UnForm® User Guide

Version 6.0
UnForm is published under license by:
Synergetic Data Systems, Inc.
2195 Talon Drive
Latrobe, CA 95682
USA

Phone: (530)-672-9970
Fax: (530)-672-9975
Email: sdsi@synergetic-data.com
Web page: http://synergetic-data.com

UnForm® Page Enhancement Software
License Agreement

NOTICE: OPENING THIS PACKAGE INDICATES YOUR ACCEPTANCE OF THE FOLLOWING TERMS AND CONDITIONS. PLEASE READ THEM. IF YOU DO NOT AGREE WITH THEM, RETURN THE PACKAGE UNOPENED, AND RETURN OR DESTROY ANY COPIES OF THE PROGRAM IN YOUR POSSESSION. THE DEALER FROM WHOM YOU PURCHASED THE SOFTWARE WILL REFUND YOUR PURCHASE PRICE.

"Program", as used herein, refers to both this documentation and the software programs described by this documentation.
"Developer", as used herein, refers to Allen D. Miglore. "Publisher" as used herein refers to Synergetic Data Systems, Inc.

LICENSE
You may use the Program on a single machine, and you may copy the Program into any machine-readable format for backup purposes only. If you transfer the Program to another machine, you agree to destroy the Program, together with all copies, in whole or in part, on the original machine.

You may not copy, modify, or transfer the Program, in whole or in part, except as expressly provided herein. You may not sublicense, assign, or otherwise transfer the Program to any third party except by the express written consent of the Developer or Publisher.

TERM
The license is effective until terminated. You may terminate at any time by destroying the Program together with all copies of the Program in your possession. It will also terminate automatically upon failure to comply with any of the terms of this agreement. You agree upon such termination to destroy the Program together with all copies in your possession in any form.

CONFIDENTIALITY OF THE PROGRAM
You understand that the Program is proprietary to the Developer, and agree to maintain the confidentiality of the Program. You agree that neither you, nor any person or entity acting on your behalf, will copy or otherwise transfer the Program, in whole or in part, in any form (including printed source code), to any third party. You agree to retain the Developer's copyright notices, in all forms, throughout the Program. You agree not to de-encrypt or de-compile the Program.

LIMITATION OF LIABILITY
The Program is provided "AS IS" without warranty of any kind, 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 Program is with you.

In no event will the Developer or Publisher be liable to you for any damages, including any lost profits or other incidental or consequential damages arising out of the use or inability to use the Program, even if advised of the possibility of such damages.

SUPPORT
Support for the Program should be obtained from the Dealer from whom it was purchased. Support pricing and terms are established by the Dealer, not the Developer or Publisher.

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 YOU AND THE DEVELOPER AND PUBLISHER AND IT SUPERSEDES ANY PRIOR AGREEMENT, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATION BETWEEN YOU AND THE DEVELOPER RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

TABLE OF CONTENTS

TABLE OF CONTENTS **3**

INTRODUCTION **6**

CLIENT-SERVER ARCHITECTURE **7**

SERVER INSTALLATION **8**

CLIENT INSTALLATION **11**

CONFIGURING THE SERVER **13**

CONFIGURING EXTERNAL PROGRAMS **15**

TCP/IP MONITOR **18**

INTEGRATING UNFORM WITH BBX **21**

INTEGRATING UNFORM WITH PROVIDEX **23**

INTEGRATING UNFORM WITH NON-BUSINESS BASIC APPLICATIONS **25**

LICENSING **27**

UNFORM COMMAND LINE OPTIONS **31**

VERSION 6 FEATURES **42**

FLOW OF PROCESSING **45**

CONCEPTS, PRIMER, AND TIPS **48**

RULE FILES **53**

CONTENT-BASED RULE SETS 54
ACROSS 55
ATTACH 56
AUTHOR 58
BARCODE (PCL,PDF) 59
BARCODE (ZEBRA) 62
BIN 65
BOJ, BOP, EOJ, EOP 66
BOLD, ITALIC, LIGHT, UNDERLINE 67
CBOLD, CITALIC, CLIGHT, CUNDERLINE 67
BOX, CBOX 68
BOXR, CBOXR 71
COLS 74
COMPRESS 75
CONST 76
COPIES, PCOPIES 77
CPI 78

CROSSHAIR 79
DETECT 80
DOWN 82
DPI 83
DSN_SAMPLE 84
DUMP 85
DUPLEX 86
EMAIL 87
ERASE, CERASE 89
FIXEDFONT 90
FONT, CFONT 91
GS 94
HLINE 95
HSHIFT 96
IF COPY … END IF 97
IF DRIVER … END IF 98
IMAGE 99
ITALIC 103
KEYWORDS 104
LANDSCAPE, RLANDSCAPE 105
LIGHT 106
LPI 107
MACRO 108
MACROS 109
MARGIN 110
MERGE 111
MICR 112
MOVE, CMOVE 113
NOTEXT 115
OUTLINE 116
OUTPUT 117
PAGE 118
PAPER 119
PORTRAIT, RPORTRAIT 120
PRECOPY, PREDEVICE, PREJOB, PREPAGE 121
POSTCOPY, POSTDEVICE, POSTJOB, POSTPAGE 121
PROTECT 124
ROWS 125
SHADE, CSHADE 126
SHIFT 128
SUBJECT 129
SYMSET 130
TEXT 131
TITLE 136
TRAY 137
UNDERLINE 138
UNITS 139
VLINE 140
VSHIFT 141

WORKING WITH MACROS 142

REGULAR EXPRESSIONS 144

SAMPLE RULE FILES 145

SIMPLE1 - INVOICE RULE SET (SIMPLE.RUL) 146
SIMPLE2 – INVOICE RULE SET (SIMPLE.RUL) 148
SIMPLE3 – INVOICE RULE SET (SIMPLE.RUL) 151
SIMPLE4 – INVOICE RULE SET (SIMPLE.RUL) 154
INVOICE - INVOICE FOR PRE-PRINTED FORM (ADVANCED.RUL) 158
STATEMENT - PLAIN PAPER FORM, TWO PAGE FORMATS IN SAME JOB (ADVANCED.RUL) 165
AGING REPORT - ENHANCED AGING REPORT (ADVANCED.RUL) 171
LABELS – TEXT LABELS TO LASER LABELS (ADVANCED.RUL) 177
132X4 – MULTI-UP, SCALED REPORTING (ADVANCED.RUL) 179
ZEBRA LABEL – ZEBRA® LABEL PRINTER EXAMPLE (ADVANCED.RUL) 180
PDF OUTLINE SAMPLE (ADVANCED.RUL) 182

PROGRAMMING CODE BLOCKS 184

BASIC SYNTAX 185
INTERNAL VARIABLES 190
INTERNAL FUNCTIONS 194
VERBS AND FUNCTIONS 198
ERROR CODES 204

EMAIL INTEGRATION 206

HTML OUTPUT 213

CREATING HTML 214
HTML CONFIGURATION 216
HTML OUTPUT TEMPLATES 218
HTML RULE SETS 221
BORDER 222
COLDEF 223
COLWIDTH 227
FRAME 228
HDRON, HDROFF, HDRTD 229
LOAD 230
MULTIPAGE 231
NULLROW 232
OUTPUT 233
OTHEROPT 234
PAGESEP 235
PREJOB, PREPAGE, POSTJOB, POSTPAGE 236
ROWDEF 238
TITLE 241
TOC 242
WIDTH 243
SAMPLE HTML RULE SET 244
AGING REPORT SAMPLE 244

INDEX 248

INTRODUCTION

UnForm is a software product designed to work as a filter between an application and an output device like a LaserJet printer or a program like a fax product. Most applications can be simply configured to print through UnForm, which in turn processes the output from the application, determines if custom processing is necessary, and then applies any enhancements before it is output.

For example, if a UNIX program sends output to the spooler like this:

cat *file-name* | lp -dlaser -s 2>/dev/null

then the output can be changed to use UnForm (note the use of the –oraw option, which can vary by operating system):

cat *file-name* | uf60c -f acct.rul | lp -dlaser –oraw –s 2>/dev/null

or for better performance, UnForm can print directly from the server:

cat *file-name* | uf60c –f acct.rul –o ">lp –dlaser -oraw"

UnForm can also work in Windows environments, as long as the application can produce a file and then execute UnForm to process the file and produce output.

UnForm is unique in its ability to analyze report output to determine what, if any, customization to apply. When a report is detected that requires enhancements, UnForm can add line drawing, shading, attributes, font control, and text to the form. UnForm can also handle the processing of multiple copies, multiple output devices, attachments, overlays, and graphic images, and includes support for the complete Business Basic programming environment to add true programmed intelligence to any form.

The enhanced output can be used to simulate pre-printed forms, or to change the look of plain-paper forms, for which headings and dashed lines are printed by the application, from crude to professional. UnForm can also be used to enhance reports, such as financial statements or aging reports, raising them from mundane to board room quality.

UnForm can produce enhancements on any printer or device that offers the HP PCL5 printer language. This includes all HP LaserJet and compatible printers beginning with the HP LaserJet III, many UNIX faxing software packages, and other products.

UnForm can also produce virtually identical output in Adobe's Portable Document Format (PDF), and similar output in Zebra's ZPL II language, supported on many Zebra thermal label printers. With proper configuration, UnForm can automatically convert its PDF output to any format supported by Ghostscript, including postscript, tif, jpeg, png, and more. Lastly, UnForm can parse column and row oriented reports and produce formatted HTML output.

CLIENT-SERVER ARCHITECTURE

UnForm Version 6 introduces a new client-server architecture, where the UnForm processing of documents can occur on a different machine from the application. The resulting enhanced document can be printed, emailed, sent to a fax gateway, or stored at the server, or can be returned to the client machine for printing or storage from its perspective. One important benefit of using a client-server model is that the application process that is sending jobs to UnForm via the client software need not wait for the job to finish if the server will be handling the output. This provides better performance to the application user, particularly for large or complex jobs that take time for UnForm to process.

The UnForm server can run on either UNIX or Windows systems. The server provides the UnForm processing logic and a listener, which handles job requests from clients located on the network.

The UnForm clients can be installed anywhere on the network, on Windows or UNIX systems. On Windows, the client is a native Windows executable. On UNIX, the client is a Perl program, so UNIX systems require Perl level 5 or above. Clients perform the application interface work, taking input from the application, submitting it to the server, and in many cases, returning the result back to the client for processing.

There is nothing to prevent the same machine from acting as both client and server, and in fact, the server installation automatically installs a client on that machine. Submitting a job to 'localhost' when the client and server run on the same machine can improve performance, as job data need not be transferred over the network.

For complete information about how to operate the client and server programs, read the Command Line Options chapter. In general, on Windows the server is operated from the Status Monitor option or as a Windows service, and on UNIX the server is operated like this:

uf60d start
uf60d stop

The client supports an extensive set of options. Some simple examples:

cat sample1.txt | uf60c –f simple.rul | lp –dhp –oraw
uf60c –i sample1.txt –f simple.rul –o ">lp –dhp –oraw"
uf60c –i sample1.txt –f simple.rul –p pdf –o client:sample1.pdf

In the first example, uf60c submits the job and returns the result to its spooler. In the second example, uf60c submits the job and the server prints the result to its spooler. In the third example, uf60c submits the job requesting PDF output, and returns the result to its file sample1.pdf.

SERVER INSTALLATION

UNIX Server CD installation instructions:

1. Login as root.

2. Mount the CD as a file system that supports lowercase file names. If you are unsure how to do this, check your man pages: **man mount**. The following table illustrates sample mount commands for various operating systems, assuming standard CD device names and that the mount directory /mnt is available. You may need to adjust these commands according to your configuration.

SCO UNIX OS5	mount –o lower /dev/cd0 /mnt
SCO UNIX	mount –r –f HS,lower /dev/cd0 /mnt
Unixware	mount –F cdfs –r /dev/cdrom/cdrom1 /mnt
AIX	mount –vcdrfs –r /dev/cd0 /mnt
Sun Solaris	mount –rt hsfs /dev/sr0 /mnt
HP/UX	mount –r –F cdfs –o cdcase /dev/dsk/c1d0s2 /mnt

3. Change to the UnForm60 UNIX directory in the mount directory: **cd /mnt/unform60/UNIX**

4. Run the install script: **./install.sh**, or if you do not have execute permission to the file, **sh install.sh**. You must agree to the license agreement, then you will be presented a list of operating system versions. Choose the correct version for your system.

5. UnForm will then be installed to the selected directory, and the set up script ./ufsetup.sh will be automatically executed in the UnForm directory.

   The ufsetup.sh script will create two scripts, called /usr/bin/uf60c and /usr/bin/uf60d. The uf60c program is the client, while uf60d manages the server.

6. Activate demo mode, or activate permanently, using **./license.sh**.

7. Start the server: **uf60d start**

8. Use the **uf60c –v** command to ensure UnForm is installed and set up correctly. The output from this command will display information about the installation. Note that uf60c requires Perl version 5 or higher.

See the **Licensing** section for additional activation information.

Note that you will probably want to place the **uf60d start** command in your system boot scripts, often found in the /etc/init.d directory or a similar location, depending on your version of UNIX.

UNIX Server download installation instructions:

1. Login as root.

2. Create a directory to hold the UnForm files, and change to that directory.

   Example: **umask 0**
   mkdir /usr/unform6
   cd /usr/unform6

3. Uncompress and extract UnForm from the download file.

   uncompress uf60_xxx_tar.Z
   tar xvf uf60_xxx_tar

4. Execute the UnForm set up script.

   ./ufsetup.sh

   The ufsetup.sh script will create two scripts, called /usr/bin/uf60c and /usr/bin/uf60d. The uf60c program is the client, while uf60d manages the server.

5. Activate demo mode, or activate permanently, using **./license.sh**.

6. Start the server: **uf60d start**

7. Use the **uf60c –v** command to ensure UnForm is installed and set up correctly. The output from this command will display information about the installation. Note that uf60c requires Perl version 5 or higher.

See the **Licensing** section for activation information.

Note that you will probably want to place the **uf60d start** command in your system boot scripts, often found in the /etc/init.d directory or a similar location, depending on your version of UNIX.

Windows Server installation instructions:

1. From the CD, use Explorer to locate the D:\unform60\win directory, and double-click the setup.exe program (*use Control Panel Add/Remove Programs if the system supports Terminal Services*). If you downloaded UnForm from the Internet, simply execute the downloaded executable (*use Control Panel Add/Remove programs if the system supports Terminal Services*). Follow the on-screen prompts from the installer to install UnForm to your system. This will install both the uf60d.exe server program and the uf60c.exe client program. The client program and its associated support files will be installed in the Windows directory, enabling a command line launch without a full path, as the Windows directory is always included in the PATH environment variable.

2. Click the **Server Configuration** option from the Start menu. This will conditionally rename certain files and prompt for several configuration values. The values entered are stored in several local .ini files in the UnForm server directory. You can also use the **Configure** button from within the UnForm Server Manager.

3. Click the **Server Manager** option from the Windows Start, Programs, UnForm 6.0 Server menu.

4. Activate the demo mode, then if desired, activate permanently, by pressing the **Licensing** button and using the form that displays. On line help is available if needed.

5. Click the **Start** button from the Server Manager to start the server manually.

6. Use the **Server Version** option from the Start menu to ensure the server is running properly and the client can operate from the server computer. The output from this command will display the version and licensing information.

7. If desired, and you are running the server on Windows NT, 2000, XP or any of the Windows variants that support NT Services, you can install the server as a service by running the **Install as a Service** option. When the UnForm server is run as a service, it is automatically started when Windows boots up. You must start and stop the service using the Windows Services applet, found in the Control Panel Administrative Tools option. The UnForm Server Manager options for starting and stopping the server are disabled.

See the **Licensing** section for activation information.

CLIENT INSTALLATION

The uf60c client software can be used to submit jobs to UnForm from anywhere on your network after the server is installed and operating. The client software is automatically installed on the same machine as the server, so jobs can be submitted locally. However, you can install the client software on any network computer. Any client can talk to any server, so you can mix and match different operating systems as you need. For example, you could install the Windows server, and have both Windows and UNIX clients submit jobs to it.

Clients must be installed on any machine that will be submitting jobs to UnForm. For example, in a Windows network, with the UnForm server installed on a single network server, each workstation that will be submitting jobs must have a client installed and configured to communicate with that server.

The UNIX client is installed from the file uf60c_tar.Z, while the Windows client installer is called uf60c_setup.exe.

The UNIX install steps are as follows:

- Ensure the system has Perl level 5 or higher: **perl –v**
  If not, Perl can be obtained from http://perl.com or http://cpan.org.
- Create a directory for the client, such as **mkdir /usr/lib/sdsi/uf60client**
- Set permissions on that directory: **chmod 777 /usr/lib/sdsi/uf60client**
- Copy the uf60c_tar.Z file to that directory and **cd** to that directory
- Uncompress the file: **uncompress uf60c_tar.Z**. If you have gzip, then the gunzip utility can also uncompress the file.
- Extract the files: **tar xvf uf60c_tar**
- Run the setup script: **./uf60csetup.sh**
- Edit the uf60c.ini file to set up the client configuration:

The uf60c.ini file looks like this:

```
[defaults]
server=localhost
port=2714
#logfile=uf60c.log
#mailto=root
retry=30
wait=2
```

Change the server= line to point to the server host name or IP address, and the port line to the proper listening port configured in the server's uf60d.ini file. The port default is 2714, and will not normally be changed. Note that the server and port can also be specified on the uf60c command line. The values entered here serve as defaults.

If you want uf60c to log errors, uncomment the logfile= line, setting the value to a log file name.

If you want uf60c to email (using the UNIX mail command) error messages to an administrator, uncomment the mailto= line, setting the value to an email address available from the client computer. Note that the Windows client does not support emailing of error messages.

The retry and wait lines set the number of times, and delay between tries, that the client will attempt to connect to the server before giving up. If any retries are needed, and the log file is specified, then a message will be logged.

On Windows, the installation steps are:

- Run the **uf60c_setup.exe** installer program.
- Run the **Configure UnForm Client** option from the Start menu. Enter the appropriate values for the server and port, and optionally the log file.

CONFIGURING THE SERVER

The server is configured via the uf60d.ini file, which can be edited with any text editor. On Windows, many of these options can be configured with the Configure button in the Server Manager. In addition to these items, you can also configure access to Ghostscript, Image Magick, or Image Alchemy elsewhere in the uf60d.ini file. See the Configuring External Programs chapter for more details.

In the defaults and security sections, here are the values available:

[defaults] section	
port=*n*	Sets the primary listing TCP/IP port to *n*. The default is 2714. Note that if you use NAT translation or if you have a firewall between the clients and server, then this port (along with the procports defined below) must be configured to allow clients access.
logfile=*path*	Sets the name of the server's log file to *path*. By default, it is stored in the UnForm directory. Standard log entries include connection information. Detailed logging includes verbose data transactions.
logdetail=*n*	Set *n* to 0 for standard logging, 1 for detailed logging. You should not leave detailed logging enabled for normal use, as the log file can grow very large.
timeout=*n*	Set *n* to the number of seconds that a connection can remain idle before closing. The default value is 3600, or one hour. Setting this value to 0 will avoid timeout-based disconnects. This value primarily affects designer connections, which can remain active for long periods.
age=*n*	This value sets the maximum age, in days, of job log entries. When jobs are submitted, basic job information is kept in a log file. If errors were recorded, the error file also remains in the temp directory under the UnForm server. After this many days, the files and log entries are automatically removed.
rulefile=*path*	Sets the default rule file to *path*, used for jobs that do not specify a rule file on the command line.
bbpath=*path*	If the bbxread() function is used, this value points to the BBx executable that is invoked when required, such as /usr/lib/basis/pro5/pro5.
library=*path1*;*path2*; …	Sets directory paths that are automatically searched for rule files, images, and attachments. By default, UnForm searches the UnForm directory and also supports full paths.
[security] section	
allow=*list*	This is a semi-colon delimited list of valid IP addresses or wildcards that are allowed to connect to the server. Note that the loopback I address 127.0.0.1 is always allowed to connect. The default list is 192.*.*.*;10.*.*.*, which allows the two standard non-routable LAN spaces to work.
procports=*list*	This is a comma-delimited list of ports or port ranges that are used for secondary connections. When a client connects to the server on the primary port, a second task is launched to handle that connection. That task uses a procport value to communicate with the client. You should define at least as many procports as you have job and designer licenses.

	Also note that if you use NAT translation or if there is a firewall between the clients and the server, these ports need to be configured in addition to the primary listening port.
hideconn=*n*	This value, which defaults to 1, is supported on Windows installations. If *n* is 0, and the server is running as an application rather than a service, connections show up on the task bar. Otherwise, they are hidden.
[tcpports] section	
port=options	Each line defines a port on which the server listens for raw print job deliveries, such as from Windows TCP/IP ports. Each job submission is then processed using a uf60c command line configured with a pre-defined –ix option plus any other *options* defined. For more information, see the TCP/IP Monitor chapter.

Note also many parameters are stored in the ufparam.txt file. You can create a custom version of this file, called ufparam.txc, which will be used instead of ufparam.txt. Any new parameters that are added during a release cycle are documented in the readme.txt file, and can be added manually to keep ufparam.txc up to date if necessary.

CONFIGURING EXTERNAL PROGRAMS

The UnForm server supports the use of three external programs for handling two tasks: image scaling and conversion, and document imaging conversion.

For image scaling, you can configure either Image Alchemy, a commercial product available from Handmade Software (http://handmadesw.com), or Image Magick, an open source product available from http://imagemagick.com. Once configured, image scaling is automatically used when an image command contains size information and the image file is not a native laser or PDF file.

For document imaging conversion, you can configure Ghostscript, an open source or commercial product available from http://ghostscript.com. Document imaging is managed by the –p command line argument, and it enables a series of additional drivers, such as tif, postscript, and png.

Once the appropriate programs are installed, then edit the uf60d.ini file to configure them.

Use the [images] section to configure Image Alchemy or Image Magick, first by defining a converter=*path* entry, where *path* is the execution path of the alchemy or convert programs. If the path is in the operating system's PATH variable, then just a simple name will be required. Since the server, uf60d, will be executing the program, you should make sure that the user under which it runs includes the proper environment variable definitions. For example, Image Magick uses a variable called MAGICK_HOME.

In addition to the executable, define several command line argument lines for pcl, pclc, and pdf, and optionally others that can be called out by the option item of the image command. Generally, you can simply uncomment the proper lines for Alchemy or Magick. The pcl command is invoked for laser output, and the pdf command is invoked for PDF output. If the image command's color option is used, or the –ci command line option is used, then the pclc command is invoked. Below is a sample uf60d.ini [images] section, with Image Magick enabled:

```
[images]
# External image conversion/scaling program setup

# 1) Define program path: converter=pathname
# Use a full path if necessary, as this becomes a system call in UnForm.
# On Windows, this will very likely be necessary.

# 2) Define arguments to be passed to converter here for pcl, pclc, and pdf.
# Use %i for input image, %o for output, %d for dpi, %x for width, %y for height
# pdf should not contain %x/%y, as scaling is performed by Acrobat.

# Options passed from image command line can be appended to the name with a dash.
# i.e. image 10,10,10,10,"image.bmp",option 123 would use pcl-123 or PDF-123.
# Options can be up to 10 characters long, and are case sensitive.

# Examples for Image Alchemy:
#converter=alchemy
#pcl="%i" "%o" -o -Q -D %d %d -+ -Xc%x -Yc%y -P 103 >/dev/null 2>&1
#pclc="%i" "%o" -o -Q -D %d %d -+ -Xc%x -Yc%y --r 9 >/dev/null 2>&1
```

```
#PDF="%i" "%o" -o -Q -D %d %d --d -8 >/dev/null 2>&1


# Examples for ImageMagick:
converter=convert
pclc="%i" -density %dx%d -colors 256 -dither -resize %xx%y "%o" >/dev/null 2>&1
pcl="%i" -density %dx%d -monochrome -resize %xx%y "%o" >/dev/null 2>&1
PDF="%i" -density 300x300 -colors 256 "%o" >/dev/null 2>&1
#PDF-72="%i" -density 72x72 -colors 256 "%o" >/dev/null 2>&1

# PDF-72, above, is a 72 dpi image conversion, and would be specified
# with 'option 72' in an image command.  The resulting file will be much
# smaller than the 300 dpi image shown in PDF=, though quality may suffer
# too much for use, depending on the image itself.
```

Use the [drivers] section to define the Ghostscript-hosted imaging drivers. When this feature is enabled, the –p *driver* option supports a series of new names, all derived from an intermediate PDF document that is converted at the end of the job to the specified format. First, enable the gs=*path* line to instruct UnForm how to run Ghostscript. On UNIX, this is often just the word "gs", while on Windows it is often a full path to the gswin32c.exe program.

Other entries are simply *name=device,multipage,dpi*, where *name* is the UnForm driver name, *device* is the –sDEVICE name used by Ghostscript, *multipage* is a 0 or 1, where 1 means the output is multi-page=multi-file and 0 means all pages go to a single file, and *dpi* is the dots-per-inch resolution.

Note that the use of multi- or single-page output is often dependent on the image format. For example, bmp files do not support multiple pages per file, while tif files do.

Note that the graphical designer may rely on the png entry shown, depending on how it is configured.

```
[drivers]
# enable ghostscript drivers by uncommenting the gs= line
gs=gs
# windows would typically need a full path
# gs=c:\gs\gs8.xx\bin\gswin32c.exe

# driver lines are structured as name=gsdevice,multipage,density
# gsdevice is the ghostscript sDEVICE value
# multipage is boolean 0 or 1, 1 means -o file is file<page>.ext
#    Many formats require a 1, as the image format supports only a
#    single image per file.
# density is output density, as hhh[xvvv] (horizontalxvertical) dpi

bmp=bmp256,1,300
bmpmono=bmpmono,1,300

tif=tiffcrle,0,300
tifmono=tiffg3,0,300

png=png256,1,300
pngmono=pngmono,1,300

jpeg=jpeg,1,300
```

```
ps=pswrite,0,300
eps=epswrite,1,300
deskjet=deskjet,0,300
```

TCP/IP MONITOR

Beginning with version 6.0.07, UnForm includes a TCP/IP monitor program that can watch for raw print jobs arriving from network computers, similar to how an HP JetDirect card would. In effect, the UnForm server can serve one or more virtual JetDirect ports, each with an associated UnForm client command line.

The monitor is automatically started if there are one or more port configuration lines defined in the [tcpports] section of uf60d.ini. For example:

This line would print to the server's spooler –dlaser device, processing jobs through the acme.rul file:

```
9100=-o ">lp –dlaser –oraw" –f acme.rul
```

This line would print to a Windows server shared UNC printer, processing jobs through the acme.rul file:

```
9101=-o \\winsrv\laser1 -f acme.rul
```

This line would generate pdf files to the path specified, using the date and job number to generate unique names:

```
9102=-o "/usr/pdfs/%d.%j.pdf" –p pdf
```

The following subsititions are made in the command line definition:

Characters	Substitution
%d	The date in YYYYMMDD format.
%t	The time in HHMMSS format, using a 24 hour clock.
%p	The process ID (this is not necessarily unique).
%j	The sequential job number, which is an ever-increasing unique number.

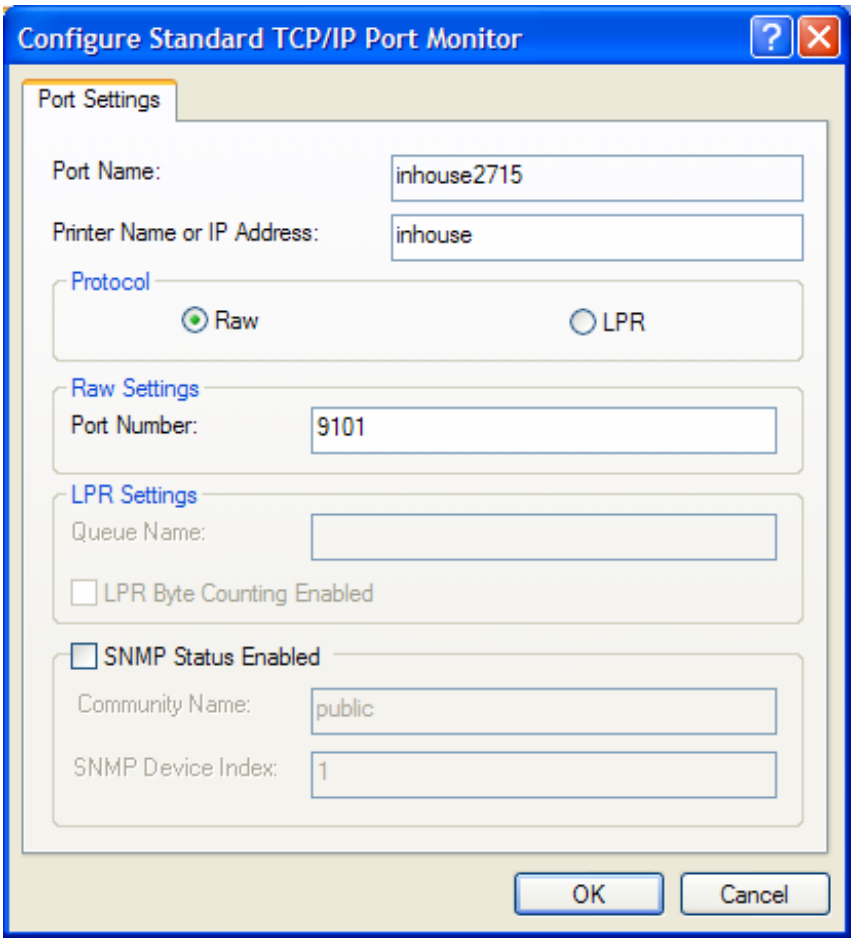
When jobs are submitted to the UnForm server in this manner, it is important to realize that the submission is one-way, and once printed the job resides entirely on the server. It is therefore not possible to print a job and have data returned to the client (i.e. –o client:*device*), or to have PDF previews generated on the submitting workstation (-p winpvw). Once the job is submitted to the TCP/IP monitor, it becomes local to the UnForm server, as if uf60c is physically run on the server (which, in fact, is what happens).

When jobs are submitted, they are dropped into the rpq/ subdirectory under the UnForm server installation. All submission files are given a unique name with a ".in" extension, and a companion file with a ".cmd" extension is also created that contains the command line options. As jobs are received, and also at least once every 5 seconds, a sweep is made of newly submitted jobs, each submitted to the

server via the server's local "uf60c" program. As a byproduct, you can drop jobs into this directory independently of the server, being careful to create the ".cmd" file first, then the associated, complete ".in" file, using your own unique naming algorithm. Note that the sweep assumes that any *.in file is a complete file and will have an associated .cmd file, so it is incorrect to open a .in file and begin writing to it, as the sweep may attempt to process an incomplete file. Instead, create the file with a different extension and then rename it when it is ready for processing.

To configure Windows printers to submit jobs to this monitor, you can use the built-in Windows support for TCP/IP printers. When configuring a printer, you can choose to Add a Port, selecting Standard TCP/IP Port. The Printer name or IP address of the "printer" will be the UnForm server, the Protocol is "Raw", and the Port Number is the number of the configured port line defined in uf60d.ini. Be sure to use the Generic/Text Only print driver when defining printers to use this port, as UnForm requires plain text input streams.

The picture below shows a Windows XP example of the configuration screen:



Note that other operating systems also support methods of supporting raw TCP/IP printers. For example, Linux contains the "jetdirectprint" script that is used by LPRng to send jobs raw TCP/IP devices.

INTEGRATING UNFORM WITH BBX

BBx handles printers via *alias* lines in a configuration file, typically called config.bbx. Printer alias lines identify a name, an output designation, a description, and several mode options. To incorporate UnForm into the configuration file on a UNIX system, you need only include an UnForm command line as part of the output designation.

BBx output designations can specify files, physical devices, or pipes, and UnForm can be installed to work with any type of definition. Note that any escape sequences configured in modes like PTON, SP, and CP are sent to UnForm and therefore need to be PCL sequences. UnForm understands how to strip a job of PCL codes, but not other printer codes. In some cases, when UnForm sends a job straight through without enhancements, these PCL sequences will also be passed on.

UNIX Aliases

A printer alias line on UNIX generally pipes to a program, such as the uf60c client program. This client program in turn can pipe its output to the spooler, or to a file, or it can instruct the server to handle the output from its end, by specifying the –o option.

Here is a sample alias line that pipes through UnForm to the local spooler:

alias P1 "|uf60c -f my.rul | lp -dxyz –oraw -s 2>/dev/null" "Printer Name" … *various modes* …

Here is a sample alias line that instructs the server to print the job to its spooler. The advantage of this type of configuration is that the client doesn't have to wait for the job to finish. It submits the job to the server and exits quickly.

alias P1 "|uf60c -f my.rul -o '>lp -dxyz –oraw'" "Printer Name" … *various modes* …

Note the use of the –oraw option in the above examples. It is important for UnForm's output to be handled as binary data by the spooler. The –oraw option is used by some UNIX spoolers, such as the SCO LaserJet model script, and the CUPS printing system. Other spoolers require different options, such as "-o-dp" for AIX, –T pcl for Unixware, -b for some older Linux installations. Check your lp configuration tools or man pages for the appropriate settings for options such as "binary", "raw", or "pass-thru" printing.

UnForm can also print directly to a device, as in this example:

alias P1 "|uf60c -f my.rul -o /dev/lp0" "Printer Name" … *various modes* …

Note that this line will behave differently with the UnForm pipe than without. When opening and sending output directly to a device, printing will occur immediately, without closing the device. However, with the pipe to UnForm, the output will not appear until the device is closed. The application may need to be modified to account for this if UnForm is to be used in this circumstance.

Windows Alias Lines

Under Windows, where pipes are not available, change the printer definition to create a file, and then use a post-processing mode, called EXECOFF, to execute UnForm with options to read the file and output to a device.

A Windows alias line will look similar to this:

alias P1 C:/TEMP/P1.TXT "UnForm Printer" CR, LOCK=C:/TEMP/P1.LCK, O_CREATE, SPCOLS=132, SP=1B451B287331362E3636481B266B3247, EXECOFF="uf60c.exe -ix C:/TEMP/P1.TXT -o *device* -f my.rul"

In the above example, a file called P1.TXT is created, using the mode O_CREATE to create the file if it doesn't exist, and using a lock file to prevent two users from writing to the same file at the same time. Note that if a file is specified with a local workstation path, such as C:\\P1.TXT, then a lock file is probably unnecessary. Just remember to specify the same path in the –ix option. Once the printer is closed by the application, the code specified by the EXECOFF mode is executed, which runs UnForm as an executable, using the P1.TXT file as input and the printer as output.

Note that pathnames containing backslashes will need double backslashes, due to the way BBx parses the command line. For example, to refer to "uf60c.exe -i c:\data\p1.txt ...", you would need to specify "uf60c.exe -i c:\\data\\p1.txt ...". You can also use forward slashes in place of backslashes, and you don't need to double them.

The *device* in the –o argument can be one of two things:

- An LPT*n* port, which can be mapped to a UNC device name with the Windows "net use" command.

- A UNC device name, defined by sharing a printer, so the name becomes //*system*/*printer*, where *system* is the system with the shared printer, and *printer* is the "share name" of that printer.

Another variety of alias line can generate a temporary PDF file and display it on the client PC, assuming you have an Adobe Acrobat Reader installed. This alias doesn't require a –o argument, but will honor it as the client-side file name for the PDF document generated. The driver selected by the –p option must be either win or winpvw, like this:

alias PUNF C:/TEMP/PUNF.TXT "UnForm Printer" CR,LOCK=C:/TEMP/PUNF.LCK,O_CREATE,SPCOLS=132, EXECOFF="uf60c.exe -ix C:/TEMP/PUNF.TXT -p winpvw -f my.rul"

Note that **the uf60c client software must be installed locally on any workstation that will execute it** to submit jobs.

INTEGRATING UNFORM WITH PROVIDEX

Simple UNIX Integration
On UNIX systems, you can integrate UnForm within the link file as the output device, and use a standard LaserJet or plain text print driver. The device used in the link file would be simply a re-direct to the uf60c program (if using ProvideX 6.0 features, a pipe (| rather than >) can be used as well), such as ">uf60c –f acme.rul –o '>lp –dhp –oraw'".

Note that this option was not available in prior versions of UnForm.

Integration using the ProvideX Print Driver uf6ptr
This method works for both UNIX and Windows environments, and provides more program control over the UnForm options when executing the uf60c client.

UnForm 6 installation includes a ProvideX print driver **uf6ptr** which should be copied to your ProvideX lib/_dev directory. This driver provides platform-independent support for UnForm, along with additional capabilities for managing UnForm command lines from the ProvideX application. In addition, it supports WindX-based output. Once copied to your ProvideX lib/_dev directory, this driver is available to use when defining ProvideX link files, which are used as printers in ProvideX.

Version 6 differs considerably from prior releases when configuring ProvideX printers to use UnForm. In prior releases of UnForm, there was a unique driver for each type of UnForm output, and the drivers often needed modification. With Version 6, there is a new universal print driver that should never need modification. All that is necessary is that the uf60c UnForm client be installed on the system where ProvideX is executed. When ProvideX and the UnForm server are installed on the same machine, the uf60c client will be available. If not, then simply install the desired UNIX or Windows client and configure the uf60c.ini file to point to the UnForm server.

To use the uf6ptr print driver:

When a link file is defined, you specify an output device and a driver program. The output device is generally something system specific, like ">lp –dhp –oraw" on UNIX, or //SERVER/PTR on Windows, or it can be a special driver name for Windows, such as *windev**, or [WDX]*windev*. In some cases, it can be /dev/null or NUL, if the driver will be directing output somewhere for the user.

The uf6ptr driver determines a default output device based upon the link file's specified output, and then re-routes the printer output to a temporary work file.

It then looks for a configuration file for additional uf60c command line parameters. This file is simply a text file named *linkfile*.unf. For example, for a link file named P1, uf6ptr will look for a file called P1.unf for additional parameters. In this text file can be one or more lines with uf60c command line options.

Once the file-based parameters have been loaded, uf6ptr then looks for the OPT value that was used in the OPEN directive, if any, for additional parameters. Any parameters named in the OPT value will override those found in the configuration file.

When all parameters have been resolved, a uf60c command line is built for execution at the end of the job. In cases where the output needs to be returned to a WindX client, the driver handles uf60c appropriately to create local output and copy that output back to the WindX PC.

Example 1:

LP is a link file pointing to device /dev/null.
LP.unf contains: -p pdf.

```
invoiceno$="00015"
OPEN(1,opt="-o /archive/"+invoiceno$+".pdf")"LP"
```

The result will be an uf60c command like this, which executes when the printer is closed:

uf60c –i *workfile* –p pdf –o /archive/00015.pdf

Example 2:

P1 is a link file pointing to device ">lp –dhp4000 –oraw".
No P1.unf file is defined

```
OPEN(1,OPT="-f acme.rul")"P1"
```

This will override the default rule file defined at the server, using acme.rul. Output will go to ">lp –dhp4000 -oraw" on the machine where the UnForm server is running. Typically this is the same machine that runs ProvideX. If it is not, add a –server *servername* option to OPT or *linkfile*.unf. In such a case, if the >lp command isn't valid locally, you will need to add a –o option to the configuration and change the link file to point to /dev/null (or NUL on Windows).

Example 3:
P2 is a link file pointing to device [WDX]*windev*.
Opening P2 will result in laser output being produced and sent to the WindX printer selected.

Example 4:
P3 is a link file pointing to device NUL.
P3.unf contains –p winpvw.
Opening P3 will cause production of a temporary PDF file. This file will automatically be viewed on a WindX client or in a Windows ProvideX session.

INTEGRATING UNFORM WITH NON-BUSINESS BASIC APPLICATIONS

UnForm is capable of interfacing with any application that can provide it with text input. On UNIX, this integration is generally performed via pipes, similar to the way it is integrated with BBx. On Windows, your application must print to a text file, and then launch uf60c.exe when the printing is complete.

If your application prints by opening a pipe to the spooler, just insert UnForm into the pipeline:

Before: |lp –dprinter –s 2>/dev/null

After: |uf60c –f *rulefile* | lp –dprinter –oraw –s 2>/dev/null

|uf60c –f *rulefile* –o '>lp –dprinter –oraw'

The second option, above, submits the job for printing on the server, while the first option will wait for the server to return the job for local printing on the client.

If your application prints to a device, such as "/dev/lp0", then you can probably modify it like this:

Before: /dev/lp0

After: >uf60c –f *rulefile* –o /dev/lp0

Note the use of the –oraw option in the above spooler examples. It is important for UnForm's output to be handled as binary data by the spooler. The –oraw option is used by some UNIX spoolers, such as the SCO LaserJet model script, and the CUPS printing system. Other spoolers require different options, such as "-o-dp" for AIX, –T pcl for Unixware, -b for some older Linux installations. Check your lp configuration tools or man pages for the appropriate settings for options such as "binary", "raw", or "pass-thru" printing.

In the case of direct device output, you will need to develop a site-specific mechanism for turning off post-processing on the device, either permanently, or while an UnForm-modified job is printing.

If your application cannot print to a pipe, or runs on Windows, then your application can be modified to print a text file, then execute UnForm when complete. Your environment may provide a way to do this automatically, such as the EXECOFF mode in Visual PRO/5 noted earlier. Here is a simple Visual Basic example of creating a file and launching UnForm:

```
open "work.txt" for output as #1
print #1,tab(35); "INVOICE"
… more printing …
```

```
close #1
if shell("uf60c.exe –i work.txt –o //server/hplaser –f rulefile",6)=0 then
        end
else
        msgbox "UnForm failed to start."
end if
```

LICENSING

UnForm is licensed based on the number of concurrent jobs it can process, with counts available as 1, 3, 5, 10, 15, 20, 25, 30, 40, 50, 75, 100, and unlimited. The UnForm Design Environment checks out a special "Designer" license, and it is available in different concurrent counts as well.

Licensing is controlled entirely by the server process, uf60d. You can install the uf60c client programs freely anywhere on your network.

Each UnForm installation has a serial number. There is one special serial number, UF0099999, reserved for demo mode use on any machine. All permanent licenses are assigned a unique serial number and must be licensed to a single machine installation. Serial numbers and their associated PIN codes are assigned by SDSI when UnForm is purchased. In order to obtain permanent or emergency temporary activation keys, the serial number and PIN code are required.

There are up to three activation keys that must be entered for full operation of UnForm: a system key, a jobs key, and a designers key. The system key enables UnForm to operate on a specific computer. The jobs and designers keys determine the number of concurrent job and design tasks that may run. For demo mode operation, just a temporary system key is required; demo mode operation automatically enables 3 jobs and 3 designers.

There are three types of system activation keys:

30-Day Demo
This license has a fixed serial number (UF0099999) and can run on any machine for 30 days. While running under this serial number, UnForm will print "Demonstration Version" phrases on any enhanced output, and will print a trailer page for each job. This is the first mode activated after an installation, as it enables the retrieval of a System ID and Machine Class needed for permanent licensing later, as well as allowing UnForm to operate in demo mode.

Permanent
This license has an assigned serial number, and requires a System ID and Machine Class to activate. A permanent license does not expire, enabling UnForm to run perpetually on the machine where installed and licensed. The System ID is derived from a given installation machine and attributes of a file in the UnForm rt\lib\keys directory (Windows) or the rt\lib directory (UNIX), so it will change if the installation is moved to a new machine, or even to a new location on the same machine. Once the System ID changes, the permanent activation key will no longer work, and UnForm must be re-activated.

If the original permanent installation of UnForm is no longer used, then you can request a reset of the permanent license to enable a new System ID and Machine Class to be associated with the permanent activation key. Contact sales@synergetic-data.com to request resets.

Emergency Temporary

This license is assigned a serial number, like a permanent license, but it does not require a System ID or Machine Class to activate. This allows you to re-install UnForm on a different machine than originally licensed, and operate it for 30 days. Once a temporary license has been issued for a given serial number, another temporary license cannot be issued for 45 days.

UNIX Licensing
To activate UnForm on UNIX, perform the following steps:

- Login as root.
- **cd** to the unform directory (i.e. cd /usr/lib/sdsi/uf60).
- Execute **./license.sh**.

The license.sh script prompts for the following options:

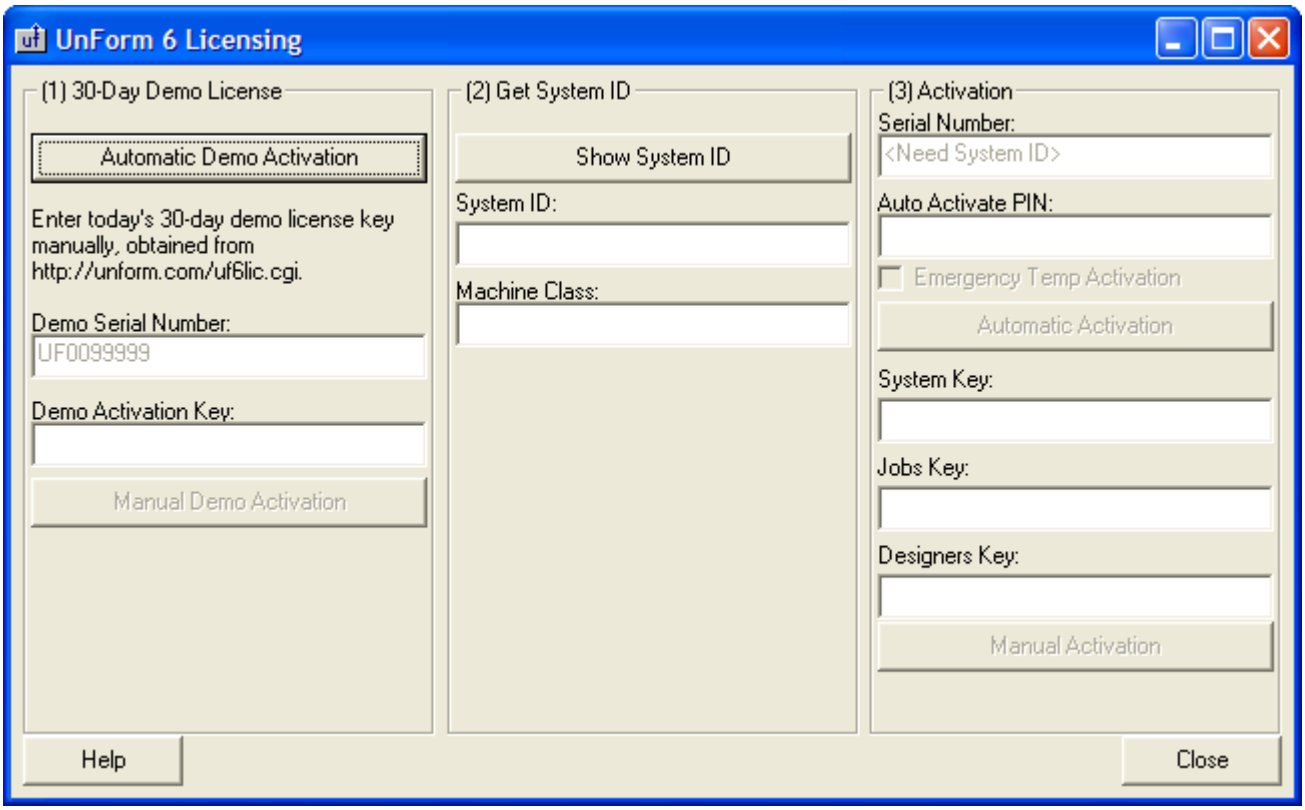
```
UNFORM LICENSING OPTIONS

Use the following options if this machine is connected to the Internet:
-----------------------------------------------------------------------
1 - Permanent Activation (requires serial number and PIN code)
2 - Emergency Temporary Activation (also requires SN and PIN)
3 - 30-Day Demo Mode Activation

Use the following options for manual activation.  Activation keys
can be obtained from http://unform.com/uf6lic.cgi.
-------------------------------------------------------------
4 - Display System ID and machine class (needed for option 5)
5 - Enter Permanent Activation
6 - Enter Emergency Temporary Activation
7 - Enter 30-Day Demo Mode Activation

q - quit
Enter selection:
```

To obtain either a permanent or emergency temporary activation, you will need to know your serial number and PIN code previously assigned by SDSI. These values are not necessary to obtain a 30-day demo mode activation.

If your machine has Internet access, you can perform activation easily by choosing options 1 through 3. Options 1 and 2 will prompt you for your serial number and PIN. Each of the three options will use the Internet to retrieve the desired activation key.

If the Internet is not available from the install machine, then you can perform activation manually by using another machine to visit http://unform.com/uf6lic.cgi. Use option 4 to display the System ID and Machine Class, which will be required to obtain a permanent activation key from this web site. Options 5 and 6 will prompt for a serial number, system key, jobs key, and designers key, in sequence. Option 7 will only prompt for a system key.

Windows Licensing



The first step after an installation is to activate demo mode. This initializes the system ID file, enabling a permanent license to be obtained. If you get an error message after pressing the Show System ID button, then this installation has never been initialized, and you must activate demo mode first.

To activate demo mode:
If you are connected to the Internet, press the Automatic Demo Activation button. This will obtain a current demo mode activation key from SDSI's website and activate the run-time engine.

If you are not connected to the Internet, go to a computer that is, and go to http://unform.com/uf6lic.cgi, then click the link to get a 30-day trial. Note the activation key returned, and enter it exactly the same way in the Demo Activation Key field, then click the Manual Demo Activation button.

To verify the activation, click the Show System ID button. If the System ID and Class fields get filled in, then it worked.

To activate permanent mode:
To activate automatically over the Internet, you need to click the Show System ID button to get the System ID and Machine Class fields. Then fill in your serial number and PIN code, and click the Automatic Activation button. This will use your information to obtain a permanent activation key for the system, as well as your job and designer activation keys, and activate everything.

To activate UnForm manually, note your System ID and Machine Class, then go to http://unform.com/uf6lic.cgi. Enter your serial number and PIN code, then click the button to get a permanent license. When prompted, enter the System ID and Machine Class exactly as noted on this screen. Note the three activation keys returned, and enter them exactly as provided in the three entry fields, then click the Manual Activation button.

To activate in emergency temporary mode:
To obtain a temporary activation over the Internet or manually, follow the steps for a permanent license, but check the Emergency Temp Activation option. The System ID and Machine Class are not used for temporary activations.

Activation Errors

Permanent activation keys are dependent on the system ID and machine class information generated by an installation. Therefore, a permanent activation key will only work on the original installation for which it was generated. If UnForm needs to be moved or re-installed, a new permanent activation key must be generated. This is only possible if SDSI resets the permanent key for your serial number, so you must contact SDSI, certify that the original installation is no longer in use, and request a reset.

In the meantime, you can obtain an emergency temporary activation to allow your serial number to be used on a new installation for 30 days.

If you attempt to get a new permanent activation key and are notified that one has already been assigned, then contact SDSI to request a reset. If this cannot be done in a timely fashion, get an emergency temporary key instead, and then contact SDSI at a later time.

Note that temporary keys are issued at most once every 45 days. If you get an error message indicating the temporary key availability has not expired, then you must contact SDSI to get a reset.

Designer Evaluation Period
In addition to the 30-day UnForm demonstration mode, if you license UnForm for jobs but not designers, an additional 30-day designer demo period is enabled. This demonstration period begins the first time a designer connects to a licensed UnForm server.

UNFORM COMMAND LINE OPTIONS

The uf60d server program can be started with the following options:

UNIX command lines	
uf60d start	Starts the server daemon.
uf60d stop	Stops the server daemon.
uf60d restart	Stops, then starts the server daemon.
Windows command lines	
uf60d.exe /install	Installs the server as a Windows Service. It will be started automatically on the next boot, and may be started and stopped manually from the Control Panel, Administrative Tools, Services applet. This option is available in the Windows Start menu.
uf60d.exe /uninstall	Uninstalls the server as a Windows Service. This option is available in the Windows Start menu.
uf60d.exe /configure	Displays the configuration window for the server. This option is available in the Windows Start menu.
uf60d.exe /a	Displays the server status window. If the server is NOT running as a service, then it can be started and stopped from this window. This option is available in the Windows Start menu.
uf60d.exe /start	Manually starts the server if it is NOT installed as a service.
uf60d.exe /stop	Manually stops the server if it is NOT installed as a service.

The uf60c client program offers many options, which control various aspects of how it communicates with the server and how the server is told to execute the job. Note that if the command line becomes too long for the operating system, you can use the –z or –zx options, which cause command line options to be read from a text file.

Standard Options	
Option	**Description**
-300	Causes UnForm to suppress 300 dpi settings within the PCL output file. Some PCL devices don't support the PCL unit of measure command, and instead include it as printed output. If this option is used, any images (dump files) or attachments must also be generated for 300 dpi and suppress any unit of measure settings.
-act	Causes UnForm to prompt for a new activation key. This is used when you change from a demonstration copy of UnForm to a live copy.
-c *copies*	Causes UnForm to issue multiple copies of the entire report. This differs from the -pc option. If *copies* is set to less than 2, this option is ignored. This option and the "-pc" option are mutually exclusive; also, rule sets can specify copy options that will override command line options.

-ci	Forces pcl image conversions to retain color rather than force black and white. See the image command for more information about automated image conversion and scaling. This also implicitly sets the –gw option.
-cmp or –compress	Either of these options causes UnForm to attempt compression of PDF output using the RLE compression algorithm. This is most effective if the report data contains repetitions of characters or spaces, and can result in PDF files that are as much as 30% smaller. Some additional processing time is used when this option is selected. You can turn on compression for individual jobs using the compress command in a rule set.
-cols *n*	Sets the default columns per page when a job is using default scaling, as when the –p pdf or –p laser options are used and no rule set is detected or specified. See also the –rows option.
-e *error-file*	Causes UnForm to output any errors to the file specified. Error files reside on the client system, not the server.
-emattach "*value*" -embcc "*value*" -emcc "*value*" -emfrom "*value*" -emlogin "*value*" -emmsgtxt "*value*" -emoh "*value*" -empswd "*value*" -emsubject "*value*" -emto "*value*"	These options supply values for an automatic email command. See the email command documentation for descriptions of each option. The –emto option is required, all others are optional, though certainly the –emsubject and -emmsgtxt are likely required for a given application. For emailing to work, the job must be a PDF job, and the server's mailcall.ini file must be properly configured with a server= line defining the SMTP server.
-exec *"launch-program*"	When the HTML output option (-p html) is used, UnForm can launch a program once the first page of output is available for viewing. The program launched must be resident on the machine where UnForm is operating. Typically this will be a Web browser, but it can be any executable program. UnForm will search the "*launch-program*" for the character "@", and substitute the file name of the HTML document produced. If no "@" symbol is present, then the file name is appended to the end of the *launch-program* value. If *launch-program* contains any spaces, it must be quoted.

-f *rule-file*	Establishes a different rule file than the default specified during the installation. Rule files are text files that contain descriptions of the form enhancements for one or more forms. The enhancement options are described in detail under Rule Files, below. UnForm will always search for the rule file first in the UnForm server directory, then by the full pathname given. Rule files must reside on the server machine, not the client. By convention, rule files have a .rul suffix, though this is not a requirement, and the *rule-file* value can be any file name. The UnForm Designer tool maintains a .rud suffix for working rule files and a .rul suffix for published rule files.
-gb|-greenbar [*options*]	Adds alternating shade patterns to simulate green bar paper. If the *options* parameter is supplied, it should be in the form defined by the shade command for repeating shade values. If no option value is supplied, the default is 3 lines shaded at 10%, 3 lines skipped, repeated until the end of the page.
-gs	Causes UnForm to generate laser driver shade regions graphically, rather than using internal PCL shade commands. The result is finer shading detail, especially at 600 dpi. Using this option will add between 2K and 4K per job. The gs command can also be used in rule sets to control graphical shading at a copy level.
-gw	Forces UnForm to pass through PCL image width and height escape sequences to the printer. This is generally necessary on color laser images to avoid a black stripe from the right image edge to the right margin. However, if you are using PCL images, then it is important that all images on a form contain width and height values so they won't conflict with one another. Some image generating programs don't store the width and height values.
-i *input-file*	Names an input text file for UnForm to process as input. If not specified, or if it is a dash (-i -), then standard input (std input) is read. Under Windows, standard input cannot be used, so an input file must be supplied. Note that the input file must reside on the client's computer, not the server.
-ix *input-file*	Same as the –i option except the input text file is removed upon completion of task. Note that the input file must reside on the client's computer, not the server.
-land	Turns on landscape print mode as the default. A portrait command in a rule set will override this option. Note that landscape printing usually requires a reduction in the number of rows per page, as compared with portrait printing, in order to produce usable results.
-macros	Turns on macros.
-macrocopy *n*	Used in conjunction with the –makemacro option. A macro will be created for the designated rule set copy.

-makemacro *n*	Causes UnForm to simply create the appropriate macro for the designated rule set and designate it as the number *n*. It must be used jointly with the –r option and can be used in conjunction with the -macrocopy option. See special section discussing macros later in this documentation.
-nn	Indicates that an error message should be issued if the input stream is empty. The value used for the error message is in the [defaults] section of ufparam.txt, in the entry nullmsg=*message text*.
-nohpgl	Reverts to full PCL, rather than a mixture of PCL and HP/GL output. A number of laser printed features use HP/GL, which is a standard feature of the PCL5 language. Some PCL interpreters, such as those that may be included in some fax or viewing software, may not support HP/GL, so this option can be used to force standard PCL5 coding for many options, such as box drawing and text alignment. A few features, such as rounded corner boxes, require HP/GL and are not supported if this option is specified.
-nointr	With this flag set, the Unix/Linux client will ignore interrupt signals once the connection to the server is established. This allows it to keep running even if a parent process receives an interrupt signal on platforms that propogate the signal to child tasks.

-o *output-file*	Specifies an output file or device. If not specified, then standard output (stdout) is used. Under Windows, an output file must be supplied unless one of the special drivers, win or winpvw, is used. On UNIX, the output can be a redirect or pipe to another program, such as lp or lpr. Output names that contain spaces must be quoted. The output file or output will, by default, be generated on the server machine. If the name is prefixed with the phrase "client:", then it is returned to the client for local handling. Here are some examples: Server output: **-o ">lp –dhplaser –oraw –s 2>/dev/null"** **-o "/tmp/archive/12345.pdf"** Client output: **-o client://prntsrv/laser** **-o client:c:\archive\12345.pdf** Note that on UNIX, if there is no –o specified, or if the output is simply a dash (-o -), then output goes to the client's standard out. A special output of /dev/tty is also recognized as client-side output to the /dev/tty device, often used for slave printing (see the –slon/-sloff options). If the output will be handled by the server, the client will generally exit as soon as the job has been successfully started on the server. If the output is to be returned to the client (or the –wait option is specified), then the client will wait for the server to finish.

-p *output-format*	Specifies the output format for the job. It may be one of the following values: **laser** (or **pcl**), which produces PCL5 or PCL5c (color) output. The default format is PCL5, but if this option is specified, and no rule set is detected or specified, then the output is scaled to fit the page in conjunction with the –cols and -rows options, or the content itself. Without any –p option, and without a rule set, the job is passed through unmodified. **pdf,** which generates files viewable by Adobe Acrobat Reader or PDF viewers. If no rule set is detected or specified, then a scaled text job is created, based on the –cols and –rows options, or the content itself. **zebra*n***, which produces ZPL II output at *n* dots per mm (6, 8, or 12 – default of 12) for Zebra label printers. For special Zebra media handling, you can append the following to **zebra*n***: • Media tracking (Y=standard, N=non-standard label stock). Standard label stock is non-continuous. NOTE: changing between standard and non-standard requires recalibrating the printer. • Set print modes (T=tear-off, R=rewind, P=peel-off, C=cutter). The default values are YT. For continuous labels, 8 dpmm, with a cutter, you would specify **–p zebra8NC**. **html,** which generates Web pages from reports, based on a special set of rule set keywords. **win, win5, winpvw**, which automatically produces a PDF file and launches the Acrobat PDF viewer on the Windows client. **win** will automatically print the document, using the Windows shell "print" option. This generally prints the document to the default printer. The **win5** option uses a command line launch of Acrobat with a /p option, which was the technique UnForm 5 used. This generally results in a printer selection dialog. Both **win** and **win5** options are dependent on the behavior of Adobe Acrobat, which can vary from version to version (and from Windows version to version). **winpvw** will provide a print preview. These options only work in Windows clients, and require both Internet Explorer and an Adobe Acrobat Reader as an Explorer plug in. Special Ghostscript-driven drivers are also available if Ghostscript is available on the server machine, and if you have configured the uf60d.ini file [drivers] section. The configuration specifies the path to Ghostscript and a set of driver names with Ghostscript sDEVICE names, a multi-page flag, and a resolution. For example: [drivers] gs=gs bmp=bmp256,1,300 If the command line contains **–p bmp –o imagefile.bmp**, then UnForm will generate an interim PDF file, and execute the gs command to convert that to the format bmp256, with output files imagefile-1.bmp, imagefile-2.bmp, and

-page *lines*	Specifies the number of lines per page that UnForm should read from the input. Normally, UnForm will find form-feed characters to delimit pages. However, if the application simply prints even numbers of lines per page, this can be used to define that value so UnForm can properly parse the input stream. The rule file page command is normally used rather than this command line option, since different reports can have different page sizes. However, this option is useful when doing cross hair prints (the -x option) to properly parse individual pages.
-paper *paper* -ps *paper*	Specifies the paper size used by the printer. Valid values include letter, legal, ledger, executive, a3, and a4. The default is letter. For a complete list, see the [paper] section of ufparam.txt. For Zebra printers, the *paper* setting is generally required, and is in the format *width*x*height*, where *width* and *height* are decimal numbers indicating width and height in inches of each label. **3.25x5.5**, for example, would define a label size of 3.25 inches wide by 5.5 inches high. The default size is 4x6.
-pc *copies*	Causes UnForm to issue multiple copies of the report, page by page. If *copies* is less than 2, this option is ignored. This option and the "-c" option are mutually exclusive; also, rule sets can specify copy options that will override command line options.
-pdfauthor "*value*" -pdfkeywords "*value*" -pdfprotect "*value*" -pdfsubject "*value*" -pdftitle "*value*"	These options supply default values for the author, keywords, protect, subject, and title commands, respectively. All options are used exclusively with PDF output.
-port *n*	Specifies the port that the server is listening on, if other than the default of 2714. The –server line can also be used to specify the port, in the format *server*:*port*. The uf60c.ini file also can contain the default port to use in the absence of this option.
-printblanks -pb	Causes UnForm to process blank pages the same as non-blank pages. Normally, blank pages are suppressed.
-prm "*parameters*"	Provides the ability for the application to send parameters to UnForm on the command line. This might be used, for instance, to pass a company number for use in a code block. The format for *parameters* is "*parameter-1=value-1[;parameter-2=value-2;...]*" Any number of parameters can be specified within the limits imposed by the operating system for command line length. Each *parameter* becomes a global string in Business Basic (use the GBL() function to retrieve), and each is set to the *value* specified. Multiple parameters need to be delimited by semi-colons (;). **-prm "company=01;name=Acme Paint"**, for example, would establish two global strings: company and name. These could be referenced within code blocks (prepage, precopy, etc.) as GBL("company") and GBL("name").
-quiet	Forces the Windows version of uf60c to route any errors to the log file defined in uf60c.ini, or “uf60c.log” by default, and to any –e file named on the command line. Without this option, errors are reported in message boxes.

-r *rule-set*	Used to specify a rule set name to use for the job. The rule set specified must exist in the rule file used for the job (see the –f option). If this option is not used, UnForm will attempt to automatically detect what form is being processed based on specifications contained in the rule file. If no form is detected, then UnForm creates a simple text job or may pass the job through to the output unmodified. If the *rule-set* contains spaces, it should be quoted. Rule set names are not case sensitive.
-rland -rport	Turn on reverse landscape or reverse portrait orientation. These options are only valid on laser output.
-rows *n*	Sets the default rows per page when a job is using default scaling, as when the –p pdf or –p laser options are used and no rule set is detected or specified. See also the –cols option.
-s *sub-file*	Specifies a text file to be used as a substitution file. Substitutions are used by UnForm when placing text in the form output. If the text can vary from one form to another, such as company names and addresses, then multiple substitution files can be defined, each containing different names and addresses, and the proper one identified with this command line option. See the **text** keyword for more information. The default substitution file is called "subst". If *sub-file* is not a full path, UnForm will look for it in the UnForm directory. UnForm will automatically generate stbl("*@name*") definitions for each line in the substitution file. Code blocks and expressions can use the stbl() function (gbl() on ProvideX) to return these values.
-server *server*	Specifies the server, if the default server found in uf60c.ini is incorrect. The *server* value can be a hostname or IP address of the system running the UnForm server, and may optionally include :*port* suffix, such as ourserver:2714. The port can also be specified with the –port option.
-shift *n*	Causes all input text to shift *n* columns to the right, similar to the action of the shift command. This can be useful in conjunction with the –x crosshair option to force text to match the alignment it would have with a shift *n* command in a rule set.

-slon "*codes*" -sloff "*codes*"	Causes local (client side) output to be started with the slon *code* and ended with the sloff *code*. This option is only supported in the UNIX client. The *code* can contain text and special escaped characters: \e Escape \n Newline \r Carriage return \0*nn* Octal character *nn* (i.e. \033 is an escape) \x*hh* Hex character *hh* (i.e. \x1b is an escape) These values are typically set in conjunction with a **–o /dev/tty** option, in order to send a job back to the client-side terminal device for slave printing. Use of these options also causes the UNIX client to attempt to change the stty setting of the –o device to "raw" for the duration of the output. A typical slave print client command line might look like this: **cat sample1.txt | uf60c –f simple.rul –slon "\e[5i" –sloff "\e[4i" –o /dev/tty**
-status -nostatus	Overrides the default behavior of the status window when submitting jobs in the Windows client uf60c.exe. The default behavior is to show the window for jobs that will be returned to the client, and not show the window for jobs that will be printed by the server.
-testpr *font symset*	Generates a test print showing nearly all characters (ASCII 1 to 254) in the *font* and *symset* codes identified. For a list of font codes and symbol sets, see the ufparam.txt file, sections [fonts] and [symsets], respectively. This option supports both laser and pdf drivers. To generate a PDF file, add "-p pdf" to the command line. Output can be sent to a file or device with the "-o" option, or on UNIX can be piped to standard output. Note that with the pdf driver, the only symbol set used is 9J.
-timeout *n*	Sets the socket timeout, for connecting to the server, to *n* seconds. If the server takes more than this amount of time to accept the connection, the client produces an error. The default value is 10 seconds.
-v	Causes UnForm to print version information and exit.
-vshift *n*	Causes all input text to shift *n* rows down, similar to the action of the vshift command. This can be useful in conjunction with the –x crosshair option to force text to match the alignment it would have with a vshift *n* command in a rule set.

-wait	Causes the client to wait for job completion, even if the server is printing the job. Normally, when the client submits a job to the server, it will exit as quickly as the server acknowledges the job has started (not, of course, if the output needs to come back to the client). By including the –wait option, the client will wait until the server job is complete, even if the output will be handled by the server. The purpose of this option is to allow client reporting of any errors the server might encounter once the job starts.
-x [*page*[,*page*, …]] -xl [*page*[,*page*, …]]	Causes the first page of input, or the pages specified, to be printed with a cross hair pattern. This is typically done once to assist in determining placement of text, and then removed. Sometimes, a special printer definition is set up within an application, using the -x option, so that any form can be printed to that printer for layout purposes. Note that setting the environment variable UFC to "y" will cause this option to be automatically implemented. Optionally, specify one or more (comma-delimited with no spaces, or hyphenated for ranges) page numbers to get UnForm to produce cross hair patterns on specific pages of the input stream. For example, '-x 1,3-5' would produce cross hair patterns on pages 1, 3, 4, and 5, suppressing all others. If the input doesn't contain form-feed page delimiters, be sure to use the –page option as well. When the –x option is used, no rule set is applied to the job. See the **crosshair** command if you want to apply a grid to enhanced output. The –xl option will produce a landscape version of the crosshair printing.
-z *filename* -zx *filename*	Adds command line options contained in the text file *filename* to the command line as if they were part of the command line itself. This option is helpful if a command line length exceeds the operating system limit. If the –zx option is used, then *filename* is erased once it has been read. The file is simply a text file with arguments separated by white space or new lines. Lines beginning with a # character are not included.
Job Status Viewing Options	

-jobs -myjobs	These options trigger the viewing of jobs submitted to the server. The –jobs option shows all jobs submitted to the server, while –myjobs shows just those jobs submitted by the current user. Job records are kept for a configurable amount of time, determined by the age= setting in the uf60d.ini file on the server. By default, the data displayed includes the job number, date/time, user, input size, pages complete, percentage complete, and status. The –detail option, below, adds the rule set, driver, and error message columns. In Windows, the jobs display in a grid. On UNIX, the jobs are displayed continuously on the display terminal, and will generally need to be processed through page filters. For example, to view a paginated display of any jobs that ended with errors: uf60c –jobs | grep 'Errored'| more
-detail	This option will cause the job listing to include additional data, including the ruleset, driver, and any ending error message.
-active	This option will limit the job display to jobs that are currently processing on the server.

VERSION 6 FEATURES

Graphical Design Tool
A new Windows-based graphical design environment is available as an optional product for Version 6. This tool is designed primarily as a rule file editor, with syntax highlighting and custom forms for editing commands, setting job properties, viewing watch lists, previewing and drawing, and test printing.

Client-Server Model
UnForm 6 has a new architecture that allows a server to be running on a Windows or UNIX network server, and client programs to be run on any computer, Windows or UNIX, that runs the user's application software.

PDF Enhancements
New rule set commands include author, keywords, protect, and subject, in addition to the existing title and outline commands. The protect option adds encryption options to the PDF file. Each of these PDF options now has a command-line equivalent, so more command-line control is available when creating PDF files in a scripted environment.

Email Enhancements
The email command has been enhanced to add cc, bcc, additional attachments, optional RFC header support, and login/password options for mail servers that require authentication. All the email command options are now also available as command line options, making it easy to use UnForm for generic report emailing in a scripted environment.

In addition to the email command, which works on a single job at a time, the new email() function can be used in a code block to send an email at any point in the job stream. The new sub-job control functions, described below, make it easy to generate sub-jobs of UnForm to create PDF files out of job fragments, which can be coupled with the email function to easily support batch emailing of PDF files.

Sub-job Support
The traditional way to manage the sub-jobs of a nested UnForm procedure involved lots of sophisticated, platform-specific code block work. In addition, it occupied a run-time user slot. Version 6 adds four code block functions to easily manage sub-jobs without any impact on the license. The functions are jobstore(), jobfile(), jobexec(), and jobclose(). They are fully documented in the Programming Code Blocks chapter.

Image Conversion and Scaling
UnForm can be configured to use an external image program to convert and/or scale images on the fly to pcl or pdf format. Two external, platform-independent image products are supported: Image Alchemy (commercial) and Image Magick (open source). In addition, the images can be cached if desired, thereby only requiring a conversion once for any given set of characteristics, though conversion time for typical images is generally very fast. See the image command for information about configuring this feature.

Image Alchemy is available from Hand Made Software, http://handmadesw.com.

Image Magick is available from http://imagemagick.com.

Ghostscript-based Drivers
UnForm can be configured to use Ghostscript to generate various formats of output from an interim PDF output file. For example, an option such as **–p tiff –o mydoc.tif** can be used to generate a tif image document. This feature also provides an ability to generate Postscript and Deskjet output for printers not normally supported by UnForm. Ghostscript can be obtained from http://www.ghostscript.com.

General Enhancements
The new boxr and cboxr commands can be used to draw rounded corner boxes.

An improved graphical shading model uses less overhead and now can be applied to shaded text as well, giving text watermarks an improved look, especially at dpi settings of 600 or 1200. In addition, graphical shading can now be turned on and off in a code block using the gs$ variable.

HP/GL has been implemented in many elements, including justified text, resulting in performance improvements in many cases.

The shade options of the box, font, and text commands now support numeric expressions.

New lcolor and scolor options are available on the box command, to provide distinct colors for lines and internal shading.

The grid options of the box command support a fourth segment value to shade the grid section with a specified color name or rgb value.

The barcode command is more flexible with regard to data lengths, so the correct symbology code of a given family is used without specifying the one that correctly matches the length.

The across and down commands have a new, optional *gap* parameter that is used to calculate inter-label gap pixels, which is helpful when formatting small labels to match a label stock.

The fit option has been added to the font command, so a given text region can be scaled to ensure it fits in a defined number of columns.

Style and weight options have been added to the font and text command, adding support for pcl fonts that require specification of a non-standard style or weight.

Added the getoffset and getcols options to the text command, for use in relative text commands (with a search string or pattern), so the data to print can be derived from the content stream. This matches a capability provided in the barcode command in Version 5.

Added the ccols option to the text command, as a complement to the cols option, the difference being that ccols specifies the ending column, while cols specifies the number of columns.

Added special handling of an input file called nul, null, or /dev/null, so that rule set output can be generated without requiring any input.

Added special handling of an input stream that starts with a rule set, so that a developer can dynamically include a rule set in a job stream, without relying on an external file.

Added a new –nn (not null) command line option, along with a configurable error message, to generate a message if the input stream is not empty.

Added a new –gb or –greenbar command line option to simulated green bar paper.

Added new behavior if the driver is specified as laser (-p laser) to automatically scale output to the page size if no rule set is detected or specified. This is similar to the behavior with –p pdf, in which a plain PDF file is generated and scaled to make the page content fit the physical page size.

Added –cols and –rows command line options to specify default columns and rows parameters for pass-through jobs.

Added –z and –zx command line options to read subsequent options from a file, providing support for long command lines that exceed operating system limits. The –zx option erases the file after it has been read.

Added a new string template (composite string) variable UF$ that contains numerous sub-values related to the command line and the UnForm environment. For example, uf.home$ contains the home directory of the UnForm server. This variable is available in code blocks and expressions, and is fully documented in the Programming Code Blocks chapter.

Added the new functions env(), parse(), parseq(), right(), left(), and sub() for use in a code block or expression. These functions are documented in the Programming Code Blocks chapter.

Added support for a configurable work file directory, using environment variables or a global string which can be defined in a prejob code block. UnForm now looks for the stbl/gbl("$tempdir"), and failing that, the UFTEMP, TMP, and TEMP environment variables, in that order.

FLOW OF PROCESSING

UnForm processes jobs in a complex but defined manner. The following list describes in general what occurs when a job is submitted:

The client program is executed with options, generally including input and output specifications, a rule file, and any other command line arguments. On UNIX, it is possible for the input and/or the output to be "standard input" and "standard output", so that the client can process jobs in a pipe. Here are a few examples:

uf60c –i sample1.txt –o ">lp –dlaser –oraw" –f acme.rul

cat sample1.txt | uf60c | lp –dmylaser –T pcl

cat sample1.txt | uf60c –p pdf >/home/mypdfs/xyz.pdf

uf60c –i sample1.txt –o client:myfile.pdf –p pdf

In all cases, the input file comes from the client and is sent to the server. With a –o option, the output normally stays on the server, though if the output designation is prefixed with "client:", then it is returned to the client. On UNIX, if "standard output" is designated, the output is also returned to the client. The rule file specified with the –f option resides on the server.

For performance reasons, it is normally desirable to specify a server-based output designation with the -o option. In that circumstance, the client only runs long enough to submit the job and ensure the command line arguments are acceptable to the server, then returns to the application. If the client will be receiving the output, it must wait for the job to finish and retrieve it, which can be time consuming (though certainly less so if the client and server are the same machine).

When the server receives the job, it stores the input in a temporary file, and calls the UnForm processor to handle the job.

UnForm reads the input file to obtain the first page. It looks for a form-feed, or if no form-feed is found, it reads the first 255 lines. It then strips the data of any PCL escape sequences in order to get a plain text array of lines. Lines must be terminated with line-feed characters (ASCII 10) or carriage-return, line-feed sequences (ASCII 13, 10).

This first page is processed against the rule file. If a –r *ruleset* command line argument was used, then the rule file is scanned for the specified rule set. Otherwise, each rule set's detect statements are tested using the first page of text. When the rule set is found, it is parsed into commands and code blocks. If no rule set is found, then the job is handled by pass-through logic, or if a rule set was specified with –r and not found, an error occurs and the job exits.

If the parsed rule set indicates a page size with the page *n* command, any excess lines read from the first page are returned to the input buffer. As the input stream is read for additional pages, UnForm will read only *n* lines per page. Note that if a form-feed character is encountered before *n* lines have been read, then the page is also considered complete.

If a prejob code block is present, it is executed.

Now processing of the job begins. Each page is processed in the following order:

- The prepage code block is executed.
- Any command expression values are resolved.
- For each copy:
    - The precopy code block is executed.
    - Command expressions are resolved.
    - Any hshift or vshift commands are executed (if shiftfirst=1 in ufparam.txt [defaults]).
    - Move commands are executed.
    - Font, bold, italic, underline, and light commands are executed.
    - Shade commands are executed.
    - Box commands are executed.
    - Text commands are executed.
    - Hline and vline commands are executed.
    - Erase commands are executed.
    - Any hshift or vshift commands are executed (if shiftfirst=0 in ufparam.txt [defaults])
    - Attach commands are executed.
    - Image commands are executed.
    - Barcode commands are executed.
    - The application text, with any font attributes applied, is added.
    - Micr commands are executed.
    - The postcopy code block is executed.
- The postpage code block is executed.
- When all pages have been processed, the postjob code block is executed.
- As the job is processed, the output designation for each copy is checked, and if the output is changed, predevice and postdevice code blocks are executed. When running a PDF job, the only time the output can be changed is in the prejob code block, or with an output command that is non-copy specific. The postdevice code block is executed after the output is complete and closed, making it suitable for handling the output file itself (for emailing, faxing, etc.).

Once the job is complete, it is available to return to the client, if the client's command line requires it. The client has monitored the job for completion in that case, and it then retrieves the job output. Note that if the rule set has overridden the output designation for the job, or part of the job, then the client will only be able to retrieve what was sent to the original output designation.

So the following scenario will conflict:

- **uf60c –i sample1.txt –o client:/tmp/invoice.pdf –f advanced.rul –r invoice**
- In the invoice ruleset is this: **output "/home/pdfs/invoice.pdf"**
- The server will send output to its /home/pdfs/invoice.pdf file, leaving the temporary output for the client empty. The client /tmp/invoice.pdf file will be an invalid, empty file.

CONCEPTS, PRIMER, AND TIPS

UnForm is a very powerful tool, with dozens of commands and features. It can be difficult to grasp the basics from such a large toolset, but the basics are really very simple. Once UnForm is installed by an administrator, the only skills required to develop typical business forms are an ability to edit text files on your system, and an ability to execute UnForm as needed to test your changes.

Here are some basic concepts that you should understand before proceeding:

- UnForm processes text input and produces formatted output. The input can come from a file or, on UNIX, can come from UnForm's standard input. The output can go to a file or a device on either the server or the client, or on UNIX can go to the client's standard output.

- UnForm uses a *rule file* to define all the form and print jobs it might process. In that rule file are one or more *rule sets*, each of which represents one form or print job. Rule files and the rule sets they contain are simply text files with command lines, which you can edit with any text editor. The rule file should be stored in the UnForm directory, and specified with the "-f *rulefile*" command line argument. If you don't specify the rule file on the command line, then the default rule file named at installation is used.

- Unless the "-r *ruleset*" command line option is used, UnForm reads the first page of input and compares that first page with all the **detect** statements found in each rule set. These statements instruct UnForm to look for text or patterns at specified locations or lines (or anywhere on the page). If all the detect statements for a given rule set match the contents of the first page, then UnForm selects that rule set and begins to produce output. If a match is not found, then the next rule set is tested, and so on until all the rule sets have been tested. If no match is found, then UnForm will pass the job through without any changes or enhancements, or in the case when a pdf or pcl driver is specified with a **–p** *driver* command line option, will produce a text job scaled to fit each page.

- Each job has its own *geometry*, that is, the basic columns and rows to which UnForm scales everything. If you specify **cols 85**, then UnForm will scale each character and all the enhancement positions and sizes to 1/85th of the printed space between the margins. In a sense, the job wraps enhancements around the text input as it is sent to the output.

- The commands in the rule set determine what enhancements are applied. These can be text additions, font changes, boxes, shade regions, barcodes, images, and more. Each change is controlled by a command line in the rule set, such as **box 5.5,2,20,4**.

  Some commands don't add output, but instead modify the text input to UnForm. The text will normally print in the Courier font, scaled to the number of columns you specify. You can change the attributes of that text in any rectangular region with **font** command, or manipulate it with the **move** and **erase** commands.

- Some commands control the printer. For example, the **tray** command can select the input tray on a laser printer, and the **bin** command can select an output bin.

- You can have UnForm generate multiple copies of each page of input. Each copy can have unique characteristics by using **if copy *n*** blocks. This is a simple structure that starts with a line "**if copy *n***", where *n* is the copy number, followed by any number of lines of enhancement commands, followed by a line "**end if**".

Creating Rule Files with the UnForm Graphical Designer

- Obtain sample output from your application for the form you want to design. Most applications provide the means to print to a text file. If no other means exists, you can define a printer that prints to UnForm with a **–debug** command line option, in which case UnForm will leave a copy of the input stream on the server, under the UnForm directory, in temp/*jobno*.in. You can find job numbers and their print times and size with the **uf60c –myjobs** command.

  Store this text file in the UnForm directory on the server.

- Start the UnForm Designer on a Windows system, and connect to the UnForm server when prompted. Create a new rule file, then a new rule set, then set the sample to the file created above. The UnForm Designer is a rule file editor with on line help, command editors, and drawing and preview capabilities. More information about using it can be found in the on line help that comes with the product.

Manual Rule Set Creation Steps

- Obtain sample output from your application for the form you want to design. This output can be printed to a text file, or you can simply use two printers defined with UnForm, one with the crosshair option (-x), the other with normal output. If you are working on a Windows system or have network access from a Windows system to the server where UnForm operates, you can use the pdf driver and an Acrobat Reader to save paper while developing the design.

- Print your sample through UnForm with the crosshair option turned on. This will provide you with a grid of text positions printed by your application. If you have a file printed by your application, the command line for a grid would look like this: uf60c –x 1-99 –i *input-file* –o *output-device* or uf60c -x 1-99 –i *input-file* | lp -d*xxx* . If your sample does not contain form-feeds, you can add a **–page *n*** option to tell UnForm how many lines are to be read per page.

- Since you will be printing this sample many times, you may wish to create a script or batch file to automate the command line, which will be something like: uf60c –i *input-file* –f *rule-file* –o *output-device* or uf60c –i *input-file* –f *rule-file* | lp -d*xxx*.

- Looking at the text of the input file, determine what makes this job unique. Sometimes there is a title, such as "PURCHASE ORDER", printed at a specific position. That may be enough to determine the uniqueness of the document so just add detect *column*, *row*, "PURCHASE ORDER". You might need to find multiple patterns by using more than one detect statement. Patterns are specified by starting the detect string argument with a ~ character. The balance of the string is a regular expression. Common syntax elements for regular expressions include "." to match any character, [0-9] to match any digit, [A-Z] to match any capital letter, and * to match any number of repetitions of the prior match character. A more complete description of regular expressions is in the Regular Expressions chapter.

  To try out your detect statement(s), try adding just those statements plus a single text command, then print the job. If your job prints with that text in addition to the text from your application, then your detect statements are working. This is what the rule set will start to look like:

  ```
  [purchase_order]
  detect 40,2,"PURCHASE ORDER"
  text 1,1,"Test Text"
  ```

  Note that it is possible to execute a rule set without detect statements, by adding "-r *ruleset*" to the command line.

- The rest of the form design is simply a matter of adding commands for text, boxes, and shade regions. It is usually best to work consistently from top to bottom, left to right in the different sections of the form. Use comments (lines starting with #) liberally; they make the rule set easier to follow when you come back later to make a change.

A good place to see complete rule sets are the sample rule files provided with UnForm, simple.rul and advanced.rul. These two files are thoroughly documented in Sample Rule Sets chapter. In addition to simple form designs, the samples show techniques with complex designs, such as jobs with multiple formats of input, and jobs that have embedded programming capabilities.

Tips and Techniques

- Always start with a crosshair pattern, so the basic text provided by the application, and its exact placement, can be seen. As the crosshair mode prints just the first page, use short versions of the reports or forms. There are several ways to create a crosshair version of a report:

  - Print the report to a file, then process that file with UnForm's command line, such as **uf60c -i *filename* -o *output-device* –x**

  - Add a printer configured with the "-x" option, and print to that printer.

  If your report doesn't contain form-feed characters at the end of the page, then you should print just

one page worth of data, or add a **–page *n*** option to the command line. Otherwise, UnForm will assume the page is made up of as many lines as are printed, up to 255 lines.

- Use **detect** statements to identify each form. UnForm is designed to process all your reports and just enhance those it can identify; all others are passed through unchanged. This is easier to set up than forcing a given printer device to be named for every form or report, as is required of most form packages.

- Specify the columns and rows for the form or report using the **cols** and **rows** commands. If this isn't done, then UnForm will assume 80 columns by 66 rows. An exception to this assumption is that if a **page** keyword is used, then the rows will be taken to be that value unless a rows command is also present.

- Remove unwanted text with the **erase** command, or move it with the **move** command. In programming code, such as in the prepage or precopy routines, you can modify the text$[] array directly or via the set() function.

- Apply attributes to the text with the **bold**, **italic**, **light**, or **underline** commands. These apply to the text generated by the application (not to text you add with the **text** keyword). Or use the **font** command, which can apply any of these attributes as well as apply other characteristics to the application text data.

- Use the **font** command to modify the font of text from the application, All text printed by the application will print in Courier unless changed with the font command. When changing to a proportional font, be sure to make the changes to specific logical regions, such as a column of prices. If you change the font for the entire page, then columns will not align properly.

- Add text, such as headings or messages, with the **text** command. Text can be literals enclosed in quotes, named values from a substitution file if prefixed with "@", environment variables prefixed by $, or an expression enclosed in { } characters. Text can be rendered at any size and in any font supported by the printer or device. Remember that fixed pitch fonts, such as Courier, are sized in characters per inch, while proportional fonts are sized in points. The larger the cpi, the smaller the font. The larger the point size, the larger the font.

- Add shading and box drawing with the **shade** and **box** commands. Reverse shading is accomplished by shading a region with 100% (black) shading, and using a **font** or **text** command to modify the text to shading of 0% gray (white). Simply using a row or column value of 1 will draw lines. To draw a box and shade the interior, use the shade option of the **box** keyword.

- Add logos and other images with the **image** command. With this command, UnForm normally looks specifically for PCL raster images (or PDF images if the pdf driver is used) in the file. UnForm can also be configured to use Image Magick or Image Alchemy for on-the-fly conversion of tradional image formats to native PCL or PDF.

- Use the **attach** command to add overlays or attachments. This command does not search only for image data. It does, however, search for and remove initialization and form-feed codes.

  Attachments should be treated as separate copies: use the **pcopies** command to allocate enough copies, then use **if copy *n*** to add the attachment, **notext** to suppress the application text output, and make sure your other enhancements don't apply to the attachment copy.

  To create an overlay, use the **attach** command, but allow the text and enhancements to also be applied on the same copy. Attachment documents for PCL output can be created using a PCL5 printer on Windows, selecting the Print to File option or setting it up to use a FILE: port. For PDF attachments, use Adobe Distiller, choosing non-optimized, ASCII output options.

- If the application doesn't use form-feeds at the end of each page, then use the **page** keyword to tell UnForm how many lines are used for each page. Many applications, especially with forms, will use just line-feeds when scrolling to the top of each form. UnForm will need to be told where the end of a page is, in this case.

Use Business Basic programming as a powerful macro language. All the data that is sent by the application to each page is available for your use. Use this data to get fax numbers and generate faxed copies, or to print shipping labels derived from the invoice ship-to addresses while packing lists are printed, or to add additional information such as costs or comments to forms, or to print logs or send email. See the precopy{} command reference, and the Programming Code Blocks chapter for more information.

RULE FILES

Rule files are text files that contain descriptions of form enhancements. There can be any number of these enhancements, called *rule sets*, in a rule file. A header line composed of a unique name enclosed in square brackets indicates a new rule set. For example, an invoice form rule set would begin with the line **[Invoice]**, followed by lines indicating enhancements to the invoice output sent by the application. Without a rule set to work with, UnForm will not perform any enhancements. UnForm determines which rule set to work with based on either a command line option (-r), or **detect** commands contained in the rule set.

The enhancements that follow the [*form-name*] line are made up of commands and (usually) a list of parameters separated by commas. The available enhancements are described on the following pages.

Unless otherwise noted, all column and row specifications are 1-based (i.e. the first column is 1, rather than 0).

Commands that have parameters accept either a space or an equal sign between the keyword and the first parameter; **page 66** and **page=66** are equivalent.

If a command and its parameters require a large amount of text, it is possible to split a command across multiple lines by adding a backslash character at the end of a line to indicate the command continues on the next line. You can have as many continuation lines as necessary. UnForm removes leading spaces and tabs from continuation lines, so you can use indention to improve readability, as long as you remember to place any required spaces before the backslash on the initial line. For example:

```
text 1,30,"This line of text is continued \
        on this line.",12,cgtimes
```

Note that the UnForm Design Tool puts continuation lines back together, so this feature is useful only when using a text editor for rule file development.

The driver differences and support for different keywords is noted. Note, however, that when a command indicates all drivers, this doesn't necessarily indicate support by html. For the HTML driver, please refer to the HTML chapter.

CONTENT-BASED RULE SETS

In addition to rule files, it is also possible to include a rule set in the content of a job, by beginning the job with a name in square brackets, like [ruleset]. If UnForm sees this line structure as the first line of a job, it then reads the input stream until it encounters a form-feed (ASCII 12, hex 0C), and then doesn't process the rule file at all. Instead, it uses the rule set provided for the job. The first character after the form-feed is treated as the start of the document, so take care that you don't have an extra line-feed that would throw off line numbers.

Using this technique, it is possible for applications such as report generators to enhance output programmatically.

ACROSS

Syntax

across *n* [,*gap*]

Description

This instructs UnForm to allocate virtual pages across the physical page, evenly spaced within the left and right margins. Use this feature for multi-up printing of standard reports, or for laser labels.

UnForm will automatically scale text (to as small as 4 point), boxes, and shading. It will not scale images, barcodes, or attachments. Also see the **down** command.

Across can be used inside an 'if copy' block, but is only compatible with non-collated copies. As a result, copy-specific **across** is only available in the laser driver, and only in conjunction with the **copies** command, not **pcopies**.

If the optional *gap* value is specified, it indicates the number of horizontal pixels between each virtual page. If it is not specified, the default is to use one *column* (as opposed to pixels).

See the 132x4 rule set in advanced.rul for an example of using the across and down commands.

Drivers: laser, pdf

ATTACH

Syntax

attach "*filename*" | {*expr*}

Description

This will add the specified file to the output. The file will be added before any other text or data for a given copy is sent to the printer, so this can work as an overlay file, or it can be placed in the output instead of any text or other output, appearing like a stand-alone attachment.

If *expr* is used, then it should be a valid Business Basic expression that resolves to a string value, which will be interpreted as the file name as each copy prints.

When used as an attachment, assign a copy to the attachment, and use the **notext** keyword to suppress printing of text, like this:

```
if copy 1
 # the standard format
 # duplexing?  add duplex 1 in this copy
 text …
 box …
 etc…
end if

if copy 2
 # the attachment
 attach "/usr/unform/attach/attach1.pcl"
 notext
end if
```

When processing the file, UnForm will remove any printer initialization codes and page ejects from the file.

The easiest way to create an attachment file is to use a Windows workstation and install a PCL5 type printer, such as the HP LaserJet III or higher. Set the port for the printer to FILE:. Then create the attachment using any word processor and print to that printer. Windows will ask for a file name, and when printing is complete, the resulting file is suitable for use as an attachment. If your document contains fonts that are not present in the printer you will be using, be sure to modify the print driver to print True Type Fonts as graphics, if possible. Also, it is sometimes necessary to use a PCL5 type driver, rather than a PCL6 driver.

To create an attachment file for the pdf driver, use Adobe Distiller, part of the Adobe Acrobat product. When using Distiller, be sure to set the job options to turn OFF the "Optimize PDF" flag, and ON the

ASCII flag. UnForm's PDF parser relies on a standard (old) PDF file format, which the optimization does not produce.

Drivers: laser (pcl format), pdf (PDF format)

AUTHOR

Syntax

author "*authorstring*" | {*expression*}

Description

If this command is present, then PDF document creation adds an author *authorstring*, or the result of *expression*, to the document content. This value is available in the General Properties Display dialog in the Adobe Acrobat Reader.

Drivers: pdf only

BARCODE (PCL,PDF)

Syntax

1. barcode *col*|{*numexpr*}, *row*|{*numexpr*},"*value*"|{*expr*},*symbology*,*height*,*spc-pixels*

2. barcode "*text*|~*regexpr*|!=*text*|!~*regexpr*[@*left*,*top*,*right*.*bottom*]", *col*|{*numexpr*}, *row*|{*numexpr*}, "", *symbology*, *height*, *spc-pixels*, getoffset *cols*, getcols *cols*, eraseoffset *cols*, erasecols *cols*

Description

col and *row* determine the upper left corner of the barcode. If used, *numexpr* is a Business Basic expression that generates a numeric value for the column or row.

value is a text string, up to 28 characters, to barcode. Often this is symbology-dependent. If check digits are required, they are generated internally in UnForm. Within barcode families, if a unique symbology is associated with a specific length, then UnForm will internally select the correct symbology. For example, if a 9-digit zip code is specified with symbology 900 (5-digit postnet), then symbology 905 will be used automatically.

expr is a Business Basic expression that generates the text to barcode.

symbology is one of the following numbers:

Code	Description
100	UPC VERSION A
105	UPC VERSION A + 2 DIGIT SUPPLEMENTAL ADD-ON
110	UPC VERSION A + 5 DIGIT SUPPLEMENTAL ADD-ON
125	UPC VERSION E
126	UPC VERSION E supporting number series 1, 6-digit input
130	UPC VERSION E + 2 DIGIT SUPPLEMENTAL ADD-ON
135	UPC VERSION E + 5 DIGIT SUPPLEMENTAL ADD-ON
150	UPC/EAN/IAN – 13
155	UPC/EAN/IAN – 8
200	INTERLEAVED 2 OF 5 – 2:1 CHECK DIGIT
205	INTERLEAVED 2 OF 5 – 2:1 NO CHECK DIGIT
220	INTERLEAVED 2 OF 5 – 3:1 CHECK DIGIT
225	INTERLEAVED 2 OF 5 – 3:1 NO CHECK DIGIT
300	STANDARD CODE 2 OF 5 – 2:1 CHECK DIGIT
305	STANDARD CODE 2 OF 5 – 2:1 NO CHECK DIGIT
320	STANDARD CODE 2 OF 5 – 3:1 CHECK DIGIT
325	STANDARD CODE 2 OF 5 – 3:1 NO CHECK DIGIT
400	CODE 39 (3 OF 9) – 2:1 NO CHECK DIGIT
405	CODE 39 (3 OF 9) – 2:1 CHECK DIGIT

410	CODE 39 (3 OF 9) – 2:1 NO CHECK DIGIT (FULL 128 ASCII)
415	CODE 39 (3 OF 9) – 2:1 CHECK DIGIT (FULL 128 ASCII)
440	CODE 39 (3 OF 9) – 3:1 NO CHECK DIGIT
445	CODE 39 (3 OF 9) – 3:1 CHECK DIGIT
450	CODE 39 (3 OF 9) – 3:1 NO CHECK DIGIT (FULL 128 ASCII)
455	CODE 39 (3 OF 9) – 3:1 CHECK DIGIT (FULL 128 ASCII)
500	CODE 93
600	CODE 128 – SERIES "A"
605	CODE 128 – SERIES "B"
610	CODE 128 – SERIES "C"
700	CODABAR – NO CHECK DIGIT
705	CODABAR – CHECK DIGIT
900	USPS Postnet – 5 DIGIT
905	USPS Postnet – 9 DIGIT
910	USPS Postnet ABC – 11 DIGIT

height is expressed in points or pixels. If it is an integer, such as 50 or 175, then it is treated as pixels at 300 dpi. If it is a floating-point number, like 18.7 or 12.0 (it contains a decimal point), then it is treated as points (1 point=1/72 inch). The maximum height is 3000 pixels.

spc-pixels is the number of pixels allocated to spacing between bars, from one to 50, the default being 2.

In syntax 2, triggered by a quoted value as the first argument, barcodes will be generated at all locations on a page where the *text* or the regular expression *regexpr* occurs. The value(s) to barcode will be based upon what text matches occur. Each match will determine the value to barcode based on the word found (up to the first space or the end of the line), and the placement of the barcode. The value to barcode can be adjusted by the getoffset *cols* (integer columns from the location of the match) and getcols *cols* (number of columns to use for the value). The location of the barcode can be adjusted by the *col* and *row* parameter, where 0,0 is the location where the match is found. The match text found can be erased from the report by setting eraseoffset *cols* and erasecols *cols*.

If the syntax "!=*text*" or "!~*regexpr*" is used, then the search is for positions NOT equal to the text or NOT matching the regular expression. When using the NOT syntax, only one search is performed per line in the search region.

The search for *text* or *regexpr* can be limited to a region on the page by adding a suffix in the format '*@left,top,right,bottom*'. To use a literal "@" character in *text* or *regexpr*, it is necessary to specify "\@".

Version 5 Note: The positioning algorithm for PDF versions of the barcode was modified in Version 5 to match the positioning of laser barcodes. If your application depends on this older algorithm, then you can modify your ufparam.txt file (preferably by copying it to ufparam.txc and then modifying that file, to avoid losing your changes during an update) to add (or change) 'v4pdfbcd=1' in the [defaults] section.

Drivers: laser, pdf

Examples:

barcode 10.5,22,{get(10,21,5)},900,12.0,2 will add a 12.0 point high, 5-digit postnet barcode based on a zip code found at column 10, row 21.

barcode "bcd:@16,22,20,55",0,0,"",600,75,2, getoffset 4, getcols 10, erasecols 14 will search for data starting with "bcd:" in the region starting at column 16, row 22, through column 20, row 55, barcode the 10 characters following it, and erase the underlying text.

BARCODE (ZEBRA)

Syntax

barcode *col*|{*numexpr*}, *row*|{*numexpr*}, ("*value*" | {*expr*}), *symbology*, *height*, *spc-pixel*s, text [above|yes|no], rotate [90|180|270], ratio *rvalue*, checkdigit, start *startc*, stop *stopc*, ucc, mode *m*, security *s*, cols *c*, rows *r*

Description

col and *row* define the upper left corner of the barcode. If used, *numexpr* is a Business Basic expression that generates a numeric value for the column or row.

value is a literal value to barcode, *expr* is a Business Basic expression that generates the text to barcode.

symbology is one of:

Symbology Code	Name
1	Code 11
2	Interleaved 2 of 5
3	Code 39
8	EAN-8
9	UPC-E
A	Code 93
C	Code 128
E	EAN-13
I	Industrial 2 of 5
J	Standard 2 of 5
K	ANSI Codabar
L	LOGMARS
M	MSI
P	Plessey
S	UPC/EAN extensions
U	UPC-A
Z	Postnet
4	Code 49
7	PDF417
B	CODEABLOCK
D	UPS Maxicode

For Maxicode, you may specify a *mode* of 2 for UPS US addresses, 3 for UPS non-US addresses, or 4 for non-UPS coding (the default is 2). The data must consist of 2 segments:

Segment 1:

- Mode 2: 3-digit class of svc, 3-digit country code, 9-digit zip code
- Mode 3: 3-digit class of svc, 3-digit country code, 6-character zip code

Zebra requires this segment; the remaining segment format is specified by UPS.

Segment 2:

- Data content as required by UPS, starting with the "[)>"+$1E$ header.

For modes other than 2 or 3, segment 2 can contain variable content.

height is either an integer, interpreted as the number of pixels, or a decimal number, such as 20.0 or 40.6, interpreted as points (1/72 inch).

spc-pixels is the narrow bar width in pixels, from one to 10, defaulting to 2.

Following spc-pixels, the options can be in any order.

Rotate will rotate the barcode the given number of degrees.

Ratio will modify the wide bar to narrow bar ratio, from 2.0 to 3.0 in 0.1 increments. The default ratio is 2.0. Some symbologies have fixed ratios.

text or **text yes** will print the human readable value below the barcode. **text above** (or just **above**) will print this value above the barcode.

text no will not print the value, even if that is the default for the given symbology.

checkdigit will cause a checkdigit to be calculated and printed by the printer.

start *char* will set the start character, if used by the symbology.

stop *char* will set the stop character.

ucc will set the UCC Case Mode on code 128 barcodes.

mode *m* will set the mode code, which is symbology dependent. The UCC Case Mode may be set for code 128 with 'mode U'. The code 49 mode can be A for auto, or 0-5 as defined in the ZPL programmers' guide.

security *n* well set the security and/or error correction level for the PDF417 bar code. *n* can be a digit from 0 to 8.

cols *c*, **rows *r*** will set the cols and rows values for the PDF417 barcode. If not set, this barcode will assume a 1:2 row to column aspect ratio. *c* can range from 1 to 30, *r* from 3 to 90, and the product of *c* x *r* can't exceed 927.

Drivers: zebra only

BIN

Syntax

bin *bin-number*

Description

The **bin** keyword is used to specify the output bin for any copy. Larger, departmental laser printers often have two or more bins, allowing print job output to be separated. In UnForm, you can specify a bin for each copy, or for the whole job.

bin-number is printer-specific, with one generally being the top, face-down bin, and 2 being a side or rear face-up bin. Some models may offer additional bins; see your printer's documentation for additional bin codes.

Drivers: laser only

BOJ, BOP, EOJ, EOP

Syntax

1. {boj | bop | eoj | eop} *hex codes*
2. {boj | bop | eoj | eop}"*text string*"

Description

These keywords provide the ability to add escape codes to the beginning of the job (after the printer is initialized but before any data prints), before each page of each copy, after each page of each copy, and after the job ends, just before the printer is re-initialized.

The escape sequences can be entered either as hex codes, such as 1b28633045 (interleaved with spaces if desired), or as a text string. To enter a text string, the value must be quoted.

When entering a text string, it is possible to include non-printable characters with angle bracket notation, such as "<27>&k10G", where "<27>" is used to include an escape character.

UnForm will normally provide all the control needed for a job. These keywords are included to handle unusual requirements, such as perhaps adding PJL coding to a job for special paper handling requirements.

Examples:

This example shows adding PJL codes to a job, setting the title to "Title Of Job".

boj "<27>%-12345X@PJL<10>@PJL JOB NAME=<34>Title Of Job<34><10>@PJL ENTER LANGUAGE=PCL<10>"

Drivers: laser only

BOLD, ITALIC, LIGHT, UNDERLINE

CBOLD, CITALIC, CLIGHT, CUNDERLINE

Syntax

1. bold|italic|light|underline *col*|{*numexpr*}, *row*|{*numexpr*}, *cols*|{*numexpr*}, *rows*|{*numexpr*}

2. bold|italic|light|underline "*text*|!=*text*|~*regexp*|!~*regexp*[*@left,top,right.bottom*]", *col*|{*numexpr*}, *row*|{*numexpr*}, *cols*|{*numexpr*}, *rows*|{*numexpr*}

If **cbold**, **citalic**, **clight**, or **cunderline** is used, then *columns* and *rows* are interpreted to be the opposite corner of the region, and columns and rows are calculated by UnForm.

Description

The region indicated by the *col*, *row*, *cols*, and *rows* parameters will have the indicated attribute (**bold**, **italic**, **light**, **underline**) applied. All text in the input within that region, but not text generated by **text** keywords, will be affected. If used, *numexpr* is a Business Basic expression that generates a numeric value for the column, row, columns, or rows.

If syntax 2 is used, then the region is defined relative to any occurrence of the *text*, or of text that matches the regular expression *regexpr*. In these cases, there may be no affected regions, or several. c*olumn* and *row* are 0-based in these formats. The search for *text* or *regexpr* can be limited to a region on the page by adding a suffix in the format '*@left,top,right,bottom*'. To use a literal "@" character in *text* or *regexpr*, it is necessary to specify "\@".

If the syntax "!=*text*" or "!~*regexpr*" is used, then the search is for positions NOT equal to the text or NOT matching the regular expression. When using the NOT syntax, only one search is performed per line in the search region.

Note that the **font** command is a more powerful alternative to these commands, and it also offers support for fonts that support specific weights or styles other than these.

Examples:

bold 1,5,30,4 bolds a region from column 1, row 5, for 30 columns and 4 lines.

underline "TOTAL:",0,0,36,1 underlines a region beginning at a position where the text "TOTAL:" is found, extending for 36 columns. If "TOTAL:" isn't found, the keyword is ignored until the next page is analyzed.

Drivers: laser, pdf. **underline** and **light** is supported on laser only. Not all pcl fonts support the **light** and **bold** options.

BOX, CBOX

Syntax

1. box *col*|{*numexpr*}, *row*|{*numexpr*}, *cols*|{*numexpr*}, *rows*|{*numexpr*} [,*thickness]* [,*shade*] [,*color*] [,rgb *rrggbb*] [,dbl|double [*gap*]] [,left *l*] [,right *r*] [,top *t*] [,bottom *b*] [,icols=*gridcols*] [,irows=*gridrows*] [,ccols=*gridcols*] [,crows=*gridrows*] [,lcolor=*color*] [,lcolor rgb=*rrggbb*] [,scolor=*color*] [,scolor rgb=*rrggbb*]

2. box "*text*|!=*text*|~*regexp*|!~*regexp*[*@left,top,right.bottom*]", *col*|{*numexpr*}, *row*|{*numexpr*}, *cols*|{*numexpr*}, *rows*|{*numexpr*} [,*thickness]* [,*shade*] [,*color*] [,rgb *rrggbb*] [,dbl|double [*gap*]] [,left *l*] [,right *r*] [,top *t*] [,bottom *b*] [,icols=*gridcols*] [,irows=*gridrows*] [,ccols=*gridcols*] [,crows=*gridrows*] [,lcolor=*color*] [,lcolor rgb=*rrggbb*] [,scolor=*color*] [,scolor rgb=*rrggbb*]

If **cbox** is used, then *columns* and *rows* are interpreted to be the opposite corner of the box, and columns and rows are calculated by UnForm.

Description

A box of the indicated dimensions will be drawn. All dimensions can be specified to 2 decimal places, in the range of -255 to +255. Whole number *col* and *row* represent center points; lines are drawn to the center point of the character position identified in order to facilitate connections between lines. This differs from the **shade** keyword, which shades full character cells. It may be easier to use the **box** keyword's shade parameter than to calculate shade positions that are offset from similar box parameters. To draw lines rather than boxes, simply set the *cols* or *rows* to 1. If both *cols* and *rows* are 1, then a vertical line is drawn 1 character high. To draw a box that is 1 column wide or 1 row deep, use 1.01 or .99. If used, *numexpr* is a Business Basic expression that generates a numeric value for the column, row, columns, or rows.

If syntax 2 is used, then the box is drawn relative to any occurrence of the *text*, or of text that matches the regular expression *regexpr*. In these cases, there may be no boxes drawn, or several. *column* and *row* are 0-based in these formats and can be negative if required. The search for *text* or *regexpr* can be limited to a region on the page by adding a suffix in the format '*@left,top,right,bottom*'. To use a literal "@" character in *text* or *regexpr*, it is necessary to specify "\@".

If the syntax "!=*text*" or "!~*regexpr*" is used, then the search is for positions NOT equal to the text or NOT matching the regular expression. When using the NOT syntax, only one search is performed per line in the search region.

Line Thickness
The optional *thickness* parameter may be a number from 1 to 99, indicating the number of dots or pixels to use when drawing the box outline. The default thickness is 1 dot. UnForm always uses dots at 1/300 inch. If a shade parameter is desired, then the thickness parameter is required.

The left, right, top, and bottom options override the specified *thickness* for any given side of the box. Setting "left 0", for example, would erase the left side of the box, while "right 4" would set the right side to 4 pixels wide.

The double or dbl option indicates a double-lined box. Both the inner and outer lines will be drawn at the normal thickness, and the optional *gap* may be specified to set the pixels between each line. The default *gap* is 1 pixel. The *gap* must be a digit between 1 and 9.

Shading
The optional *shade* parameter may be used to specify a "percent gray" value from 1 to 100. Most laser printers can only print about 8 different shades of gray, so a value of 45, for example, may print the same pattern as 50. Note that if you specify a shade level of 0, this differs from not specifying any shade at all: a shade level of 0 will force a white interior, even if another box or shade command draws shading inside the bounds of the box. If an interior color is specified, shading is ignored.

Color
Color can be specified as "white", "cyan", "magenta", "yellow", "blue", "green", "red", or "black", or you can name an RGB value as a 6-character hex string with "rgb *rrggbb*", where *rr* is red (00-FF), *gg* is green (00-FF), and *bb* is blue (00-FF). To distinguish colors between the line and the shade region, use "lcolor" or "lcolor rgb" for lines, and "scolor" or "scolor rgb" for shade.

Grids
The *gridcols* and *gridrows* settings are used to draw grid lines and/or shade regions inside the box. *gridcols* specifies one or more vertical column settings in the structure of *column*[:*thickness*[:*shade*[:*color*|*rrggbb*]]]. Multiple columns can be delimited by any character other than digits, the decimal point (.), and the colon. Each column designates a vertical line to draw from the top to bottom edges of the outer box. If a thickness is specified, then the line is drawn using that thickness (0 would draw no line at all). The default thickness is 1. If shade is specified, then a shade region is drawn from the left edge or prior column. *gridrows* is identical in structure to *gridcols*, but specifies the horizontal rows rather than vertical columns. The "icols" and "irows" introducers indicate columns and rows relative to the upper-left corner of the outer box. The "ccols" and "crows" introducers indicate absolute columns and rows. In each case, any column or row specification outside the bounds of the box is ignored.

For partial shading, partial color shading, or multiple color shading, see the **shade** keyword. You can improve the look of shade regions on laser printers, especially at medium shade levels and 600 or higher dpi settings, by using the **gs** command.

Examples:

box 5.5,2.5,34,3,2,10 will draw a box 34 columns wide and 3 lines high, at column 5.5, line 2.5. The box border will be 2 dots wide (1/150 inch). It will be filled with 10% gray shading.

box 1,1,55,1 will draw a horizontal line, 55 columns wide, at column 1, line 1.

box "Customer Total",-1,-1,60,3 will draw a box around the text "Customer Total", beginning 1 column before and 1 row up, for 60 columns and 3 rows.

cbox 12,{start_row-.5},40,{end_row+.5} will draw a box with the top and bottom lines based on two numeric variables, which would have been previously calculated in a prepage or precopy code block. In using the **cbox** version, the second pair of numbers indicates the lower-right corner, rather than the number of columns and number of rows. The code block used to calculate these positions might look something like this code, which finds the first and last rows that contain any data in the row range of 22 through 55:

```
prepage{
start_row=0,end_row=0
for line=22 to 55
  if trim(text$[line])>"" then if start_row=0 then start_row=line
  if trim(text$[line])>"" then end_row=line
next line
}
```

cbox .5,22,80.5,66,3, ccols=10.5 30 55.5 67.5, crows=23.25:1:20 60 will draw a box from column 0.5, row 22 through column 80.5, row 66. The lines of this outer box will be 3 pixels wide. Inside this box will be vertical lines at columns 10.5, 30, 55.5, and 67.5. Also inside the box will be a 1 pixel high horizontal line at row 23.25, with 20% shading from row 22 to row 23.25, and another 1 pixel horizontal line at row 60.

Drivers: all (*gridcols* and *gridrows* options supported only in laser and pdf)

BOXR, CBOXR

Syntax

1. boxr *col*|{*numexpr*}, *row*|{*numexpr*}, *cols*|{*numexpr*}, *rows*|{*numexpr*} [,*thickness]* [,*shade*] [,*color*] [,rgb *rrggbb*] [,tl=*topleft*] [,tr=*topright*], [,bl=*bottomleft*], [,br=*bottomright*] [,icols=*gridcols*] [,irows=*gridrows*] [,ccols=*gridcols*] [,crows=*gridrows*] [,lcolor=*color*] [,lcolor rgb=*rrggbb*] [,scolor=*color*] [,scolor rgb=*rrggbb*]

2. boxr "*text*|!=*text*|~*regexp*|!~*regexp*[*@left,top,right.bottom*]", *col*|{*numexpr*}, *row*|{*numexpr*}, *cols*|{*numexpr*}, *rows*|{*numexpr*} [,*thickness]* [,*shade*] [,*color*] [,rgb *rrggbb*] [,tl=*topleft*] [,tr=*topright*], [,bl=*bottomleft*], [,br=*bottomright*] [,icols=*gridcols*] [,irows=*gridrows*] [,ccols=*gridcols*] [,crows=*gridrows*] [,lcolor=*color*] [,lcolor rgb=*rrggbb*] [,scolor=*color*] [,scolor rgb=*rrggbb*]

If **cboxr** is used, then *columns* and *rows* are interpreted to be the opposite corner of the box, and columns and rows are calculated by UnForm.

Description

A box with rounded corners of the indicated dimensions will be drawn. All dimensions can be specified to 2 decimal places, in the range of -255 to +255. Whole number *col* and *row* represent center points; lines are drawn to the center point of the character position identified in order to facilitate connections between lines. If used, *numexpr* is a Business Basic expression that generates a numeric value for the column, row, columns, or rows.

If syntax 2 is used, then the box is drawn relative to any occurrence of the *text*, or of text that matches the regular expression *regexpr*. In these cases, there may be no boxes drawn, or several. *column* and *row* are 0-based, in these formats, and can be negative if required. The search for *text* or *regexpr* can be limited to a region on the page by adding a suffix in the format '*@left,top,right,bottom*'. To use a literal "@" character in *text* or *regexpr*, it is necessary to specify "\@".

If the syntax "!=*text*" or "!~*regexpr*" is used, then the search is for positions NOT equal to the text or NOT matching the regular expression. When using the NOT syntax, only one search is performed per line in the search region.

Line Thickness
The optional *thickness* parameter may be a number from 1 to 99, indicating the number of dots or pixels to use when drawing the box outline. The default thickness is 1 pixel. UnForm always uses dots at 1/300 inch. If a shade parameter is desired, then the thickness parameter is required.

Corner Rounding
To specify the degree of rounding for different sides, specify values for tl, tr, bl, and br, as desired. The specification for each corner is *col*:*row*:*scale*, where *col* is the number of columns from the corner to

begin the rounding, *row* is the number of rows from the corner to begin rounding, and *scale* is the level of rounding, from –100 for fully convex, to 100 for fully concave, where 0 becomes a straight line from the column and row break points. If no rounding options are specified at all, then UnForm will apply default rounding to all four corners. If any rounding is specified, then any unspecified corners become square corners.

Shading
The optional *shade* parameter may be used to specify a "percent gray" value of from 1 to 100. Most laser printers can only print about 8 different shades of gray, so a value of 45, for example, may print the same pattern as 50. Note that if you specify a shade level of 0, this differs from not specifying any shade at all: a shade level of 0 will force a white interior, even if another box or shade command draws shading inside the bounds of the box.

Color
Color can be specified as "white", "cyan", "magenta", "yellow", "blue", "green", "red", or "black", or you can name an RGB value as a 6-character hex string with "rgb *rrggbb*", where *rr* is red (00-FF), *gg* is green (00-FF), and *bb* is blue (00-FF). To distinguish colors between the line and the shade region, use "lcolor" or "lcolor rgb" for lines, and "scolor" or "scolor rgb" for shade.

Grids
The *gridcols* and *gridrows* settings are used to draw grid lines and/or shade regions inside the box. *gridcols* specifies one or more vertical column settings in the structure of *column*[:*thickness*[:*shade*[:*color*|*rrggbb*]]]. Multiple columns can be delimited by any character other than digits, the decimal point (.), and the colon. Each column designates a vertical line to draw from the top to bottom edges of the outer box. If a thickness is specified, then the line is drawn using that thickness (0 would draw no line at all). The default thickness is 1. If shade is specified, then a shade region is draw from the left edge or prior column. *gridrows* is identical in structure to *gridcols*, but specifies the horizontal rows rather than vertical columns. The "icols" and "irows" introducers indicate columns and rows relative to the upper left corner of the outer box. The "ccols" and "crows" introducers indicate absolute columns and rows. In each case, any column or row specification outside the bounds of the box is ignored.

For partial shading, partial color shading, or multiple colors shading, see the **shade** keyword. You can improve the look of shade regions on laser printers, especially at medium shade levels and 600 or higher dpi settings, by using the **gs** command.

Examples:

boxr 10,9.5,70,4.25,2,5,lcolor=blue will draw a box with default rounding on all corners, with a 2 pixel edge and 5% shading. The edge line will be drawn in blue if the output device supports color.

cboxr 0.5,60,80.5,66,1,0,bl=3:1.5:75,br=3:1.5:75 will draw a box with corners 0.5,60 and 80.5,66, with a 1 pixel border, no shading, and just the bottom left and right corners rounded. The rounding will start 3 columns and 1 row from the corners, and be rounded outward.

Drivers: laser, pdf

COLS

Syntax

cols *n*

Description

This keyword specifies the number of columns to use for the form or report. The base font is scaled to accommodate this many columns. If present, this value will override any calculation based on the **cpi** keyword.

The number of columns *n* can be any value up to 255.

Examples:

cols 80 will set the print pitch to accommodate 80 columns per page.

Drivers: all

COMPRESS

Syntax

compress

Description

If this command is present, then PDF output is compressed using the RLE compression algorithm. This is most effective when repeated characters like spaces are present in the output, such as wide reports with empty space between columns. Pdf output can be reduced by as much as 30%, though in some jobs there may be little or no change. Compression requires extra processing and will therefore affect performance.

Compression can also be turned on with the –compress command line option.

Drivers: pdf only

CONST

Syntax

const *ID=value*

Description

The **const** keyword provides the capability to use a named value as a parameter to other keywords. If, for example, you want to place a series of text values at a certain column position, but may need to adjust the position in the future, and then set a constant *ID* to the column position *value*, then use the *ID* in the column position of all the text values.

```
const COLPOS=22.25
text COLPOS,30,"Text line 1"
text COLPOS,31,"Text line 2"
text COLPOS,32,"Text line 3"
```

A given constant ID can be reused, and references to it in subsequent rule set lines will reflect the new value. Also, a constant defined before the first rule set in the rule file will apply to any rule sets in the file, unless the same ID is reused in any particular rule set.

Note that case does make a difference. "COLPOS" and "colpos" are different constants. Take care not to use constant names that may inadvertently cause unintended replacements. For example, it may be tempting to use a constant named "font", but this would conflict with any font command. There would be no conflict, however, between a constant named FONT and a lower-case font command.

Constant names are limited to 25 characters, and constant values are limited to 75 characters. If you use a quoted value, the outer quotes are removed before the value is substituted into the rule file commands.

Drivers: all

COPIES, PCOPIES

Syntax

copies *copies*
pcopies *copies*

Description

These keywords are used to generate multiple copies of the form. The number of copies is specified by the number *copies*. If the **copies** form is used, then the entire print job is duplicated the number of times indicated. If the **pcopies** form is used, then each page is duplicated as it is printed, so the pages come out collated.

The two versions of this keyword are mutually exclusive; the last one that is found in the rule set is the one used. Note also the **-c** and **-pc** command line options can be used, though these keywords take precedence, if specified.

Individual copies can be managed to any degree necessary via "if copy *n*" rule set logic, and also full programming logic with the "precopy {}" and "postcopy {}" logic entry points. Use this to modify the output device for specific copies, or to modify the content of specific copies.

To add attachments that are separate pages from the standard form pages, assign a copy to the attachment, and add a **notext** keyword for that copy.

```
copies 2

if copy 2
notext
attach "/usr/unform/attachments/attach1.pcl"
end if
```

Examples:

copies 2 will print the entire report twice.

pcopies 3 will print each page three times.

Drivers: all, pdf driver treats copies as pcopies

CPI

Syntax

cpi *characters-per-inch*

Description

The **cpi** keyword indicates what pitch UnForm should use when printing the text of a form or report. From this, along with the paper dimensions, UnForm can determine the columns per page and ensure that the proper pitch is selected. As UnForm uses **cpi** to calculate a **cols** value, **cpi** values are rounded to allow even character spaces. It is advisable to use **cols** rather than **cpi**.

See also **lpi**, **cols**, **rows**.

Examples:

cpi 16.66 will set the character spacing to a common "compressed" character pitch.

Drivers: laser, pdf, zebra

CROSSHAIR

Syntax

crosshair

Description

If this command is present in a rule set, then UnForm will generate a crosshair grid over the page, making rule file development easier. Crosshair mode can also be turned on from a code block with the crosshair$ variable.

Drivers: laser, pdf

DETECT

Syntax

detect *column(s),row(s),*"[^[!]]*text"*
detect *column*(s),*row(s),*"[^[!]]~*regexpr*"

Description

This option is used to identify a form from the data read by UnForm. If the **-r** option is used on the UnForm command line, then **detect** keywords are ignored. Otherwise, each rule set's detects are analyzed until a match is found. If more than one **detect** keyword is specified for a rule set, then the job must match all of them. Detection occurs only at the start of the job, using the first page of data read from the input stream.

If *column* and *row* are 0, then the whole page is scanned for the occurrence of the text. If *column* is 0 and *row* is greater than 0, then the whole line is scanned.

column and *row* can contain ranges in the format *from-through*, such as '20-25' for the columns (or rows) 20 through 25.

The format of the quoted third parameter determines how the detection scan is handled. If plain text is specified, then a literal match for *text* is performed. If the text begins with the prefix character ~, then a regular expression search for *regexpr* is performed.

If the text begins with ^, then a case insensitive match is performed.

Following the optional ^ character, but before the ~ character, may be a ! character, indicating a scan for NON-matches.

The following prefix sequences are valid: ^, ^~, !, !~, ^!, ^!~, meaning, respectively: case insensitive text, case insensitive regular expression, text not found, regular expression not found, case insensitive text not found, case insensitive regular expression not found.

Examples:

detect 0,2,"INVOICE" would search for INVOICE anywhere on line 2.

detect 10-12,4,"~../../.." would match a date format at column 10, 11, or 12, on row 4.

detect 65-66,6-8,"!~../../.." would match a date format NOT occurring at column 65 or 66, on rows 6 through 8.

detect 0,2-3,"^invoice" would match INVOICE, Invoice, invoice, etc. anywhere on lines 2 or 3.

Drivers: all

DOWN

Syntax

down *n* [,*gap*]

Description

This instructs UnForm to allocate virtual pages down the physical page, evenly spaced within the top and bottom margins. Use this feature for multi-up printing of standard reports, or for laser labels.

UnForm will automatically scale text (to as small as 4 point), boxes, and shading. It will not scale images, barcodes, or attachments. Also see the **across** command.

Down can be used inside an 'if copy' block, but is only compatible with non-collated copies. As a result, copy-specific **down** is only available in the laser driver, and only in conjunction with the **copies** command, not **pcopies**.

If the optional *gap* value is specified, it indicates the number of vertical pixels between each virtual page. If it is not specified, the default is to use 1 *row* (as opposed to pixels).

See the 132x4 rule set in advanced.rul for an example of using the across and down commands.

Drivers: laser, pdf

DPI

Syntax

dpi 300 | 600 | 1200

Description

The **dpi** keyword instructs PCL printers to print at the specified dots per inch. The default dpi value is 300; however, many printers are capable of printing at 600 or 1200 dpi (or possibly even higher values). This takes more printer memory, but results in crisper characters and lines.

Drivers: laser only

DSN_SAMPLE

This command is used exclusively by the UnForm Designer tool, to store the name of a sample text file to apply to previews generated in the design environment.

DUMP

See the **image** command.

DUPLEX

Syntax

duplex *mode* [, *left-offset*] [, *top-offset*]

Description

Duplex printing, if supported by your printer, causes printing on both sides of the paper.

mode can be 1 for long-edge binding, or 2 for short-edge binding. A *mode* of 0 will print in simplex (single-sided) mode.

left-offset and *top-offset* are optional values in decipoints ($1/720^{th}$ inch) that indicate how far to shift the page printing from the left and top edges, respectively. Note that margins may need to be adjusted (with the **margin** keyword) if offsets are used.

Note that any duplex command will cause a page eject on a laser printer, so timing of the duplex command is important. For example, if you use pcopies 2, and the second reserved for a back side attachment, the duplex command should be in the 'if copy 1' block. This forces copy 1 to be on the front side and copy 2 to follow on the back side. This concept is shown in the example below.

Examples:

```
pcopies 2
if copy 1
 duplex 1
 # complete form for front of page
end if
if copy 2
  # attachment for back of page
  notext
  attach "terms.pcl"
end if
```

Drivers: laser

EMAIL

Syntax

email { *to* | {*toexpr*} }, { *from* | {*fromexpr*} }, { *subject* | {*subjectexpr*} }, { *msgtxt* | {*msgtxtexpr*} } [,cc "*cc*"|{*ccexpr*}] [,bcc "*bcc*"|{*bccexpr*}], [,attach "*attach*"|{*attachexpr*}] [,otherhead|oh "*otherhead*"|{*otherheadexpr*}] [,login "*login*"|{*loginexpr*}] [,password|pswd "*password*"|{*passwordexpr*}]

Description

The PDF document being created will be emailed as an attachment upon completion, using the information supplied. The name of the attached file is supplied with the "-o" argument on the UnForm command line, or can be overridden by setting the variable output$ in a prejob code block.

Each of the first 4 values is positional, and each can be a literal value or an expression enclosed in curly braces. The *to* value is the only required value, and must be a fully qualified email address, or a comma-separated list of email addresses. The *from* value, if supplied, must also be a fully qualified email address. If it is not supplied, then a default address will be used from the mailcall.ini file.

Note that the expressions are resolved as of the last copy of the last page of the job. If you need to use data from an initial page, use a prejob code block to assign variables, and then use those variables in the expressions.

In order to use this command, the mailcall.ini file must be edited to configure a mail server (server=*value*) line. See the Email Integration chapter for more detail about configuration, and also for information about using direct calls to the MailCall program bundled with UnForm. Direct calls enable more control over email processing.

The *msgtxt* value can contain line-feed characters to break lines. These characters can be added in expressions as CHR(10) functions or as $0A$ hex literals, or with the literal backslash-n (\n) character sequence. Note that if the message text starts with a structure "<*value*>", then it is assumed to be an HTML message, and the appropriate header tag is set to send the message as HTML.

Optional arguments can follow the message text value in any order, prefixed by the appropriate option name:

cc	Followed by a literal that is, or an expression in curly braces that resolves to, a list of email addresses separated by commas. These addresses become the CC, or carbon copy, list for the email.
bcc	Followed by a literal that is, or an expression in curly braces that resolves to, a list of email addresses separated by commas. These addresses become the BCC, or blind carbon copy, list for the email. Blind carbon copy addresses are stripped from the email header before the message is sent.
attach	Followed by a literal that is, or an expression in curly braces that resolves to, a list

	of additional attachment files, separated by commas. Note that the PDF job itself is always emailed as an attachment, so only use this option for adding additional attachments to the message.
otherhead or oh	Followed by a literal that is, or an expression in curly braces that resolves to, one or more line-feed or "\n" delimited custom email headers.
login	Followed by a literal that is, or an expression in curly braces that resolves to, a login name. Some mail servers are configured to require a login and password for authentication. This value and the password value are then required.
password or pswd	Followed by a literal that is, or an expression in curly braces that resolves to, a login password. Some mail servers are configured to require a login and password for authentication. This value and the login value are then required.

Example

```
prejob{
email_to$=trim(get(1,1,50))
invoice_no$=get(60,5,6)
}

email {email_to$}, "sales@acme.com", {"Invoice number "+invoice_no$}, "Please pay the attached invoice promptly.\n\nBest regards,\n\nAcme Distributing", cc "accounting@acme.com"
```

Drivers: pdf only

ERASE, CERASE

Syntax

1. erase *col*|{*numexpr*}, *row*|{*numexpr*}, *cols*|{*numexpr*}, *rows*|{*numexpr*}

2. erase "*text*|!=*text*|~*regexp*|!~*regexp*[*@left,top,right.bottom*]", *col*|{*numexpr*}, *row*|{*numexpr*}, *cols*|{*numexpr*}, *rows*|{*numexpr*}

If **cerase** is used, then *columns* and *rows* are interpreted to be the opposite corner of the region, and columns and rows are calculated by UnForm.

Description

The text from the input, in the region indicated by the *column*, *row*, *columns*, and *rows* parameters, is erased. This keyword may be used to easily clear unwanted text from the output. The text is erased after text expressions and prepage and precopy code blocks are executed, so the information to be erased is available to those routines. If used, *numexpr* is a Business Basic expression that generates a numeric value for the column, row, columns, or rows.

If syntax 2 is used, then the region is defined relative to any occurrence of the *text*, or of text that matches the regular expression *regexpr*. In these cases, there may be no erased regions, or several. *column* and *row* are 0-based in these formats. The search for *text* or *regexpr* can be limited to a region on the page by adding a suffix in the format '*@left,top,right,bottom*'. To use a literal "@" character in *text* or *regexpr*, it is necessary to specify "\@".

If the syntax "!=*text*" or "!~*regexpr*" is used, then the search is for positions NOT equal to the text or NOT matching the regular expression. When using the NOT syntax, only one search is performed per line in the search region.

Also see the erase option of the **hline** and **vline** keywords.

Examples:

erase 1,5,30,4 erases text from a region from column 1, row 5, for 30 columns and 4 lines.

erase "John Smith",0,0,10,1 erases all occurrences of "John Smith" from the page.

Drivers: all

FIXEDFONT

Syntax

fixedfont *fontcode*

The **fixedfont** keyword overrides the default fixedfont setting found in the [default] section of the ufparam.txt file. If there is no fixedfont value in that file, then the *fontcode* 4099 (Courier) is used.

The *fontcode* specified is used for the text sent to UnForm by the application. It must be a non-proportional, scaleable font, except in the circumstance where a non-scaleable font provides the exact pitch required by UnForm to lay out the columns within the margins.

Drivers: laser only

FONT, CFONT

Syntax

1. font *col*|{*numexpr*}, *row*|{*numexpr*}, *cols*|{*numexpr*}, *rows*|{*numexpr*} [,*fontname*] [,font *fontcode*] [,symset *symset*] [,*size*] [,bold] [,italic] [,underline] [,light] [,shade *percent*] [,fixed | proportional] [,*color*] [,rgb *rrggbb*] [,*justification*] [,upper|lower|proper] [,fit] [,weight *w*|*weightname*] [,style *style*|*stylename*]

2. font "*text*|!=*text*|~*regexp*|!~*regexp*[*@left,top,right.bottom*]", *col*|{*numexpr*}, *row*|{*numexpr*}, *cols*|{*numexpr*}, *rows*|{*numexpr*} [,*fontname*] [,font *fontcode*] [,symset *symset*] [,*size*] [,bold] [,italic] [,underline] [,light] [,shade *percent*] [,fixed | proportional] [,*color*] [,rgb *rrggbb*] [,*justification*] [,upper|lower|proper] [,fit]] [,weight *w*|*weightname*] [,style *style*|*stylename*]

If **cfont** is used, then *columns* and *rows* are interpreted to be the opposite corner of the region, and columns and rows are calculated by UnForm.

Description

The **font** keyword applies font control to all input stream text in the defined region of column, row, columns, and rows. The other parameters are all optional. If used, *numexpr* is a Business Basic expression that generates a numeric value for the column, row, columns, or rows.

If syntax 2 is used, then font attributes are applied relative to the occurrence of *text* or the regular expression *regexpr*. In these cases, there may be no attribute regions, or several. *column* and *row* are 0-based in these formats, and can be negative if required. The search for *text* or *regexpr* can be limited to a region on the page by adding a suffix in the format '*@left,top,right,bottom*'. To use a literal "@" character in *text* or *regexpr*, it is necessary to specify "\@".

If the syntax "!=*text*" or "!~*regexpr*" is used, then the search is for positions NOT equal to the text or NOT matching the regular expression. When using the NOT syntax, only one search is performed per line in the search region.

Font Names and Numbers
fontname can be Courier (the default), CGtimes, or Univers. These fonts are standard on virtually all PCL5 compatible printers. Alternately, font *fontcode* can specify a specific fontcode supported by your printer. For example, if your printer supports True Type Arial, specify "font 16602". Bitmap fonts (as opposed to scaleable fonts) should not be used. *fontname* and *fontcode* can also be specified from the "ufparam.txt" file. UnForm 6 uses HP/GL by default for laser output, and justification is supported on all native printer fonts. However, if the –nohpgl command line option is used, then only certain, known fonts (found in fonts.txt in the UnForm directory) can be properly justified, if the center, decimal, or right *justification* option is used. When producing PDF output, only native PDF fonts are supported. All others are mapped to one of these fonts: Courier, Helvetica, or Times-Roman.

Symbol Sets
symset can be any symbol set supported by your printer. The default symbol set is "9J", using a Windows ANSI character set. *symset* can also be a name from the "ufparam.txt" file. The pdf driver only supports the Windows ANSI symbol set.

Point and Pitch Sizes
size is a numerical value that specifies the point size of a proportionally spaced font or the pitch size of a fixed font. Values range from about 4 to 999.75. The default is based on the rows per page. Note that for proportional fonts, the larger the number, the larger the size printed. Fixed fonts are the opposite.

Attribute Styles
The words "bold", "italic", "underline", and "light" will apply the indicated attribute(s) to the text.

Shaded Text
percent indicates the percent gray to print the text, from 0 (white) to 100 (black). The default is black.

Fixed and Proportional Text
Any font code below 4100 is presumed to be fixed (mono-spaced), and codes 4100 and up are presumed to be proportional. To override this assumption, specify one of the words "fixed" or "proportional".

Color
Color can be specified as "white", "cyan", "magenta", "yellow", "blue", "green", "red", or "black", or you can name an RGB value as a 6-character hex string with "rgb *rrggbb*", where *rr* is red (00-FF), *gg* is green (00-FF), and *bb* is blue (00-FF).

Justification
justification can be one of the following words: "left", "center", "right", or "decimal". UnForm will remove leading and trailing spaces from the text and justify it within the column specification. Decimal justification will use a "." character unless a "decimal=*character*" line is placed in the ufparam.txt file under the [defaults] section.

Text Case Conversion
The mutually exclusive "upper", "lower", and "proper" options will convert the text in the fonted region to all UPPER, lower, or Proper case. Proper case capitalizes the initial letter of each word or word segment preceded by a non-letter or non-digit character.

Fit to Width
If the "fit" option is used, then each line in the font region is scaled down, if necessary, to fit within the defined number of columns for the region. This differs from the text command's fit option, in that each line is treated distinctly, rather than the entire set of lines being calculated as a unit.

Weight and Style
Some laser printer fonts must be specified with given weight or style in order to be selected by the printer. For example, the font Clarendon Condensed is only available if the condensed style is specified, by adding "style 4" or "condensed" to the font command. Style and weight options and codes can be found in the ufparam.txt file. Note that fonts are expressly designed for certain weights and styles, and

simply specifying an unsupported value does not produce the desired result. In fact, it may result in selection of a different font entirely. Check your printer's documentation or control panel prints for supported fonts.

Note that if you use identical font commands for two adjacent or overlapping regions, UnForm will combine the regions. For proportionally spaced fonts, the result will be misaligned columns. To avoid this, you can add non-operational options, like "black" or "shade 100" to alternating commands, so UnForm will not treat them as identical.

Examples:

font 10,20,29,50,cgtimes,12,center will change the text in the region starting at column 10, row 20, for 29 columns and 50 rows, to 12-point cgtimes. The text will be centered within the 29 column width.

cfont 1,20,132,52,courier,16.67 will change the font of the region specified to 16.67 pitch courier. Since courier is a mono-spaced font, the number 16.67 is interpreted as a pitch (characters per inch) rather than a point size.

cfont {pos("Description"=text$[22]},23,{pos("Units"=text$[22])-1},60,univers,10 will calculate the starting and ending column based upon where "Description" and "Units" occur in line 22, and change the font for that column range, for rows 23 through 60.

Drivers: all, but note the following:

PDF: maps pcl font names and numbers to Courier, Helvetica, or Times-Roman. Symbol set 9J is the default and the only symbol set supported.

zebra: symbol sets are not supported. *size* is limited to scalability of the font in the printer's firmware, typically integer multiples of the base font size in dots. Color is not supported, nor is justification. Shading can be either 100% (black) or 0% (white). Font names are not mapped. Specify fonts instead as font codes, which must be internal font identifiers, such as a-f, 0-9. See the ZPL documentation for font codes.

The fit option is only supported in laser and pdf drivers.

GS

Syntax

gs [yes | on]

Description

The **gs** command can be used to control graphical shading. The command by itself or followed by the words "yes" or "on" will turn on graphical shading. Any other parameter value will turn graphical shading off, resulting in the highly efficient, though not as finely rendered, internal laser shade commands. The –gs command line option can be used to specify graphical shading by default.

If dpi is set to 600 or above (and the printer supports 600 dpi printing), graphical shading is even more finely rendered. Note that some faxing products that convert pcl code into low-density bitmaps provide more readable output without graphical shading. You can selectively turn graphical shading on or off within "if copy" blocks.

Using the **gs** command will add approximately 2000 bytes of additional overhead to a job.

Example:

```
gs on

if copy 2
  gs off
  output "|vfx –n " + faxnumber$ +" –F pcl"
end if
```

Drivers: laser only

HLINE

Syntax

hline "*text*" [,erase] [,extend] [,*thickness*]

Description

Any horizontal occurrence of the *text* indicated, of at least the length indicated, will be replaced with a horizontal line. The *text* must be composed of a single character repeated any number of times. There can be multiple **hline** keywords in a rule set, if needed. For example, if both dashes (-) and equal signs (=) are used for lines in a form, both can be specified in separate **hline** keywords.

This keyword is useful if the application already produces boxes and lines with standard characters. Also see the **vline** keyword.

As with all box drawing, UnForm will consider line endpoints to be at the center position of a character, which may impact how lines intersect. Lines are drawn 1 pixel (1/300 inch) thick.

If the "erase" option is used, then no line is drawn. Instead, the horizontal text values are simply removed from the output.

If the "extend" option is specified, the lines are extended ½ character left and right. The *thickness* parameter specifies a pixel width to draw.

The search for *text* can be limited to a region on the page by adding a suffix in the format '*@left,top,right,bottom*'. To use a literal "@" character in *text*, it is necessary to specify "\@".

Example:

hline "---" will search the report for 3 or more horizontal dashes. All such dashes found will be replaced with a horizontal line.

Drivers: all

HSHIFT

See the **shift** command.

IF COPY … END IF

Syntax

if copy *n,n,…*
…
end if

Description

The **if copy** command will cause any following commands, up to an **end if** command, to apply only to the copy or copies specified. The feature is used to manipulate the content of various copies. For example, you may wish to add a text message on a specific copy, or suppress a region of text with a white shade. When combined with **attach** and **notext** keywords, attachments can be added without the printing of text.

end if indicates that conditional processing of the rule set is done, and keywords apply to all copies again. The **end if** keyword may also be entered as **endif** or **fi**.

Examples:

if copy 2 will process keywords following this line, until an **endif** keyword is found, and apply keywords only to copy 2.

if copy 3,4,6 will apply keywords to the 3 copies identified.

Drivers: all

IF DRIVER … END IF

Syntax

if driver *n*
…
end if

Description

The command **if driver** will cause any commands to apply only when the rule set is evaluated under the driver *n*. The driver is specified with the command line option "-p", and defaults to "laser".
end if indicates that conditional processing of the rule set is done, and keywords apply to all copies again. The **end if** keyword may also be entered as **endif** or **fi**.

Example:

This example will use the image "pdflogo.pdf" when "-p pdf" is used on the command line.

```
if driver pdf
  image 1.5,2,15,6,"pdflogo.pdf"
end if
```

Drivers: all

IMAGE

Syntax

image *col*|{*numexpr*}, *row*|{*numexpr*} [, *cols*|{*numexpr*}, *rows*|{*numexpr*}], {"*file*" | {*expr*}} [,color], [,cache] [,option *code*] [,shade *percent*]

Description

The **image** command is used to print an image file specified by *file* to each page when the output position is the *column* and *row* indicated. This option is typically used to add graphic logos to forms. The column and row can be specified with decimal fractions to 1/100 character. The image file must be in the native format for the driver being used: pcl raster for laser, PDF for pdf, zpl for zebra. An exception to this is that if image conversion is configured, then most image formats can be converted to pcl or pdf as needed. See Automated Image Conversion, below.

If the *row* is 0 or 255, then UnForm will apply no positioning to the output. In this case, the positioning desired should be present in the file. UnForm will scan the file, looking for image information and possibly position data. Just that information will be sent to the output device. If the row is greater than 0 and less than 255, then UnForm will ignore any positioning that might be contained in the image file, and instead place the upper left corner of the image where specified.

The optional *cols* and *rows* parameters are used in some circumstances. If not supplied, and scaling is possible, then each defaults to 10. The following list specifies how *cols* and *rows* are used:

- PDF images are scaled so that they fit within the *cols* and *rows* specified.
- Laser images are scaled only if automated image conversion is enabled (see below).

If used, *numexpr* is a Business Basic expression that generates a numeric value for the column, row, columns, or rows.

If *expr* is used, then it should be a valid Business Basic expression that resolves to a string value, which will be interpreted as the file name as each copy prints.

If UnForm is producing PDF output, and the image file name ends in .pcl, .prn, or .rtl, then the file name is modified to have a .pdf extension automatically. This allows a single fixed file name to accommodate both laser and PDF output without special logic. If automated image conversion is enabled, and the extension is other than one of these four values, then the image is converted to the appropriate format and size.

Shading of an image, most often used for producing a watermark, can be specified using the shade *percent* option. This causes a shade mask of the *percent* specified to be applied as a mask to the image, resulting in a portion of the pixels being converted to white.

Creating Native Image Formats Manually
The most commonly used method is to use the publisher's web site image conversion utility, available from the UnForm page at http://synergetic-data.com. You can upload an image file and receive back images in PCL, RTL, or PDF format.

Another way to generate a PCL image for UnForm is to set up a HP LaserJet III or higher printer on a Windows workstation, and specify the "port" to be a file. You don't need a physical printer, just the Windows printer driver. Then use a graphics or word processing tool to display the image and print to that printer. Make sure that the properties are set to raster graphics and not vector graphics. Windows will prompt for a file name, and produce that file as a PCL raster image that UnForm can use. Note that even if the file has a .prn extension, it will still be a PCL file. Do not select a driver that uses PCL6, as that may not produce a PCL raster image. Choose PCL5 driver, or a sub-level, such as PCL5e.

Note that for color laser printers, UnForm requires a HP RTL (raster transfer language) format file. Color LaserJet printer drivers for Windows do not produce RTL images. Image Alchemy, from Handmade Software Inc. (http://www.handmadesw.com), is recommended to create RTL files, or you can use the image conversion utility mentioned above.

To create an image file for the pdf driver, use either Adobe Acrobat Distiller or rely on the automated image conversion, if configured. If you use Distiller, be sure to set the job options to turn OFF the "Optimize PDF" flag, and ON the ASCII flag. UnForm's PDF parser relies on a standard, non-optimized PDF file format.

PDF Image Considerations
Image file names can be up to 75 characters in length.

The default value for *cols* and *rows* is 10, if not supplied. Version 5 and prior allowed image sizes to vary by device by simply not enforcing a scaled size.

PDF images are scaled to the largest size that will fit within *cols* and *rows* while maintaining the aspect ratio of the source image. Version 5 and prior scaled images to both *cols* and *rows*, even if the result stretched the original image.

Automated Image Conversion and Scaling
UnForm can be configured to use external image management software to perform scaling and conversion as needed, if a supplied image is not in pcl or PDF native format. Two commonly used products are the commercial Image Alchemy, available from http://handmadesw.com, and open source ImageMagick from http://imagemagick.org. The configuration for this is entered into the [images] section of the uf60d.ini file. One entry, converter=*program* sets the path of the converter executable (usually "convert" or "convert.exe" for ImageMagick, and "alchemy" or "alchemy.exe" for Image Alchemy). Entries for command lines for pcl, pclc, and PDF are configured for pcl, color pcl, and PDF, image conversion lines, respectively.

Related to the conversion are three options: **color**, **cache**, and **option *code*.** These three options have the following use:

If the word **color** is present, then the command line configured for pclc is used, rather than the default pcl line. Note that this has no effect on PDF output, and it should not be used if the target device is a black and white laser printer, as the image produced will probably be incompatible with that type of printer. The **–ci** command line option can also be used to specify color images by default.

If the word **cache** is present, then UnForm will store converted files by their base name and characteristics in the images sub-directory in the UnForm directory. UnForm will then use this pre-converted file in subsequent jobs calling for the same file and options. Note that this technique will not work if different paths are used for the same base file name. For example, if a standard file called "signature.bmp" is found in different users' home directories, it would not work to cache the images, as every user's signature would be have the same name. If the source image ever changes, simply remove the file(s) from the images directory, and UnForm will re-convert the files as needed.

The **option *code*** entry can be used to reference secondary conversion lines in the uf60d.ini file. By referencing *code*, different conversion command lines can be configured and specified by the image command. The name referenced will be pcl-*code* or PDF-*code*, as required. Option code values can be up to 10 characters long and are case sensitive.

Within each line, UnForm will replace the following markers with appropriate values determined from the image command:

%i	for the input image file
%o	for the output PDF or pcl image file that UnForm will use
%d	for resolution in dots per inch
%x	for image width in pixels
%y	for image height in pixels

Here is an example of configuration using Image Magick for UNIX, using the convert command:

```
# Examples for ImageMagick (note pcl requires 5.5.7+)
converter=convert
pclc=%i %o -density %dx%d -dither -resize %xx%y >/dev/null 2>&1
pcl="%i" -density %dx%d -monochrome -resize %xx%y "%o" >dev/null 2>&1
pdf="%i" -density 300x300 -colors 256 "%o" >/dev/null 2>&1
pdf-72="%i" -density 72x72 -colors 256 "%o" >/dev/null 2>&1
```

Examples:

image 0,255,"/usr/unform/logo.pcl" will place the named file on each page. The file should contain the desired cursor positioning.

image .5,1.25,"/usr/unform/logo.pcl" will place the raster image contained in the named file at column .5, row 1.25.

**image {icol},{irow},{icols},{irows},{logo$}** will place an image file specified in the variable logo$ at the position specified by the variables icol and irow. If used in a pdf driver or when automated conversion and scaling is invoked, the variables icols and irows would specify the image size (more specifically, its bounding box) in columns and rows. All the variables would have to be created in a code block, such as prejob{} or prepage{}.

Drivers: all.

Laser requires pcl raster format, pdf driver requires PDF format, zebra requires zpl format. If automatic image conversion is configured, then laser and PDF images can be produced from various formats supported by the configured converter. Shading applies only to laser images.

ITALIC

See the **bold** keyword.

KEYWORDS

Syntax

keywords "*keywordstring*" | {*expression*}

Description

If this command is present, then PDF document creation adds a keyword *keywordstring*, or the result of *expression*, to the document content. This value is available in the general properties display dialog in the Adobe Acrobat Reader.

Drivers: pdf only

LANDSCAPE, RLANDSCAPE

Syntax

landscape or rlandscape

Description

This keyword will ensure that UnForm produces output in landscape (horizontal) orientation. The default orientation is portrait (vertical), unless UnForm encounters a PCL control code setting landscape mode (hex 1B266C314F) on the first page. Use of this keyword will force landscape mode regardless of PCL control codes found in the input.

The **rlandscape** command will turn on reverse landscape mode.

Note that landscape is supported inside 'if copy' blocks, allowing different copies to be in different orientations.

Also see the **portrait** keyword.

Drivers: laser, pdf (rlandscape is laser only)

LIGHT

See the **bold** keyword.

LPI

Syntax

lpi *line-height*

Description

The **lpi** keyword indicates the vertical line height UnForm should use when printing the text of a form or report. From this, along with the paper dimensions, UnForm can determine the rows per page and ensure that the proper vertical placement is selected for each line. To save time and effort, use the **rows** keyword and UnForm will calculate the lpi.

See also **cpi**, **cols**, **rows**.

Examples:

lpi 8 sets 8 lines per inch.

lpi 6.6 uses a common laser printer value based on 66 lines in a 10 inch printable page length on letter paper.

Drivers: all

MACRO

Syntax

macro *n*

Description

This keyword will cause UnForm to invoke macro number *n* in the LaserJet printer. This macro must be defined and downloaded to the printer as a permanent macro. This keyword could be used to call a macro for a company letterhead, for example. For more information, see the Working With Macros chapter.

Drivers: laser only

MACROS

Syntax

macros on|off

Description

This keyword causes UnForm to invoke (or not invoke) macros for fixed raster elements (**box**, **shade**, **text**, **image**, and **attach**). Macro usage can significantly reduce the data transfer requirements to the printer, most noticeably on a serial or parallel connection with many pages of similar output. The printer must have enough memory to store and execute the macros.

The default macros setting is "off"; the "-macros" command line option establishes the default macros setting to "on". This keyword overrides either default for this rule set.

Macros are numbered from 0 to 32767. UnForm will start macro definitions at 32000 unless the "[defaults]" section, "macrono" field is set to a different value in the ufparam.txc file. If a site uses macros and finds a conflict with this number, then the value should be changed to allow an available contiguous range for UnForm.

Drivers: laser only

MARGIN

Syntax

margin[s] *left, right, top, bottom*

Description

The **margin** keyword is used to increase the margins used by UnForm when calculating row and column positions. Normally, UnForm will use a 0.25 inch margin on all 4 sides, based on the paper size in use. If you need to increase any margin, you can specify the dot offsets desired. Note that the values for *left, right, top,* and *bottom* are entered in dots, which default to 300 dpi, but can be modified by the **dpi** keyword.

For example, **margin 75,75,0,150** (at 300 dpi) would set left and right margins to 0.5 inches, the top margin would remain at 0.25 inches, and the bottom margin would be 0.75 inches.

Drivers: laser, pdf

MERGE

Syntax

merge "*ruleset*" [, "*rulefile*"]

Description

This command will insert the contents of the *ruleset* into the currently parsed rule set. If the *rulefile* parameter isn't supplied, the current rule file is used. Otherwise, *rulefile* is opened in the UnForm directory or by full path, if specified, and is scanned for *ruleset*. This command can be used to incorporate common elements into many rule set formats. For example, a name and address heading could be placed into a rule set called "address_header", and various forms could use the command **merge "address_header"** to include the commands it contains.

Note that if no *rulefile* is specified, then the rule file specified for the job is used for the merge, even if the merge is nested within another merge that specifies another rule file.

Unlike other UnForm commands, **merge** works within code blocks, such as precopy or prepage, as well as outside of code blocks.

Drivers: laser, pdf, zebra

MICR

Syntax

micr *col*|{*numexpr*}, *row*|{*numexpr*}, *"account"*|{*expr*}, *"check"*|{*expr*}

Description

Prints MICR font at the *col* and *row* specified, for laser check printing. If used, *numexpr* is a Business Basic expression that generates a numeric value for the column and row. The account number must be in the format **:123456789:xxx"**, where the colons surround the 9-digit bank number, and the balance of the account number is terminated with, or contains, a quote. Quotes can be identified in a text literal with <34>. A space after the bank number and terminating colon is optional. When the MICR code is generated, colons or A become a "transit" symbol, B becomes an "amount" symbol, quote or C become an "on us" symbol, and a hyphen or D becomes a dash . Account numbers can contain these symbols, spaces, and digits. The check number can be up to 12 digits long. This keyword supports 8 inch checks only, not the smaller 6 inch variety, which requires a different format for the MICR.

If no "on us" symbol is present in the account number (i.e. no <34> or C character), then one is appended automatically.

The fixed bank number is typically hard-coded, but can be an expression if enclosed in braces {}. The check number will generally be an expression, which can use get() to retrieve the number from the application print, or can be a variable defined in a prepage{} block.

Note that with proper soft font configuration, you can use the text command to print MICR encoded data in any format, such as that required by a deposit slip. The same MICR soft fonts included for use with this command can be used as text soft fonts.

Example:

micr 6,42.25,":123456789:9999-1234<34>",{trim(get(65,5,6))} would print a MICR encoded line with the indicated bank and account number, and a check number derived from the input stream data printed at column 65, row 5, for 6 characters.

Drivers: laser only

MOVE, CMOVE

Syntax

1. move *col*|{*numexpr*}, *row*|{*numexpr*}, *cols*|{*numexpr*}, *rows*|{*numexpr*}, *newcol*|{*numexpr*}, *newrow*|{*numexpr*} [,retain]

2. move "*text*|!=*text*|~*regexp*|!~*regexp*[*@left,top,right.bottom*]", *col*|{*numexpr*}, *row*|{*numexpr*}, *cols*|{*numexpr*}, *rows*|{*numexpr*}, *movecols*|{*numexpr*}, *moverows*|{*numexpr*} [,retain]

Description

cmove causes *cols* and *rows* to be interpreted as the opposite corner of the region to be moved.

The **move** keyword moves a block of text to a new location on the page. Syntax 1 moves the region indicated by *col*, *row*, *cols*, and *rows* so the new upper left point is at *newcol*, *newrow*. Syntax 2 searches for occurrences of *text* or the regular expression *regexpr*, respectively, and uses each location found as a point from which *col* and *row* are measured (0-based movement). The rectangular region specified is then moved *movecols* left or right, and *moverows* up or down. The search for *text* or *regexpr* can be limited to a region on the page by adding a suffix in the format '*@left,top,right,bottom*'. To use a literal "@" character in *text* or *regexpr*, it is necessary to specify "\@".

If used, *numexpr* is a Business Basic expression that generates a numeric value for the column, row, columns, or rows, and also the "new" column and row (syntax 1) and the "move" columns and rows (syntax 2).

If the syntax "!=*text*" or "!~*regexpr*" is used, then the search is for positions NOT equal to the text or NOT matching the regular expression. When using the NOT syntax, only one search is performed per line in the search region.

The optional "retain" parameter will cause UnForm to leave the text in its original location, in effect copying the text rather than moving it.

Move commands simply shift text around in an internal array, so it is possible for moves to cascade to other moves. Moves that specify positions (syntax 1) are performed in the order found in the rule set, then moves that are relative to text (syntax 2) are performed in the order found in the rule set.

Note that **move** commands occur *after* any **shift** or **vshift** commands. If you would like to move data based on positions before the **shift** and **vshift** commands, consider using a **text** command with an expression using the cut() or mcut() functions.

Examples:

move 5,10,40,4,20,20 moves text at column 5, row 10, 40 columns wide and 4 rows high, to the region 20,20,40,4.

move "Date",0,0,4,1,-4,0 moves all occurrences of the word Date left by 4 columns.

Drivers: laser, pdf

NOTEXT

Syntax

notext

Description

This keyword specifies that no report text should be printed. Typically, this would be placed inside an "if copy *n*" block in order to add an attachment and prevent overwriting of the form text.

Example:

```
if copy 2
        attach "/usr/unform/attachments/attach1.pcl"
        notext
end if
```

Drivers: all

OUTLINE

Syntax

outline [*level*]

Description

The **outline** keyword turns on the production of PDF outlines (also called bookmarks) and the automatic display of the outline when the document is displayed in an Adobe Acrobat Reader. The content of the outline is set page by page, by setting the variable "outline$" in a precopy or prepage code block. Multi-level outlines can be specified by delimiting the levels with vertical bar (|) characters in the outline$ string.

If *level* is supplied, it must be an integer greater than zero. This indicates the highest outline level that will be initially opened when Acrobat displays the document. The default behavior is to have all levels open, but with exceptionally large reports, it may be desirable to have just the first 1 or 2 levels initially opened.

See the outline rule set in advanced.rul for and example.

Drivers: pdf only

OUTPUT

Syntax

1. output "*output-device*"

2. output {*expression*}

Description

The **output** keyword is used to modify the output device of any copy. Normally, all copies are printed to the output device specified in the "-o" option, or to standard out on UNIX. However, it is sometimes desirable to have copies of forms sent to different devices, such as a different laser printer, or a fax product.

The *output-device* can be a printer device, a pipe or re-direct (starting with | or >), or a filename. Beware of pipes or redirects on UNIX, noting that any shell-aware characters, such as ampersands (&), must be quoted.

If the second syntax is used, *expression* is evaluated after each page of input has been loaded and the prepage subroutine has been executed.

When used inside an **if copy** block, the output for that copy only is changed. Note that this feature is only supported in the laser driver. When using the pdf driver, any change to output for different copies is ignored.

The "output$" variable can also be set in a code block for equivalent results.

Example:

```
if copy 2
output "|lp -daccounting -s"
end if
```

The above example would send the second copy of the form to the printer named "accounting".

Drivers: laser, pdf only for a job-wide specification outside of "if copy" blocks as PDF output cannot be changed during printing.

PAGE

Syntax

1. page *rows*

2. page *cols*, *rows*

Description

Syntax 1 specifies an input page length of no more than *rows* lines. If a form-feed character is encountered first, then the page is considered complete also. This keyword is useful if the application creates a form with line-feeds rather than form-feeds.

If syntax 2 is used, then each page worth of rows is divided into column groups of *cols* wide and treated as virtual pages from left to right. For example, if an application prints mailing labels as 4-up labels each 30 columns wide and 6 rows deep, then the command **rows 30,6** would produce 4 pages, each 6 rows. This can be useful to convert *n*-up continuous label print jobs into laser label jobs using the **across** and **down** commands.

If no **rows** or **lpi** keyword is specified, then *n* is assumed to be the rows per page.

Examples:

page 42 treats each 42 lines of input as a full page.

page 42
rows 66 treats each 42 lines input as a full page, but produces output scaled to 66 lines per page.

Drivers: all

PAPER

Syntax

paper *size*

Description

The **paper** keyword overrides the "-paper" command line option. It tells UnForm the paper size to instruct the printer to use, and also defines the page size from which UnForm calculates column and row widths.

Common sizes for laser and PDF output include the following, plus any sizes defined in the [paper] section of the ufparam.txt file (or ufparam.txc if defined).

Value	**Size**
Letter	8.5 x 11 inches
Legal	8.5 x 14 inches
Ledger	11 x 17 inches
Executive	7.25 x 10.5 inches
A4	210 x 297 mm
A3	297 x 420 mm

For Zebra printers, indicated by the "-p zebra*n*" command line option, the *size* is given as a single word made up of the width in inches, a letter "x", and the height in inches. For example, a 3-inch by 5.25-inch label would be specified by **paper 3x5.25**.

If you specify the "custom" paper size for laser output, UnForm will use the defined size for scaling and will issue the proper custom paper command to the printer, but you may still have to modify the custom paper setting via the printer's control panel to avoid prompts to load custom paper into the printer.

Drivers: all

PORTRAIT, RPORTRAIT

Syntax

portrait or rportrait

Description

This keyword ensures that UnForm will print pages oriented in portrait (vertical) fashion. If, while analyzing the report text, UnForm detects a PCL control sequence to turn on landscape mode, then landscape will be the default orientation. Use this keyword to guarantee that the orientation will be vertical.

The **rportrait** command turns on reverse portrait mode.

Note that **portrait** is supported inside **if copy** blocks, allowing different copies to be in different orientations.

See also the **landscape** keyword.

Drivers: laser, pdf (rportrait is laser only)

PRECOPY, PREDEVICE, PREJOB, PREPAGE

POSTCOPY, POSTDEVICE, POSTJOB, POSTPAGE

Syntax

precopy | postcopy | prejob | postjob | prepage | postpage {
code block
}

Note: the opening brace "{" needs to be on the same line as the keyword. The closing brace may follow the last statement, or be on the line below the last statement.

Description

These keywords are used to add Business Basic processing code to the form or report. They represent six different subroutines that UnForm executes at specific points during processing. The *code block* can be an arbitrary number of Business Basic statements; the total number of statements in all code blocks can be about 6,000.

- **prejob** executes after the rule set has been read, and after the first page is read, but before any printing takes place. Use this code to open files, define string templates, create user-defined functions, and initialize job variables.
- **postjob** executes after the last page has been printed. Use this to close out your logic, such as adding totals to log reports. There is no need to close files, since UnForm will RELEASE Business Basic.
- **predevice** executes just after a device has been opened. With the laser driver, the output device can be changed with the **output** command or by modifying the output$ variable in a prepage or precopy code block. Whenever a new device is opened for any given copy, this code block is executed. The programmer can then store information from the page that causes the device to be opened, such as a customer code or fax information.
- **postdevice** executes just after the output device has been closed. Use this code block to perform processing with prior output device, once UnForm has closed the device. For example, if the output device changed when the customer number changed, then one or more pages for a given customer would be in the output file and could be sent as a group to a fax product.
- **prepage** executes after each page is read, but before any printing takes place. Use this to gather data associated with any page, or to modify the content of the text if you need such modifications to apply to all copies.
- **postpage** executes after the last copy of each page has printed.
- **precopy** executes before each copy is printed. Use this to modify copy text content, to skip specific copies, or to modify a copy's output device.

- **postcopy** executes after each copy is printed.

Any valid Business Basic programming code can be entered, including I/O logic, loops, variable assignments, and more. Program to your heart's content. UnForm will add extensive error handling code within your code, and report syntax errors to the error log file or a trailer page.

Note that the **merge** command, while not executable code, is honored within a code block. The merged data must be valid code block syntax.

For more details about programming code blocks, see the Programming Code Blocks chapter.

Important Note for BBx Developers

As UnForm 6 provides its own ProvideX-based run-time engine, it is important to note that code blocks now operate under a ProvideX interpreter. UnForm includes lexical substitution for nearly all BBx syntax; however, in order to read BBx data files, you must use the new bbxread() function, or develop your own method of accessing the files via a native BBx execution in a pipe or one that uses a socket or other means of communication. The bbxread() function is provided expressly for this purpose, and allows for reading of records or record-based string templates with a file-name and a key.

Here is an example of using bbxread:

```
prepage{
ky$=get(65,5,6)
dim rec$:"id:c(5*=10),*:c(1*=10),…,fax:c(15*=10)"
bbxread("/usr/data/CUSTOMER",ky$,rec$,errcode)
if errcode=-1 then faxnum$=rec.fax$
}
```

In order to use the bbxread function, you must specify the bbpath value in the uf60d.ini file.

Example:

This example shows how to use various routines to make copy 2 of a form be a conditionally faxed invoice, which is logged to another printer for verification.

```
prejob {
cust=unt; open(cust)"custfile.dat"
dim cust$:"id:c(5),name:c(30),*:c(100),faxnum:c(12)"
}

prepage {
if trim(get(10,2,30))="Acme Systems" comp$="01" else comp$="02"
```

```
dim cust$:fattr(cust$)
read record (cust,key=comp$+get(10,5,6),err=next)cust$
}

precopy {
if copy=2 if cvs(cust.faxnum$,3)>"" output$="|fx -n "+ \
cust.faxnum$, log$=log$+cust.name$+$0d0a$ else skip=1
}

postjob {
if log$="" goto endjob
log=unt; open(log)">lp -dprinter"
print (log)"Fax Verification Log"
print (log)log$,chr(12),
close(log)
endjob:
}
```

Drivers: all, but predevice and postdevice are only supported by laser and pdf drivers.

PROTECT

Syntax

protect [print] [,annotate] [,extract] [,modify]

Description

Without the **protect** command, UnForm generates a standard PDF document that can be opened, viewed, printed, and modified by a user. This suffices for most business documents, but if an application requires protection of the PDF contents, then this command can be used. It adds encryption and protection to a PDF document.

By default, only viewing access is provided to users. Additional access can be granted by including the following options:

print adds the ability to print the document.

annotate adds the ability to add text annotations and fill in form fields.

extract adds the ability to copy text or graphics from the document for pasting into other applications.

modify adds the ability to modify document contents.

Drivers: PDF only

ROWS

Syntax

rows *n*

Description

This keyword specifies the number of output rows to use for the form or report. The placement of each line is calculated to accommodate this many rows within the printable area of the paper. For example, with letter paper, the printable area is about 10.5 inches; **rows** 66 will cause each line to be 10.5/66 inches high. If present, this value will override any calculation based on the **lpi** keyword.

The number of rows (*n)* can be any value up to 255. It will default to 66 if no **rows**, **lpi**, or **page** keywords are present.

Note there is an important distinction between the **page** and **rows** commands. **Rows** refers to output scaling, whereas **page** defines the number of text lines to read per page from the input stream. However, if a **page** command is used, and a **rows** command is not, then the **rows** defaults to the value of the **page** command.

Examples:

rows 80 will set the line height to accommodate 80 rows per page.

Drivers: all

SHADE, CSHADE

Syntax

1. shade *col*|{*numexpr*}, *row*|{*numexpr*}, *cols*|{*numexpr*}, *rows*|{*numexpr*}, *percent* [,extend] [,*color*] [,rgb *rrggbb*]

2. shade *col*|{*numexpr*}, *row*|{*numexpr*}, *cols*|{*numexpr*}, *rows*|{*numexpr*}, *percent, skip, times* [,extend] [,*color*] [,rgb *rrggbb*]

3. shade "*text*|!=*text*|~*regexp*|!~*regexp*[@*left,top,right.bottom*]", *col*|{*numexpr*}, *row*|{*numexpr*}, *cols*|{*numexpr*}, *rows*|{*numexpr*}, *percent* [,extend] [,*color*] [,rgb *rrggbb*]

If cshade is used, then *cols* and *rows* are interpreted to be the opposite corner of the shade region, and columns and rows are calculated by UnForm.

Description

The region indicated by *col*, *row*, *cols*, and *rows* will be shaded, using the *percent* as the percent-gray value. The region parameters can be specified as decimal values to 1/100 character. The region is based on the full character cell, starting at the upper left corner of the cell. This differs from the **box** keyword, which measures from the center point of a cell. The *percent* can be any value from 0 to 100, where 0 is white (useful for erasing regions), and 100 is black. The default shade value is 5% (which renders as 10% in PCL5 devices). PCL5 printers actually support only eight levels of gray, generally: 2%, 10%, 20%, 35%, 55%, 80%, 99%, and 100%. Given values less than these are rounded up to the next supported value.

For compatibility with Version 1 rule files, Version 2 and above will convert shade values of 1, 2, 3, and 4 to 2%, 20%, 55%, and 100%, respectively.

If used, *numexpr* is a Business Basic expression that generates a numeric value for the column, row, columns, or rows.

Syntax 2 provides for repeating regions to be easily specified. The *skip* parameter is a number indicating the number of blank lines that follow the shade region. The *times* parameter is the number of times to repeat the shade/blank pattern. UnForm will generate multiple rows of shading until either the number of repetitions is met or the end of the page is found. For example, **shade 1,21,80,2,1,2,8** would produce 8 shaded regions, each 80 columns by 2 rows with shade grade level 1. Two blank lines would separate the shade regions. These two parameters are ignored if the first parameter is a text string, as in syntax 3.

If syntax 3 is used, then the shading is drawn relative to any occurrence of the *text*, or of text that matches the regular expression *regexpr*. In these cases, there may be no shaded regions, or several. *column* and *row* are 0-based, in these formats, and can be negative if required. The search for *text* or

regexpr can be limited to a region on the page by adding a suffix in the format '*@left,top,right,bottom*'. To use a literal "@" character in *text* or *regexpr*, it is necessary to specify "\@".

If the syntax "!=*text*" or "!~*regexpr*" is used, then the search is for positions NOT equal to the text or NOT matching the regular expression. When using the NOT syntax, only one search is performed per line in the search region.

All formats support the **extend** option. This simply expands the shade region by ½ character in all directions, making it easy to fill in a box that is placed at the mid-point of each character position surrounding the shade region.

Note that the **box** keyword also supports shading, and may be more convenient to use if an outlined shaded region is desired.

Color can be specified as white, cyan, magenta, yellow, blue, green, red, or black, or you can name a RGB value as a 6-character hex string with rgb *rrggbb*, where *rr* is red (00-FF), *gg* is green (00-FF), and *bb* is blue (00-FF).

You can improve the look of shade regions on laser printers, especially at medium shade levels and 600 or higher dpi settings, by using the gs command.

Examples:

shade 41,3,40,6,2 will fill the indicated region with a medium (20%) shade.

shade 10.5,3.01,40,4.98,25 will shade the indicated region with 25% gray.

shade "No. Item/Desc",0,0,79,1,10,extend will shade from the position the noted text is found, for 79 columns and 1 line. The shaded region will then be extended ½ column and row in each direction. 10% gray will be used.

shade 1,14,80,2,1,2,12 will produce a repeated pattern of 80 columns wide, 2 lines high, light shading, followed by two blank lines. The pattern will be repeated 12 times, occupying a total of 48 lines.

Drivers: all, zebra only supports 0% or 100%.

SHIFT

Syntax

shift *n*

Description

The text in the report is shifted *n* characters to the right (or left, if *n* is negative). If a report starts in column 1, but doesn't extend all the way to the right edge of the page, it is possible to shift the data to the right to allow for box drawing around text elements on the left margin.

The placement of relative shading, drawing, and attributes is determined *before* any shift.

See **vshift** also, for shifting text vertically.

Example:

shift 1 will shift all text 1 character to the right.

Drivers: all

SUBJECT

Syntax

subject "*subjectstring*" | {*expression*}

Description

If this command is present, then PDF document creation adds a subject *subjectstring*, or the result of *expression*, to the document content. This value is available in the general properties display dialog in the Adobe Acrobat Reader.

Drivers: PDF only

SYMSET

Syntax

symset "*symbolset"*

Description

The **symset** keyword overrides the default symbol set setting found in the [defaults] section of the ufparam.txt file. If there is no [defaults] section, then the symbol set 10U is used. Symbol set values for the LaserJet are always integers followed by an uppercase letter. Be sure to quote the *symbolset* value to maintain the uppercase letter (unquoted values in rule sets get converted to lowercase by UnForm's rule file parser).

Symbol sets are used to display specific international character sets or symbols. See your LaserJet documentation for symbol set codes supported by your printer.

If you plan to use the pdf driver in addition to the laser driver, you should specify your symbol sets as 9J if you intend to use special characters in the ASCII 128 to 255 ranges.

Drivers: laser only

TEXT

Syntax

1. text *col*|{*numexpr*}, *row*|{*numexpr*}, "*text"* | *@name* | $*name* | {*expression*} [,*fontname*] [,font *fontcode*] [,symset *symset*] [,*size*] [,bold] [,italic] [,underline] [,light] [,shade *percent*] [,rotate 90 | 180 | 270][,fixed | proportional | prop] [,*color*] [,rgb *rrggbb*] [,*justification*, cols *ncols*|icols ncols|ccols *endcol*] [,wrap] [,fit] [,spacing *spacing*] [,weight *w*|*weightname*] [,style *style*|*stylename*]

2. text "*text*|!=*text*|~*regexp*|!~*regexp*[*@left,top,right.bottom*]", *col*|{*numexpr*}, *row*|{*numexpr*}, { "*text"* | *@name* | $*name* | {*expression*} } [,*fontname*] [,font *fontcode*] [,symset *symset*] [,*size*] [,bold] [,italic] [,underline] [,light] [,shade *percent*] [,rotate 90 | 180 | 270][,fixed | proportional | prop] [,*color*] [,rgb *rrggbb*] [,*justification*] [,cols *ncols*|icols ncols|ccols *endcol*], [eraseoffset *cols*, erasecols *cols*] [,getoffset *cols*, getcols *cols*] [,wrap] [,fit] [,spacing *spacing*] [,weight *w*|*weightname*] [,style *style*|*stylename*]

Description

The *text* indicated in quotes will be printed at the column and row indicated by *col* and *row*. The column and row can be specified to 1/100 character. The position specified becomes the baseline left edge for the first character. If used, *numexpr* is a Business Basic expression that generates a numeric value for the column, row, columns, or rows.

If *text* begins with "@", such as **@company**, then the substitution file is searched. In the example above, if a line **company=ABC Company** was found, the text "ABC Company" is used. The substitution file defaults to "subst", but may be specified on the command line with the "-s" option.

If *text* begins with "$", then the operating system environment is searched for the indicated variable and its value is used. For example, **$USER** would use the value stored in the environment variable "USER".

If *text* should be a literal value that starts with @ or $, then use \@ or \$, respectively.

If braces surround *text*, then it is taken to be an expression to be evaluated after each page of input has been loaded and the **prepage** subroutine has been executed. The expression can be any valid Business Basic statement that would appear on the right side of an assignment statement and produces a string data type result. Some UnForm supplied functions and data can be useful, such as TEXT$[], which contains the text of the page in an array, and GET(col,row,length), a function that returns data from the TEXT$ array. For example, {"Copy 2, generated on "+date(0)} would generate text similar to this: "Copy 2, generated on 03/31/99". See the Programming Code Blocks chapter for more information about programming expressions.

If *text* contains line-feed characters (CHR(10) or $0A$), or the mnemonic character string "\n", then UnForm will break the text into multiple lines and space them according to the *spacing* value. For example, if the point size is 12, and *spacing* is set to 1.5, then line spacing is set to 18 points. The

default *spacing* is calculated from the number of rows per page, so multi-line text data will match the vertical placement of single line text data.

If syntax 2 is used, then UnForm will search for occurrences of *text* or the regular expression *regexpr*. In this case, *col* and *row* become 0-based offsets from each location where matches are found. In addition, the erasecols *cols* and eraseoffset *cols* can be used to remove match text. The search for *text* or *regexpr* can be limited to a region on the page by adding a suffix in the format '*@left,top,right,bottom*'. To use a literal "@" character in *text* or *regexpr*, it is necessary to specify "\@".

If the syntax "!=*text*" or "!~*regexpr*" is used, then the search is for positions NOT equal to the text or NOT matching the regular expression. When using the NOT syntax, only one search is performed per line in the search region.

Font Names and Numbers

fontname can be Courier (the default), CGtimes, or Univers. These fonts are standard on virtually all PCL5 compatible printers. Alternately, a specific *fontcode* supported by your printer can be specified by its font number. For example, if your printer supports True Type Arial, specify "font 16602". Bitmap fonts (as opposed to scaleable fonts) may be specified, but proper use depends on the form's or report's cpi value matching that of the font. Bitmap fonts have low *fontcode* values, like 0 for Line Printer, or 4 for Helvetica. *fontname* and *fontcode* values can also be specified from the "ufparam.txt" file.

Note that font 15002 is configured by default (in ufparam.txt) as a reference to the default MICR soft font, and can be used (with 'fixed, 8' options) to print MICR encoded text lines in cases where the micr command can't be used, such as with deposit slips, unusual bank account numbers, or non-standard check sizes.

Symbol Sets

symset can be any symbol set supported by your printer. The default symbol set is "10U", using the PC-8 character set. Other examples include 19U for Windows ANSI or 0Y for Postnet Bar Code. You can also specify symbol sets by name from the "ufparam.txt" file. Only symbol set 9J is supported by the pdf driver.

To include non-printable characters, such as control codes or 8-bit characters from a specific symbol set, include the character's numeric (ASCII) value in angle brackets. For example, to include a copyright symbol from the Desktop (7J) symbol set, use something like this: "<165>2000 Synergetic Data Systems Inc.".

Point and Pitch Size

size is a numerical value that specifies the point size of a proportionally spaced font or the pitch size of a fixed font. The values range from about 4 to 999.75 with a default of 12. PCL printers generally round this value to the nearest or smallest ¼ point. Note that for proportional fonts, the larger the number, the larger the size printed. Fixed fonts, such as Courier, are the opposite. If you specify the "fit" option, then the *size* value represents the largest acceptable size.

Fit and Wrap Options
The "fit" option will scan *text* for line breaks and decrease the *size* value as necessary to ensure that all lines will fit in the number of specified *ncols* or through *endcol*. The smallest point size that will be used is 4, and the largest pitch that will be used is 30.

The "wrap" option will scan *text* and insert line breaks as needed to ensure no line at the specified *size* will exceed the specified *ncols*. If no spaces exist in word that exceeds the line width, UnForm will print the word in its entirety, exceeding the allocated space.

The "fit" and "wrap" options are mutually exclusive, and in either case, if no *ncols* or *endcol* value is specified with the "cols" option, then *ncols* defaults to the page width in columns minus *column*.

Attribute Styles
The attribute words "bold", "italic", "underline", and "light" will apply the indicated attribute(s) to the text.

Shading
percent indicates the percent gray to print the text, from 0 (white) to 100 (black). The default is black. Note that the **gs** command can be used to improve laser printer shading.

Rotation
The "rotate" option will cause the text to be rotated around the baseline left edge at 90, 180, or 270 degrees. PCL5 supports rotation only in these increments.

Fixed and Proportional Spacing
Specify "fixed" or "proportional" (or "prop") to override the default of fixed for Courier (or any *fontcode* below 4100), and proportional for all else. For example, if a mono-spaced font, such as the MICR soft font, has a font code higher than 4100, then the "fixed" option is required in order to ensure the proper font is selected, rather than a default proportional font. Proportional vs. fixed carries a very high priority when a printer chooses a font, and if the desired font is not specified with the correct spacing, a different font will be chosen by the printer.

Color
Color can be specified as "white", "cyan", "magenta", "yellow", "blue", "green", "red", or "black", or you can name an RGB value as a 6-character hex string with "rgb *rrggbb*", where *rr* is red (00-FF), *gg* is green (00-FF), and *bb* is blue (00-FF).

Justification
justification can be one of the following words: "left", "center", "right", "decimal". UnForm will remove leading and trailing spaces from the text and justify it within the column specification. Decimal justification will use a "." character unless a "decimal=*character*" line is placed in the ufparam.txt file under the [defaults] section.

For justification, you must also specify *ncols* or *endcol* with the "cols", "icols", or "ccols" option, so that UnForm can determine the right edge of the justification region.

Weight and Style
Some laser printer fonts must be specified with given weight or style in order to be selected by the printer. For example, the font Clarendon Condensed is only available if the condensed style is specified, by adding "style 4" or "condensed" to the font command. Style and weight options and codes can be found in the ufparam.txt file. Note that fonts are expressly designed for certain weights and styles, and simply specifying an unsupported value does not produce the desired result. In fact, it may result in selection of a different font entirely. Check your printer's documentation or control panel prints for supported fonts.

Get Text From Input Stream
If "getoffset" and "getcols" are specified in a syntax 2 command, then the value printed is taken from the data stream at the offset and length specified from each occurrence (any *text* value supplied is ignored). Further, "eraseoffset" and "erasecols" can be used to remove any data stream text from the point of occurrence as well.

Barcode Note
The text command can be used to print a human-readable version of a barcode value, which can be useful in cases where the human readable value differs from the supplied value, such as UPC-E, or when a check digit value is needed.

Text in this syntax: "bcd*sss*|*value*" to print the human readable barcode value for symbology *sss* and barcode text *value*, "ck1*sss*|*value*" to print check digit 1, or "ck2*sss*|*value*" to print check digit 2. See the barcode command for symbology values.

Special Symbol Fonts
There is a difference between PDF and laser output for special symbols. In the laser printer environment, you need to select a symbol set *and* font that contains the special characters you want, but in the PDF environment, you need only select the font (font 4141 for Dingbat and 16686 for Symbol). Once a symbol set or font is identified, use the appropriate decimal value of text to print the character you want. The easiest way to do this is with angle bracket notation in a literal, like "<182>", or with the CHR function in an expression, like {CHR(182)}.

On many LaserJet printers, the available symbol sets and fonts differ from those specified in UnForm's ufparam.txt file, and the only way to know for sure what is available is to do a font list print on the printer. This should show you the proper symbol set and font number to use for your printer.

Examples:

text 10,2,"SOLD TO" prints the text SOLD TO at the indicated position.

text 120,3,$LOGNAME prints user's login name at column 120, line 3.

text 1.25,63.25,{"Printed on "+date(0)}, cgtimes, 6, italic would place a small (6 point), italic note about the date near the lower left corner of a page.

text "TOTAL:",-1,0,"Total:",cgtimes,12,bold,eraseoffset 0, erasecols 6 changes words TOTAL: to Total: in CGTimes, 12 point, after backing up 1 column from where TOTAL: is found. It also erases the word TOTAL: to avoid overprinting.

text 67,21,"bcd125|00010000654",univers,12 will print the UPC-E human readable barcode value.

text 20,62,{terms$},cgtimes,10,cols 40,wrap,spacing 1 will print a paragraph of text contained in terms$ between column 20 and 59, in CGtimes 10 point text, word-wrapping as necessary, using a nominal line height matching the 10 point text.

text {pos("Item"=text$[20])},21,"Number",cgtimes,12 will print the word "Number" on line 21, in the same column where the word "Item" is found in line 20.

text 20.5,20,{mcut(10,20,12,40,"","y","y")},cgtimes,12,right will cut text from the data stream, at column 10, row 20, for 12 columns, 40 rows, retaining line breaks, and print it as a column of 40 rows at column 20.5, row 20. The column will be printed in the font CGtimes, 12 point size, right justified.

text 1,60,{mcut(1,60,200,5,"","","")},univers,10,wrap,cols 60 will cut a large message block from the data stream, at column 1, row 60, for 200 columns, 5 rows, removing line breaks. It will then print it at column 1, row 60, at 10 point size and word wrapping to make it fit within 60 columns.

Drivers: all. pdf driver fonts map to Courier, Helvetica, or Times-Roman, and support only symbol set 9J (Windows ANSI characters). Zebra fonts are limited in scalability, and the font codes are letters or numbers that identify internal font codes specified in the ZPL documentation. Zebra shading is limited to 0% or 100%. Zebra doesn't support colors or justification. **Wrap** and **fit** options are only available on pcl and pdf drivers. **Light** and **underline** options are are only supported by the pcl driver.

TITLE

Syntax

title "*titlestring*" | {*expression*}

Description

If this command is present, then PDF document creation adds a title *titlestring*, or the result of *expression*, to the document content. This value is available in the general properties display dialog in the Adobe Acrobat Reader.

Drivers: PDF only

TRAY

Syntax

tray *paper-source*

Description

The **tray** keyword can be used to specify the paper source for any copy or for the print job. If, for example, you have two input trays, one with letterhead stock and one with plain stock, you can specify which paper stock to use for any form or copy of a form.

The *paper-source* is printer dependent. Typically, tray 1 is an upper tray source, tray 2 is a manual feed source, and tray 4 is a lower tray paper source. These will likely not coincide with physical tray numbers labeled on the printer itself, unfortunately. To determine the proper tray values, see your printer's documentation for the paper source command.

Drivers: laser only

UNDERLINE

See the **bold** keyword.

UNITS

Syntax

units dpi | char

Description

As UnForm parses a rule set, column and row specifications are normally interpreted as decimal column and row numbers that align enhancement elements such as boxes and shade regions with characters in the source data. If you need to specify absolute dot positions, however, you can change the units to dpi. From that point in the rule set, until a **units char** is found, row and column values are interpreted as integer dot positions. Note that the **dpi** keyword has a direct impact on dpi units, though no impact on char units.

For example, the following will print two text phrases at column 1 inch, row 1.5 inch.

```
units dpi
text 300,450,"Hello, world"
dpi 600
text 600,900,"Over printing hello world"
units char
```

Drivers: laser, PDF

VLINE

Syntax

vline "*text*" [,erase] [,extend] [,*thickness*]

Description

Any vertical occurrence of the *text* indicated, of at least the length indicated, will be replaced with a vertical line. The *text* must be composed of a single character repeated any number of times. There can be multiple **vline** keywords in a rule set, if needed.

This keyword is useful if the application already produces boxes and lines with standard characters. See also the **hline** keyword.

As with all box drawing, UnForm will consider line end-points to be at the center position of a character, which may impact how lines intersect. Lines are drawn one dot (1/300th inch) thick.

If the "erase" option is used, then no line is drawn. Instead, the vertical text values are simply removed from the output.

If the "extend" option is used, the lines are extended ½ characters up and down. The *thickness* parameter specifies a pixel width to draw.

The search for *text* can be limited to a region on the page by adding a suffix in the format '*@left,top,right,bottom*'. To use a literal "@" character in *text*, it is necessary to specify "\@".

Example:

vline "|" will search the report for pipe characters. All such characters found will be replaced with vertical line draw (box) characters.

Drivers: all

VSHIFT

Syntax

vshift *n*

Description

The **vshift** keyword shifts text vertically down (or up, if *n* is negative) the indicated number of lines. The shifting is done before placement of any fixed shading or boxes. Lines shifted out of the printing region (line 1 through the page specification, or 255 if not specified) are not printed. See the **shift** keyword, also, for horizontal shifting.

The placement of relative shading, drawing, and attributes is determined *before* any shift.

Example:

vshift 1 shifts all text down 1 line, providing room for a box definition at the top of the page.

Drivers: all

WORKING WITH MACROS

Using macros can increase the speed and efficiency of printing your enhanced forms and documents by storing fixed raster graphics (e.g. logos) on the printer instead of transmitting these graphics on every page being printed. With the graphics stored on the printer, only 12 to 14 bytes are transmitted to the printer to select the macro to print. The time savings for printing are most noticeable when your system can't communicate to your printer at a high speed. For parallel or local network connections, macro usage doesn't often make too much difference. However, if you use serial connections or wide area network printing with low- or shared-bandwidth, then implementing macros can help performance. The more graphics used in enhancing forms, the more print transmission time you can save by using macros.

The PCL5 specification defines two types of macros: temporary and permanent. Temporary macros are downloaded at the start of a print job, and can be executed by the printer until it is reset at the end of the job. Permanent macros remain in printer memory until the printer power is turned off. A number from 1 to 32767 always identifies macros.

To access permanent macros, simply add **macro *n*** (n=macro #) to the rule set. To instruct UnForm to utilize temporary macros, add the **macros on** command to the rule set. UnForm will then generate temporary macros for any fixed elements of the job, download them at the start of the job, and execute them as the job is printed.

If you print large batches of forms at one time, and use a serial or low-bandwidth network connection, temporary macros can produce considerable time savings by reducing the amount of data transmitted to the form. For example, if a logo image is 20,000 bytes, and line drawing and shading add another 5,000 bytes, a 50 page form will save about 49 x 25,000 bytes, or about 1.2MB. At typical serial throughput, this could save as much as 10 minutes of print time. High-speed printer connections (parallel or local network) only produce minimal time savings, which is sometimes offset by the extra overhead incurred by UnForm to manage the macros in memory.

UnForm also provides the ability to generate permanent macro files. Permanent macros can be downloaded when the printer is turned on, and then UnForm can execute them without the overhead of downloading them at the start of a job. To utilize this enhanced functionality, you must modify the rule file and create a command line script to load the graphics into the printer.

To use this capability, you should split a rule set into two rule sets. One will be used to generate the permanent macros (there can be a macro for each copy defined in the rule set); the other will be used as before, but will replace the elements placed in the macros with **macro *n*** commands.

The rule set used to generate the macro can contain these commands that are in fixed positions: **image**, **attach**, **box**, **shade**, and **text**. It can also contain **if copy** blocks. It should not contain any other commands or any of the named commands if they incorporate relative positioning. **Detect** commands are ignored; you will use the "-r *ruleset*" command line option instead. The remaining commands should be left in the original rule set, and **macro *n*** commands added based upon the macro numbers assigned in the command described below.

Next, you need to generate macro files for each copy that is used in the rule set. To do this, use this command line:

uf60c –makemacro *macro-number* –f *rulefile* –r *macro-rule-set* –macrocopy *copy* –o *output-file*

UnForm will generate a permanent macro in *output-file*, numbered as *macro-number*. This is the same number you would then specify in the regular rule set, as macro *macro-number*. On UNIX, the output can be piped directly to the spooler, either by removing the –o option or by using a quoted pipe as the output file: –o "|lp –o raw –d *printername*".

REGULAR EXPRESSIONS

Regular expressions are supported in many of UnForm's keywords, and can be used to great advantage in detect statements and relative enhancements. Regular expressions are similar to, but much more powerful than, MS-DOS or UNIX *wildcards*.

A regular expression is used to match patterns in text. By using special characters, called *meta characters*, UnForm can be instructed to search for patterns, such as dates or codes, and use them in processing. Below is a description of the various meta characters and how to use them.

- The simplest regular expression contains no meta characters. It just matches itself. **John** will match any occurrence of the text "John".

- Brackets can be used to match any of a group of values: **[Jj]ohn** will match both "John" and "john".

- If a range of letters or numbers is valid in a position, then the range can be indicated in a similar manner: **[A-Za-z]ohn** will match any letter, upper or lower case, followed by the letters "ohn".

- If single character positions are not enough, then groups of options can be used with parentheses and vertical bars, like this: **(John|Jack|Jill) Smith**, which matches any of the first names, along with "Smith".

- If any character will do in a position, use a dot: **Jo.n** will match "Jo", followed by any single character, followed by "n".

- To repeat any pattern, including a dot, use an asterisk (*) for 0 or more repetitions, or + for 1 or more repetitions: **J.*n** will match a "J", followed by 0 or more characters, followed by "n". **Jo+n** would match a "J" followed by one or more "o"s, followed by "n".

- You can include multiple meta characters and patterns in the expression. For example, to search for 3 digits followed by 2 letters: **[0-9][0-9][0-9][A-Z][A-Z]**.

- To disable the special meaning of any of the meta characters, prefix it with a backslash. For example, a phone number might include parentheses; to include them in the expression, they must be disabled: **\(...\)-...-.....**

- The meta characters are: ., *, +, (,), |, [,], ^, and $.

SAMPLE RULE FILES

UnForm is supplied with several sample report text files and associated rule sets. A description of each report and rule set follows. Each of the sample reports is in the UnForm directory, named "sample*n*.txt." All example rule sets can be found in the files simple.rul and advanced.rul in the UnForm directory.

The simple.rul file contains a series of examples that use the sample invoice text file, sample1.txt. Beginning with the rule set simple1, and incrementally advancing in capabilities through simple4, this rule file is designed to help a new user learn fundamental UnForm concepts. To try these out, use this command, varying the rule set name (-r argument) simple1 with one of the four samples, simple1, simple2, simple3, or simple4:

uf60c –i sample1.txt –f simple.rul –r simple1 –o *output-device*

The advanced.rul file contains rule sets that show a variety of topics, and is designed to show advanced concepts. To produce these samples on your own laser printer or to a PDF file, you can use the following command, substituting the proper sample text file:

uf60c –i *sample-file* –f advanced.rul –o *output-device*

For the output device, you can use a device name, like LPT1 or /dev/lp0, a file name, or a quoted pipe command to a spooler. For example, to print the first sample to a spooler, use something like this:

uf60c –i sample1.txt –f advanced.rul –o "|lp –dhp –oraw"

To produce PDF versions of these files, change the output device to a PDF file name, and add "-p pdf" to the command line:

uf60c –i sample1.txt –f advanced.rul –p pdf –o invoice_sample.pdf.

Change "-o invoice_sample.pdf" to "-o client:invoice_sample.pdf" to store the output on the client's system.

A few of the samples don't support detection capabilities, and they must be specified on the command line with a "-r *ruleset*" option. If necessary, the documentation will state this requirement.

SIMPLE1 - INVOICE RULE SET (SIMPLE.RUL)

This is the first example of an invoice rule set, found in simple.rul. To produce this example:

uf60c –i sample1.txt –f simple.rul –p pdf –o client:simple1.pdf

A title header prefixes all rule sets, which is just a unique name enclosed in brackets.

```
[simple1]
```

Detect statements are used to identify this form from any other report that the application might send to the printer through UnForm. Unlike most form packages, UnForm doesn't dedicate a printer name to a particular form (though it can be configured to do so). Instead, it reads the first page of data, then compares it to the detect statements found in the various rule sets in the rule file.

The detect statements below indicate that

- *a date (mm/dd/yy format) followed by 2 spaces, followed by 7 more characters will appear at column 61, row 5*
- *6 characters will appear at column 9, row 11*
- *a date, a space, and 6 characters will appear at column 10, row 21*

```
detect 61,5,"~../../..  ......."        # invoice date and #
detect 9,11,"~......."                  # customer code
detect 10,21,"~../../.. ......"         # ord date and cust code
```

The following lines define that the dimensions of the page are 80 columns by 66 rows. All positioning will be based on 80 columns and 66 rows appearing within the printed margins of the page.

```
cols 80                                 # max output columns
rows 66                                 # max output rows
```

The header section draws a box around the entire form with a cbox command, then adds a logo and some header text. The" \n" character sequence represents a line break, so you can print a column of text easily. All the text commands are using the univers font, which is standard in all supported laser printers and which maps to Helvetica in PDF output.

```
# header section
cbox .5,.5,80.5,66.5,5
image 1,1,12,6,"sdsilogo.pcl"
text 15,2,"Company Name",univers,14,bold
text 15,3,"Company Address\nCompany City, St Zipcode\nCompany Phone",univers,12,bold
text 15,6,"Web: www.myweb.com\nEmail: sales@myweb.com",univers,11,bold
text 70,2,"INVOICE",univers,16,bold
```

The upper right of the form contains a box with grid lines and some title text, placed around the existing text supplied from the input stream (in this example, the file sample1.txt). The cbox command draws an outer box using the primary dimensions, and then adds internal horizontal lines at rows 6 and 8, and internal vertical lines at columns 69 and 78. The second row simply duplicates the bottom row, but adds 20% shading between rows 6 and 8.

Additional heading and box sections are drawn in a similar manner, for the remainder of the form. All the drawing simply adds details on top of, or around, the input data stream.

```
# invoice # section
cbox 60,4,80.5,8,crows=6 8::20,ccols=69 78
text 61,7,"Date",univers,italic,10
text 70,7,"Invoice #",univers,italic,10
text 79,7,"Pg",univers,italic,10

# bill to / ship to section
cbox .5,10,80.5,18.5,5,ccols=7::20 43.5 50::20
text 2,11,"Sold To",cgtimes,italic,10
text 45,11,"Ship To",cgtimes,italic,10

# ribbon section
cbox .5,18.5,80.5,22.5,5,crows=20.5::20,ccols=9 18 25 65
# special internal grid in ribbon box
cbox 29,18.5,65,21.5
cbox 42,18.5,56,21.5
text 1,19,"Order\nNumber",univers,italic,10
text 10,19,"Order\nDate",univers,italic,10
text 19,19,"Cust.\nNumber",univers,italic,10
text 26,19,"Sls\nPrs",univers,italic,10
text 30,19,"Purchase\nOrder No.",univers,italic,10
text 43,19,"\nShip Via",univers,italic,10
text 57,19,"Ship\nDate",univers,italic,10
text 66,19,"\nTerms",univers,10,italic

# detail section
cbox .5,22.5,80.5,56.5,5,crows=24.5::10,ccols=5 10 16 51 55 69
text 1,23,"Qty\nOrd",univers,italic,10
text 6,23,"Qty\nShip",univers,italic,10
text 11,23,"Qty\nBkord",univers,10,italic
text 17,23,"\nItem & Description",univers,italic,10
text 52,23,"\nU/M",univers,italic,10
text 56,23,"Unit\nPrice",univers,italic,10
text 70,23,"Extended\nPrice",univers,italic,10

# footer section
cbox 57,57,80.5,65,crows=59 63,ccols=69::20
text 58,58,"Sales Amt",univers,11
text 58,61,"Sales Tax",univers,11
text 58,62,"Freight",univers,11
text 58,64.25,"TOTAL",univers,bold,14
```

SIMPLE2 – INVOICE RULE SET (SIMPLE.RUL)

This is a somewhat more advanced rule set than simple1, demonstrating how to add fonting, justification, and text movement to the job. Additional notes are supplied to highlight these concepts. To prevent simple1's detection code from selecting the job, add a –r option to the command line:

uf60c –i sample1.txt –f simple.rul –r simple2 –p pdf –o client:simple2.pdf

```
[simple2]
# to use this rule set, you need to FORCE the rule set with the -r option
# or remark (#) out the detect command in the rule sets above.
#
# This rule set takes the rule set above and improves it by adding
# fonting and justification.

# It also cuts and pastes the invoice #/date/pg fields which allows
# more room for company name and address to be centered

# Also notice first use of relative expression in a text command to fix
# a problem with fonting a series of rows. Put a # in front of this
# command to see the problem that occurs. See memo section.
#

detect 61,5,"~../../..  ......."        # invoice date and #
detect 9,11,"~......"                   # customer code
detect 10,21,"~../../.. ......"         # ord date and cust code

cols 80                                 # max output columns
rows 66                                 # max output rows
```

The header section has changed to use center and right justification. Note the use of cols=79 *in each text command, which tells UnForm the bounds of the justification region. For example, the text "Company Name" is centered in the region starting at column 1, for 79 columns.*

```
# header section
cbox .5,.5,80.5,66.5,5
image 1,1,12,6,"sdsilogo.pcl"
text 1,2,"Company Name",univers,14,bold,center,cols=79
text 1,3,"Company Address\nCompany City, St Zipcode\nCompany
Phone",univers,12,bold,center,cols=79
text 1,6,"Web: www.myweb.com\nEmail: sales@myweb.com",univers,11,bold,center,cols=79
text 1,2,"INVOICE",univers,16,bold,right,cols=79
```

The Invoice number section is re-formatted here, by first drawing a new, vertically-oriented grid and heading section, then by using text commands with expressions that use the cut() function. The expression is indicated by the curly braces, such as {cut(61,5,8,"")}, *which directs UnForm to resolve the function as the job processes each page. In the line starting with* "text 75,5", *the data from the input stream at column 61, row 5, for 8 characters is cut and replaced with nothing (""), and it becomes the value printed at column 75, row 5.*

Further down, in the bill to/ship to section, is an example of the mcut() function, which cuts multiple lines and replaces them with "", retaining line breaks and trimming each line of leading and trailing spaces.

```
# invoice # section
cbox 67,4,80.5,10,1,crows=6 8,ccols=74::20
text 68,5,"Date",univers,italic,10
text 68,7,"Invoice",univers,italic,10
text 68,9,"Page #",univers,italic,10
# cut data from old position and place in new
text 75,5,{cut(61,5,8,"")},cgtimes,bold,10
text 75,7,{cut(71,5,7,"")},cgtimes,bold,10
text 75,9,{cut(79,5,2,"")},cgtimes,bold,10

# bill to / ship to section
cbox .5,10,80.5,18.5,5,ccols=7::20 43.5 50::20
text 2,12,"Sold To",cgtimes,italic,10,center,cols=5
cfont 8,11,40,11,cgtimes,bold,10,left
cfont 8,12,40,15,cgtimes,bold,10                          # sold to address
text 45,12,"Ship To",cgtimes,italic,10,center,cols=5
cfont 51,11,80,11,cgtimes,bold,10,left
text 51,12,{mcut(51,12,30,4,"","Y","Y")},cgtimes,bold,10

# ribbon section
cbox .5,18.5,80.5,22.5,5,crows=20.5::20,ccols=9 18 25 65
# special internal grid in ribbon box
cbox 29,18.5,65,21.5
cbox 42,18.5,56,21.5
text 1,19,"Order\nNumber",univers,italic,10,center,cols=8
text 10,19,"Order\nDate",univers,italic,10,center,cols=8
text 19,19,"Cust.\nNumber",univers,italic,10,center,cols=6
text 26,19,"Sls\nPrs",univers,italic,10,center,cols=3
text 30,19,"Purchase\nOrder No.",univers,italic,10,center,cols=12
text 43,19,"\nShip Via",univers,italic,10,center,cols=13
text 57,19,"Ship\nDate",univers,italic,10,center,cols=8
text 66,19,"\nTerms",univers,italic,10,center,cols=14
```

This section changes the fonts of the input data stream in the invoice ribbon section. For example, the first cfont command changes the data in column 1, row 21 through column 8, row 21, to cgtimes, bold, 10 point, centered text. Note how the font command applies to the incoming data stream, which differs from the text command, which adds additional output to the job. Font commands therefore work with integer positions, as they modify the character-base data stream as it passes through to the output.

```
cfont 1,21,8,21,cgtimes,bold,10,center      # order #
cfont 10,21,17,21,cgtimes,bold,10,center    # order date
cfont 19,21,24,21,cgtimes,bold,10,center    # cust #
cfont 26,21,28,21,cgtimes,bold,10,left      # sls prs code
cfont 26,22,64,22,cgtimes,bold,10,left      # sls prs name
cfont 30,21,41,21,cgtimes,bold,10,center    # po #
cfont 43,21,55,21,cgtimes,bold,10,center    # ship via
cfont 57,21,64,21,cgtimes,bold,10,center    # ship date
cfont 66,21,80,22,cgtimes,10,center         # terms

# detail section
cbox .5,22.5,80.5,56.5,5,crows=24.5::10,ccols=5 10 16 51 55 67
text 1,23,"Qty\nOrd",univers,italic,10,right,cols=4
```

```
text 6,23,"Qty\nShip",univers,italic,10,right,cols=4
text 11,23,"Qty\nBkord",univers,10,italic,right,cols=4
text 17,23,"\nItem & Description",univers,italic,10
text 52,23,"\nU/M",univers,italic,10,center,cols=3
text 56,23,"Unit\nPrice",univers,italic,10,right,cols=11
text 68,23,"Extended\nPrice",univers,italic,10,right,cols=12
```

This section performs two distinct fonting functions. First, the detail item columns are fonted. Note that you can't simply font the entire detail section in a proportional font, as the spacing between columns would be lost. Instead, each column is fonted individually.

However, the data stream for the invoice also contains memo lines in the middle of the detail item lines, and those memo lines should not be broken into individual columns.

Therefore, an additional font command is added after the column fonting, which will override any font characteristics defined for any given data position in a prior font command. This memo section fonting uses a technique that will scan the page for a pattern (in this example, 4 spaces in the region outlined by column 1, row 25 through column 4, row 56), and change font characteristics relative to those locations found. In this example, wherever the 4 spaces are found, fonting will occur 17 columns to the right, 0 rows down, for 60 columns and 1 row. These are the memo lines found in the midst of the item detail lines.

```
cfont 1,25,4,56,cgtimes,10,bold,right        # qty ord
cfont 6,25,9,56,cgtimes,10,bold,right        # qty shipped
cfont 11,25,15,56,cgtimes,10,bold,right      # qty b/o
cfont 17,25,50,56,cgtimes,10,left            # item # & desc
cfont 52,25,54,56,cgtimes,10,bold,center     # u/m
cfont 56,25,66,56,cgtimes,10,bold,right      # unit price
cfont 68,25,79,56,cgtimes,10,bold,right      # extended

# memo section
font "    @1,25,4,56",17,0,60,1,cgtimes,10,left

# footer section
cbox 57,57,80.5,65,crows=59 63,ccols=67::20
text 58,58,"Sales Amt",univers,11
cfont 58,60,66,60,univers,11,left
text 58,61,"Sales Tax",univers,11
text 58,62,"Freight",univers,11
text 58,64.25,"TOTAL",univers,bold,14
cfont 68,58,79,65,cgtimes,bold,right,14                # totals
```

SIMPLE3 – INVOICE RULE SET (SIMPLE.RUL)

This rule set adds copy handling, a watermark, and a barcode. To produce this sample, use this command:

uf60c –i sample1.txt –f simple.rul –r simple3 –p pdf –o client:simple3.pdf

A title header prefixes all rule sets, which is just a unique name enclosed in brackets.

```
[simple3]
# to use this rule set, you need to FORCE the rule set with the -r option
# or remark (#) out the detect command in the rule sets above.
#
# This rule set takes the rule set above and improves it by adding
# copies, watermark, and a barcode.

dsn_sample "sample1.txt"
detect 61,5,"~../../..  ......."        # invoice date and #
detect 9,11,"~......."                  # customer code
detect 10,21,"~../../.. ......"         # ord date and cust code

cols 80                                 # max output columns
rows 66                                 # max output rows
```

This rule set produces two copies of each page, with each copy sequentially produced as each page is read from the data stream. The pcopies command indicates this page-oriented copy production. There is also a copies command, which produces job-oriented copies for laser jobs. Note that PDF output always is produced as page-oriented copies, whether copies or pcopies is used.

When copies are produced, all rule set content that is not bracketed within 'if copy' blocks is produced on all copies. The majority of this rule set is outside of such blocks, so the majority will be applied to both copies. Near the bottom of the rule set is some code that will apply distinctly to each copy.

```
# copies
pcopies 2

# header section
cbox .5,.5,80.5,66.5,5
image 1,1,12,6,"sdsilogo.pcl"
text 1,2,"Company Name",univers,14,bold,center,cols=79
text 1,3,"Company Address\nCompany City, St Zipcode\nCompany
Phone",univers,12,bold,center,cols=79
text 1,6,"Web: www.myweb.com\nEmail: sales@myweb.com",univers,11,bold,center,cols=79
text 1,2,"INVOICE",univers,16,bold,right,cols=79

# invoice # section
cbox 67,4,80.5,10,1,crows=6 8,ccols=74::20
text 68,5,"Date",univers,italic,10
text 68,7,"Invoice",univers,italic,10
text 68,9,"Page #",univers,italic,10
# cut data from old position and place in new
```

```
text 75,5,{cut(61,5,8,"")},cgtimes,bold,10
text 75,7,{cut(71,5,7,"")},cgtimes,bold,10
text 75,9,{cut(79,5,2,"")},cgtimes,bold,10

# bill to / ship to section
cbox .5,10,80.5,18.5,5,ccols=7::20 43.5 50::20
text 2,12,"Sold To",cgtimes,italic,10,center,cols=5
cfont 8,11,40,11,cgtimes,bold,10,left
cfont 8,12,40,15,cgtimes,bold,10                          # sold to address
text 45,12,"Ship To",cgtimes,italic,10,center,cols=5
cfont 51,11,80,11,cgtimes,bold,10,left
text 51,12,{mcut(51,12,30,4,"","Y","Y")},cgtimes,bold,10

# ribbon section
cbox .5,18.5,80.5,22.5,5,crows=20.5::20,ccols=9 18 25 65
# special internal grid in ribbon box
cbox 29,18.5,65,21.5
cbox 42,18.5,56,21.5
text 1,19,"Order\nNumber",univers,italic,10,center,cols=8
text 10,19,"Order\nDate",univers,italic,10,center,cols=8
text 19,19,"Cust.\nNumber",univers,italic,10,center,cols=6
text 26,19,"Sls\nPrs",univers,italic,10,center,cols=3
text 30,19,"Purchase\nOrder No.",univers,italic,10,center,cols=12
text 43,19,"\nShip Via",univers,italic,10,center,cols=13
text 57,19,"Ship\nDate",univers,italic,10,center,cols=8
text 66,19,"\nTerms",univers,italic,10,center,cols=14

cfont 1,21,8,21,cgtimes,bold,10,center      # order #
cfont 10,21,17,21,cgtimes,bold,10,center    # order date
cfont 19,21,24,21,cgtimes,bold,10,center    # cust #
cfont 26,21,28,21,cgtimes,bold,10,left      # sls prs code
cfont 26,22,64,22,cgtimes,bold,10,left      # sls prs name
cfont 30,21,41,21,cgtimes,bold,10,center    # po #
cfont 43,21,55,21,cgtimes,bold,10,center    # ship via
cfont 57,21,64,21,cgtimes,bold,10,center    # ship date
cfont 66,21,80,22,cgtimes,10,center         # terms

# detail section
cbox .5,22.5,80.5,56.5,5,crows=24.5::10,ccols=5 10 16 51 55 67
text 1,23,"Qty\nOrd",univers,italic,10,right,cols=4
text 6,23,"Qty\nShip",univers,italic,10,right,cols=4
text 11,23,"Qty\nBkord",univers,10,italic,right,cols=4
text 17,23,"\nItem & Description",univers,italic,10
text 52,23,"\nU/M",univers,italic,10,center,cols=3
text 56,23,"Unit\nPrice",univers,italic,10,right,cols=11
text 68,23,"Extended\nPrice",univers,italic,10,right,cols=12

cfont 1,25,4,56,cgtimes,10,bold,right        # qty ord
cfont 6,25,9,56,cgtimes,10,bold,right        # qty shipped
cfont 11,25,15,56,cgtimes,10,bold,right      # qty b/o
cfont 17,25,50,56,cgtimes,10,left            # item # & desc
cfont 52,25,54,56,cgtimes,10,bold,center     # u/m
cfont 56,25,66,56,cgtimes,10,bold,right      # unit price
cfont 68,25,79,56,cgtimes,10,bold,right      # extended

# memo section
font "    @1,25,4,56",17,0,60,1,cgtimes,10,left
```

This text line adds a large text watermark on line 56, centered horizontally. The text is printed in cgtimes, 120 point, with 10% shading applied.

```
.
# watermark
text 1,56,"INVOICE",cgtimes,120,shade=10,center,cols=80,fit

# footer section
cbox 57,57,80.5,65,crows=59 63,ccols=67::20
text 58,58,"Sales Amt",univers,11
cfont 58,60,66,60,univers,11,left
text 58,61,"Sales Tax",univers,11
text 58,62,"Freight",univers,11
text 58,64.25,"TOTAL",univers,bold,14
cfont 68,58,79,65,cgtimes,bold,right,14                    # totals
```

The barcode command can be used to add barcodes in many symbologies. It is similar to other commands, in that you provide a column, row, and value. In addition, you specify a symbology (400 is Code 3 of 9), a point size or pixel height (14.0, being a decimal rather than integer value, is treated as point size), and a bar spacing value in pixels. Like most commands, you can use expressions in the value element of the command. In this example, the data stream data from column 9, row 11, for 6 characters is used on each page, using the get() function within an expression.

```
text 2,58,"Customer code as 3 of 9 barcode",univers,italic,10
barcode 2,58.67,{get(9,11,6)},400,14.0,4
```

The following lines produce different output for each of the two copies. Copy 1 is labeled with a text command to say it is the "Customer Copy", while copy 2 is labeled as "Accounting Copy". Any commands outside of 'if copy' blocks are applied to all copies.

```
# copy name section
if copy 1
    text 1,65.5,"Customer Copy",univers,12,bold,center,cols=80
end if
if copy 2
    text 1,65.5,"Accounting Copy",univers,12,bold,center,cols=80
end if
```

SIMPLE4 – INVOICE RULE SET (SIMPLE.RUL)

This rule set demonstrates the use of constants, graphical shading, colors, and expressions to produce explanatory notes in the document. To produce this sample, use this command:

uf60c –i sample1.txt –f simple.rul –r simple4 –p pdf –o client:simple4.pdf

A title header prefixes all rule sets, which is just a unique name enclosed in brackets.

```
[simple4]
# to use this rule set, you need to FORCE the rule set with the -r option
# or remark (#) out the detect command in the rule sets above.
#
# This rule set takes the rule set above and improves it by adding
# constants, graphical shading, increases the resolution,
# and adds explanatory text commands for cust # and ship to #.
#
# Also adds a copy for a packing slip with no prices.

dsn_sample "sample1.txt"
detect 61,5,"~../../..  ......."        # invoice date and #
detect 9,11,"~......."                  # customer code
detect 10,21,"~../../.. ......"         # ord date and cust code
```

Constants are simple text names that are replaced by values later in the rule set. They can be used to simplify maintenance of the rule set, or to make it easier to read. In this example, a series of constants is defined using the const command, and you will find the names referenced throughout the balance of the rule set.

```
const MAXCOLS=80                        # max cols to output
const MAXRCOLS=79                       # MAXCOLS-1
const LEFTCOL=.5                        # use 1 if empty
const RIGHTCOL=80.5                     # use LEFTCOL for symmetry
const MAXROWS=66                        # max rows to output

cols MAXCOLS
rows MAXROWS

# copies
const CUSTOMER_COPY=1
const FILE_COPY=2
const PACK_COPY=3
pcopies 3
```

The dpi setting applies to laser output only, and instructs the printer to produce output at 600 dpi, providing a typically crisper, more professional look. In addition, the gs on *command turns on graphical shading mode, so that shade regions and shaded text are rendered as graphical data rather than using pcl's internal, typically coarse, shade macros.*

```
dpi 600
gs on                                   # turn on graphical shading

# enhancement constants
const HSHADE=30
const ISHADE=20
const DSHADE=10
const HFONT=univers,11
const IFONT=univers,italic,10
const DFONT=cgtimes,10
const DBFONT=DFONT,bold

# header section
cbox LEFTCOL,.5,RIGHTCOL,{MAXROWS+.5},5
image 1,1,12,6,"sdsilogo.pcl"
text 1,2,"Company Name",HFONT,14,bold,center,cols=MAXRCOLS
text 1,3,"Company Address\nCompany City, St Zipcode\nCompany
Phone",HFONT,12,bold,center,cols=MAXRCOLS
text 1,6,"Web: www.myweb.com\nEmail:
sales@myweb.com",HFONT,bold,center,cols=MAXRCOLS
text 1,2,"INVOICE",HFONT,16,bold,right,cols=MAXRCOLS

# invoice # section
cbox 67,4,RIGHTCOL,10,1,crows=6 8,ccols=74::ISHADE
text 68,5,"Date",IFONT
text 68,7,"Invoice",IFONT
text 68,9,"Page #",IFONT
# cut data from old position and place in new
text 75,5,{cut(61,5,8,"")},DBFONT
text 75,7,{cut(71,5,7,"")},DBFONT
text 75,9,{cut(79,5,2,"")},DBFONT
```

The cbox command shown here uses constants defined above, plus shows the use of color options, which are supported by PDF and color laser output. In this example, the interior is colored in cyan, and the lines are colored in blue. Alternately, RGB hex triplets (such as 800000 for dark red) can be specified using the rgb, scolor rgb, or lcolor rgb options.

```
# bill to / ship to section
cerase 1,11,MAXCOLS,11    # erase cust#,ship# used later in text commands
cbox LEFTCOL,11,RIGHTCOL,18.5,5,cyan,lcolor=blue,ccols=7::ISHADE 43.5 50::ISHADE
text 2,12,"Sold To",IFONT,center,cols=5
cfont 8,11,40,11,DBFONT,left
cfont 8,12,40,15,DBFONT                         # sold to address
```

This text command shows an example of how to use an expression to construct a message using a combination of hard-coded text and information from the data stream. In this example, the phrase "Your customer code is" is concatenated with the data at column 9, row 11, for 6 characters, on each page, and the result is printed at column 9, row 18, using the specifications provided by the constant IFONT, defined earlier in the rule set.

```
text 8,18,{"Your customer code is "+get(9,11,6)+"."},IFONT

text 45,12,"Ship To",IFONT,center,cols=5
cfont 51,11,80,11,DBFONT,left
text 51,12,{mcut(51,12,30,4,"","Y","Y")},DBFONT
```

```
text 51,18,{"Your ship to code is "+get(55,11,6)+"."},IFONT

# ribbon section
cbox LEFTCOL,18.5,RIGHTCOL,22.5,5,lcolor=blue,crows=20.5::ISHADE:cyan,ccols=9 18 25
65
# special internal grid in ribbon box
cbox 29,18.5,65,21.5
cbox 42,18.5,56,21.5
text 1,19,"Order\nNumber",IFONT,center,cols=8
text 10,19,"Order\nDate",IFONT,center,cols=8
text 19,19,"Cust.\nNumber",IFONT,center,cols=6
text 26,19,"Sls\nPrs",IFONT,center,cols=3
text 30,19,"Purchase\nOrder No.",IFONT,center,cols=12
text 43,19,"\nShip Via",IFONT,center,cols=13
text 57,19,"Ship\nDate",IFONT,center,cols=8
text 66,19,"\nTerms",IFONT,center,cols=14

cfont 1,21,8,21,DBFONT,center      # order #
cfont 10,21,17,21,DBFONT,center    # order date
cfont 19,21,24,21,DBFONT,center    # cust #
cfont 26,21,28,21,DBFONT,left      # sls prs code
cfont 26,22,64,22,DBFONT,left      # sls prs name
cfont 30,21,41,21,DBFONT,center    # po #
cfont 43,21,55,21,DBFONT,center    # ship via
cfont 57,21,64,21,DBFONT,center    # ship date
cfont 66,21,80,22,DBFONT,center    # terms

# detail section
if copy PACK_COPY
     erase "~\.[0-9][0-9]@62,25,79,56",-6,0,11,1
endif
cbox LEFTCOL,22.5,RIGHTCOL,56.5,5,crows=24.5::DSHADE,ccols=5 10 16 51 55 67
text 1,23,"Qty\nOrd",IFONT,right,cols=4
text 6,23,"Qty\nShip",IFONT,right,cols=4
text 11,23,"Qty\nBkord",IFONT,right,cols=4
text 17,23,"\nItem & Description",IFONT
text 52,23,"\nU/M",IFONT,center,cols=3
text 56,23,"Unit\nPrice",IFONT,right,cols=11
text 68,23,"Extended\nPrice",IFONT,right,cols=12

cfont 1,25,4,56,DBFONT,right        # qty ord
cfont 6,25,9,56,DBFONT,right        # qty shipped
cfont 11,25,15,56,DBFONT,right      # qty b/o
cfont 17,25,50,56,DFONT,left        # item # & desc
cfont 52,25,54,56,DBFONT,center     # u/m
cfont 56,25,66,56,DBFONT,right      # unit price
cfont 68,25,79,56,DBFONT,right      # extended

# memo section
font "    @1,25,4,56",17,0,60,1,DFONT,left

# watermark
if copy CUSTOMER_COPY,FILE_COPY
     text 1,56,"INVOICE",DFONT,120,shade=DSHADE,center,cols=MAXCOLS,fit
endif
if copy PACK_COPY
     text 1,56,"PACK SLIP",DFONT,120,shade=DSHADE,center,cols=MAXCOLS,fit
endif

# footer section
```

```
cbox 57,57,RIGHTCOL,65,lcolor=red,crows=59 63,ccols=67::HSHADE
text 58,58,"Sales Amt",HFONT
cfont 58,60,66,60,HFONT,left
text 58,61,"Sales Tax",HFONT
text 58,62,"Freight",HFONT
text 58,64.25,"TOTAL",HFONT,bold,14
cfont 68,58,79,65,DBFONT,right,14                  # totals

text 2,58,"Customer code as 3 of 9 barcode",IFONT
barcode 2,58.67,{get(9,11,6)},400,14.0,4
```

Note the use of constants to make this section easier to read.

```
# copy name section
if copy CUSTOMER_COPY
    text 1,65.5,"Customer Copy",HFONT,12,bold,center,cols=MAXCOLS
end if
if copy FILE_COPY
    text 1,65.5,"Accounting Copy",HFONT,12,bold,center,cols=MAXCOLS
end if
if copy PACK_COPY
    text 1,65.5,"Packing Slip",HFONT,12,bold,center,cols=MAXCOLS
end if
```

This text line demonstrates the use of multi-line text forced to fit within a certain number of columns. UnForm scans each of the two lines (delimited by the \n character sequence, or it could contain data with line-feed or carriage-return line-feed delimiters) to determine the width, beginning with the point size 12 specified in the command. The size is reduced until both lines will fit within the 20 columns specified with the cols option. Once the correct point size is determined, the lines are spaced normally for that height. For example, if the size required is 8.25 points, then the lines will be spaced 8.25 points apart. If spacing had been set to 1.5, then the lines would be spaced 12.33 points apart.

```
text 2,62,"This sample message text, which contains\nline breaks,  is sized to fit
in 20 columns.",cols 20,cgtimes,12,fit,spacing 1
```

INVOICE - INVOICE FOR PRE-PRINTED FORM (ADVANCED.RUL)

This sample is an invoice that is intended for a pre-printed form. The data generated by the application doesn't include any headings or simulated line drawing like a plain-paper invoice might. In this case, UnForm must simulate the entire pre-printed invoice form.

uf60c –i sample1.txt –f advanced.rul –p pdf -o client:invoice.pdf

A title header prefixes all rule sets, which is just a unique name enclosed in brackets.

```
[Invoice]
```

Detect statements are used to distinguish this form from any other report that the application might send to the printer through UnForm. Unlike most form packages, UnForm doesn't dedicate a printer name to a particular form (though it can be configured to do so). Instead, it reads the first page of data, then compares it to the detect statements found in the various rule sets in the rule file.

The detect statements below indicate that

- *a date (mm/dd/yy format) followed by 2 spaces, followed by 7 more characters will appear at column 61, row 5*
- *6 characters will appear at column 9, row 11*
- *a date, a space, and 6 characters will appear at column 10, row 21*

```
detect 61,5,"~../../..  ......."        # invoice date and #
detect 9,11,"~......."                  # customer code
detect 10,21,"~../../.. ......"         # ord date and cust code
```

The following lines define several constants that are used elsewhere in the rule set. Wherever the constant names appear in a command, the value is substituted. Constants are not variables and are not interpreted while the job is processed. They are simply literal placeholders used while UnForm reads rule set lines.

```
# set up document constants
const MAXCOLS=80                        # max cols to output
const MAXRCOLS=79                       # MAXCOLS-1
const LEFTCOL=.5                        # use 1 if empty
const RIGHTCOL=80.5                     # use LEFTCOL for symmetry
const MAXROWS=66                        # max rows to output
```

The following lines define the page size and orientation. The dpi command sets the printer to 600 dots per inch. The rows and cols commands set the dimension for positioning and scaling. All positioning will be based on 80 columns and 66 rows appearing within the printed margins of the page. The gs *on*

command triggers the use of graphical shading, which improves the look of shade regions over the native pcl shading of most laser printers, especially at higher dpi settings and shade percentages. In addition, UnForm will generate two copies of the job, with each page producing two copies as processed (collated).

```
portrait
dpi 600
gs on                                   # graphical shading
cols MAXCOLS                            # max output columns
rows MAXROWS                            # max output rows

# to print more copies, increase value and add copy titles in prejob
pcopies 2                               # max # of copies
```

If this rule set is used to produce a PDF document, then the title of "Sample Invoice" will be added to the PDF file. For laser output, the title command is ignored.

```
title "Invoice Sample"                  # view in PDF properties
```

The prejob code block is executed once at the beginning of the job, after the first page of data has been read and the rule set parsed. This example is simply setting a variable form_title$ *to a literal value INVOICE. This variable is used later in the rule set.*

The prepage code block is executed once per page, just after UnForm has read the text for the page, but before any copies of that page have been printed. Within a prepage code block, you can insert any valid Business Basic code (though you need to be careful not to insert any UnForm commands.) This code initializes a variable shipzip$ to null, then looks for a regular expression pattern of 5 digits on line 15. If the pattern is found, it sets shipzip$ to the zip code. After the code block is closed, a barcode command is used to place a postnet barcode below the shipping address. The barcode command uses the syntax "{shipzip$}", indicating the expression shipzip$ should be used to generate the data to barcode.

Once the prepage code block creates shipzip$, it then scans a range of rows looking for special memo format lines. It marks these lines with the characters "mL" in the first two columns. Later in the rule set, you'll see how these markers are used to treat memo lines differently than standard invoice lines.

The order of execution is controlled by UnForm. There is actually no need to place the barcode command below the prepage code block, as UnForm will properly execute the code block before any form commands are executed at run-time.

```
prejob {
     # set up variables needed by merged routines below
     # if form title changes per page,
     # set up in prepage routine below
```

```
     form_title$="INVOICE"
}

prepage {
     # find zip code in city,state,zip line for bar code
     shipzip$=""
     # regular expression of 5 digits on line 15
     x=mask(text$[15],"[0-9][0-9][0-9][0-9][0-9]")
     if x>0 then shipzip$=get(x,15,5)

     # mark memo lines for special handling in detail section below
     # memo start in column 28 with all spaces before
     for i=25 to 56
          if len(text$[i])>27 and trim(text$[i](1,27))="" then \
               text$[i](1,2)="mL"
     next i
}
```

The pdf driver supports the ability to email the PDF file created using the email command. The commented # email line below provides an example of the command. It requires four arguments, each of which can be a literal string value or a string expression enclosed in braces. In order for the email command to work, the mailcall.ini file must be properly configured for your system.

```
# When run in PDF mode, and if mailcall.ini is configured properly,
# and if the system can communicate with the mail server, then the
# next line would send the PDF invoice as an attachment to an email.
# email "someone@somewhere.com","me@mycompany.com", \
#    {"A test invoice "+cvs(get(71,5,7),3)}, \
#    "Attached is a sample invoice\n"
```

The next group of commands creates a page header with box and text commands. The box commands are given as the cbox variant, which accepts two pairs of numbers as opposite corners of the box. Some of the commands are stored in a different rule set, called "Mrg Form Header". This rule set is also located in the advanced.rul file. The lines in that rule set are merged in here as if they were part of this rule set.

Note that some of the text commands, and also a barcode command, use an expression rather than a literal. An expression is an executable value assignment enclosed in braces. For example, one text command uses an expression {cut(61,5,8,"")}*,which cuts out the text at column 61, row 5, for 8 columns, returning the result, while setting those positions to "". The result is printing at position 75,5 what was at position 61,5.*

```
# heading section
const HFONT=univers,12                          # headings
cbox LEFTCOL,1,RIGHTCOL,MAXROWS,5               # complete page box
merge "Mrg Form Header"                         # merge std hdr rules
```

```
# right top ribbon
const HFONT=univers,11,italic                # headings
const DFONT=cgtimes,11,bold                  # data

# draw info box with internal grid and shading
# horizontal lines at 6 and 8
# vertical line at 74 with shading between 67 and 74
cbox 67,4,RIGHTCOL,10,5,crows=6 8,ccols=74::20
text 68,5,"Date",HFONT
text 68,7,"Invoice",HFONT
text 68,9,"Page #",HFONT
# cut data from old position and place in new
text 75,5,{cut(61,5,8,"")},DFONT
text 75,7,{cut(71,5,7,"")},DFONT
text 75,9,{cut(79,5,2,"")},DFONT

# sold to section
cbox LEFTCOL,10,41,18.5,5
cbox LEFTCOL,10,41,11.25,0,10
text 8,10.75,"SOLD TO",HFONT,bold
cfont 8,12,40,15,DFONT                       # sold to address
if copy 1
     barcode 8,16,{shipzip$},900,9.0,2
end if
text 2,18,{"Your customer code is "+cut(9,11,6,"")+"."},8,cgtimes

# ship to section
cbox 41,10,RIGHTCOL,18.5,5
cbox 41,10,RIGHTCOL,11.25,0,10
text 48,10.75,"SHIP TO",HFONT,bold
# cut ship to address and place in new position
text 48,12,{mcut(51,12,30,4,"","Y","Y")},DFONT
text 43,18,{"Your ship to code is "+cut(55,11,6,"")+"."},8,cgtimes
```

This section draws order detail boxes and headings. The first cbox command draws a grid, using the internal crows and ccols options. In addition to the boxes and headings, the font used for the data from the input stream is changed using a series of cfont commands, one for each section.

```
# ribbon section
const L1=19
const L2=20
# draw info box with internal grid and shading
# horizontal line at 20.5 with shading between 18.5 and 20.5
# vertical lines at 9, 18, 25, and 65
cbox LEFTCOL,18.5,RIGHTCOL,22.5,5,crows=20.5::20,ccols=9 18 25 65
# special internal grid in ribbon box
cbox 29,18.5,65,21.5
```

```
cbox 42,18.5,56,21.5
# ribbon headings
text 1,L1,"Order",HFONT,right,cols=8
text 1,L2,"Number",HFONT,right,cols=8
text 10,L1,"Order",HFONT,center,cols=8
text 10,L2,"Date",HFONT,center,cols=8
text 19,L1,"Cust.",HFONT
text 19,L2,"Number",HFONT
text 26,L1,"Sls",HFONT
text 26,L2,"Prs",HFONT
text 30,L1,"Purchase",HFONT
text 30,L2,"Order No.",HFONT
text 43,L2,"Ship Via",HFONT
text 57,L1,"Ship",HFONT,center,cols=8
text 57,L2,"Date",HFONT,center,cols=8
text 66,L2,"Terms",HFONT
# ribbon data
cfont 1,21,8,21,DFONT,right                          # order #
cfont 10,21,17,21,DFONT,center                       # order date
cfont 19,21,24,21,DFONT                              # cust #
cfont 26,21,28,21,DFONT                              # sls prs code
cfont 26,22,64,22,DFONT                              # sls prs name
cfont 30,21,41,21,DFONT                              # po #
cfont 43,21,55,21,DFONT                              # ship via
cfont 57,21,64,21,DFONT,center                       # ship date
cfont 66,21,MAXCOLS,22,DFONT                         # terms
```

This section of lines controls the formatting of the invoice detail lines. A grid is drawn around the column headers and detail lines. The column headers are shaded. Item detail lines are fonted using a series of font commands that look for the pattern "~\.[0-9][0-9][0-9][0-9]" *which is a period followed by 4 digits. Wherever that occurs, font changes are made relative to that position. Similarly, the memo lines identified by the prepage code block and marked with the text marker "mL" are fonted with a different column structure. In addition to the font command, an erase command is used to remove the text markers.*

```
# detail section
# detail headings
const L1=23
const L2=24
# draw info box with internal grid and shading
# horizontal line at 24.5 with shading between 22.5 and 24.5
# vertical lines at 5, 10, 16, 51, 55, and 67
cbox LEFTCOL,22.5,RIGHTCOL,56.5,5,crows=24.5::10, \
     ccols=5 10 16 51 55 67
text 1,L1,"Qty",HFONT,right,cols=4
text 1,L2,"Ord",HFONT,right,cols=4
text 6,L1,"Qty",HFONT,right,cols=4
text 6,L2,"Ship",HFONT,right,cols=4
text 11,L1,"Qty",HFONT,right,cols=5
```

```
text 11,L2,"Bkord",HFONT,right,cols=5
text 20,L2,"Item & Description",HFONT
text 52,L2,"U/M",HFONT,center,cols=3
text 56,L1,"Unit",HFONT,right,cols=11
text 56,L2,"Price",HFONT,right,cols=11
text 68,L1,"Extended",HFONT,right,cols=12
text 68,L2,"Price",HFONT,right,cols=12
# detail data
# Modify fonts for lines. As comments may be present in the same rows,
# use a pattern to locate the .nnnn in the price column,
# which indicates a part number line.
# Use a prepage routine to find the comments and change their font.
font "~\.[0-9][0-9][0-9][0-9]",-61,0,4,1,DFONT,right   # qty ord
font "~\.[0-9][0-9][0-9][0-9]",-56,0,4,1,DFONT,right   # qty shipped
font "~\.[0-9][0-9][0-9][0-9]",-50,0,4,1,DFONT,right   # qty b/o
font "~\.[0-9][0-9][0-9][0-9]",-42,0,30,2,DFONT        # item # & desc
font "~\.[0-9][0-9][0-9][0-9]",-10,0,3,1,DFONT,center  # u/m
font "~\.[0-9][0-9][0-9][0-9]",-6,0,11,1,DFONT,right   # unit price
font "~\.[0-9][0-9][0-9][0-9]",6,0,12,1,DFONT,right    # ext price

# handle memo lines
# inserted 'mL' in prepage above
font "mL@1,25,2,56",10,0,63,1,HFONT
erase "mL@1,25,2,56",0,0,2,1
```

Watermark text is placed in the middle of the detail lines. This text is centered between column 1 and MAXCOLS, is rendered at 120 points, and is printed at 20% gray shading.

```
# watermark - large font with light shading
text 1,52,{form_title$},cgtimes,120,shade 20,center,cols=MAXCOLS
```

The totals section is formatted like the other sections, with a grid, text headings, and font changes that apply to the input stream text.

```
# totals section
# draw info box with internal grid and shading
# horizontal lines at 59 and 63
# vertical line at 69 with shading between 58 and 69
cbox 58,57,RIGHTCOL,65,5,ccols=69::20,crows=59 63
text 59,58,"Sales Amt",HFONT
text 59,61,"Sales Tax",HFONT
text 59,62,"Freight",HFONT
text 59,64.25,"TOTAL",HFONT,bold,14
cfont 59,60,68,60,HFONT                                 # disc hdr
cfont 70,58,MAXRCOLS,65,DFONT,14,decimal                # totals
```

These text lines simply demonstrate some of UnForm's paragraph features. The first text command forces the longest line in the paragraph to fit within the number of defined columns. The maximum

point size is 12, but that may be adjusted down to accommodate the longest line. Lines are delimited by the \n character sequence, or by a CHR(10) within an expression. Line spacing is determined by the final point size, and may be adjusted with the spacing option. For example, if the rendered size is 8 point, then the spacing of 1 will result in 9 lines per inch (9 x 8=72), while spacing of 1.5 would result in 6 lines per inch (9/1.5=6).

The second example will force use the defined point size to render the text, but any lines wider than the specified columns will be word-wrapped.

The third example shows how to use a specified ASCII value in a text command. The ASCII value 174, when printed using the symbol set 9J, is a trademark symbol. This technique can be used to print Latin characters and special symbols. The symbol set determines what any given character value prints as. The 9J symbol set is the default. See the –testpr command line option for viewing printed tables of different symbol sets.

```
# footer section
# These lines show fitting and wrapping of text
text 2,60,"This sample message text, which contains\nline breaks, \
     is sized to fit in 20 columns.",cols 20,cgtimes,12, \
     fit,spacing 1

text 28,60,"This sample message text is word wrapped to not exceed \
     20 columns, while retaining the specified 12 point size.",\
     cgtimes,cols 20,12,wrap,spacing 1

text 2,64,"This sample was generated by UnForm<174>.",7,cgtimes, \
          symset 9J,blue
```

This set of commands places the phrase "Customer Copy" on copy 1, and "Remittance Copy" on copy 2. The text is placed at row 65.5, and is centered within the columns defined at column 1 and the constant MAXCOLS, which represents the whole page width.

```
# copy name section
const ROW=65.5
if copy 1
     text 1,ROW,"Customer Copy",HFONT,bold,center,cols=MAXCOLS
end if
if copy 2
     text 1,ROW,"Accounting Copy",HFONT,bold,center,cols=MAXCOLS
end if
```

STATEMENT - PLAIN PAPER FORM, TWO PAGE FORMATS IN SAME JOB (ADVANCED.RUL)

In this sample, a two-page, plain paper statement is printed. The two pages contain two slightly different formats, with the second page containing detail lines and a customer aging, and the first page containing some more detail lines and the phrase "CONTINUED" at the bottom. In the same statement print run, some statements may contain a single page, others two or more pages.

The trick here is to get UnForm to produce two formats based on the content of each page. In order to accomplish this, we define the job to produce multiple copies, and assign certain copies to certain formats. Using a precopy{} code block, we can then control the printing of the different formats.

uf60c –i sample2.txt –f advanced.rul –p pdf –o client:statement.pdf

This statement header identifies this rule set.

```
[Statement]
```

The word STATEMENT appears at column 34, row 2, and a date appears at column 65, row 7. To further clarify, a date format is matched at position 65, 7.

```
detect 34,2,"STATEMENT"
detect 65,7,"~../../.."                 # statement date
```

The page dimensions are 66 rows and 75 columns. The text input to UnForm doesn't contain any form-feeds to indicate the end of a page, so the command "page 66" tells UnForm to consider each 66 lines to be a page.

Pcopies 4 is used to tell UnForm to print 4 copies of each page, with copies following each other in sequence for each page (collated). You will find later that UnForm doesn't actually print all copies of each page, but instead simply prints selected copies, depending on the format required. As each page is processed, if the page contains aging totals, UnForm prints 2 copies of that format, and if it does not contain aging totals, then UnForm prints 2 copies of the second format.

```
# set up document constants
const MAXCOLS=75                        # max cols to output
const MAXRCOLS=74                       # MAXCOLS-1
const LEFTCOL=1                         # use 1 if empty
const RIGHTCOL=75                       # MAXCOLS for symmetry
const MAXROWS=66                        # max rows to output

portrait
```

```
dpi 300
gs on                                   # graphical shading
cols MAXCOLS                            # max output columns
rows MAXROWS                            # max output rows
page MAXROWS                            # no form-feeds used

# to print more copies, increase value and add copy titles in prejob
# Copy 1,2     Statement with aging totals
# Copy 3,4     Statement w/o  aging totals
pcopies 4                               # max # of copies
```

If this rule set is used to produce a PDF document, then the title "Statement Sample" will be added to the PDF file. For laser output, the title command is ignored.

```
title "Statement Sample"                # view in PDF properties
```

The prejob command defines a string variable form_title$, assigning it the value "STATEMENT". This variable is used later in the rule set for a page heading and also in a watermark.

```
prejob {
     # set up variables needed by merged routines below
     # if form title changes per page,
     # set up in prepage routine below
     form_title$="STATEMENT"
}
```

The prepage code block performs 2 functions. It checks the input data for the word "CONTINUED" at position 66, 64. If that word is present, then a variable continued$ is assigned to the phrase "Continued"; otherwise it is set to null. In addition, at three individual lines (16, 62, and 64), there may be single ! characters used as character-mode vertical lines in the input data. Elsewhere in the rule set is a 'vline "!!", erase' command, which erases instances of 2 or more ! characters vertically on the page. This code takes care of the single-row instances.

```
prepage {
     # get continued if it exists
     continued$=get(66,64,9)
     if continued$="CONTINUED" then continued$="Continued" \
          else continued$=""

# remove single ! from line draw regions
        x=pos("!"=text$[16]; \
        while x>0; text$[16](x,1)="",x=pos("!"=text$[16]);wend

        x=pos("!"=text$[62]; \
        while x>0; text$[62](x,1)="",x=pos("!"=text$[62]);wend
```

```
        x=pos("!"=text$[64]; \
        while x>0; text$[64](x,1)="",x=pos("!"=text$[64]);wend

}
```

The precopy code block is executed as each of the 4 copies are about to be printed. The logic here indicates the copies 1 and 2 are for pages that do not contain the word "CONTINUED" (remember the prepage code block?), and copies 3 and 4 do contain that word. By setting the variable skip to a non-zero value, the copy being processed is skipped. Only 1 of the 2 formats is printed, depending on the content of the page.

```
precopy {
     if copy=1 or copy=2 then if continued$="Continued" then skip=1
     if copy=3 or copy=4 then if continued$<>"Continued" then skip=1
}
```

The following lines remove most of the existing character-mode line drawing elements from the input data. The hline and vline commands scan for places where at least the indicated number of characters, horizontally or vertically, occur on the page. The erase option removes them rather than replacing them with graphical lines.

```
#remove existing lines
hline "--",erase
hline "==",erase
vline "!!",erase
cerase 1,1,1,MAXROWS                          # erase all 1st column
cerase MAXCOLS,1,MAXCOLS,MAXROWS              # erase all last column
```

The following lines draw the page headings. Some of the commands are stored in another rule set, "Mrg Form Header", which is merged as the rule set is parsed. The headings already exist, and are moved and fonted with text commands using expressions, such as {cut(66,5,4,"")}.

```
# heading section
const HFONT=univers,12                        # headings
cerase 1,1,MAXCOLS,10
cbox LEFTCOL,1,RIGHTCOL,MAXROWS,5             # complete page box
merge "Mrg Form Header"                       # merge std hdr rules

# right top ribbon section
const HFONT=univers,11,italic                 # headings
const DFONT=cgtimes,11,bold                   # data
# draw info box with internal grid and shading
# horizontal line at 6
# vertical line at 68 with shading between 63 and 68
```

```
cbox 63,5,MAXCOLS,9,5,crows=7,ccols=68::20
text 64,6,{cut(66,5,4,"")},HFONT              # page #
text 64,8,{cut(59,7,4,"")},HFONT              # date
text 69,6,{trim(cut(71,5,3,""))},DFONT        # page #
text 69,8,{trim(cut(65,7,8,""))},DFONT        # date

# customer section
# draw info box with internal grid and shading
# vertical line at 10 with shading between 1 and 10
cbox LEFTCOL,10,MAXCOLS,15,5,ccols=10::10
text 2,11,{cut(2,10,2,"")},HFONT              # to
text 4,13,{trim(cut(15,10,10,""))},DFONT          # cust code
cfont 12,11,MAXCOLS,14,DFONT                  # address
```

The detail section contains several columns of information. There are fewer detail lines on pages with the aging data, so the grid drawing is made specific to particular formats with "if copy 1,2" and "if copy 3,4" sections. Then two groups of font changes are used, first for the column headings and then for the data columns.

```
# detail section
# detail headings
# draw info box with internal grid and shading
# horizontal line at 6
# vertical line at 68 with shading between 63 and 68
if copy 1,2
     cbox LEFTCOL,15,MAXCOLS,56,5,crows=17::20, \
          ccols=10 18 27 39 48 60 63
end if
if copy 3,4
     cbox LEFTCOL,15,MAXCOLS,61,5,crows=17::20, \
          ccols=10 18 27 39 48 60 63
end if
const ROW=16
cfont 2,ROW,9,ROW,HFONT,center                    # date
cfont 11,ROW,17,ROW,HFONT                         # inv #
cfont 19,ROW,26,ROW,HFONT,center                  # due date
cfont 28,ROW,38,ROW,HFONT,right                   # due amt
cfont 40,ROW,47,ROW,HFONT,center                  # pmt date
cfont 49,ROW,59,ROW,HFONT,right                   # pmt amt
cfont 61,ROW,62,ROW,HFONT,center                  # type
cfont 64,ROW,MAXRCOLS,ROW,HFONT,right             # balance
# detail data
const DFONT=cgtimes,11                            # data
cfont 2,18,9,60,DFONT,center                      # date
cfont 11,18,17,60,DFONT                           # inv #
cfont 19,18,26,60,DFONT,center                    # due date
cfont 28,18,38,60,DFONT,right                     # due amt
cfont 40,18,47,60,DFONT,center                    # pmt date
```

```
cfont 49,18,59,60,DFONT,right                       # pmt amt
cfont 61,18,62,60,DFONT,center                      # type
cfont 64,18,MAXRCOLS,60,DFONT,right,BOLD            # balance
```

A watermark prints the form title as large, lightly shaded text. Its position depends upon the format, hence the use of if copy blocks.

```
# watermark - large font with light shading and rotation
if copy 1,2
     text 39,56,{form_title$},cgtimes,75,shade 20,center, \
          cols=MAXCOLS,rotate 90
end if
if copy 3,4
     text 44,61,{form_title$},cgtimes,85,shade 20,center, \
          cols=MAXCOLS,rotate 90
end if
```

The footer section differs considerably between the two formats. Copies 1 and 2 are associated with pages that have aging data, so you see the fonting of the aging columns defined there. Copies 3 and 4 are printed when the word "CONTINUED" appears, and that word is printed below, though as the value stored in continued$ ("Continued").

```
# footer section
# remarks
if copy 1,2
     cbox LEFTCOL,56,RIGHTCOL,61,5
     cfont 2,57,MAXRCOLS,60,HFONT
endif
# totals
const DFONT=cgtimes,11,bold                         # data
if copy 1,2
     cbox LEFTCOL,61,RIGHTCOL,64.5,5,crows=63::20, \
          CCOLS=14 26 38 50 62
     const ROW=62
     cfont 1,ROW,13,ROW,HFONT,right                      # current
     cfont 15,ROW,25,ROW,HFONT,right                     # 1-15
     cfont 27,ROW,37,ROW,HFONT,right                     # 16-30
     cfont 39,ROW,49,ROW,HFONT,right                     # 31-45
     cfont 51,ROW,61,ROW,HFONT,right                     # over 45
     cfont 63,ROW,MAXRCOLS,ROW,HFONT,right,bold,12       # total due
     const ROW=64
     cfont 1,ROW,13,ROW,DFONT,right                      # current
     cfont 15,ROW,25,ROW,DFONT,right                     # 1-15
     cfont 27,ROW,37,ROW,DFONT,right                     # 16-30
     cfont 39,ROW,49,ROW,DFONT,right                     # 31-45
     cfont 51,ROW,61,ROW,DFONT,right                     # over 45
     cfont 63,ROW,MAXRCOLS,ROW,DFONT,right,bold,12       # total due
```

```
endif

if copy 3,4
     cerase 1,62,MAXCOLS,66
     text 1,65,{Continued$},HFONT,right,cols=MAXRCOLS
endif
```

Finally, within the two formats are two physical copies. Each of these copies is either for the customer to keep or for the customer to return with their payment. Copy 1, the first page of format 1, and copy 3, the first page of format 2, get the "Customer Copy" footer. The others get the "Remittance Copy" footer.

```
# copy name section
const ROW=65.5
if copy 1,3
     text 1,ROW,"Customer Copy",HFONT,bold,center,cols=MAXCOLS
end if
if copy 2,4
     text 1,ROW,"Remittance Copy",HFONT,bold,center,cols=MAXCOLS
end if
```

AGING REPORT - ENHANCED AGING REPORT (ADVANCED.RUL)

In this third example, an aging report is enhanced to be more readable. This shows the use of *relative* enhancements, which are those applied relative to the occurrence of text or regular expressions anywhere on the page.

uf60c –i sample3.txt –f advanced.rul –p pdf –o client:aging.pdf

This statement header identifies this rule set.

```
[AgingReport]
```

The only detect statement required is this one, looking for the report title at column 50, row 2.

```
detect 50,2,"Detail Aging Report"
```

These constants are used throughout the rule set.

```
# set up document constants
const MAXCOLS=131                       # max cols to output
const MAXRCOLS=130                      # MAXCOLS-1
const LEFTCOL=.5                        # use 1 if empty
const RIGHTCOL=131.5                    # LEFTCOL for symmetry
const MAXROWS=66                        # max rows to output
```

This report should print in landscape orientation, rather than the default portrait. UnForm will scale the columns and rows to 131 by 66.

```
landscape
dpi 1200
gs on                                   # graphical shading
cols MAXCOLS                            # max output cols
rows MAXROWS                            # max output rows

pcopies 1                               # max # of copies
```

The title "Aging Sample" will appear in PDF document properties. It is ignored for laser output.

```
title "Aging Sample"                    # view in PDF properties
```

The following prejob code demonstrates the use of sdOffice™ to mine data from this report and export it to Microsoft Excel®. SdOffice can be running anywhere on your network on a system with Excel. The code relies on your setting two variables correctly. First, the sdo$ variable should be set to the path to the sdOffice client program sdofc_e.bb. In addition, the value of gbl("$sdhost") needs to be set to the address or hostname of the system running sdOffice. An optional way of doing this is to define an environment variable prior to running UnForm, called SDHOST. If you use that alternative, then comment out the x$=gbl("$sdhost") line.

The code here contains enough error handling to ignore the code if sdOffice isn't present or fails to execute.

```
prejob {
     # set up sdOffice export to Excel
     # set to path to your sdoffice *.pv programs
     sdo$="/u0/sdofc/sdofc_e.pv"

     # You can set the environment variable SDHOST, or use this
     # stbl function to define the sdOffice server address
     x$=gbl("$sdhost","bcj")

     # initialize excel
     call sdo$,err=prejob_done,"newbook","",errmsg$
     if errmsg$>"" then goto prejob_done
     sdofc_init=1
     call sdo$,"show","",""
     call sdo$,"setdelim |","",""
     call sdo$,"writerow ID|Name|Phone|Over 60|Total","",""
     call sdo$,"format row=1,font=Arial,size=12,bold","",""
prejob_done:
}
```

The prepage code block starts with code that exports data to Excel, but only if the prejob code block successfully initializes the sdOffice connection. In addition to that code, it also sets two numeric variables, colw and scol, based upon positions and widths of report column headers. These values are used later in the rule set for fonting and line drawing.

```
prepage{
     # if prejob hasn't initialized sdoffice, skip this code
     if sdofc_init<>1 then goto sdofc_complete

     for row=1 to 66
          ln$=text$[row]

          # customer heading row contain phone numbers
          x=mask(ln$,"\(...-...-....\)")
```

```
          while x
               custid$=mid(ln$,1,6)
               custname$=trim(mid(ln$,8,30))
               custphone$=trim(mid(ln$,38,14))
               x=0
          wend

          # totals - 50 plus spaces followed by digit-.-digit-digit
          x=mask(ln$,"^"+fill(50)+".*[0-9]\.[0-9][0-9]")
          while x
               amount60=cnum(mid(ln$,87,11))
               amount90=cnum(mid(ln$,98,11))
               amount120=cnum(mid(ln$,109,11))
               over60=amount60+amount90+amount120
               total=cnum(mid(ln$,120,11))

               export$=custid$+"|"+custname$+"|"+custphone$+"|"
               export$=export$+str(over60)+"|"+str(total)
               call sdo$,"writerow "+export$,"",""
               x=0
          wend

     next row
sdofc_complete:

     # Now for some tricky code.
     # Agings can have different headings and column widths
     # To use version 5 features allowing variable columns and rows,
     # the following code will calculate starting positions
     # and column widths. It assumes a consistency in column widths,
     # 1 char negative, and 1 blank space between each column
     hd1$=text$[7]                 # temp heading line with agings
     x=pos("Type"=hd1$)
     xhd1$=trim(hd1$(x+4))         # remove all except agings
     x=pos(" "=xhd1$)
     x$=xhd1$(1,x-1)               # get first column header
     xhd1$=trim(xhd1$(x))
     x=pos(x$=hd1$)                # find true position
     x1=x+len(x$)-1                # get end of first column
     # now find end of 2nd column
     x=pos(" "=xhd1$)
     x$=xhd1$(1,x-1)               # get second column header
     x=pos(x$=hd1$)
     x2=x+len(x$)-1                # get end of second column
     # now calculate mask width less space between columns
     colw=x2-x1-1
     # now calculate start of first field
     scol=x1-colw+2
}
```

The postjob code block performs some closing formatting control if the job is exporting data to Excel. If sdOffice is not being used, based upon the attempt to initialize it in the prejob code block, then this code is skipped.

```
postjob{
     # if prejob hasn't initialized sdoffice, skip this code
     if sdofc_init<>1 then goto sdofc_complete2

     call sdo$,"leaveopen","",""
     call sdo$,"format autofit","",""
     call sdo$,"format col=1,numberformat=@","",""
     call sdo$,"format col=4,numberformat=""###,##0.00""","",""
     call sdo$,"format col=5,numberformat=""###,##0.00"",bold","",""

     call sdo$,"insertrow 1","",""
     call sdo$,"mergecells range=A1:E1","",""
     call sdo$,"writecell range=A1,value="+$22$+ \
          "Over 60 Aging Values as of "+date(0)+$22$,"",""
     call sdo$,"format range=A1:E1,center,size=15,bold","",""
sdofc_complete2:
}
```

Here, finally, are the commands to enhance the formatting of the report. The initial commands use text commands with cut expressions to move the report header data around and change the fonting.

```
# heading section
const BLFONT=univers,10,bold,italic
const BSFONT=univers,9,bold,italic
cbox .5,.5,RIGHTCOL,5,5,30
# line 1
text 2,1.25,{trim(cut(1,1,10,""))},BSFONT                    # date
text 1,1.25,{trim(cut(20,1,100,""))},BLFONT,center, \
     cols=MAXRCOLS  # comp name
text 1,1.25,{trim(cut(121,1,15,""))},BSFONT,right, \
     cols=MAXRCOLS  # page #
# line 2
text 2,2.35,{trim(cut(1,2,10,""))},BSFONT                    # time
text 1,2.35,{trim(cut(20,2,100,""))},BLFONT,center, \
     cols=MAXRCOLS  # rpt title
# line 3
text 1,3.45,{trim(cut(20,3,100,""))},BSFONT,center, \
     cols=MAXRCOLS  # sub heading
# line 4
text 1,4.45,{trim(cut(20,4,100,""))},BSFONT,center, \
     cols=MAXRCOLS  # sub heading
```

This section formats the column headings. The left portion is drawn with text commands, while the aging columns are fonted with font commands, which use the positions from the values calculated in the prepage code block.

```
# detail heading section
const HFONT=univers,10,italic
cbox LEFTCOL,5.25,RIGHTCOL,7.5,1,20
# line 1
cerase 1,6,MAXCOLS,6
text 1,6,"Customer # & Name",HFONT
text 38,6,"Phone #",HFONT,center,cols=14
text 54,6,"Contact",HFONT

# line 2
cerase 1,7,49,7
text 3,7,"Invoice #",HFONT
text 12,7,"Due Date",HFONT,center,cols=8
text 21,7,"P/O #",HFONT
text 32,7,"Order #",HFONT
text 39,7,"Terms",HFONT,center,cols=5
text 45,7,"Type",HFONT,center,cols=4
# using variables from prepage, enhance aging headings
font {scol},7,{colw-1},1,HFONT,right
font {scol+1*(colw+1)},7,{colw-1},1,HFONT,right
font {scol+2*(colw+1)},7,{colw-1},1,HFONT,right
font {scol+3*(colw+1)},7,{colw-1},1,HFONT,right
font {scol+4*(colw+1)},7,{colw-1},1,HFONT,right
font {scol+5*(colw+1)},7,{colw-1},1,HFONT,right
font {scol+6*(colw+1)},7,{colw},1,HFONT,right,bold
```

The report body is enhanced using UnForm's ability to scan for patterns and anchor enhancements to those patterns. The first series of font commands scan for two spaces in the region from column 1, row 9 through column 2, row 66 (defined as the constant MAXROWS above). At each point in that search region, if the two spaces are found, a font command is issued relative to the location. This changes the font of the input data at that location.

The second series of font commands looks for customer heading line types, by searching for any alpha or digit character in the region 1,9 though 2,66. A different set of font commands is then issued for those positions.

```
# detail data section
const BDFONT=cgtimes,10,bold
const DFONT=cgtimes,10
# invoice line
font "  @1,9,2,MAXROWS",2,0,8,1,DFONT
font "  @1,9,2,MAXROWS",11,0,8,1,DFONT,center
font "  @1,9,2,MAXROWS",20,0,10,1,DFONT
```

```
font "  @1,9,2,MAXROWS",31,0,7,1,DFONT
font "  @1,9,2,MAXROWS",38,0,5,1,DFONT,center
font "  @1,9,2,MAXROWS",44,0,4,1,DFONT,center
# using variables from prepage, enhance agings
font "  @1,9,2,MAXROWS",{scol},0,{colw},1,DFONT,decimal
font "  @1,9,2,MAXROWS",{scol+1*(colw+1)},0,{colw},1,DFONT,decimal
font "  @1,9,2,MAXROWS",{scol+2*(colw+1)},0,{colw},1,DFONT,decimal
font "  @1,9,2,MAXROWS",{scol+3*(colw+1)},0,{colw},1,DFONT,decimal
font "  @1,9,2,MAXROWS",{scol+4*(colw+1)},0,{colw},1,DFONT,decimal
font "  @1,9,2,MAXROWS",{scol+5*(colw+1)},0,{colw},1,DFONT,decimal
font "  @1,9,2,MAXROWS",{scol+6*(colw+1)},0,{colw+1},1,BDFONT,decimal

# cust line
font "~[A-Z0-9]@1,9,2,MAXROWS",0,0,6,1,BDFONT
font "~[A-Z0-9]@1,9,2,MAXROWS",7,0,28,1,BDFONT
font "~[A-Z0-9]@1,9,2,MAXROWS",37,0,14,1,BDFONT,center
font "~[A-Z0-9]@1,9,2,MAXROWS",53,0,36,1,BDFONT
shade "~[A-Z0-9]@1,9,2,MAXROWS",0,-.15,{RIGHTCOL-1.5},1,20
```

The following commands look for sequences of dashes, which indicate sub total lines. Wherever a sequence of 50 dashes occurs, a box is drawn and input data is bolded. In addition, the original dashes are removed with the hline command.

```
# customer totals
hline "---",erase
# example of UnForm command with continuation to next line
box "--------------------------------------------------", \
     -1,.25,{RIGHTCOL-53},1.25
bold "--------------------------------------------------",0,1,120,1
```

Finally, grand total lines are treated with special fonting and a box.

```
# grand totals
const DFONT=cgtimes,11,bold
# sample of box command with increased thickness and double lines
box "Grand Total:",-9.5,-1.25,MAXRCOLS,2.25,5,30,dbl 9
font "Grand Total:",0,0,12,1,BDFONT,13
font "Grand Total:",{scol-10},0,{colw},1,DFONT,decimal
font "Grand Total:",{scol-10+1*(colw+1)},0,{colw},1,DFONT,decimal
font "Grand Total:",{scol-10+2*(colw+1)},0,{colw},1,DFONT,decimal
font "Grand Total:",{scol-10+3*(colw+1)},0,{colw},1,DFONT,decimal
font "Grand Total:",{scol-10+4*(colw+1)},0,{colw},1,DFONT,decimal
font "Grand Total:",{scol-10+5*(colw+1)},0,{colw},1,DFONT,decimal
font "Grand Total:",{scol-10+6*(colw+1)},0,{colw+1},1,DFONT,decimal
```

LABELS – TEXT LABELS TO LASER LABELS (ADVANCED.RUL)

UnForm is capable reading rows of input, parsing those rows into logical pages, and reproducing the output with different dimensions. A typical situation that can take advantage of this is if your application is designed to print mailing labels on continuous label stock on dot matrix printers. The labels can be 1-up, 2-up, or any other dimensions. As long as each label has a consistent number of rows and columns, UnForm can parse each label and treat each label as a logical page with the across and down commands. To use this sample, you must add "-r labels" to the command line.

uf60c –i sample4.txt –f advanced.rul –r labels –p pdf –o client:labels.pdf

This statement header identifies the rule set. The name is used in the –r command line option.

```
[labels]
```

Each label "page" is 35 columns and 6 rows of input text. If each line is 106 to 140 characters wide, then four labels are parsed from the columns. When the output is produced, each label will be 30 columns by 6 rows. The labels will be arranged 3 rows across and 10 down the page. UnForm will actually print 3x30=90 columns and 10x6=60 rows on each physical page.

Most laser label stock has ½ inch top and bottom margins. The margin command adds 75 dots (¼ inch) to the standard UnForm top and bottom margins, which default to ¼ inch.

In this sample, the text of the labels is printed from lines 1 to 4. By using the vshift 1 command, UnForm will move the text to lines 2 through 5. The shift command moves the text to the right.

```
page 35,6
rows 6
cols 30
across 3
down 10
font 1,1,40,6,cgtimes,12
margin 0,0,75,75
vshift 1
shift 2
# manual feed tray is usually 2
# tray 2
```

The barcode command supports both 5 and 9-digit formats of the postnet barcode. To get either to print, the prepage code block sets one or the other variable (zip$ or zip9$), and both commands are issued. A null value is not barcoded. The prepage code extracts the zip code from line 3 or 4 of the label. It then determines the length and sets zip$ or zip9$ appropriately.

```
barcode 2,6,{zip$},900,11.0,2
barcode 2,6,{zip9$},905,11.0,2

prepage{
# get zip code from line 3 or 4
zip$="",zip9$="",zipline$=""
if trim(text$[4])>"" then zipline$=trim(text$[4])
if zipline$="" then if trim(text$[3])>"" then zipline$=trim(text$[3])
while zipline$>""
     x=mask(zipline$,"[0-9][0-9][0-9][0-9][0-9]")
     if x>0 zip$=zipline$(x)
     zipline$=""
wend
# remove possible hyphen and validate length
x=pos("-"=zip$); if x=6 then zip$=zip$(1,5)+zip$(7)
if len(zip$)<>5 and len(zip$)<>9 then zip$=""
if len(zip$)=9 then zip9$=zip$,zip$=""
}
```

132X4 – MULTI-UP, SCALED REPORTING (ADVANCED.RUL)

This sample rule set will work with any 132 column by 66 row report. To use it, you must add "-r 132x4" to the command line. The report uses the across and down commands to scale the report to print four logical pages to a physical page.

uf60c –i sample3.txt –f advanced.rul –r 132x4 –p pdf –o client:132x4.pdf

The rule set header identifies the name.

```
[132x4]
```

The page dimensions are defined as 132 columns by 66 rows. UnForm will scale each page to fit 2 across and 2 down on a physical page (264 columns and 132 rows). The report is printed in landscape orientation. A box is drawn around each page, and the hline command will convert all occurrences of 3 or more dashes to horizontal lines.

```
cols 132
rows 66
across 2
down 2
landscape
cbox .5,.5,132.5,66.5
hline "---"
```

ZEBRA LABEL – ZEBRA® LABEL PRINTER EXAMPLE (ADVANCED.RUL)

UnForm offers an optional Zebra printer driver, which produces ZPLII code. Within the limits of the ZPL language, UnForm produces enhanced forms for Zebra printers in much the same way it does for laser printers. Some key differences are: fonts are identified differently and are limited in scalability, shading is either 100% (black) or 0% (white), and the **barcode** command is more extensive and capable than the laser printer **barcode** command.

When executing a Zebra run, it is critical to tell UnForm how large the labels are. This is done with a special syntax on the "-page" command line option. Also, UnForm needs to know what print density is used by the printer. This is determined by the "-p zebra*n*" option, where *n* is either 6, 8, or 12 dots per millimeter. You may need to adjust this sample command line to match your Zebra printer, as it assumes an 8 dpmm printer and 3.25 by 5.5 inch label stock.

uf60c –i samplez.txt –f advanced.rul –p zebra8 –paper 3.25x5.5 –o *zebra-device*

This label is scaled to 40 columns and 35 rows.

```
[zebra label]
detect 0,1,"Zebra Barcode"
cols 40
rows 35
```

The prepage code block gets the PO number, setting it into a variable po$, and removing the PO number from the text with a set() function.

```
prepage{
po$=""
po$=cvs(get(2,16,10),3)
trash$=set(2,16,10,"")
}
```

The From and To sections draw boxes, change fonts, and re-allocate the lines of text from row 10 to row 14 with a series of text commands followed by an erase command.

```
# From section
box 1,1,39,8,3
text 2,2,"From:",font A
font 2,3,35,6,font 0,9

# To section
box 1,9.75,39,10.5,5
#text 2,10.6,"To:",font 0
text 3,11,{get(2,11,30)},font 0,12
```

```
text 3,12.25,{get(2,12,30)},font 0,12
text 3,13.5,{get(2,13,30)},font 0,12
text 3,14.75,{get(2,14,30)},font 0,12
text 3,16,{get(2,15,10)},font 0,12
erase 2,11,30,5
```

This group of commands prints three different barcodes on the label. First, a postnet code is printed from the zip code located at column 2, row 15, for up to 10 characters. Then a UPS maxicode barcode is printed with SDSI's address. Last, a "3 of 9" barcode is printed using the variable po$, derived in the prepage{} code block above.

```
# bar codes
barcode 10,18.25,{trim(get(2,15,10))},Z,33

text 2,24,"Maxicode",font 0,10
barcode 2,25,{"999840956820000" + $0a$ + "SDSI"+ $0a$ + "2195 Talon
Drive" + $0a$ + "Latrobe, CA 95682"},D

box 17,25,22,12,3
text 18,25.75,"Our PO No (in code 39):",font A,21
barcode 20,28,{po$},3,120,2,text above
```

PDF OUTLINE SAMPLE (ADVANCED.RUL)

UnForm supports PDF outlines (or bookmarks) when using the pdf driver. Outlines can be multiple levels, and each outline tree can be different levels deep. UnForm assumes each outline branch points to a page. To control the text shown in the outline, you set the variable outline$ in a prepage or precopy code block. This variable is parsed as each page is printed. Multi-level entries are created by delimiting the text of the levels with a vertical bar (|) within the contents of the variable.

The file sample5.txt contains the contents of a 14-page report featuring two sort and subtotal levels, as well as grand totals and a recap page. The outline tree for this report will be based on the salesperson (outer sort) and class code (inner sort), along with specific page entries for the report total and recap page. As there are no detect statements, you need to specify the –r option on the command line, as shown.

uf60c –i sample5.txt –f advanced.rul –r outline –p pdf –o client:outline.pdf

```
[outline]
```

Set the page dimensions and turn on the outline feature with the outline keyword. The default outline title for each page is simply "Page n", but a code block can override the outline text by setting the variable outline$.

```
cols 132
rows 66
outline
```

The prepage code block looks on each page for the following cases, in order:

- *A 3-digit salesperson number at the first column on line 7*
- *A salesperson subtotal heading on line 8*
- *A report total heading on line 8*
- *A recap page heading on line 2*

For the first two types of pages, a two level outline entry is created (level 1|level 2 structure). For the report total and recap pages, a single level outline entry is created.

```
prepage{
# default outline setting matches prior page
outline$=lastoutline$

# if line 7 starts with 3 digits, set 2-level outline slsp+class
if mask(get(1,7,3),"[0-9][0-9][0-9]") then \
 outline$="Slsp "+get(1,7,3)+"|Class "+get(13,7,2)

# if line 8 contains this, it is a salesperson subtotal
if pos("SALESPERSON: "=text$[8])>0 then \
 outline$="Slsp "+get(14,8,3)+"|Totals"
```

```
# if line 8 contains this, it is a report title
if pos("*Report"=text$[8])>0 then \
 outline$="Report Total"

# if line 2 contains this, it is the recap page
if pos("RECAP PAGE"=text$[2])>0 then \
 outline$="Recap Page"

lastoutline$=outline$

}
```

PROGRAMMING CODE BLOCKS

The prejob, predevice, prepage, and precopy subroutines (and their associated post*xxx* routines) open the world of Business Basic programming to the report and form designs. With a full programming language at your disposal, it is possible to customize and manipulate the forms, and to interact with other applications and devices, or with the operating system.

An experienced BBx or ProvideX programmer (ProvideX is the actual dialect used, with lexical compatibility added for most BBx syntax) typically performs the programming of these subroutines. However, programmers experienced in other languages, particularly other dialects of Basic, can easily learn the fundamentals of Business Basic and perform these programming tasks. Several of the sample forms include some programming, and there is a complete reference guide available from the ProvideX web site: www.pvx.com. In this manual, we have provided some basic (no pun intended) information that will assist developers experienced in other programming environments.

BASIC SYNTAX

Statements
A statement consists of a single verb and any arguments or parameters suitable for that verb. Multiple statements can be placed on a single line by separating them with a semicolon (;). Statements can be preceded by a label, which consists of a label name followed by a colon. Label names must follow the same naming conventions as numeric variables.

Variables
There are two types of variables in Business Basic: string and numeric. Variables that end in a "$" character are treated as string variables. They can hold any amount of text data, limited only by system memory. Numeric variables can contain any number or integer. UnForm sets precision to 10, so that up to 10 digits to the right of the decimal are maintained accurately.

Variable names can be up to 31 letters, digits, and underscore characters, and must start with a letter.

work$, account01$, and cust_name$ are valid string variables.
cust-name$ is invalid.
amount, period_12, and six are valid numeric variables.

Arrays can be defined for both string and numeric variables. Arrays must be defined to a fixed number of elements with a DIM statement, and array elements can then be referenced as variables. Arrays can contain up to three dimensions.

dim amount[12] defines a 13-element array, a[0] … a[12].
dim x$[1:6,1:20] defines a 2-dimensional string array. The first dimension ranges from 1 to 6, the second from 1 to 20. x$[2,20] would be a valid element in this array.

The dim statement can also be used to initialize strings to a specified length. Dim a$(12), for example, will set a$ to 12 spaces.

There are special string constructs available in ProvideX. These are called string templates or composite strings. Details about these constructs can be found in the language manual for ProvideX, available from www.pvx.com.

Functions
Many functions are available in Business Basic. Most will be familiar to a Basic programmer. Functions consist of a word, an opening parenthesis, one or more arguments, and a closing parenthesis. The function returns a string or numeric result, which is typically used as part of an expression, or in an assignment. Wherever a string or numeric value can be used, a string or numeric function can be used. In addition to internal Business Basic functions, UnForm also provides some functions that perform tasks typical to print stream environment in which it runs.

String and numeric representation
Strings are made up of concatenated bytes. They can be represented as literals inside double quotes, such as "Name:", or as hexadecimal strings inside "$" delimiters, such as $1B45$ for Escape-E. They can also be made up of combinations of literals, hex strings, string variables, and functions that return string values. These values are combined using the "+" operator to concatenate each string together. For example, a string containing quotes could be constructed one of these ways: chr(34)+"some text"+chr(34); or 22+"some text"+22, or quote$+"some text"+quote$. Since chr(34) and 22 both represent a quote character, and it would be possible for the variable quote$ to contain the same, all these expressions can represent the same string.

Substrings can be derived from a string variable with the syntax *stringvar*(*start* [,*length*]). For example, if account$ is "01-567", then account$(4,3) would return the value "567". Substrings references with positions that aren't in the string result in errors, so care must be used. To avoid the possible errors, the mid() function can be used.

Numbers can be represented as integers or decimal numbers, or, like strings, can be represented as expressions containing literal numbers, numeric variables, and numeric functions. With numbers, there are more operators available to produce the expressions. A literal number is just a series of digits, with an optional decimal point and an optional leading minus sign. 1995.99 and -100.433 are valid numbers. Other punctuation, such as thousands separators or currency symbols, are invalid in a number though they can be added when a number is formatted as a string for output.

Operators
Business Basic has the following standard operators:

+	concatenate strings or add numbers, depending on context
-	subtraction
*	multiplication
/	division
^	exponentiation
=	testing for equality, or assignment, depending on context
>	testing for greater than
>=	testing for greater than or equal to
<	testing for less than
<=	testing for less than or equal to
<>	testing for inequality
()	controlling precedence
and	combining expressions with logical "and" in conditions
or	combining expressions with "or" in conditions

If Then Else
The structure of IF…THEN…ELSE statements is simple and unblocked. The IF must be followed by an expression to test. The expression can be simple or complex, and must resolve to a single boolean or

numeric result. For numeric results, a 0 is considered false, and anything else is considered true. Once resolved, if true the THEN clause is executed, otherwise the ELSE clause, if present, is executed.

Both the THEN clause and the ELSE clause can contain any statements, including nested IF statements. A closing END_IF after a THEN or ELSE clause will terminate the conditional nature of statements following it.

Here are some examples of IF statements:

```
if amount < 0 then text$="Credit Balance"
if x$="A" then desc$="Acme Rental" else if x$="S" then desc$="Smith & Sons" else desc$="N/A"
if testmode then dummy$=set(1,1,10,"Test Mode") end_if; goto exitsub
```

While Wend Loops

One of Business Basic's looping structures is the WHILE..WEND loop. At the top of the loop is a while *condition* statement, where the condition is evaluated like an IF clause. As long as the condition is true, or returns a non-zero value, the statements up until the closing wend statement are repeated. To escape the loop, you can use the EXITTO label verb, or set variables such that the condition is false before executing the wend verb.

Here is a simple WHILE…WEND syntax that substitutes (") with (') in a string:

```
x=pos($22$=work$)
while x > 0
        work$(x,1)="'"
        x=pos($22$=work$)
wend
```

For Next Loops

Another commonly used loop structure is the FOR…NEXT loop. A FOR statement identifies a variable, a start value, an end value, and an optional step value. The variable is set to the start value; the loop statements are executed until a NEXT statement is encountered; the variable is incremented by the step value; and, until the end value is exceeded, the loop statements are repeated. To exit the loop before the end value is reached, use the EXITTO label verb. Here is an example that would perform the same substitution shown above (though more slowly):

```
for i=1 to len(work$)
        if work$(i,1)=$22$ then work$(i,1)="'"
next i
```

File Handling

Business Basic has very powerful facilities for handling files. Not only are there intrinsic keyed file types, but also text files and pipes can be used.

If the application with which UnForm is integrated is written in ProvideX, then full native access to the data files is available. If the application is written in BBx, then the bbxread() function can be used to obtain record data via an instance of BBx.

If UnForm is working with a non-Business Basic application (e.g. C, Cobol, Informix, Oracle, etc.), there are additional means to obtain data, via ODBC on Windows or pipes on UNIX.

Opening Files

File access is performed through an open file channel. The OPEN statement opens the file on a numeric channel in preparation for later file access. Open(99)"customers.dat" opens the named file on channel 99. Channel numbers can range from 1 to 32767, though the operating system will typically impose a limit on the number of simultaneous channels that can be opened. Channel numbers must be unique. Once opened, that channel number is no longer available until closed. To avoid conflicts with channel numbers, it is common to use a special function that returns an available channel number, UNT. Here is a typical syntax:

```
cust=unt
open(cust)"customers.dat"
```

After that, file access verbs can use the cust variable to access the "customers.dat" file.

To open a pipe channel, you could do the following:

```
faxlist=unt
open(faxlist)"|sqlexec 'select cust,faxnum from customers'"
read(faxlist)line1$

labelprt=unt
open(labelprt)">lp –dlabels"
print(labelprt)"To: "+name$
print(labelprt)"    "+address1$
```

Reading Files

There are two verbs used for reading channels: READ, and READ RECORD. The READ verb understands line and field separators, whereas the READ RECORD verb reads blocks of a specified size or whole records, in the case of intrinsic keyed file types. The READ verbs accept several options, including "key=*string*", "ind=*index*", "err=*linelabel*", "end=*linelabel*", and others. Full details can be found in the language reference manuals. A special syntax of "err=next" is used by UnForm to simply drop through to the next statement if an error occurs.

To read from an intrinsic keyed file (ProvideX files only), you might use one of these:

```
read(cust,key=custkey$,err=next)*,name$,*,*,*,*,faxnum$
```

```
read record(cust,key=custkey$,err=next)custrec$
name$=custrec$(7,30),faxnum$=custrec$(112,10)
```

To read from a pipe or a text file, you may not use a key= clause, so you just read sequentially through the file:

```
read(faxlist,end=done)cust$,faxnum$
```

Writing files

You probably would not want to write to your application files, but you may well want to write to external devices or log files. Writing is performed with these verbs: WRITE or WRITE RECORD and PRINT. Each uses a channel number and arguments to print. PRINT and WRITE terminate their values with a line-feed character, unless a comma follows the last argument. WRITE RECORD will write a single string variable without any termination so it is suitable for binary or blocked output.

```
print (logfile)"Customer: "+custname$+" printed on "+date(0,tim:"%D-%M-%Y:%Hz:%mz")
dim block$(128); block$(1)=custname$,block$(31)=str(amount:"000000.00"); write record(log)block$
```

INTERNAL VARIABLES

In addition to your own variables, UnForm provides a list of variables that you can use, or in many cases, set to a desired value.

across$	Can be set to values described in the across command. Available only in prepage and precopy.
bin$	Can be set to values described in the bin command. Available only in prepage and precopy.
cols$	Can be set to values described in the cols command. Available only in prepage and precopy.
copies **pcopies**	Can be set to the number of copies to generate for a page. You can change this value to dynamically adjust the number of copies. If the number you specify is higher than the number specified by the rule set, then that highest defined copy's text and enhancements will be repeated until your specified copies are complete. This value is reset after each page to the rule set default, so you can't set it in the prejob routine. If you set pcopies, that is also honored like the pcopies command.
copy	Contains the current copy number in precopy. Generally you shouldn't modify this value. If you need to skip printing of a copy, use the **skip** variable instead.
crosshair$	Can be set to "Y" or "y" to enable crosshair grid printing over the output (laser and PDF output only).
down$	Can be set to values described in the down command. Available only in prepage and precopy.
driver$	Stores the current driver as "laser", "PDF", or "zebra". The win and winpvw drivers are considered variants of PDF, and driver$ is set to "PDF" when used. This variable should not be changed.
duplex$	Can be set to values described in the duplex command. Available only in prepage and precopy.
gs$	Can be set to the values described in the gs command. Available only in prepage and precopy.
margin$	Can be set to values described in the margin command. Available only in prepage and precopy.
orientation$	Can be set to "landscape", "portrait", "rlandscape", or "rportrait". It can also be set to a literal digit: "0"=portrait, "1"=landscape, "2"=reverse portrait, or "3"=reverse landscape.
outline$	Can be set to an outline string used when the PDF outline feature is turned on, by use of the outline command. Multiple levels of outlines can be defined by delimiting levels with vertical bars, such as outline$="Customer type "+get(1,6,4)+"|Page

<table>
<tr><td></td><td>"+str(pagenum). This example would produce a 2-level outline structure with a customer type code being the top level, and page numbers as child levels.</td></tr>
<tr><td>output$</td><td>In laser output, this can be changed in prejob, prepage, or precopy, and is tracked by copy. Set it to the device or file name desired for output on the server. If it changes for a given copy in the middle of a laser job, UnForm will close the prior output channel and reopen the new one. This can be used to send a copy to a different printer, or to a fax device. You can set the value to any printer alias known to UnForm (in the unform.cnf file), any file, or a pipe or redirect, such as ">vfx -n "+faxnum$. When using a UNIX redirect or pipe, be sure to add quote characters (CHR(34)) around any data that might contain ampersands (&) or other shell-aware characters.

For PDF output, you can set this value in the prejob code block to override any –o command line setting. Setting this value in any other code block is ignored.</td></tr>
<tr><td>pagenum</td><td>Can be referenced as the current page number. The value should not be changed.</td></tr>
<tr><td>paper$</td><td>Can be set to values described in the paper command. Available only in prepage and precopy.</td></tr>
<tr><td>rows$</td><td>Can be set to values described in the rows command. Available only in prepage and precopy.</td></tr>
<tr><td>skip</td><td>Can be set to a non-zero value in prepage or precopy, to skip printing of that page or copy, respectively.</td></tr>
<tr><td>text$[all]</td><td>Stores the text for the page as a one-dimensional array. For example, text$[2] is the second line of text on the page. In prejob, it contains the content of the first page. In prepage and precopy, it contains the content of each page in sequence. You can use the array directly in code, or you can use the built in get(), mget(), set(), cut(), and mcut() functions to retrieve or manipulate its contents.</td></tr>
<tr><td>tray$</td><td>Can be set to values described in the tray command. Available only in prepage and precopy.</td></tr>
<tr><td>uf.xxx$</td><td>A string template or composite string that can provide access to many attributes of the UnForm environment and command line.
<table>
<tr><td>uf.cols</td><td>Columns for the current page.</td></tr>
<tr><td>uf.copies</td><td>Copies defined for the job.</td></tr>
<tr><td>uf.dfrule$</td><td>Default rule file from the environment.</td></tr>
<tr><td>uf.driver$</td><td>Driver for the current job.</td></tr>
<tr><td>uf.emattach$</td><td>Command-line –emattach value.</td></tr>
<tr><td>uf.emfrom$</td><td>Command line –emfrom value.</td></tr>
<tr><td>uf.emlogin$</td><td>Command line –emlogin value.</td></tr>
<tr><td>uf.emmsgtxt$</td><td>Command line –emmsgtxt value.</td></tr>
</table></td></tr>
</table>

uf.emoh$	Command line –emoh value.
uf.empswd$	Command line –empswd value.
uf.emsubject$	Command line –emsubject value.
uf.emto$	Command line –emto value.
uf.errfile$	Command line –e file value (dynamically determined by the server).
uf.home$	Home directory of the UnForm server.
uf.inputfile$	Command line –i file value (dynamically determined by the server).
uf.job	Current job number.
uf.maxdatacols	Maximum column in the current page.
uf.maxdatarows	Maximum row in the current page.
uf.outputfile$	Command line –o file value. For server-based output, this is the –o option sent by the client. For client-based output, this is dynamically determined by the server.
uf.page	Number of input lines per page. Do not confuse this with the pagenum variable, which holds the current page number.
uf.paper$	Paper size name.
uf.pcopies	Pcopies defined for the job.
uf.pdfauthor$	Command line –pdfauthor value.
uf.pdfkeywords$	Command line –pdfkeywords value.
uf.pdfprotect$	Command line –pdfprotect value.
uf.pdfsubject$	Command line –pdfsubject value.
uf.pdftitle$	Command line –pdftitle value.
uf.prm$	Command line –prm value.
uf.rows	Rows for the current page.
uf.rulefile$	Command line –f rule file value.
uf.ruleset$	Selected rule set for the current job.
uf.shift	Horizontal shift value.
uf.subjob	Set to 1 (can be treated as a boolean) if this is a sub-job executed by the jobexec() function.
uf.subst_file$	Command line –s file value.
uf.vshift	Vertical shift value.

	uf.warn$	Job warning messages, delimited by line-feeds. For example, to add your own message: uf.warn$=uf.warn$+"My message"+chr(10).

INTERNAL FUNCTIONS

In addition to the intrinsic functions available in the run-time Business Basic engine, the most common of which are documented later in this chapter, UnForm provides a set of functions specific to its operating environment. Some functions are macros that perform an action, rather than return a value.

bbxread(*file$,key$,rec$,errcode*)	Executes an instance of BBx, configured with the bbpath=*path* line in uf60d.ini, and obtains the record specified by key$ in file$. If an error occurs in the BBx instance, it is returned in errcode. An errcode value of –1 indicates no error occurred. The variable rec$ can be DIMed as a string template, but be sure to use '=10' to define field separators, as the default separator in the ProvideX engine is a hex 8A rather than the BBx default hex 0A. If it is not defined as a template, the raw record data is returned and may be parsed.
	
	Here is an example:
	
	`prepage{`
	`ky$=get(65,5,6)`
	`dim rec$:"id:c(5*=10), *:c(1*=10), …, fax:c(9*=10)"`
	`bbxread("/u/data/CUSTOMER",ky$,rec$,ec)`
	`if ec=-1 then faxnum$=rec.fax$`
	`}`

cnum(*expression*)	Returns a number from a text string, after stripping formatting characters such as commas and dollar signs. Parentheses and minus signs indicate negative numbers. Use this function, rather than the intinsic num() function, to convert text to numbers if the text can contain punctuation.
cut(*col,row,cols,value$*)	Returns the value text at position *col, row*, for *cols* columns, after setting the specified position to *value$*. If *value$* is null ("") or spaces, cut() effectively erases the text. This is useful for moving data in text commands, such as **text 10,60,{cut(10,59,10,"")}**, which would cut text from 10,59 and move it to 10,60.
email(*to$, from$, subject$, body$, attach$, cc$, bcc$, otherheaders$, login$, password$*)	Sends an email, assuming emailing is properly configured in the mailcall.ini file, using the information supplied. The arguments are positional but need not all be supplied. For example, **email(trim(get(81,1,40)), info@acme.com,**

	"Please review", messagebody$) will send a plain message to the address stored at column 81, row 1, for 40 characters in the current page. No attachment, carbon copy, etc. information will be used. As the arguments are positional, if you need to supply a login and password for the mail server to perform authentication, then all the arguments must be supplied, even if simply null (""). Note that this email function is different from the email command, in that the job itself is not sent, and multiple emails can be sent during the job stream within code blocks. This is useful, particularly in combination with the jobstore and jobexec functions, to develop batch email jobs.
env(*name$*)	Returns the value of the operating system environment variable in *name$*, or in a literal quoted string. Returns null ("") if the variable does not exist.
err=next	May be used for any err=*label* option in any function or statement. Forces UnForm's error trapping to ignore an error. You may, of course, name your own err=*label* if desired.
exec(*expression*)	Executes a barcode, bold, box, erase, font, image, italic, light, micr, move, shade, text, or underline command from within the code block. *Expression* must be a single string value that contains the text of such a command, such as **exec("box "+str(col)+","+str(row)+",30,2.5")**. You can use the **exec**() function to add enhancements to a print job within the code block. The function can be used in either **prepage**{} or **precopy**{} blocks. Remember that some commands need quoted parameters to work properly. For example, if you exec() a text command, be sure to add quote characters around the text to be printed, using one of three methods: double any internal quotes, use an expression that uses 22 for quotes, or use an expression that uses CHR(34) for quotes. For example, **exec("text 10,10," + chr(34) + message$ + chr(34) + ",cgtimes,10")**, or **exec("text " + str(col) + "," + str(row) + ","""Quoted Text""",univers,12")**.
get(*col,row,cols)*	Returns text from the text$[all] array, without substring or array out-of-bounds errors.
jobclose(id$…)	Closes and erases the temporary storage file associated with id$. Open jobs are all automatically closed at the end of the primary job.
jobexec(id$,output$,driver$,argstring$)	Executes a sub-UnForm job using the parameters given. The id$ identifies a job with one or more pages previously stored with the jobstore() function. The output$ value

	defines where the sub-job's output should go. This can be a file name, like "/archive/"+invoice$+".pdf", a device name, like "//printsrv/hp4000", or a pipe/redirect, like ">lp –dhp4000 –oraw". The driver$ argument can be set to one of the –p drivers supported by UnForm, such as laser or PDF. The argstring$ contains any additional command line parameters you wish to add to the sub-job command line. You can use any parameter supported by the uf60c client, though the -i, -o, and -p options are specified using the other three function arguments. A rule set can check uf.subjob, as "if uf.subjob" or "if uf.subjob=1", to test if an instance is running from a jobexec() function.
jobfile(id$)	Returns the temporary text file associated with id$.
jobstore(id$)	Stores the content of the current page in a temporary file, identified by id$. The value in id$ is user-defined, and each unique value stores content in a different temporary file. The other job-related functions use the id$ value to select which file to use. For example, you could store a whole job with an id$ of "job", and individual documents in jobs identified by their document number. Each would be stored separately and could be jobexec'd separately.
left(*str$,length*)	Returns the leftmost *length* characters from str$, padding with spaces on the right to enforce *length*. Note also the mid() and right() functions.
lower(*expression*)	Returns text in lowercase.
mcut(*col,row,cols,rows,value$,lf$,trim$*)	Returns multiple lines of text, optionally with line-feed delimiters and/or trimmed of spaces. The lf$ argument can be set to "Y" or "y" to add a line-feed character between each line; likewise, the trim$ argument can be set to "Y" or "y" to cause each line to be trimmed before returned. In addition, mcut() assigns each line in the cut region to *value$*. Use null ("") or spaces to erase the source text.
mget(*col,row,cols,rows,lf$,trim$*)	Returns multiple lines of text into a single string, optionally with a line-feed delimiter and/or trimmed of spaces. This function is useful in conjunction with multi-line functionality of the text command. The lf$ argument can be set to "Y" or "y" to add a line-feed character between each line; likewise, the trim$ argument can be set to "Y" or "y" to cause each line to be trimmed before returned.
mid(*arg1$,arg2,arg3*)	Safely returns a substring without generating an error 47 if the value in *arg1$* isn't long enough to accommodate position *arg2* and length *arg3*. Note also the left() and

	right() functions.
parse(*str$,n,delimter$*)	Returns the *n*th element of the string str$, when parsed by the delimiter specified. For example, parse("one,two",2,",") would return "two". If the delimiter is null, then any white space delimiter is used.
parseq(*str$,n,delimiter$*)	This is the same as parse(), except that honors quoted values in the string str$, ignoring delimiters contained in them.
proper(*expression*)	Returns text in Proper Case.
right(*str$,length*)	Returns the rightmost *length* characters from str$, padding with spaces on the left to enforce *length*. Note also the left() and mid() functions.
set(*col,row,cols,value$)*	Returns *value$*, after it places *value$* in the text$[all] array at the position indicated.
sub(*str$,old$,new$*)	Returns a string where all occurrences of old$ in str$ are replaced with new$.
trim(*expression*)	Returns *expression* after trimming spaces from the left and right side.
upper(*expression*)	Returns text in UPPERCASE.

When using variables and line labels, you should avoid using any values that begin with "UF". UnForm reserves all such variables and labels for its use. You may use a backslash (\) at the end of a line to continue the statement on the next line. Lines prefixed with "#" are not added to the code.

Two data elements from the command line can be referenced in code blocks using the stbl() function (use gbl() in ProvideX environments). The –s *sub-file* option will generate stbl values as "*@name*". For example, if the substitution file contains the line 'company=Smith Produce', then stbl("@company") will return "Smith Produce". Further, the –prm command line option will directly create stbl values.

VERBS AND FUNCTIONS

The following list is a summary of verbs and functions that are commonly used in UnForm applications. Note that all functions accept an ",err=*linelabel*" or "err=next" argument, and all verbs accept the same after any parameters, to branch if an error occurs. Optional arguments are shown inside braces {}.

ASC(*string*)	Returns the ASCII numeric value (0-255) of the first character of *string*.
ATH(*string*)	Returns a binary equivalent of a human readable hex string. ATH("1B") returns an escape character.
BIN(*integer*,*length*)	Returns a binary integer representation of the specified length. The inverse function of this is the DEC() function.
BREAK	Breaks out of a loop structure. Equivalent to EXITTO *linelabel* if *linelabel* is the line after the closing WEND or NEXT.
CHR(*integer)*	Returns a character string whose ASCII value is *integer*, between 0 and 255. CHR(27) returns an escape character.
CONTINUE	Executes the next iteration of a loop structure. Equivalent to GOTO *linelabel*, if *linelabel* is the closing WEND or NEXT.
CVS(*string*,*arg*)	Returns a converted string according to the cumulative value of the integer *arg*. Values: 1=strip leading spaces, 2=strip trailing spaces, 4=uppercase, 8=lowercase, 16=non-printable characters to spaces, 32=multiple spaces to single spaces. CVS(a$,3) trims both leading and trailing spaces.
DATE(*julian* {,*time*} {:*mask*}) DTE(*julian* {,*time*} {:*mask*})	returns a human readable date and/or time, based on the julian date (see the JUL() function), a decimal time (hour and fraction of hour – 12.5=12:30PM), and a format mask. The mask can contain combinations of placeholder characters and modifiers. The placeholders are %M=month, %D=day, %Y=year, %H=hour (24 hour clock), %h=hour (12 hour clock), %m=minute, %s=second, %p=AM/PM. Modifiers include z=zero fill, s=short text, l=long text. Examples on June 30, 1999 at 1:30 in the afternoon: date(0) returns "06/30/99", date(0:"%Ml %D, %Yl") returns "June 30, 1999", date(0,tim:"%hz:mz %p") returns "01:30 PM".
DEC(*string*)	Returns the decimal conversion of the binary integer in *string*. The counterpart to the BIN() function. To treat *string* as an unsigned integer, you should use the form DEC(00+*string*).
DIM *string*(*length* {,*char*})	Returns a string of *length* size, of spaces or the specified *char* character.
DIM *name*[*dim1*{,*dim2*{,*dim3*}}]	Creates a numeric or string array variable. Dimensions can

	be simple integers, indicating an index range of 0..*dim*, or two integers separated by a colon, like 1:12.
DIR("")	Returns the current disk directory. On Windows, DIR(*driveletter*) will return the current directory for the specified disk drive.
EPT(*number*)	Returns the 10's exponent value of *number*. EPT(100)=3, EPT(12)=2.
ERASE *filename*	Erases a file. Obviously, care should be taken to only erase temporary work files.
EXITTO *linelabel*	Exits a loop structure (current level only, in nested structures) and jumps to the specified *linelabel*.
FBIN(*number*) I3E(*number*)	Returns a 64-bit IEEE number in natural left to right ordering.
FDEC(*string*) I3E(*string*)	Returns the decimal value of a 64-bit IEEE number.
FID(*channel*)	Returns a file identification string for the file opened on *channel*. For devices, just the device name is returned. For files, the first byte indicates the file type (00=indexed, 01=serial, 02=keyed, 03=text, 04=program, 05=directory, 06=mkeyed, etc.) You can verify a file is a plain text file like this: test$=fid(filechan); if test$(1,1)=03 then x$="text file".
FILL(*integer*{,*string*}) DIM(*integer*{,*string*})	Returns a string if *integer* length, made up of successive iterations of *string*, or spaces if no *string* is provided. FILL(7,"abc") will return "abcabca".
FIN(*channel*)	Returns additional file information not found in the FID() function. A common use of this function is to determine file size, which is stored as a binary integer in the first four bytes. To get the length of a file: x$=fid(filechannel); length=dec(00+x$(1,4)). Additional potentially useful information can be found as well. See the language reference manual for more details.
FOR *numvar=start* TO *end* {STEP *increment*}	Initiates a loop, using a numeric variable initialized to *start* the first pass through the loop, incrementing by 1 or the specified *increment*, which can be negative, until the variable exceeds (or goes below in the case of a negative *increment*) *end*. The statements following this command, until a NEXT *numvar* statement, are executed. The loop can be broken from with the BREAK or EXITTO verbs.
FPT(*number*)	Returns the fractional portion of a number. FPT(100.66) returns .66.
GOSUB *linelabel*	Jumps to the specified *linelabel*. Statements will be executed until a RETURN verb is encountered, and execution will return to the statement after the GOSUB.
GOTO *linelabel*	Jumps to the specified *linelabel*.

HTA(*hexstring*)	Returns a human readable hex string of *hexstring*. HTA(CHR(2)) returns "02". HTA("0") returns "30".
IF *test* THEN *statement(s)* {ELSE *statement(s)*} {END_IF or FI}	Conditionally executes *statements*. *test* must be a simple expression that produces a boolean or numeric result (0=false, non-0=true). Multiple statements can follow the THEN or ELSE clause by separating them with semi-colons. Statements following a END_IF are executed without regard to the condition of the last IF test. Nested IF statements are accepted without practical limit.
INT(*number*)	Returns the integer portion of a number. INT(99.645)=99.
JUL(*year,month,day*)	Returns the julian integer of the specified date elements. The year should be specified, if possible, as a 4-digit year. Otherwise the function will assume a century of 1900. The complement of this function is the DATE() function.
LEN(*string*)	Returns the length of the string.
LET *var=value*{,*var=value…*}	Assigns variables to values. The variables can be numeric, string, or array variables. The values can be any compatible numeric or string expression. LET is implied when an assignment is performed in context. "LET a=1" and "a=1" are equivalent.
MASK(*string{,regexpr}*) MSK(*string{,regexpr}*)	Returns the position where a regular expression pattern was found in the *string*, or 0 If not found. If *regexpr* is not specified, then the last *regexpr* used is re-used. This provides a performance benefit for repeated uses of the same *regexpr*. The length of the string matched is returned by the TCB(16) function.
MAX(*num*{,*num…*})	Returns the largest number found in the list of *num*s.
MIN(*num*{,*num…*})	Returns the smallest number found in the list of *num*s.
MOD(*num1,num2*)	Returns the remainder of dividing *num1* by *num2*. MOD(4,3)=1, MOD(6,3)=0.
NUM(*string*)	Returns the decimal value of a string, assuming the string is a well-formatted value containing digits, a single optional period (decimal point), and a single optional leading hyphen (minus sign). Other punctuation or characters will return an error. NUM("-12.5") returns 12.5. NUM("1,456") results in an error.
ON *integer* GOTO|GOSUB *linelabel*{,*linelabel…*}	Branches to one of the indicated line labels based on the value of *integer*. If *integer* is 0 or less, branch to the first label, 1 to the second, 2 to the third, and so on. The last label is used for *integer* values greater than that of the last label.
OPEN(*integer*{,err=*linelabel*|next}{,isz=*integer*}) *string*	Opens the file named in *string* on channel *integer*. To open a file in binary mode regardless of the file type, specify a block size with the ",isz=*integer*" option.
POS(*string1 relation string2* {,*increment*	Scans *string2* for a substring having the specified *relation* to

{,*occurrence*}})	*string1*. POS("B"="ABC") returns 2. POS("B"<"ABC") returns 3. The string can be searched in even character increments: POS("02"="002002",2) will return 5, since the second and third characters, though matching the search string, are not located at an increment boundary. If the string is not found, or the requested relation, increment, and occurrence cause the string to not be found, the function returns 0.
PRINT(*channel*) *value* {,*value…*}{,}	Prints a series of values, numeric and/or string, to the file channel specified. A line-feed character is added to the channel unless the last character of the statement is a comma.
READ{ RECORD}(*channel* {,*options*}) *variable* {,*variable…*}	Reads data from the specified channel into the specified variables, looking for field terminator characters to delimit variables. Field terminators include line-feeds, carriage returns, and nulls. Valid *options* include "err=*linelabel*", "end=*linelabel*", "siz=*blocksize*". "key=*keystring*", "ind=*index*", and "dom=*linelabel*". For intrinsic keyed files, use the key= or ind= options to read specific records. For text files, use READ to process line-feed delimited files, but be aware that carriage return characters act as field separators. To read text files as binary files, use READ RECORD with a "siz=" option.
REM	Places a non-executing remark line in the code. In UnForm, you can also use a # character.
RETRY	Retries the statement that caused the last error branch to be taken.
RETURN	Returns from a GOSUB branch.
RND(*integer*)	Returns a pseudo-random number. The random number sequence can be re-seeded by providing a negative integer, so it is common at startup (like in a prejob code block) to seed the RND function with a variable number, such as MOD(JUL(0,0,0)+INT(TIM*10000),32000). The *integer* can be a number from –32767 to +32767. Positive numbers return a random integer from 0 to *integer*-1. If *integer* is 0, a random number between 0 and 1 is returned.
ROUND(*number*,*precision*)	Returns *number*, rounded to *precision*. ROUND(1.566,2)=1.57. ROUND(100.83,0) returns 101.
SCALL(*string*) SYS(*string*)	Executes the operating system command in *string*. Returns the result code provided by the operating system. Use this function to interface with the operating system or external commands. This is an alternative to opening a pipe to a command.
SETERR *linelabel*	Provides a generic error handler to catch errors not trapped by err=*linelabel* branches in functions and verbs. UnForm

	also adds error handling code to code blocks, and reports errors in a job error file (temp/*jobno*.err in the server directory).
SGN(*number*)	Returns a 1, 0, or –1, depending on the sign of *number*.
STBL(*string1*{,*string2*}) GBL(*string1*{,*string2*})	Returns and/or sets the global string table value named *string1*. If *string2* is present, then the string table is set to *string2*. In both cases, the value is returned. If *string1* has not been set, STBL(*string1*) will result in an error (trappable with err=*linelabel*, of course).
STR(*number*{:*mask*}) STR(*string*{:*mask*})	Converts a number to a string, optionally formatted with a *mask*. The mask can contain any text, plus the following placeholder characters: 0=zero filled digit, #=space filled digit, "."=decimal point, ","=thousands separator, -, (,), and CR for negative numbers. STR(99.91:"0000.00") returns "0099.91". STR(19093.255:"###,##0.00") returns "19,093.26".
STRING *filename*{,err=*label*} SERIAL *filename*{,err=*label*}	Creates a text file of the name specified. Use either a string variable or expression, or a quoted literal string. Examples: STRING "/tmp/test.txt" or STRING "/tmp/"+str(dec(info(3,0)))+".txt",err=next.
TCB(*integer*)	Returns task control information. Commonly used *integer* values include: 10=last operating system error code and 16=length of MASK() function match.
TIM	Numeric variable that returns the decimal time of day, from 0.0 to 23.99.
UNT	Numeric variable that returns the next available file channel number.
WHILE *condition*…WEND	Looping construct that performs statements between WHILE and WEND statements as long as *condition* is true or non-zero.
WRITE {RECORD} (*chan*,*options*)*data*	Writes data to a file. Numerous options are available, some depending on the type of file. See the full programming documentation available on www.pvx.com for more details.

Lexical Substitutions

With the change in Version 6 to the ProvideX run-time engine, it is possible that some BBx syntax in code blocks will be incompatible. For the most part, the lexical substitutions automatically performed by UnForm will handle any differences, with the exception of direct I/O to BBx data files, which can be handled with the bbxread() function. However, if any additional substitutions are required, they can be entered into a user-defined text file called uflexsub.usr.

The format for the lines in this file is simply *bbxsyntax=pvxsyntax*. An example is provided in uflexsub.txt, which is a file that provides some standard syntax substitutions that the internal lex capabilities do not support. You can add your own by simply creating uflexsub.usr and adding lines.

ERROR CODES

When code is executed, any errors that are not handled by err=*label* branches are reported as warnings on a job trailer page. High error code numbers are used to report errors in client-server communication. Common error codes are shown in the following table.

Error Number	Description
1	End of record error, which may occur on a buffered disk write operation if the data is too long for the record buffer. This error is rare in UnForm jobs, but could occur if output is being printed to a printer alias defined in the config.unf file.
2	End of file, which may indicate a disk full message, or a file that is too large for the operating system to handle.
10	An invalid file name was given.
11	A missing key on a keyed read operation, or a duplicate key on a keyed write operation with a DOM= option.
12	A missing file error on a file open operation, or a duplicate file error on a file creation operation.
13	Normally a file permission error.
14	A file channel conflict or locking conflict error.
16	Out of resources, such as file handles. If this error occurs, it is often due to opening too many files. This can easily occur if files are opened but not closed in a loop or call construct.
18	Normally a file or directory permission error.
20	Syntax error. Common causes include mismatched parentheses, incorrect spelling of verbs or functions, or missing or incorrect function arguments.
21	Missing statement, as referenced in an ERR=*label*, or a goto or gosub branch.
23	Missing GBL/STBL variable name.
26	String/Number mismatch, where a string variable or literal is used where a number is expected, or visa versa.
27	Stack error, such as a return without a gosub, or a wend without a while.
28	For/Next error, such as executing a next without an associated for.
29	Mnemonic error. Mnemonics are pre-defined codes inside single quotes, such as 'FF' or 'LF'. Therefore, single quotes are not valid as string literal indicators; only double quotes are.
30	Corrupt program, which indicates that UnForm itself is probably corrupted, unless this error occurs on a call statement referencing an external program.
31	Out of memory.
33	Out of memory.
36	Mismatched arguments on a call statement.
40	Numeric overflow, normally caused by a divide by zero.
41	An integer overflow or range error. Some functions require integer arguments, so a floating point number will cause this error. Also, some functions require integer arguments to fall in a certain range, and this error will occur if the function is given a value outside of the valid range.
42	Array subscript error.

Error Number	Description
43	Masking error.
46	String length error.
47	Substring error, such as a starting position of 0 or a length greater than the length of the string.
997	The client's IP address is not in the server's list of valid addresses. To correct this problem, the allow= line in the server's uf60d.ini file must be modified to match the network addresses in use, and the uf60d server restarted.
998	The maximum number of concurrent jobs licensed was exceeded.
999	The server was unable to start the secondary process to handle the job within the allotted time of 30 seconds. Possible causes include a sluggish server and network problems, such as a DNS server timeout.
1024	The Windows uf60c.exe client can report this error if the network connection to the server is too slow.
1057	The Windows uf60c.exe client can report this error if the server is not running or a firewall is blocking the primary listening port.

EMAIL INTEGRATION

UnForm includes a copy of the MailCall utility that enables emailing of attachments from within UnForm. This is most often used to send PDF files. It can be used to email laser printer (PCL5) files, as long as you know the email recipient has a compatible printer that supports any of the fonts used in your documents. If you use CGTimes, Courier, and Univers fonts, then any PCL5 laser print device should be able to properly print documents, as long as the user can copy the file directly to the printer.

The MailCall utility is used internally by both the email command, which emails a complete PDF-formatted job, and the email() function, which can send email(s) in mid-job, possibly with attachments resulting from sub-jobs managed by the job*xxx* series of code block functions. These two features are capable of handling most email requirements. However, within a code block, you can use the MailCall program directly, for any degree of control required. For example, the MailCall utility provides logging facilities that are helpful in debugging connection or communication problems. To implement logging, direct calls to the MailCall program are required.

Generally, the only requirement to get email working is to configure the server= line in the mailcall.ini file. This line needs to name the machine or IP address of the SMTP server that MailCall connects to. Other configuration options serve as default values.

The remainder of this chapter discusses the utility in depth.

Configuration

To configure MailCall, you need to edit the mailcall.ini file, using any text editor. If you don't have a mailcall.ini file, then you can rename mailcall.sds to be mailcall.ini. The following notes provide details about each option.

The most important element of the configuration is to ensure the system that executes MailCall has connectivity to your SMTP mail server. This may be an in-house system, or it may be hosted by your Internet Service Provider. A fairly foolproof way to test this is to telnet to port 25 on the mail server from your system (telnet *hostname* 25 from either UNIX or an MS-DOS Command Window). If you get a non-error response, MailCall should work.

server=*smtp-server*
This contains a reference to the IP address or domain name of the SMTP email server. This is used by the native socket interface, the mailcall.exe program, and the mailcall.pl program. If your mailer= setting uses sendmail or mmdf, this value is not used.

port=*port-number*
When native sockets are used, the default SMTP port of 25 can be overridden by setting a *port-number*. Normally, this should not be required.

from=*email-address*
Defines a default 'from' address if none is supplied when sending email.

hostname=*hostname*
If the environment does not provide a system name that is valid for the SMTP server, you can specify a value here. If no value is specified, then MailCall will determine the system hostname with the UNIX "hostname" command, or on Windows with the INFO() function in Visual PRO/5 or the NID variable in ProvideX. This element is only used by the native socket support.

login=*username*
password=*password*
If the SMTP server requires authentication, then you can define a default *username* and *password* with these elements. It is also possible to specify a *username* and *password* within the CALL interface. These values, if required, are supplied by the mail administrator, and must be supplied exactly as specified or you will probably get an authentication error and be unable to send mail.

mailer=*commandline*

NOTE: **When running MailCall under UnForm 6, there is no need to configure a mailer= line.**

If MailCall will *not* use internal sockets, then this line configures how MailCall actually sends the mail. If you are running under ProvideX or PRO/5 or Visual PRO/5 revision 2.2 or higher with a proper alias line defined, MailCall will use internal sockets and this line does not need to be configured. When required, BBx executes this command line via the SCALL() function. There must be a % character in the command line, which MailCall substitutes with the email submission file at run-time.

If no mailer value is set (all lines are commented) and a mailer is required, then a default mailer line is constructed, using "perl mailcall.pl % >mailcall.pl.log 2>mailcall.pl.err" on UNIX or "mailcall.exe %" on Windows. The proper path to the mailer is automatically generated. In other words, **if you have Perl or are on Windows, there is generally no need to configure a mailer= line**.

On Windows, *commandline* should be set to the full path for mailcall.exe plus the % argument, such as 'c:\mailcall\mailcall.exe %'. Be sure to use DOS-style backslashes rather than forward slashes.

On UNIX, you will probably want to use mailcall.pl. mailcall.pl should be in the same directory as the MailCall program, and mailer should be set to the full path to mailcall.pl. The *commandline* should be 'perl /usr/mailcall/mailcall.pl % >/dev/null' (adjust the directory path as necessary). Perl, of course, must be installed on your system for this to work. To enable logging, change the ">/dev/null" to ">*pathname*", and the conversation that mailcall.pl has with the SMTP server will be logged to that file.

If you use sendmail, the *commandline* '/usr/lib/sendmail –t <%' should work.

If you use mmdf, then the *commandline* 'echo $LOGNAME >%2; cat % >>%2; /usr/mmdf/bin/submit -uxto,cc* <%2; rm %2' is used to submit email messages. The command line argument "-uxto,cc*" instructs submit to scan for To: and Cc: headers for addresses.

Note that mmdf doesn't support Bcc: headers, while the other three methods do.

timezone=*zone*
Internet mail must include a date and time header; a properly formatted time will include your time zone. On Windows, the *zone* is added to the date and time header in the submission file. On UNIX, the time zone is determined from the date command.

charset=*charsetname*
The default character set in Internet email is "us-ascii". With this setting, it is possible to override this default for text elements of an email that includes attachments, including the body text itself.

Most configuration options have equivalent variables in the CALL string template. If you define values in the template, they override the equivalent values in the configuration file.

Implementation

Implementing MailCall requires the use of code blocks to establish temporary output files and then the execution of MailCall itself.

Here is a sample PDF rule file that can be used to email a PDF document. Since the pdf driver can only be used to produce one PDF file at a time, there is only one file to worry about.

```
[mailpdf]
cols 80
rows 66

prejob{
# set output file to a unique name using process ID
# note the pdf driver only allows output changes in prejob
output$="/tmp/email"+str(dec(info(3,0)))+".pdf"
}

postdevice{
call uf.home$+"mailcall.bb",1,x$,""
x.to$="someone@somwhere.com"
x.subject$="PDF Report attached"
x.msgtxt$="Here is a sample PDF file.\n"
x.attach$=output$
x.from$="sdsi@synergetic-data.com"
call uf.home$+"mailcall.bb",0,x$,""
erase output$
}
```

Here is a slightly more complex example, designed to email the second copy of a PCL document. PCL allows output to be split in the middle of the job, so this technique would work in a batch run where a document reference number is used to define the output name. This sample assumes the report will contain the email address at column 1, row 1 of each document.

```
[mailpcl]
cols 80
rows 66
copies 2

prejob{
# initialize mailer$ template
call uf.home$+"mailcall.bb",1,mailer$,""
}

precopy{
# set copy 2 output to document number plus extension
if copy=2 then output$=get(70,6,6)+".pcl"
}

postdevice{
# whenever the document number changes, this routine is executed
if copy<>2 then goto skip_mail
mailer.to$=trim(get(1,1,40))
mailer.subject$="Report attached"
mailer.msgtxt$="Here is the report you asked for.  Copy it to your laser printer.\n"
mailer.attach$=output$
mailer.from$="sdsi@synergetic-data.com"
call uf.home$+"mailcall.bb",0,x$,""
erase output$
}
```

MailCall Reference

CALL uf.home$+"mailcall.bb", mode, dat$, errmsg$

You may call either mailcall.bb or mailcall.pv; both are identical files for use within UnForm.

Arguments:

mode is an integer value that controls how MailCall interprets or returns data in the dat$ argument. The following are valid mode values:

0	Send mail based on data in string template dat$
1	Return a string template suitable for mode=0 in dat$

2 Return version information in dat$

For modes 0 and 1, **dat$** is a string template in the format:

from:c(1*=0),to:c(1*=0),cc:c(1*=0),subject:c(1*=0),otherhead:c(1*,msgtxt:c(1*=0),attach:c(1*=0),status:n(1*=0),forcebase64:n(1*=0),forcenotify:n(1*=0),bcc:c(1*=0),bodymime:c(1*=0),charset:c(1*=0),timeout:n(1*=0),statuspause:n(1*=0),dialog:n(1*=0),login:c(1*=0),password:c(1*=0),logfile:c(1*=0),timezone:c(1*=0),charinterface:n(1*=0),logdata:n(1*=0)"

To provide for additions to this base template, you should always use a single CALL using mode=1, which will return a usable template in dat$.

For mode 2, dat$ returns a printable string that describes the version and license status.

Here is a description of each template field:

dat.from$ contains the sender's email address. This value defaults to what is specified in the "from=*address*" line in mailcall.ini

**dat.to$** contains one or more email addresses delimited by commas. Note that if multiple addresses are desired, it is more common to place additional addresses in the cc$ field. Each address should be structured in one of two ways: *name@domain* or "*text name*" <*name@domain*>. It is important that if any data is present other than the plain internet email address, that the Internet address be enclosed in angled brackets <>.

dat.cc$** contains zero or more carbon copy addresses. Multiple addresses must be delimited with commas. Address formats are the same as for **dat.to$, above.

dat.bcc$ contains zero or more blind carbon copy addresses. Multiple addresses must be delimited with commas. A blind carbon copy address receives a copy of the email, but the Bcc: header is removed from the submission, so no other recipients know of the Bcc: recipients.

dat.subject$ contains a single line of subject text, describing the message content.

dat.otherhead$ contains additional mail headers, should they be necessary. The rfc822 specification allows for user defined headers starting with the characters "X-", in the format of "X-*name*: *value*". Each header line should be suffixed with a CRLF (or LF) delimiter ($0D0A$). There must be no blank lines in this value, and all lines should have a proper header structure of 'name <colon (:)> <space> value'.

dat.msgtxt$ is plain text for the message body. It may contain line breaks delimited with CRLF (or LF) sequences. Lines should not exceed 900 characters without line breaks. You may also use UNIX-style line break escapes (\n sequences) instead of binary CRLF characters.

**dat.bodymime$** can be used to define an alternate body text (dat.msgtxt$) MIME type. The default is "text/plain", but it is common to prepare message body text as HTML, in which case you can specify

dat.bodymime$="text/html". This must be a well-known standard value (see the mime.typ file included with MailCall), and should be of the text/* family.

dat.attach$ contains one or more file names to attach to the message, delimited with commas. If this contains names, then MailCall will produce a MIME-encoded message, with the message body as plain text, text-style files (MIME types such as text/plain or text/html) as quoted-printable attachments, and other files as base64-encoded attachments.

dat.status, if set to 1 (or any positive value), will cause a status window to display as the email is processed. This flag is honored when MailCall uses native sockets or the external mailcall.exe program. When native sockets are used, the status window operates for both generation and SMTP server submission. When the external Windows mailer is used, it only operates for submission. External UNIX mailers do not support this flag.

For logging on UNIX installations, if you are using mailcall.pl, do this:

- Verify the setting of $log=1 in mailcall.pl near the top of the program
- Direct stdout to a file or the screen by modifying the mailer= line: something like "perl /usr/mailcall/mailcall.pl % >/tmp/mailcall.log". or just "perl /usr/mailcall/mailcall.pl %".

dat.statuspause can be set to the number of seconds to pause before closing the status window after the SMTP conversation is complete. This can help the user see the process completion without a quickly flashing window. This flag is only honored when MailCall uses native sockets and the dat.status flag is set.

dat.forcebase64, if set to 1 (or any non-zero value), will cause MailCall to always encode files with base64-encoding. By default, files whose MIME type is text are encoded using quoted-printable encoding.

dat.bodymime$, if set, will override the default text/plain MIME type used for the message body.

dat.charset$, if set, will override the charset default defined in the mailcall.ini configuration file, or the default of "us-ascii", when no setting is defined. Character sets are associated with any text body or attachment.

dat.login$, dat.password$, if set, and if the SMTP server requires authentication, are used for the AUTH LOGIN authentication process. These values would be provided by the ISP or mail server administrator, and must be provided exactly as specified. These values are honored when MailCall uses native sockets or the mailcall.exe or mailcall.pl mailers.

dat.logfile$, if set to a pathname, will trigger detail logging of the SMTP conversation when MailCall is using native sockets. The file will be erased and created each time MailCall is CALLed. Be careful not to use pathnames that should not be erased.

dat.timezone$, if set, will override the normal time zone value that is applied to the Date: header. The default time zone comes from either the timezone= value in mailcall.ini (for Windows) or the UNIX 'date +%Z' command. Use this to set a relative GMT value, like "-0800" for PST.

dat.charinterface, if set to a non-zero value, will force character-mode for the dialog and status window displays, even in a GUI environment. The status window display affected is only the internal version used when native sockets are utilized, not the status window displayed by the mailcall.exe mailer.

dat.logdata, if set to a non-zero value, and if the dat.logfile$ is defined, and if a native socket is in use, will cause the mail submission file data to be logged to the log file specified in dat.logfile$. The default behavior is to only log SMTP conversation information and suppress the message data.

errmsg$ will contain the text of an error message, if one occurs.

UnForm Notes: When UnForm is running on a UNIX system, there is no usable terminal device associated with it, even if run from the command line. Therefore, the user interface options (such as dat.dialog=1) of MailCall are not available. This is not the case on a Windows installation, so long as the server is running as an application rather than a service. Note however, that any user interface presented occurs where the UnForm server is running, not necessarily where the client runs.

HTML OUTPUT

UnForm provides an optional capability to produce HTML files from reports, using a processing engine that is similar to that used for laser printer output. Using this capability, users can convert their standard text-based reports into HTML documents, which are suitable for viewing with Web browsers such as Netscape Navigator and Communicator, and Microsoft Internet Explorer.

Reports can be converted in real-time, as part of a CGI or ASP procedure that responds to a browser request to generate a report, then format it as HTML. Alternatively, reports can be converted with a periodic batch process, such as a nightly procedure that produces various reports, then converts them all to HTML for viewing the next day.

Even without a rule set, UnForm can streamline text reports by producing plain text pages with horizontal rules at the end of each page. These are constructed using HTML templates, so standard company headers and footers can be applied even to reports that are not enhanced via a rule set.

CREATING HTML

UnForm will create HTML output if you specify "-p html" on the command line. Given this parameter, and with no "-f *rulefile*" parameter, UnForm will look for the "html.rul" file rather than the default "unform.rul" file used for printer output.

By default, the HTML output is generated to standard output (on UNIX only), but it is normally preferable to specify an output file, such as "-o /usr/internet/docs/reports/aging". UnForm can then build the reports with varying styles in stages, and a browser can view interim results as soon as the first page is generated. UnForm will add a ".htm" extension automatically to the output file. UnForm will also create additional files depending on the style of the report. For example, if a table of contents is generated as a separate document, then the base file (aging.htm in the above example) will be the table of contents, and additional files will be generated for the pages of the report (aging.*page*.htm).

A sample command, therefore, might look like this:

unform -i aging.txt -o /usr/internet/docs/reports/aging -p html -f ourhtml.rul

As HTML structure is very different from that of laser printers PCL, HTML rule sets are very different from printer rule sets. UnForm uses HTML table structures to format pages. These structures have a defined hierarchy of rows, cells, and data, with attributes applied to either cells or data. HTML rule sets follow this structure in that you define rows, then within rows you define cells, and then within cells you define the attributes of the cell and text.

The HTML output that UnForm produces can be in one of several styles. The rule set options used to trigger the style are shown in parentheses:

- The simplest form is that of one document with all the pages sequentially created as tables. If no output file is specified (-o *filename*), this is what UnForm will produce regardless of any style options you specify.
- The output can be produced in one file, with a table of contents at the top of the file (toc=y or toc=l, multipage=n). As each page is generated and appended to the file, the table of contents is updated and inserted at the top. The table of contents consists of descriptions linked to the individual pages. The descriptions default to "Page number *n*", but can be created in page code blocks. Additionally, the table of contents can be created as a vertical column (toc=y), or as a bullet list (toc=l).
- The output can be produced in multiple files (multipage=y), with the table of contents being the primary one, with links to each page as a separate HTML document.
- The output can be produced as frames (frame=y), with the table of contents in one frame, and pages in the other. The target pages can be stored in a single file, multi-page document, or with each page in an individual file.

Note that all these options but the first require that a table of contents be maintained as each page is generated. In order to construct an updated document as each page is generated, UnForm must generate temporary files with which to build the HTML required. The *filename* specified by the "-o" option is re-

created as each page is completed. Therefore, if standard output is generated rather than output files, only the first style can be produced.

This interim generation of files means that the HTML output can be viewed as soon as the first page is generated. This can be very helpful when large reports are being formatted in real-time.

HTML CONFIGURATION

When generating HTML documents, UnForm uses several configuration elements to structure the output. Most of these are created in UnForm's parameter file, which is named "ufparam.txt". Note that you can create a custom parameter file for your site that will not be overwritten during an update of UnForm by copying "ufparam.txt" to "ufparam.txc". Then make any changes to the custom version.

A section in the configuration file headed by "[html]" controls HTML configuration. It will look like this:

```
[html]
page=page.htm
toc=toc.htm
both=both.htm
frame=frame.htm
pagenum=Page number
imagelib=
imageurl=
complete=Report Complete
incomplete=Report not complete (reload page to view again)
```

The following table describes each parameter:

Element	**Description**
page=*filename* toc=*filename* both=*filename* frame=*filename*	These elements point to HTML template files in UnForm's home directory. These files are used by UnForm based on the style of output being generated. To create custom templates for your site, you should copy each file to some other name, modify the file names identified in these four elements, and edit the templates for your needs. See "HTML OUTPUT TEMPLATES", below, for more information.
colwidth=*text*	The default column cell width is *text*. This can be a pixel value, such as "colwidth=9", or any other value accepted by a <td width=*value*> tag in HTML. If no value is specified, UnForm uses "2em", which indicates 2 *half-characters*, based on the average width of a character in the default font. This value can also be specified for individual reports using the **colwidth** keyword in a rule set.
pagenum=*text*	This text is used to generate the default table of contents' values. A space and the page number follow the text.
imagelib=*directory*	This points to a directory where image files are

Element	Description
	physically stored on disk. If any column definition has an option indicating it contains image file names, then the files in the column are searched for first as named, and then in this directory. If the image can be found, then the image tag can be generated with width and height parameters, which normally speeds up the page rendering speed by the browser.
imageurl=*url-prefix*	When image tags are generated in a column, the *url-prefix* is placed in front of the file name. This allows the Web server to map the name to a physical location on the server.
complete=*text* incomplete=*text*	One of these values is placed in the "$status" global string at the end of each page, depending on whether the job is complete or not. You can then place the value in the HTML template files by embedding the tag "[$status]" in the template.

HTML OUTPUT TEMPLATES

As companies develop Internet and Intranet strategies, they should employ standard formatting conventions to their HTML documents. HTML-formatted reports should likewise follow these conventions, so UnForm supports the use of HTML template files.

UnForm looks for these files in the UnForm directory, each named in the parameter file "ufparam.txc" or "ufparam.txt". UnForm is distributed with a standard parameter file and standard HTML template files. To customize these for your site, copy "ufparam.txt" to "ufparam.txc", then copy the template files to new names and reference those names in the new "ufparam.txc" file.

The names to use are specified in the "[html]" section of the parameter file, and are coded as "toc=*tocfilename*", "page=*pagefilename*", "both=*bothfilename*", and "frame=*framefilename*". In each of these files, place the text "[$toc]" where the table of contents should be placed, and "[$page]" where the page table(s) need to be placed. In the case of a frame template, the two markers are used for placement of URL links to the table of contents document and the page document(s), respectively.

UnForm determines which template files are used based on the style being used for the output. If there are separate table of contents and page documents, then the *tocfilename* and *pagefilename* are both used. If the table of contents and the pages are in the same document, then the *bothfilename* is used. This file should contain both [$toc] and [$page] tags. If frame output is used, then the *framefilename* is used for the primary document, and the *tocfilename* and *pagefilename* files are used for the target documents.

In addition to the required [$toc] and [$page] tags, you can also reference other pre-defined tags: [$title], [$date], [$time], and [$status], as well as any global strings that you define in prepage{} or prejob{} code blocks. These global strings, generated by the STBL() or GBL() functions, are embedded in the document by placing the name in square brackets anywhere in the template.

One special note: If you wish to customize the date and time masks used by UnForm, set DATEMASK$ and/or TIMEMASK$ in the prejob{} code block to the desired format based on the BBx DATE() function.

The default HTML template for a page (page=*filename*) looks like this:

```
<html>
<head>
<title>[$title]</title>
</head>
<body bgcolor=#e0e0e0>
<h3><center>[$title]</center></h3>
<hr>
[$page]
<hr>
<center><small>
&copy;1997 by Synergetic Data Systems Inc.<br>
All rights reserved.
```

```
</small></center>
</body>
</html>
```

The default template for an independent table of contents (toc=*filename*) looks like this:

```
<html>
<head>
<title>[$title]</title>
</head>
<body bgcolor=#e0e0e0>
<center>
<h3>Table of Contents</h3>
<strong>[$title]</strong>
</center>
<hr>
[$toc]
<p>[$status]
<hr>
<center><small>
&copy;1997 by Synergetic Data Systems Inc.<br>
All rights reserved.
</small></center>
</body>
</html>
```

The default template for a combined style (both=*filename*) looks like this:

```
<html>
<head>
<title>[$title]</title>
</head>
<body bgcolor=#e0e0e0>
<h3><center>[$title]</center></h3>
<center>[$toc]</center>
<hr>
[$page]
<hr>
<center><small>
Run on [$date] [$time]<p>
&copy;1997 by Synergetic Data Systems Inc.<br>
All rights reserved.
</small></center>
</body>
</html>
```

The default template for a frame style (frame=*filename*) looks like this:

```
<html>
<head><title>[$title]</title></head>
<frameset cols="25%,*">
 <frame name="toc" src="[$toc]">
```

```
 <frame name="page" src="[$page]">
</frameset>
</html>
```

HTML RULE SETS

Like PCL rule sets, HTML rule sets are stored in a text file. Each set is headed by a unique name in square brackets:

[AgingReport]
keywords…

UnForm selects a rule set to use based on either the "-r *ruleset*" command line option, or **detect** keywords in each rule set. **Detect** keywords cause UnForm to scan the first page of input, then search for a match where all **detect** keyword(s) for a given rule set match the contents of the page.

Once a rule set is selected, UnForm begins processing each page of text using the rules specified. Each page is first stripped of any PCL escape sequences so that just text remains, then the array of text rows is converted to HTML based on the rules. This HTML is then placed in the output according to the style of output defined by the rule set.

If no rule set is selected, then UnForm will process each page as plain text, using HTML <pre> and </pre> tags, with horizontal rules between pages (where form-feeds occur in the input).

The following keywords are identical in use and function with printer rule sets:

- cols
- const
- detect
- page
- rows

The **hline** and **vline** keywords are identical, except that they *always* perform an erase of the horizontal and vertical lines found.

Keywords unique to HTML generation are defined on the following pages.

BORDER

Syntax

border=*value*

Description

The tables generated by UnForm for each page will normally have borders, and will therefore set the table border option to 1: <table border=1 ...>. If you would prefer a different border setting, define it with this keyword.

See also the **otheropt** and **width** keywords.

COLDEF

Syntax

1. [coldef | ccoldef] *col, cols, options*
   { *code block* }

2. coldef "*text | ~regexpr", coloffset, cols, options*
   { *code block* }

3. coldef "*text | ~regexpr", coloffset, "to-text | ~to-regexpr", to-coloffset options*
   { *code block* }

Syntax 1 defines an absolute column region. **coldef 30,21** for example, would define a column region from column 30 for 21 columns (30-50). If the "ccoldef" syntax is used, then *col* is the starting column, and *cols* is the ending column. **ccoldef 30,50** would define the same region as above.

Syntax 2 defines a region based on a search for a starting point. For each *text* value or *regexpr* (regular expression) found, the region will begin at the column *coloffset* from the point found, and extend for *cols* columns. For example, **coldef "Customer total",-1,52** will create the region from 1 column before the occurrence of "Customer total", and extend the region for 52 columns.

Syntax 3 defines the region based on two searches, one to find the starting column, one to find the ending column to the right of the starting point. In both cases, the column position is adjusted for the offset. **coldef "Current",-1,"30-Days",-1** would define a region starting one column before the word "Current", extending to one column before the word "30-Days". If just the first string is found, then all columns from there to the last are specified. If just the last string is found, then all columns from the first through there are specified. For this reason, be sure that any absolute column regions are specified first.

Description

Column definitions are used to define columns within a row definition. Each column definition becomes a table cell (<td>…</td>), with each row in the column being separated by a line break (
). There can be up to 255 column definitions within any given row definition. Any given column will be formatted based on the first **coldef** keyword that applies to it. Columns not so defined will be displayed as mono-spaced text, using the HTML <pre> and </pre> tags.

Each column definition can define attributes that will apply to the text and cell formatting, and optionally can have a code block associated with it to add custom Business Basic coding to the data in the column.

Options are comma-separated lists of words and parameters. The options available in the column definition include:

Option	**How it gets applied**
bgcolor=*#rgb*, bgcolor=*color*	Cell gets a bgcolor=*value* attribute to control the background color. The color can be expressed as an #rrggbb hexadecimal value or as a color name supported by the target browser, such as red, blue, white, etc..
blink	Text gets <blink> attribute.
bold	Text gets <b> attribute.
bottom, top, middle	Cell gets "valign=*value*" attribute to control vertical justification. The default is "top".
center, left, right	Cell gets "align=*value*" attribute to control horizontal justification. The default is "left".
color=#rgb, color=*color*	Text gets <font color=*value*> attribute. The color can be expressed as a #rrggbb hexadecimal value or as a color name supported by the target browser, such as red, blue, white, etc..
font=*font*	Text gets <font face=*font*> attribute. Several modern browsers support this, though the *font* typeface selected may not be available on all clients.
hdr=*html text*	The top of the column gets the *html text*, followed by a line break tag. Use this option to replace top of page column headers with "in cell" column headers.
hdron=*hdron text* hdroff=*hdroff text* hdrtd=*hdrtd text*	The column header, if defined with hdr, gets these values in its <td *hdrtd*>*hdron* hdr value *hdroff*</td> structure. Be sure to turn off any *hdron text* HTML tags in *hdroff text*.
italic	Text gets <i> attribute.
image	Text is assumed to be file names that are image files, and gets treated as an <img> tag. The ufparam.txc|t file values for imagelib and imageurl are used for image processing. The imagelib value is used to locate files on the web server's file system in order to calculate width and height values (.gif and .jpg files only.) The imageurl value is prefixed to the report data when constructing the <img src="*image URL"*>.
ltrim, rtrim, trim	These three mutually exclusive options will cause UnForm to left, right, or left and right trim the text of the column when generating the HTML cell text. By default, any spaces in the data for the cell remain in the output. Use of this option may save some disk storage space and document transmission time.
noencode	If this option is present, then the text is not encoded for HTML markup entities. This should only be used if you know that the text contains valid HTML coding.

Option	How it gets applied
otheropt=*options*	The table cell gets additional attributes not otherwise specified by the other options.
size=*n*	Text gets <font size=*n*> attribute. Size ranges from 1 to 7, with 3 being considered a "normal" size.
suppress	If this word is present, then column data gets set to null.
underline	Text gets <u> attribute.

Code blocks are optional definitions associated with any given column definition. With a code block, it is possible to manipulate the text of each row in the column. A typical use of this capability might be to convert the plain text to hyperlinks, so that a column of part numbers could be linked to pages in a catalog, for example. Code blocks begin just after the opening brace "{", can extend as many lines as required, and end with a closing brace "}".

The code block is executed for each row of the column. As the code starts, the following variables can be used:

Variable	Description
attr.align$ attr.bgcolor$ attr.blink attr.bold attr.color$ attr.font$ attr.italic attr.otheropt$ attr.size$ attr.underline attr.valign$	The attr$ variable is a string template that defines the attributes to apply to the text or cell. These values match those defined above in the Options. Numeric values can be set to 0 (false) or 1 (true). String values can be set to any valid value for that attribute.
colofs	The column offset from the left edge of the text. If the column region is from column 21 through 40, then colofs will be 21. This should be treated as a read-only value.
cols	The number of columns in the region. Read only.
row	The row number within the current region, from 1 through the last row in the region. With each execution of the subroutine, the row will increase by 1. Read only.
row$	The text of the current row within the region. This can be manipulated by the code.
rowofs	The position of the current row, relative to the whole page. If you need to refer to data in some other column of the current row, use rowofs. Read-only.

Functions available for your use, in addition to any intrinsic Business Basic functions, include:

Function	**Description**
get(*col,row,cols*)	Returns text from the page, given the column, row, and cols parameters.
htmencode(*text$*)	Returns *text$* after converting HTML entities into displayable versions.
set(*col,row,cols,text$*)	Sets *text$* into the page at the given column, row, and columns.
urlencode(*text$*)	Returns *text$* after URL encoding to make it suitable for inclusion in a hyperlink.

COLWIDTH

Syntax

colwidth=*text*

Description

When UnForm generates a table for each page of a document, it defines a standard column cell width so that text that lines up vertically in the report will remain lined up in the HTML version. UnForm generates an initial single row of individual cells, using *text* as the cell width, as used in the HTML tag "<td width=*text*>".

If a *text* value, such as a pixel count or other valid HTML cell width is specified, then UnForm will use that value when defining the initial column cell sizes for each page.

FRAME

Syntax

frame=y | yes | n | no

Description

The **frame** keyword can be used in conjunction with the **multipage** keyword to control the presentation of the report. Without these options, UnForm will produce a single file (named with the **output** keyword or –o command line option, or to stdout), containing an HTML table for each page of output from the source file. With the **multipage** keyword, UnForm will produce unique files for each page of output, plus a table of contents page (whose format is controlled by the **toc** keyword). If frame is set to "y" or "yes", then an additional frame file is created for the browser to view the table of contents constantly while viewing the report pages.

The output filename generated is for the frame file if frame is set to "y" or "yes", and the table of contents file if frame is not present or is set to any other value.

This keyword is ignored if there is no *filename* specified for the output.

HDRON, HDROFF, HDRTD

Syntax

hdron=*value*
hdroff=*value*
hdrtd=*value*

Description

When a coldef **hdr=*text*** option is present, UnForm will add *text* to the top of the column, in a separate cell. In order to make a column-heading stand out, it may be desirable to give it attributes that are distinct from the column text. These keywords define HTML text attributes to add before and after any column header. **hdrtd** applies <td *value*> to the cell tag, while **hdron** and **hdroff** apply to the heading text. Values for individual row groups can be specified in the **rowdef** or **coldef** keywords.

For example, **hdron=<small>** and **hdroff=</small>** would make column headings small and bold.

Be sure to close any tags in the **hdron** value with corresponding tags in the **hdroff** variable.

LOAD

Syntax

load *filename*

Description

The **load** keyword is used to load a secondary text file into the rule file at parsing time, at the position of the **load** keyword. This provides the ability to maintain separate text files for the definitions, grouped in any manner desired. For example, a common set of options for all reports could be defined in a second file, and each report could reference that file.

UnForm will try to open the file first as named, then in the UnForm directory if it is not found. Note that the prefix setting, if present, in UnForm's config.unf file can be used to affect file searching.

Example:

```
[Report1]
load "stdoptions.txt"
```

MULTIPAGE

Syntax

multipage=y | yes

Description

If multipage is set to "y" or "yes", UnForm will generate a different document file for each page of output. The pages will be named *filename.pagenum*.htm, with *pagenum* being the sequential page number of the report.

A table of contents will automatically be generated as well, with each link in the table of contents referencing the proper document name. The table of contents file will be named one of two names: *filename*.toc.htm if a frame structure is being generated, or *filename*.htm if not. When no frame is generated, then the table of contents document becomes the base document for the output.

This keyword is ignored if there is no *filename* specified for the output.

NULLROW

Syntax

nullrow=y | yes

Description

If this value is set to "y" or "yes", UnForm will print undefined row sets as mono-spaced text, using HTML <pre> and </pre> tags. By default, UnForm will suppress any rows that have not been allocated with **rowdef** keywords.

OUTPUT

Syntax

output "*filename*"

Description

If no "-o *filename*" is specified on the command line, UnForm will use the file *filename* specified here. Use this keyword to specify a default output location for any given report.

UnForm automatically adds a ".htm" extension to *filename*.

OTHEROPT

Syntax

otheropt "*table-options*"

Description

When UnForm generates a table for each page of the document, it establishes border and width options for the table tag: <table border=*border* width=*width*>. If additional options are desired, specify them with this keyword. If present, the table tag is generated like this:

<table border=*border* width=*width table-options*>

See also the **border** and **width** keywords.

PAGESEP

Syntax

pagesep "*html code*"

Description

If a single document is generated for all pages of output (multipage is not set to "y" or "yes"), then UnForm will place a paragraph tag (<p>) between each page. If something other than a paragraph tag is desired, then specify the HTML code in the **pagesep** keyword.

The **pagesep** value can contain global string values generated from code blocks by referencing the string value name inside square brackets.

For example: **pagesep "<p><hr>[pagehdr]"** would generate a paragraph tag plus a horizontal rule, followed by the value in the global string "pagehdr", defined with the STBL() function in a prepage{} or prejob{} code block.

PREJOB, PREPAGE, POSTJOB, POSTPAGE

Syntax

prejob | postjob | prepage | postpage {
code block
}

Note: the opening brace "{" needs to be on the same line as the keyword. The closing brace may follow the last statement, or be on the line below the last statement.

Description

These keywords are used to add Business Basic processing code to the document generation process. They represent four different subroutines that UnForm executes at specific points during processing. The *code block* can be an arbitrary number of Business Basic statements; the total number of statements in all code blocks can be about 6,000 (or less, depending on program size limits imposed by the run-time environment).

- **prejob** executes after the rule set has been read, and after the first page is read, but before any printing takes place. Use this code to open files or databases, prepare SQL statements or string templates, create user-defined functions, and initialize job variables.
- **postjob** executes after the last page has been printed. Use this to close out your logic, such as adding totals to log reports. There is no need to close files, since UnForm will RELEASE Business Basic.
- **prepage** executes after each page is read, but before any printing takes place. Use this to gather data associated with any page, or to modify the content of the text if you need such modifications to apply to all copies.
- **postpage** executes after the last copy of each page has printed.

Any valid Business Basic programming code can be entered, including I/O logic, loops, variable assignments, and more. Program to your heart's content. UnForm will add extensive error handling code within your code, and report syntax errors to the error log file or a trailer page.
You may use the following variables and functions in your *code block*:

- **text$[all]** is a one-dimensional array of the text for the page. For example, text$[2] is the second line of the page.
- **mid(*arg1$,arg2,arg3*)** (or fnmid$(*arg1$,arg2,arg3*)) is a function that safely returns a substring without generating an error 47 if the value in *arg1$* isn't long enough to accommodate position *arg2* and length *arg3*.

- **get(*col,row,length)*** (or fnget$(*col,row,length*)) is a function that safely returns text from the text$[all] array, without substring or array out-of-bounds errors.
- **set(*col,row,length,value$)*** (or fnset$(*col,row,length,value$*)) is a function that places *value$* in the text$[all] array at the place indicated. It returns *value$*.
- **err=next** may be used for any err=*label* option in any function or statement, in order to force UnForm's error trapping to ignore an error. You may, of course, name your own err=*label* if desired.

When using variables and line labels, you should avoid using any values that begin with "UF_". UnForm reserves all such variables and labels for its own use. You may use a backslash (\) at the end of a line to continue the statement on the next line. Lines prefixed with "#" are not added to the code.

A discussion of programming in Business Basic is outside of the scope of this manual. If your needs require programming, then it would be advisable to hire a professional Business Basic programmer, acquire training for a technical member of your staff, or contract with SDSI for your needs.

Column definitions can also have code blocks, which are executed as each row of a column definition is generated. See the **coldef** keyword for more information.

ROWDEF

Syntax

1. [rowdef | crowdef] row, rows, *options*
   { *code block* }

2. rowdef "*text | ~regexpr", rowoffset, rows, options*
   { *code block* }

3. rowdef "*text | ~regexpr", rowoffset, "to-text | ~to-regexpr", to-rowoffset options*
   { *code block* }

Syntax 1 defines an absolute row region. **rowdef 5,3** for example, would define a row region starting with row 5, and extending 3 rows down (5-7). If the "crowdef" format is used, then *row* is the starting row, and *rows* is the ending row. **crowdef 5,7** would define the same region as **rowdef 5,3**.

Syntax 2 defines a region based on a search for a starting row that contains the text or matches the regular expression. For each *text* value or *regexpr* found, the region will begin at the row *rowoffset* from the point found, and extend for *rows* rows. For example, **rowdef "Customer total",0,1** will create a region from each row containing "Customer total" (0 offset is that row), and extending for 1 row (just that row).

Syntax 3 defines the region based on two searches, one to find the first row, one to find the ending row below the starting row. In both cases, the row used for the region is adjusted for the offset. **rowdef "Customer:",1,"Customer:",-1** would define a region between each occurrence of the text "Customer:". If just the first string is found, then all rows from there to the last are specified. If just the last string is found, then all rows from the first through there are specified. For this reason, be sure that any absolute regions are specified first.

Under format 3, if the last string is not found, UnForm will continue that row definition on the page following the first unallocated row at the time this row definition is evaluated on that page.

Description

Row definitions are used to define sets of rows for which a given group of column definitions would apply. Each row definition defines a group of rows that will be presented within a single table row (<tr> ... </tr>). Under any given row definition, place the column definitions (**coldef** keywords) that will be used to format the rows.

For example, an A/R Aging Report might contain a report heading, column headings, one or more customer headings, and, under each customer heading, one or more detail lines. At the end of the detail lines would be customer totals. This report would have five row definitions, for each type of row: report heading, column heading, customer headings, detail lines, and totals. Each of these types of rows

will have its own set of column groups (or in some cases, no column groups at all, allowing simple mono-spaced presentation.)

There can be up to 255 row definitions within any rule set.

Each row definition can define attributes that will become defaults for the text and cell formatting of all the column definitions. Additionally, row definitions can define an option called "suppress", which causes UnForm to suppress the display of the row region. A comma separates each option.

Option	**How it gets applied**
bgcolor=*#rgb*, bgcolor=*color*	Cell gets a bgcolor=*value* attribute to control the background color. The color can be expressed as an #rrggbb hexadecimal value or as a color name supported by the target browser, such as red, blue, white, etc..
blink	Text gets <blink> attribute.
bold	Text gets <b> attribute.
bottom, top, middle	Cell gets "valign=*value*" attribute to control vertical justification. The default is "top".
center, left, right	Cell gets "align=*value*" attribute to control horizontal justification. The default is "left"
color=#rgb, color=*color*	Text gets <font color=*value*> attribute. The color can be expressed as an #rrggbb hexadecimal value or as a color name supported by the target browser, such as red, blue, white, etc..
font=*font*	Text gets <font face=*font*> attribute. This is supported by several modern browsers, though the *font* typeface selected may not be available on all browser clients.
hdr=*html text*	The top of the column gets the *html text*, followed by a line break tag. Use this option to replace top of page column headers with "in cell" column headers.
hdron=*hdron text* hdroff=*hdroff text* hdrtd=*hdrtd text*	The column header, if defined, gets placed in a cell with <td> attributes specified *hdrtd text*, and text attributes *hdron text* and *hdroff text*. Be sure to turn off any *hdron text* HTML tags in *hdroff text*.
italic	Text gets <i> attribute.
noencode	If this option is present, then the text is not encoded for HTML markup entities. This should only be used if you know that the text contains valid HTML coding.
otheropt=*options*	The table cell gets additional attributes not otherwise specified by the other options.
size=*n*	Text gets <font size=*n*> attribute. Sizes range from 1 to 7, with 3 being considered a "normal" size.
suppress	The rows are not displayed.
tr	Each row in the row group gets a <tr> tag, ensuring that column definitions, even if they contain data values of

Option	How it gets applied
	varying height, will remain horizontally contiguous. If the cells contain only text, this is generally not required, but if some cells contain images, this keyword will likely be required.
underline	Text gets <u> attribute.

TITLE

Syntax

title "*title text*"

Description

The title for any report can be defined in the rule set with this keyword. Once defined, anywhere in HTML output templates that the tag "[$title]" is placed, this text will be substituted.

TOC

Syntax

toc=y | yes | li | list | sh | short

Description

If this keyword is set to "y" or "yes", UnForm will generate a simple table of contents by constructing hyperlinks to each page generated. The hyperlinks are placed either at the top of the document, in a separate main document, or in a document referred to as the table of contents in a frame.

The following templates use a table of contents. Templates refer to files in the UnForm directory, and are referenced in the parameter file under the "[html]" section: "both=" and "toc=". In each case, the placement of the table of contents is based on the placement of the tag "[$toc]" within the template file.

The text displayed for each hyperlink is generated from the "pagenum=" item of the "[html]" section of the parameter file (ufparam.txc or ufparam.txt.) This text can also be generated by Business Basic code in the prepage{} or postpage{} code blocks, by setting the string variable "toc$" to the value desired.

If the keyword is set to "li" or "list", then the hyperlinks are created within an HTML unordered list (<ul> ...), and will normally be displayed as a bullet list.

If the keyword is set to "s", "sh", or "short", then the table of contents links consist of just the pagenum descriptor followed by each page number, with no line breaks or bullets. In this case, any code that sets the value of toc$ is ignored.

This keyword is ignored if there is no *filename* specified for the output.

WIDTH

Syntax

width=*value*

Description

The tables generated by UnForm for each page will normally occupy the entire width of the page, and will therefore set the table width to 100%: <table width=100% ...>. If you would prefer a different width setting, define it with this keyword. Be sure that if the value is a percentage of the screen, it has a trailing "%".

See also the **otheropt** and **border** keywords.

SAMPLE HTML RULE SET

Below are sample rule sets defined in the sample rule file, samphtml.rul. The sample text input files used by UnForm for the PCL output examples are redefined here for HTML. Comments are interspersed in the rule sets to help clarify which keywords perform which tasks.

AGING REPORT SAMPLE

To produce this aging report sample to a file, execute the following command:

uf60c -i sample3.txt -o aging.htm -p html –f samphtml.rul

You can substitute a different path/file name for "aging" to produce the HTML file elsewhere, such as in the HTML document tree of your Web server.

The form is called "aging" to distinguish it from other rule sets. If the "-r aging" option is used on the command line, then this set will be used.

```
[aging]
```

A detect statement identifies a report as the one defined by this rule set. If no "-r ruleset*" option is used on the command line, then this detect statement will be evaluated. If the text "Detail Aging" appears in any column on row 2, this rule set is used.*

```
detect 0,2,"Detail Aging"
```

The HTML output will produce 132 columns and 66 rows per page.

```
cols 132
rows 66
```

Any text consisting of 3 or more dashes will be erased. This removes all the dashed underlines at customer totals. There are other ways to accomplish this, including defining a row set and using the suppress option, or using a prepage{} code block to erase such text from the text$[] array.

```
hline "---"
```

The title used in HTML output for this report will be "Aging Report".

```
title "Aging Report"
```

If this line were not commented out (with the #), then anytime this rule set was used and no "-o filename*" was present on the command line, the output would go to "/tmp/aging.htm."*

```
#output "/tmp/aging"
```

This report will be generated in multiple files (one per page), with a table of contents page, and with an HTML frame construct.

```
multipage=y
toc=y
frame=y
```

*Between each page will be an HTML <p> tag (a paragraph separator). Any HTML text could be supplied, including references to global strings inside square brackets ([*variablename*]). The hdron/hdroff keywords supply HTML codes to place before and after any column definition headings, defined with the hdr=*text *option in the coldef and rowdef keywords.*

```
pagesep <p>
hdron=<i><b>
hdroff=</b></i>
```

This rowdef keyword defines a row set from row 1 for 5 rows. All column definitions within this row will default to a background color RGB hex value of FFE0E0 (lots of red, high green and blue content).

```
rowdef 1,5,bgcolor=#ffe0e0
```

For the above row set, there are three column sets: 1 through 10, 11 through 110, and 111 through 132. The columns are left, center, and right justified, respectively. Otherwise, except for the background color, the browser will use its default values for displaying the data.

```
coldef 1,10,left
coldef 11,100,center
coldef 111,22,right
```

This row definition causes UnForm to suppress display of rows 6, 7, and 8 (the column heading information). The rule set will define the column headers as necessary in other row sets.

```
rowdef 6,3,suppress
```

Each customer has a heading line, distinguished by the occurrence of a phone number in those rows. The initial quoted value "~\(...-...-....\)" instructs UnForm to search for a regular expression match that looks like a U.S. phone number in parentheses. From any and all such rows, it will start at 0 rows up or down, and continue for 1 row. This defines those and only those rows that contain the phone numbers. Columns defined for those rows will be bold, with blue text on a white background. As no columns are defined under this row definition, UnForm allocates one column set the full 132 columns wide, and applies the row defaults to the text.

```
# Customer header
rowdef "~\(...-...-....\)",0,1,bold,color #0000ff,bgcolor #ffffff
```

The invoice detail lines represent the most complicated of the row definitions, as there are numerous columns with two different formats. We define constants for the two formats (left and right justification being the only difference.) Then the rows are defined as any rows that contain a date structure of 2 characters, a slash, 2 characters, a slash, and 2 more characters. Note that even though some heading

rows have this structure, those rows have already been allocated by prior row definitions and won't confuse things here. UnForm searches for any row with a date. Then starting from that row (row offset of 0), it searches for a row that contains 5 dashes. If such a row is found, then the row set goes through the row before (row offset -1) the dashes. If no such row is found, then the row set goes through the last row on the page.

```
# Invoice lines
const LEFT="bgcolor=#e8e8e8,color=black"
const RIGHT="bgcolor=#e8e8e8,color=black,right"
rowdef "~../../..",0,"-----",-1
```

Each invoice line is made up of 13 columns of information. Each has been defined by the ccoldef keyword by starting and ending column values. Additionally, each is given a header value that will appear at the top of the column, and a constant that references other attributes defined earlier in the rule set.

```
ccoldef 1,10,hdr="Invoice",LEFT
ccoldef 11,20,hdr="Due Date",LEFT
ccoldef 21,31,hdr="PO Number",LEFT
ccoldef 32,39,hdr="Ord Number",LEFT
ccoldef 40,45,hdr="Terms",LEFT
ccoldef 46,52,hdr="Type",LEFT
ccoldef 53,64,hdr="Future",RIGHT
ccoldef 65,75,hdr="Current",RIGHT
ccoldef 76,86,hdr="30 Days",RIGHT
ccoldef 87,97,hdr="60 Days",RIGHT
ccoldef 98,108,hdr="90 Days",RIGHT,color=red
ccoldef 109,119,hdr="120 Days",color=red,RIGHT
ccoldef 120,132,hdr="Balance",right,bold,RIGHT
```

The customer totals occur just below the row of dashes at the end of each customer's invoices. This row definition therefore searches for any rows containing 5 dashes, then starts 1 row down, and continues for just 1 row.

```
# Customer totals
rowdef "-----",1,1
```

The first 52 columns make up one column set. The report provides no text, so we include a code block for this column that sets row$ to "Customer Totals:". Note that if this row set contained more than a single row, we could say 'if row=1 then row$="Customer Totals:'. The remaining column sets just apply right justification to the column values.

```
ccoldef 1,52,right
{row$="Customer Totals:"}
ccoldef 53,64,right
ccoldef 65,75,right
ccoldef 76,86,right
ccoldef 87,97,right
ccoldef 98,108,right
```

```
ccoldef 109,119,right
ccoldef 120,132,bold,right
```

INDEX

[tcpports] section, 19
30 day demo activation, 27
Across, 54
Activation keys, 27
advanced.rul, 145
Alias lines, 21
Alignment, 92, 133
Application integration, 25
Attach, 55
Author, 57
Barcode
  PCL, PDF, 58
  Zebra, 61
bbpath, 14
BBx
  code block note, 122
  code blocks, 184
BBx integration, 21
bbxread function, 194
bbxread(), 14
Bin, 64
boj, bop, eoj, eop, 65
Bold, 66
Box, 68
Boxr, 71
Business BASIC
  code blocks, 184
Case-conversion, 92
Characters
  -testpr option, 38
Check printing, 112
client-server architecture, 7
Code blocks
  arrays, 185
  BASIC functions, 198
  error codes, 204
  number data, 186
  operators, 186
  precopy, etc. commands, 121
  programming, 184
  special functions, 194
  special variables, 190
  string data, 186
  variables, 185
Collation, 77
Color
  box command, 69
  font command, 92
  text command, 133
Color images
  -gw option, 33
Cols, 74
Command line length
  -z option, 39
Command line options, 31
Commands
  across, 54
  attach, 55
  author, 57
  barcode, PCL and PDF, 58
  barcode, Zebra, 61
  bin, 64
  boj, bop, eoj, eop, 65
  bold, 66
  box, 68
  boxr, 71
  cols, 74
  compress, 75
  const, 76
  copies, 77
  cpi, 78
  crosshair, 79
  detect, 80
  down, 82
  dpi, 83
  duplex, 86
  email, 87
  erase, 89
  fixedfont, 90
  font, 91
  gs, 94
  hline, 95
  hshift, 96
  if copy, 97
  if driver, 98
  image, 99
  italic, 66
  keywords, 104
  landscape, 105
  light, 66
  lpi, 107
  macro, 108
  margin, 110
  merge, 111
  micr, 112
  move, 113
  notext, 115
  outline, 116
  output, 117
  page, 118
  paper, 119
  pcopies, 77
  portrait, 120
  precopy, 121

protect, 124
rows, 125
shade, 126
shift, 128
subject, 129
symset, 130
text, 131
title, 136
tray, 137
underline, 66
vline, 140
vshift, 141
Compress, 75
Concepts, 47
Configuration, 14
Const, 76
Content-based rule files, 53
Copies, 77
Copy blocks, 97
Cpi, 78
Crosshair, 79
Crosshair pattern
-x option, 39
cut function, 194
Detect, 80
Document imaging conversion, 16
Down, 82
Dpi, 83
Dsn_sample, 84
Duplex, 86
with an attachment, 55
Email, 87
command line options, 32
configuration, 206
email code block function, 194
Emergency activation, 27
Encryption, 124
end if, 97
env function, 195
eoj, eop,boj, bop, 65
Erase, 89
Error codes, 204
exec function, 195
Firewalls, 14
Fit to width, 92
Fixedfont, 90
Flow of processing, 44
Font, 91
get function, 195
Ghostscript, 16
Graphical shading, 33, 94
Greenbar option, 33
Grid drawing, 69
Gs, 94
Hline, 95
HP JetDirect, 19
HP/GL
-nohpgl option, 34
Hshift, 96
HTML driver, 213
HTML format
-p option, 35
if copy, 97
If driver, 98
Image, 99
Image Alchemy, 16
Image conversion and scaling, 100
Image Magick, 16
Images
scaling and conversion, 16
Input file
-i option, 33
Installation
clients, 12
configuration, 14
Unix CD, 8
Unix download, 10
Windows, 11
Integration with applications, 25
IP addresses, 14
Italic, 66
job functions, 195
Job status
-jobs, -myjobs, 40
Jobs
job code block functions, 195
Justification, 92, 133
Keywords, 104
Landscape, 105
Landscape orientation
-land option, 33
Laser format
-p option, 35
left function, 196
Library, 14
Licensing, 27
Light, 66
Lines per page, 118
-page option, 36
Link file, 23
Logging, 14
lower function, 196
Lower-case, 92
Lpi, 107
Macro, 108
Macros
working with, 142
Mailcall reference, 209
mailcall.ini, 206
Margin, 110

mcut function, 196
Merge, 111
mget function, 196
MICR, 112
mid function, 196
Mono-spaced text, 92, 133
mount commands, 8
Move, 113
NAT, 14
Notext, 115
Order of operations, 44
Orientation
  -land option, 33
  landscape command, 105
  portrait command, 120
Outline, 116
Output, 117
  -o option, 34
Output format
  -p option, 35
Page, 118
Page length
  -page option, 36
Paper, 119
Paper size
  -paper option, 36
Parameter passing
  -prm option, 36
parse function, 197
Pass-through printing, 38
PCL format
  -p option, 35
Pcopies, 77
PDF
  command line options, 36
  compress command, 75
  encryption, 124
  keywords command, 104
  outline command, 116
  protect command, 124
  subject command, 129
  title command, 136
PDF format
  -p option, 35
Perl, 7, 12
Permanent activation, 27
Pitch, 92, 132
Points, 92, 132
Portrait, 120
Precopy, 121
Process flow, 44
Programming, 184
proper function, 197
Proportional text, 92, 133
Protect, 124
ProvideX
  code blocks, 184
ProvideX integration, 23
Regular expressions, 144
right function, 197
Rounded boxes, 71
Rows, 125
Rule file
  -f option, 33
Rule files, 52
  content-based, 53
Rule set
  -r option, 37
Sample rule files, 145
Security, 14
Serial numbers, 27
Server operation, 7, 11
set function, 197
Shade, 126
Shaded text, 92, 133
Shift, 128
simple.rul, 145
Slave printing, 38
SMTP server, 206
Special characters, 38
sub function, 197
Subject, 129
Sub-jobs
  job code block functions, 195
Symbol sets, 92, 130, 132
  -testpr option, 38
Symset, 130
TCP/IP
  -port option, 36
  -server option, 37
TCP/IP monitor, 19
TCP/IP ports, 14
Text, 131
Text sizes, 92, 132
Timeouts, 14, 38
Title, 136
Tray, 137
trim function, 197
uf$ variable, 191
uf60c.ini, 12
uf60d options, 7
uf60d.ini, 14, 16
uf6ptr print driver, 23
uf6ptr ProvideX print driver, 23
ufsetup.sh, 8, 10
Underline, 66
UnForm
  client-server architecture, 7
  concepts, 47
  introduction, 6

Unix
CD installation, 8
client installation, 12
download installation, 10
mount commands, 8
server operation, 7
upper function, 197
Upper-case, 92
Variables, 190
Version 6 features, 41
Vline, 140
Vshift, 141
Windows
client installation, 12
server installation, 11
server operation, 11
Zebra format
-p option, 35