific text window data table entry, or it may be CUR to indicate the current text window.

If the specified text window doesn't exist, or is deactivated, then this command returns a value of -1 to indicate failure.

NOTE: Under 1.54, this text variable used to return "0" to indicate that a text window was deactivated. This was inaccurately documented in previous RIPscrip releases because "0" could be a valid lower-right X coordinate.

If the second parameter, *TYPE*, is provided, then the *WINDOW* number parameter must be present. The type parameter defines what type of information you are requesting about the text window. The available keyword values for the type parameter are as follows:

Type	
BOUND	Lower right X coordinate of the bounding rectangle
TEXT	Lower right X coordinate of text window display rectangle

A *TYPE* of BOUND indicates that you are inquiring about the text window's bounding rectangle's lower right X coordinate. The result is returned in either world coordinates or in physical device coordinates based on the domain parameter (see below).

A *TYPE* of TEXT indicates that you are inquiring about the actual text window's display area somewhere inside the bounding rectangle. The information returned is the lower-right X graphical coordinate of the text window display rectangle. It is returned either in world coordinates or in device pixel coordinates based on the domain parameter (see below).

The third parameter is only valid if the *TYPE* parameter is provided. It indicates the *DOMAIN* under which the type parameter data is returned to the host.

Text Variable Reference

The *DOMAIN* parameter determines what type of numbers are returned for the bounding box or the text window display rectangle. If this parameter is omitted, then the coordinates are returned in world coordinates. If the parameter is specified, then the result is in either world or device coordinates depending on the value of the parameter.

If no *TYPE* parameter is specified (as previously described), then the text window's lower-right X coordinate is returned in text coordinates. This is to maintain backward compatibility with older RIP *scrip* v1.54 related commands. In fact, this is not the best way of determining where the text window is on the screen - graphical coordinates are a much better method. If a text window is defined with the *RIP_TEXT_WINDOW* command where you specify the location of the text window solely on the basis of text coordinate X/Y data, then this form of this command will return a number indicating which X character cell the text window starts at. If the text window is defined using the *RIP_EXTENDED_TEXT_WINDOW* command, where the

lower-right corner of the text window might not start on an even multiple of the window's cell size, then this command returns a value of -1 to indicate that the desired request cannot be processed because the text window isn't the right kind of text window. In this manner, the value -1 is used to indicate that an error has occurred with this command.

When graphical coordinate information is returned on the lower right X location, it is returned based on the resolution independent nature of rectangles in RIP *scrip*. This means that if the very last most device pixel of text window data is at X coordinate 299, then this command would return a value of 300 for the lower right X coordinate value. See the section on the "The Mathematics Of Graphics And Coordinates" for more details about why we use this coordinate convention.

Example: \$Twx1\$ Lower right X coordinate of the current text window in text coordinates

Returns: 5

Example: \$Twx1(CUR)\$ Lower right X coordinate of the current text window in text coordinates

Returns: 5

Example: \$TWX1(CUR,BOUND)\$ Lower right X coordinate of the current text window's bounding box in world coordinates.

Returns: 100

Example: \$TWX1(CUR,BOUND,WORLD)\$ Lower right X coordinate of the current text window's bounding box in world coordinates.

Returns: 100

Example: \$TWX1(CUR,BOUND,DEVICE)\$ Lower right X coordinate of the current text window's bounding box in device pixel coordinates.

Returns: 50

Example: \$TWX1(CUR,TEXT,DEVICE)\$ Lower right X coordinate of the text window's display box in device pixel coordinates.

Returns: 55

\$TWY0\$... Text Win Upper Left Y Coordinate

Format: \$TWY0(*window,type,domain*)\$

Syntax: \$TWY0(*opt:WINDOW, opt:TYPE, opt:DOMAIN*)\$

WINDOW	CUR	Current Text Window
	0-35 default = CUR	Text Window number 0-35
TYPE	BOUND	Upper left Y of bounding rectangle
	TEXT	Upper left Y of display rectangle
DOMAIN	WORLD	Return value in World coordinates
	DEVICE default = WORLD	Return value in Device coordinates

This command is identical in every way to the \$TWX0\$ text variable except that it returns information on the upper-left Y coordinate of the text window. The exact same parameters apply as in the \$TWX0\$ text variable (see that command for more details).

\$TWY1\$... Text Win Lower Right Y Coordinate

Format: \$TWY1(*window,type,domain*)\$

Syntax: \$TWY1(*opt:WINDOW, opt:TYPE, opt:DOMAIN*)\$

WINDOW	CUR	Current Text Window
	0-35 default = CUR	Text Window number 0-35
TYPE	BOUND	Lower right Y of bounding rectangle
	TEXT	Lower right Y of display rectangle
DOMAIN	WORLD	Return value in World coordinates
	DEVICE default = WORLD	Return value in Device coordinates

This command is identical in every way to the \$TWX1\$ text variable except that it returns information on the lower-right Y coordinate of the text window. The exact same parameters apply as in the \$TWX1\$ text variable, as do conventions for coordinates (see that command for more details).

\$UNPROT\$... Unprotects object

Format: \$UNPROT(*data_object*,*element1*,...)\$

Syntax: \$UNPROT(*req:OBJECT*, *req:ELEMENT1*, ...)\$

OBJECT	ELEMENT	
SCREEN	S0-S9	Unprotect screen slot 0-9
SCREEN	ALLSLOTS	Unprotect all screen slots 0-9
MOUSE	S0-S9	Unprotect mouse slot 0-9
MOUSE	ALLSLOTS	Unprotect all mouse slots 0-9
TW	CUR	Unprotect current text window
TW	1-35	Unprotect Text window entry 1-35
TW	ALL	Unprotect all text window entries 1-35
TW	S0-S9	Unprotect text window slot 0-9
TW	ALLSLOTS	Unprotect all text window slots 0-9
PORT	CUR**	Unprotect current viewport
BUT	CUR**	Unprotect current button entry
STYLE	CUR**	Unprotect current graphic style
PAL	CUR**	Unprotect current palette entry
ENV	CUR**	Unprotect current enviroment entry

** 1-35, ALL, S0-S9, and ALLSLOTS work same as for TW.

This command unprotects a specific element of a given data object. What is element of the data object that is to be unprotected is defined by the *ELEMENT* parameter. This parameter must be specified (you may specify more than one to unprotect multiple elements in one command). If a data object element is attempted to be unprotected, but it is not in use then this command does nothing for that parameter.

Example: \$UNPROT(TW, S7)\$

Returns: *nothing*

\$VT102OFF\$... Turn VT-102 keyboard mode OFF

Format: \$VT102OFF\$

Syntax: \$VT102OFF\$

Text Variable Reference

This Active Text Variable disables the VT-102 terminal emulation mode, returning your text windows to standard ANSI operation.

Example: \$VT102OFF\$

Returns: *nothing*

\$VT102ON\$... Turn VT-102 keyboard mode ON

Format: \$VT102ON\$

Syntax: \$VT102ON\$

This Active Text Variable enables the VT-102 terminal emulation option of the RIP *scrip* software. This affects character placement and formatting of text in a text window, and also the way that the keyboard operates.

Example: \$VT102ON\$

Returns: *nothing*

\$WDAY\$... Day of Week

Format: \$WDAY\$

Syntax: \$WDAY\$

This Text Variable returns a one-digit number representing the day of the week. Possible values are 0-6, where 0=Sunday (the first day in the week).

Example: \$WDAY\$

Returns: 2

\$WORLD\$... Set/query World coordinate frame

Format: \$WORLD(*env_no,width,height*)\$

Syntax: \$WORLD(*opt:ENV_NO, opt:WIDTH,opt:HEIGHT*)\$

<i>ENV_NO</i>	<i>CUR</i>	Set/Query current enviroment
	0-35	Set/Query a specific enviroment
	default = CUR	

WIDTH	1-65535	Set width to a specific value
	DEFAULT	Set width to default value
HEIGHT	1-65535	Set height to a specific value
	DEFAULT	Set height to default value

This command allows you to set or query an environment's world coordinate frame.

If you specify no parameters, then you are querying the contents of the current environment's world coordinate frame. The return value to the host will be a value similar to the following 1234:5678 where 1234 is the width of the coordinate frame in the horizontal X direction, and the value 5678 is the height of the coordinate frame in the vertical Y direction.

If you specify one parameter, then you are querying the world coordinate frame of a specific environment from 0-35 or CUR for the current environment. The result returned to the host is in the same format as if you specified no parameters (see above). If the requested environment isn't in use, then the value -1 is returned.

If you wish to set the world coordinate frame for an environment, you have two choices. You can set it to some basic set of default values or you can set it to a specific height and width.

If you wish to set the world coordinate frame to a set of default values then you specify two parameters - the environment number (0-35 or CUR), and the second parameter must be the value DEFAULT. The default width and height values will depend on that environment's Base Math setting. If MegaNums are in use in that environment, then the world frame is set to 1280x960. If it is UltraNums, it is set to 4096x3072.

If you wish to manually set the width and height of the world coordinate frame, then you must specify all three parameters. The first one is the environment (0-35 or CUR). The second parameter is the width of the world coordinate frame and the third and final parameter is the height of the coordinate frame. Both of these values can not exceed the value 65535. If the specified environment isn't in use, then a syntax error is

Text Variable Reference

generated. This variation of `$WORLD$` is the same as using the `RIP_SET_WORLD_FRAME RIPscrip` command.

Example: `$WORLD(CUR,1000,1000)$`

Returns: *nothing ... Sets current environment world frame to 1000x1000*

Example: `$WORLD(5, DEFAULT)$`

Returns: *nothing ... Sets environment 5's world frame to 1280x960 if MegaNums, or 4096x3072 if it's in UltraNum mode.*

Example: `$WORLD(CUR)$`

Returns: *640:350 ... Returns world frame of current environment*

`$WORLDH$` ... Vertical resolution (height) of world coordinate system

Format: `$WORLDH(env_no,height )$`

Syntax: `$WORLDH(opt:ENV_NO, opt:HEIGHT)$`

ENV_NO	CUR	Set/query current enviroment
	0-35	Set/query a specific enviroment
	default = CUR	
HEIGHT	1-65535	Set height to a specific value
	DEFAULT	Set height to a specific value

This function is used to either set or query the setting of the world coordinate system's height. If no parameters are specified (e.g., `$WORLDH$`) or a single `CUR` parameter is specified, then the current environment's world coordinate height dimension is returned to the host. If you would like to inquire about a specific environment's world coordinate height setting, specify the environment table entry number as the only parameters (e.g., `$WORLDH(5)$` inquires about environment 5's world coordinate height setting).

If you would like to alter the world coordinate system's height setting, you need to specify two parameters. The first parameter must be a table entry from `0-35` or the value `CUR` for the current environment.

The second parameter must be a world coordinate setting to set the height to.

Note, if you try to a value in an environment that is not in use, this command generates a syntax error. If you attempt to query a value from an environment that is not in use, the value -1 is returned to the host.

Example: \$WORLDH\$
Returns: 4096

Example: \$WORLDH(CUR, 1000)\$
Returns: *nothing*

\$WORLDW\$... Horizontal resolution of world coordinate system

Format: \$WORLDW(*env_no,height*)\$
Syntax: \$WORLDW(*opt:ENV_NO, opt:WIDTH*)\$

ENV_NO	CUR	Set/Query current enviroment
	0-35	Set/Query a specific enviroment
	default = CUR	
WIDTH	1-65535	Set width to a specific value
	DEFAULT	Set width to default value

This function is used to either set or query the setting of the world coordinate system's width. If no parameters are specified (e.g., \$WORLDX\$) or a single CUR parameter is specified, then the current environment's world coordinate width is returned to the host. If you would like to inquire about a specific environment's world coordinate width setting, specify the environment table entry number as the only parameters (e.g., \$WORLDX(5)\$ inquires about environment 5's world coordinate width setting).

If you would like to alter the world coordinate system's width setting, you need to specify two parameters. The first parameter must be a table entry from 0-35 or the value CUR for the current environment. The second parameter must be a world coordinate setting to set the width to.

Text Variable Reference

Note, if you try to a value in an environment that is not in use, this command generates a syntax error. If you attempt to query a value from an environment that is not in use, the value -1 is returned to the host.

Example: \$WORLDW\$
Returns: 4096

Example: \$WORLDW(CUR, 1000)\$
Returns: *nothing*

\$WOY\$... Week of current year 00-53; Sunday=1st Day of Week

Format: \$WOY\$
Syntax: \$WOY\$

This Text Variable returns a number from 00-53, representing the week in the year. Even though there are 52 weeks in a year, a week might not begin exactly on the first day of the year, so a maximum value for this variable can be 53 under these circumstances. For this variable, Sunday is considered to be the first day of the week.

Example: \$WOY\$
Returns: 32

\$WOYM\$... Week of current year 00-53; Monday=1st Day of Week

Format: \$WOYM\$
Syntax: \$WOYM\$

This Text Variable returns a number from 00-53, representing the week in the current year. Even though there are 52 weeks in a year, a week might not begin exactly on the first day of the year, so a maximum value for this variable can be 53 under these circumstances. For this variable, Monday is considered to be the first day of the week.

Example: \$WOYM\$

Returns: 32

\$X\$... X Mouse location

Format: `$X(domain)$`

Syntax: `$X(opt:DOMAIN)$`

DOMAIN	WORLD	Return in World coordinates
	DEVICE	Return in Device coordinates
	default = WORLD	

This text variable returns the current X coordinate of the mouse pointer. This can be used interactively (for example, by on-line games) to determine the location of the mouse pointer. Only the X value of the mouse (X,Y) is returned. The value is 0000-9999 depending on what the current position is.

If this text variable is used inside a "text window query", then the X coordinate is returned in text cell coordinates based on the internal dimensions of the text window. This kind of result cannot happen unless the user clicks inside a text window's display area that has a text window query command in it, so there's no possibility for the coordinate to be out of bounds for that text window's dimensions.

The results are in current world coordinates if no parameter is specified. If *DOMAIN* is *DEVICE* then the result is in raw video device coordinates. If *DOMAIN* is *WORLD* then the result is in desktop world coordinates. If the domain parameter is specified when this text variable is used in a text mode query, then it is explicitly overriding the "text coordinate" defaults of this command and returning the mouse coordinate information in the desired coordinate system.

Example: `$X(WORLD)$` ... equivalent to `$X$`

Returns: 0523

\$XFER\$... Initiate a file transfer with the hostFormat: `$XFER(dir,prot,type,filename,... )$`Syntax: `$XFER(req:DIR,req:PROT,req:TYPE,opt:FILENAME, ...)$`

DIR	SEND	Tells RIPterm prepare to send a file
	RECEIVE	Tells RIPterm prepare to receive a file
PROT	KERMIT	Kermit file tranfer
	SKERMIT	Super Kermit file transfer
	XMODEM	X Modem file transfer
	XMODEMCRC	X- Modem CRC file transfer
	XMODEM1K	X Modem 1k file transfer
	XMODEM1KG	X Modem G 1k file transfer
	YMODEM	Y Modem file transfer
	YMODEMG	Y Modem G file transfer
	ZMODEM	Z Modem file transfer
	ZMODEMCR	Z Modem with crash recovery
TYPE	NONE	Store host's or ICONS directory
	IMAGE	Display according to current Image Style setting
	RIP	Store and replay text file locally
	ACTIVE	Store, replay RIP files locally and display any images according to current Image style setting

This command starts up a binary file transfer to or from the host system. At least three parameters are required. Sometimes a fourth parameter (the filename) is required - see below for a description of file transfer protocols.

The first parameter is the *DIR* parameter which determines the direction of the file transfer.

The next parameter determines the file transfer protocol to use. This determines the "language" that will be spoken with the host system to transfer file(s). Valid settings for this parameter are as follows:

Text Variables Reference

Protocol		Receive		
KERMIT	Kermit	Yes	Yes	No
SKERMIT	Super Kermit	No	Yes	No
XMODEM	Xmodem (checksum)	Yes	Yes	No
XMODEMCRC	Xmodem (CRC)	Yes	Yes	No
XMODEM1K	Xmodem-1K	Yes	Yes	No
XMODEM1KG	Xmodem-1K (G)	Yes	Yes	No
YMODEM	Ymodem (batch)	No	Yes	Yes
YMODEMG	Ymodem (G)	No	Yes	Yes
ZMODEM	Zmodem	No	Yes	Yes
ZMODEMCR	Zmodem (w/crash recovery)	No	Yes	Yes

In the table described above we show what protocol keywords are permitted in the `PROT` parameter and what kind of protocols belong to those keywords. In addition we describe information about the filename parameter (which we will discuss in detail below). The filename parameter columns *Receive* and *Send* specify whether the filename parameter is required with this command based on the protocol. For example, Xmodem-CRC requires a filename to be specified both when sending and receiving, but a filename is only required when sending file(s) with Ymodem ! You must always specify a filename parameter when sending file(s). Some protocols allow you to send multiple files within the same file transfer (e.g., Zmodem , Ymodem , etc.). For these protocols the column *Multiple Files Allowed* will have Yes in their entries; these are considered "batch" protocols. For non-batch protocols like Xmodem , you can only transfer one file at a time. The multiple files allowed only pertains to "sending" files to the host - when this column indicates that multiple files are permitted, then you can specify more than one filename parameter with this text variable. When receiving files from the host you can only specify one filename (if the protocol requires it - some protocols don't even require a filename to be specified).

The third parameter is the *TYPE* parameter. This specifies the type of files that will be received from the host. This parameter has no significance when sending files - only when receiving. In this manner, you should set this parameter to `NONE` when sending files. The possible *TYPE* parameters are described as follows:

Text Variable Reference

Type	
NONE	Do nothing with the file. For files that are received, they are stored in the host system's directory (e.g., ICONS\, etc.).
IMAGE	Any files received that appear to be JPEG or GIF images are displayed based on the current image style settings. If the image style does not direct the image to be deleted then it is stored in the host system's directory (like the type "NONE" above)
RIP	Any files received that appear to be RIP <i>scrip</i> scene files will be played back as if a local RIP <i>scrip</i> file playback directive were received. All files received in this mode are stored in the host system's directory (as above).
ACTIVE	This mode is basically a combination of the IMAGE and RIP types described above. Any image files will be displayed in the image style settings and any RIP <i>scrip</i> files will be executed. Other files are stored in the host system's directory as above.

Essentially this command functions identically to the RIP_ENTER_BLOCK_MODE command of RIP *scrip*. There are no fundamental differences in it except that the *TYPE* parameter is a simplified version of the FILE_TYPE parameter of the block command.

The filename parameter is used when sending file(s) or when receiving files with a protocol that isn't a batch protocol (e.g., Xmodem , etc.). When sending files you must specify at least one filename parameter. Some protocols allow you to specify more than one filename parameter. Wildcards are not allowed in filenames, nor is any path information - only raw filenames. When receiving files you may or may not have to specify a filename - it depends on if the file transfer protocol is a "batch" protocol (Ymodem or Zmodem variations). If it is then you do not have to specify a filename parameter (it would be ignored if you did).

Omitting a filename during send mode or when receiving a file with a non-batch protocol like Xmodem is considered a syntax error.

Before executing this command you should verify that the RIP *scrip* terminal supports the file transfer protocol via the `$IFS()` text variable. RIPterm from TeleGrafix doesn't support Super Kermit for example - other vendors might have similar limitations.

Example: `XFER(SEND,ZMODEM,NONE,FILE1.BMP,FILE2.RIP)$`
Returns: *nothing*

Example: `$XFER(RECEIVE,YMODEM,ACTIVE)$`
Returns: *nothing*

`$XY$` ... X/Y Mouse Location

Format: `$XY(domain )$`
Syntax: `$XY(opt:DOMAIN)$`

DOMAIN	WORLD	Return in World coordinates
	DEVICE	Return in Device coordinates
	default = WORLD	

This Text Variable returns both the X and Y coordinates of the mouse pointer. A colon (:) separates the two values. The X and Y values may range from 0000-9999. The format that this value uses is: `XXXX:YYYY`

If this text variable is used inside a "text window query", then the X/Y coordinates are returned in text cell coordinates based on the internal dimensions of the text window. This kind of result cannot happen unless the user clicks inside a text window's display area that has a text window query command in it, so there's no possibility for the coordinate to be out of bounds for that text window's dimensions. The format that the coordinates are returned in is `XX:YY` where `XX` and `YY` can range from 00-99.

The results are in current world coordinates if no parameter is specified. If `DOMAIN` is `DEVICE` then the results are in raw video device coordinates. If `DOMAIN` is `WORLD` then the results are in desktop world coordinates. If the domain parameter is specified when this text variable is used in a text mode query, then it is explicitly overriding the "text coordinate" defaults of this command and returning the mouse coordinate information in the desired coordinate system.

Text Variable Reference

Example: \$XY(WORLD)\$... equivalent to \$XY\$

Returns: 0297:0321

\$XYM\$... X, Y & button status

Format: \$XYM(*domain*)\$

Syntax: \$XYM(opt:DOMAIN)\$

DOMAIN	WORLD	Return in World coordinates
	DEVICE	Return in Device coordinates
	default = WORLD	

This Text Variable returns the X and Y coordinates of the mouse pointer, and which mouse buttons are pressed (if any). A colon (:) separates the three values. The X and Y values may range from 0000-9999. *LMR* stands for Left/Middle/Right. If any of these buttons are depressed (clicked), then the corresponding position will contain a 1. If a button is not depressed, then it will contain a 0. The format that this value uses is: *XXXX:YYYY:LMR*

If this text variable is used inside a "text window query", then the X/Y coordinates are returned in text cell coordinates based on the internal dimensions of the text window. This kind of result cannot happen unless the user clicks inside a text window's display area that has a text window query command in it, so there's no possibility for the coordinate to be out of bounds for that text window's dimensions. The format that the coordinates are returned in is *XX:YY:LMR* where *XX* and *YY* can range from 00-99.

The results are in current world coordinates if no parameter is specified. If *DOMAIN* is *DEVICE* then the results are in raw video device coordinates. If *DOMAIN* is *WORLD* then the results are in desktop world coordinates. If the domain parameter is specified when this text variable is used in a text mode query, then it is explicitly overriding the "text coordinate" defaults of this command and returning the mouse coordinate information in the desired coordinate system.

Example: \$XYM(WORLD)\$... Equivalent to \$XYM\$

Returns: 0123:0297:110

\$Y\$... Y Mouse location

Format: \$Y(*domain*)\$

Syntax: \$Y(*opt*:DOMAIN)\$

DOMAIN	WORLD	Return in World coordinates
	DEVICE	Return in Device coordinates
	default = WORLD	

This Text Variable returns the current Y coordinate of the mouse pointer. This can be used interactively (for example, by on-line games) to determine the location of the mouse pointer. Only the Y value of the Mouse (X,Y) is returned. The value is 0000-9999 depending on what the current position is.

If this text variable is used inside a "text window query", then the Y coordinate is returned in text cell coordinates based on the internal dimensions of the text window. This kind of result cannot happen unless the user clicks inside a text window's display area that has a text window query command in it, so there's no possibility for the coordinate to be out of bounds for that text window's dimensions.

The results are in current world coordinates if no parameter is specified. If *DOMAIN* is DEVICE then the result is in raw video device coordinates. If *DOMAIN* is WORLD then the result is in desktop world coordinates. If the domain parameter is specified when this text variable is used in a text mode query, then it is explicitly overriding the "text coordinate" defaults of this command and returning the mouse coordinate information in the desired coordinate system.

Example: \$Y(WORLD)\$... equivalent to \$Y\$

Returns: 0244

\$YEAR\$... 2 digit year

Format: \$YEAR\$

Syntax: \$YEAR\$

Text Variable Reference

This Text Variable returns the two-digit number of the current year.

Example: \$YEAR\$

Returns: 93

