

# UnForm<sup>TM</sup> User Guide

Version 4.1

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](mailto:sdsi@synergetic-data.com)  
Web page: <http://synergetic-data.com>

UnForm is Copyright ©1994-2000 by Allen D. Miglore. All rights reserved.  
UnForm is a trademark of Synergetic Data Systems, Inc.  
Other product names used herein may be trademarks or registered trademarks of their respective owners.

## **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
INSTALLATION .....	7
INTEGRATING UNIFORM WITH BB <sup>X</sup> 4 OR PRO/5 .....	10
INTEGRATING UNIFORM WITH PROVIDEX.....	13
INTEGRATING UNIFORM WITH NON-BUSINESS BASIC APPLICATIONS.....	15
FILES USED BY UNIFORM .....	16
DEMO AND LIVE INSTALLATIONS .....	18
USER COUNTS .....	19
UNIFORM OPTIONS.....	20
VERSION 4 FEATURES .....	24
UNIFORM 4.1 FEATURES.....	28
RULE FILES .....	29
ACROSS .....	30
ATTACH.....	31
BARCODE (PCL,PDF).....	32
BARCODE (ZEBRA) .....	34
BIN.....	37
BOJ, BOP, EOJ, EOP .....	38
BOLD, ITALIC, LIGHT, UNDERLINE .....	39
CBOLD, CITALIC, CLIGHT, CUNDERLINE.....	39
BOX, CBOX .....	40
CONST.....	42
COLS.....	43
COPIES / PCOPIES .....	44
CPI.....	45
DETECT .....	46
DOWN .....	47
DPI .....	48
DUMP .....	49
DUPLEX .....	50
ERASE, CERASE .....	51
FIXEDFONT.....	52
FONT, CFONT .....	53
HLINE.....	55
IF COPY ... END IF .....	56
IF DRIVER ... END IF .....	57
IMAGE.....	58

ITALIC.....	60
LANDSCAPE.....	61
LIGHT.....	62
LPI.....	63
MACRO .....	64
MACROS .....	65
MARGIN.....	66
MICR.....	67
MOVE, CMOVE.....	68
NOTEXT .....	69
OUTLINE.....	70
OUTPUT .....	71
PAGE .....	72
PAPER .....	73
PORTRAIT .....	74
PRECOPY, PREDEVICE, PREJOB, PREPAGE .....	75
POSTCOPY, POSTDEVICE, POSTJOB, POSTPAGE .....	75
ROWS .....	79
SHADE, CSHADE.....	80
SHIFT.....	82
SYMSET .....	83
TEXT .....	84
TRAY .....	87
UNDERLINE .....	88
UNITS .....	89
VLIN.....	90
VSHIFT .....	91
<b>CONCEPTS, PRIMER, AND TIPS.....</b>	<b>92</b>
<b>WORKING WITH MACROS .....</b>	<b>97</b>
<b>REGULAR EXPRESSIONS .....</b>	<b>99</b>
<b>SAMPLE RULE SETS .....</b>	<b>100</b>
INVOICE - INVOICE FOR PRE-PRINTED FORM.....	101
STATEMENT - PLAIN PAPER FORM, TWO PAGE FORMATS IN SAME JOB .....	106
AGINGREPORT - ENHANCED AGING REPORT .....	110
LABELS – TEXT LABELS TO LASER LABELS .....	112
132X4 – MULTI-UP, SCALED REPORTING .....	114
ZEBRA LABEL – ZEBRA® LABEL PRINTER EXAMPLE.....	115
PDF OUTLINE SAMPLE.....	117
<b>PROGRAMMING FUNDAMENTALS.....</b>	<b>119</b>
<b>EMAIL INTEGRATION.....</b>	<b>129</b>
<b>NESTED RULESET EXECUTION .....</b>	<b>134</b>
<b>HTML OUTPUT.....</b>	<b>135</b>
CREATING HTML .....	136
HTML CONFIGURATION .....	138
HTML OUTPUT TEMPLATES .....	140
HTML RULE SETS .....	143

BORDER.....	144
COLDEF .....	145
COLWIDTH.....	149
FRAME .....	150
HDRON, HDROFF, HDRTD .....	151
LOAD.....	152
MULTIPAGE.....	153
NULLROW .....	154
OUTPUT .....	155
OTHEROPT .....	156
PAGESEP.....	157
PREJOB, PREPAGE, POSTJOB, POSTPAGE.....	158
ROWDEF .....	160
TITLE.....	163
TOC.....	164
WIDTH .....	165
SAMPLE HTML RULE SET.....	166
AGING REPORT SAMPLE .....	166
<b>INDEX.....</b>	<b>170</b>

# INTRODUCTION

UnForm is a software utility 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:

```
cat file-name | UnForm -f acct.rul | lp -dlaser -s 2>/dev/null
```

UnForm can also work in MS-DOS or Windows environments, as long as the application can produce a file and then execute UnForm to process the file and output it. In BB<sup>X</sup>PROGRESSION/4 or Visual PRO/5 environments, this is easily accomplished with some simple changes to the BBx configuration file.

UnForm is unique in its ability to analyze report output to determine what, if any, customization to apply. For most reports used by an application, UnForm will perform no manipulation of the text. When a report is detected that requires enhancements, however, 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 BB<sup>X</sup>PROGRESSION/4 or PRO/5 programming environment to add true programmed intelligence to any form.

The enhanced output can be used to simulate pre-printed forms, or can 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 Laserjets and compatible printers beginning with the HP Laserjet III, many UNIX faxing software packages, and other products. Using the **pdf** driver, all printers available to the **pdf** reader can be used.

# INSTALLATION

## UNIX 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 the mount directory, /mnt, is available, and standard CD device names. 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/c1b03t10 /mnt
AIX	mount -vcdvfs -r /dev/cd0 /mnt
Sun Solaris	mount -rt hsfs /dev/sr0 /mnt
HP/UX	mount -r /dev/dsk/c1d0s2 /mnt

3. Change to the UnForm4 UNIX directory in the mount directory: **cd /mnt/unform41/unix**
4. Run the install script: **./install.sh**, or if you do not have execute permission to the file, **sh install.sh**. Follow the prompts to select a directory for UnForm, and choose if you will be supplying your own run-time bbx4, pro5, or pvx engine, or installing a bundled version that includes a pro5 engine. If you install a bundled version, you will need to select the version that is appropriate for your operating system.
5. UnForm will then be installed by copying the UnForm files to the selected directory, and executing the setup script ./ufsetup in the UnForm directory. Once UnForm has been installed from the CD, ./ufsetup can be executed at any time from the UnForm directory. See step 4 in the next section for more information about ufsetup.
6. Use the **UnForm -v** command to ensure UnForm is installed and setup correctly. The output from this command will display the serial number used and state that this is a demo version.
7. See **Demo and Live Installations** section for activation instructions.

## UNIX 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/uniform**  
              **cd /usr/uniform**

3. Uncompress and extract UnForm from the download file:

**uncompress uf41\_xxx\_tar.Z**  
**tar xvf uf41\_xxx\_tar**

4. Execute the UnForm setup script, which will normally ask you where BB<sup>X</sup>PROGRESSION/4 or PRO/5 or ProvideX executable is located on your system. UnForm uses BB<sup>X</sup>PROGRESSION/4 or PRO/5 as a run-time environment. If you have a version of UnForm that includes a run-time, then this question won't be asked.

**./ufsetup**

If you see the question "What directory contains BBx4, or Pro5, or PVX?", then you must answer with the correct directory, or the installation will not complete, and you will have to try again when you know the correct directory information.

ufsetup will then prompt for a default rule file, normally uniform.rul. This will be used if the command line doesn't contain a "-f" argument.

Once a valid directory has been entered, the ufsetup script will create a script called "/usr/bin/UnForm". That is the script that is used as a pipe, and as it is in the /usr/bin/ directory, all tasks should be able to access it by using the command "UnForm".

5. Use **UnForm -v** command to ensure UnForm is installed and setup correctly. The output from this command will display the serial number used and state that this is a demo version.
6. See **Demo and Live Installations** section for activation instructions.

### **Windows 95/98/NT Installation Instructions:**

1. From the CD, use explorer to locate the D:\uniform41\windows directory, and double-click the setup.exe program. If you downloaded UnForm from the Internet, simply execute the downloaded executable. Follow the on-screen prompts from the installer to install UnForm to your system. Note that the uniform.exe program and the associated uniform.ini file are installed both in the Windows directory and in the UnForm directory.
2. Click the **Run ufsetup** icon. This will conditionally rename certain files and prompt for where your run-time Business Basic is located. The full path to the executable must be given. This



value will be stored in `unform.ini` in the Windows system directory. It will also prompt for a default rule file, to be used in case the UnForm command line doesn't contain a "-f" argument.

3. Use the **UnForm Information** icon to ensure UnForm is installed and setup correctly. The output from this command will display the serial number used and state that this is a demo version.
4. If you plan to use Windows print driver output, you will need to install Adobe Acrobat. If you already have Acrobat installed, you can skip this step. Otherwise, you should click the **Install Adobe Acrobat** icon. UnForm's default configuration for the Acrobat Reader location is `C:\Program Files\Adobe\Acrobat 4.0\Reader`, the default installation directory for Acrobat 4.0. If Acrobat is installed elsewhere, then the first UnForm execution to a Windows print driver will prompt for the correct location, or you can edit the `reader.ini` file directly in the UnForm directory.
4. See **Demo and Live Installations** section for activation instructions.

# INTEGRATING UNFORM WITH BB<sup>X</sup>4 OR PRO/5

BBx handles printers via *alias* lines in a configuration file. 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. To convert any printer definition for use with UnForm, look at the following examples.

## UNIX Spool System Alias

If your printer's alias line looks like one of these lines:

```
alias P1 ">lp -dxyz -s 2>/dev/null" "Printer Name" ... various modes...  
alias P1 "[lp -dxyz -s 2>/dev/null" "Printer Name" ... various modes...
```

Then you need to modify the alias line to look like this:

```
alias P1 "|UnForm -f my.rul | lp -dxyz -oraw -s 2>/dev/null" "Printer Name" ... various modes ...
```

This simply places UnForm in the pipeline ahead of the spooler. The modes after the printer's name can and should remain, since UnForm will pass those through on any print jobs that are not enhanced, and they will be needed to retain the formats used by the existing reports. Note that only modes compatible with PCL printers should be used. Don't use UnForm on an alias line originally designed for another printer type.

## UNIX Device Alias

If your printer definition prints directly to a device, rather than using a spooler, the alias line will look like this:

```
alias P1 /dev/lp0 "Printer Name" ... various modes...
```

You need to convert this to a pipe to UnForm, and in turn instruct UnForm to send its output directly to the device:

```
alias P1 "|UnForm -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.

If you will be printing binary files on UNIX systems, such as logos via the image command, you might need to modify the method by which UNIX talks to the printer. In most cases, UNIX performs “post processing” on everything that is sent to the printer. Post processing adds information to the output, such as a carriage return whenever a line-feed is encountered. For a binary image, this post processing causes malformed images to be printed. The method for turning off post processing varies from system to system, but generally will involve the use of the UNIX “stty” command.

On the UNIX System V spooler, the spooler print model needs to support an option to turn off post processing (usually “-oraw” on the lp command line.) In the printer interface file, make sure the following logic, or similar logic, is present in the printer interface script, and is invoked by the alias line with the “-oraw” option of the “lp” command:

```
graphics=no

for i in $options
do
    case $i in
        r|raw) graphics=yes ;;
    esac
done

if [ "$graphics" = "yes" ]
then
    stty -opost 0<&1
fi
```

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.

## Windows Alias Lines

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

If the original alias line looks like this:

**alias P1 LPT1 “Printer P1” ... *various modes* ...**

Then you can modify the line to look like this:

**alias P1 P1.TXT "Printer P1" *various modes*, LOCK=P1.LCK, O\_CREATE, O\_TRUNC, EXECOFF="unform.exe -ix P1.TXT -o LPT1 -f my.rul"**

In the above example, a file called P1.TXT is created, using the modes O\_CREATE and O\_TRUNC to force a fresh copy of the file each time, and using a lock file to prevent two users from writing to the same file at the same time. 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 “unform.exe -i c:\data\p1.txt...”, you would need to specify “unform.exe -i c:\\data\\p1.txt ...”

### **Printer Aliases Within UnForm**

While UnForm can print directly to a device, or to the standard output handle on UNIX, it may be helpful to use a BBx alias for printing. For example, in a Novell environment, where there is no “device” that can be used to send output to the Novell spool queue, a BBx print alias that names nspool as its device can be used. It can also simply be more convenient to refer to an output device by a short alias rather than a long pipe expression. Printer aliases for use by UnForm can be defined in the file config.unf in the UnForm directory. When defining such an alias, be sure that no modes are defined that would cause output to the device (SP, PTON, etc.), as these would conflict with what UnForm sends to the device. These aliases are referenced in UnForm’s command line with the “-o *alias*” option.

### **Special Issues Under Visual PRO/5**

Under Visual PRO/5, the IO device is invalid and UnForm must run under a minimized terminal alias. This alias must be defined in the file config.unf, as in the following example:

```
alias TINV syswindow “” minimized,title=”UnForm”
```

Note that you can’t use the “sysprint” device to access the Windows print driver, at least until Visual PRO/5 supports the use of the ‘BO’ mnemonic on the device. If you need to access a network printer, use the print manager to define a printer, select “Capture Printer Port”, enter the network printer address, and select the LPT*n* device. Then in UnForm, use LPT*n* as the output device.

# INTEGRATING UNFORM WITH PROVIDEX

ProvideX works with printers in a very different manner from BBx. Instead of a central configuration file, each printer is defined as a distinct disk file, called a link file. Link files contain two items: an output device, and a print driver. ProvideX print drivers are ProvideX programs stored in the \*lib/\_dev directory. These programs can be modified as any other ProvideX programs.

UnForm is provided with several ProvideX print driver programs, one for each of the different types of output UnForm can produce. These drivers are copied to the ProvideX lib/\_dev directory whenever you run ufsetup (or ufsetup.exe on Windows) and setup UnForm to run with a ProvideX executable. To use these drivers, use the ProvideX utilities to define printer link files. The link files should point to the output device you want (like ">lp -dxx", \\server\printer, or LPTx), and specify the proper UnForm driver for the type of output you need (uf\_laser, uf\_pdf, etc.)

In most cases, the drivers supplied with UnForm can be used without modification. They are designed to route application output to a temporary work file, then launch UnForm with appropriate options to produce output to the device specified in the link file. In some cases, a driver may need to be modified. For example, if the default rule file is not correct for all print jobs, a copy of a driver may be created, and a specific rule file specified. Be sure to always modify copies rather than the original driver. Otherwise, the next time you update UnForm, your customized driver will be overwritten.

Below are the possible changes to each of the driver files.

**uf\_laser** This driver is used for laser printer output on all platforms. It is possible to override the default rule file by changing the assignment of %UF\_RULEFILE\$ to the rule file name. Also, if you need to execute a script or executable named something other than UnForm, you can change %UF\_UNFORM\$. To make the changes, just look for these lines and modify them:

```
LET %UF_RULEFILE$=""  
LET %UF_UNFORM$="UnForm"
```

**uf\_pdf** Like the laser driver, you can modify the rule file or the executable. In addition, since PDF output typically goes to a file, you can override the default output device or file (from the link file definition) to some other value by modifying %UF\_PDFFILE\$ to the file name desired. You might, for example, prompt the user for a file name, set the response into a global string, and set %UF\_PDFFILE\$ to that global string.

```
LET %UF_RULEFILE$=""  
LET %UF_UNFORM$="UnForm"  
LET %UF_PDFFILE$=""
```

**uf\_win, uf\_winpww** These two drivers trigger UnForm to send output to the Windows printer or a Windows preview by producing PDF output and then launching a copy of the Adobe Acrobat Reader on either the local machine, or the WindX client if the user is running WindX. Like the other drivers, you can override the default rule file and/or change the executable name for UnForm.

If you are running on WindX, you must also do two things: 1) install an Adobe Acrobat Reader on each client PC, and 2) modify the %UF\_CLIENTREADER\$ to full path to the reader executable *from the client's perspective*. This means all clients must have the reader installed in the same location. When running under WindX, you must be running ProvideX version 4.11 or higher.

```
LET %UF_RULEFILE$=""  
LET %UF_UNFORM$="UnForm"  
LET %UF_CLIENTREADER$="c:\acrobat3\reader\acrord32.exe"
```

**uf\_html** Like the other drivers, you can modify the rule file or the executable. In addition, since html output typically goes to a file, you can override the default output device or file (from the link file definition) to some other value by modifying %UF\_HTMLFILE\$ to the file name desired. You might, for example, prompt the user for a file name, set the response into a global string, and set %UF\_HTMLFILE\$ to that global string.

```
LET %UF_RULEFILE$=""  
LET %UF_UNFORM$="UnForm"  
LET %UF_HTMLFILE$=""
```

**uf\_zebra** The UnForm zebra driver requires some command line arguments to specify the print density and label size. These are specified by the %UF\_DOTSPERMM\$ and %UF\_LABELSIZE\$ values. For a detailed description of these arguments, look at the -p and -paper options in the UnForm Options chapter.

```
LET %UF_RULEFILE$=""  
LET %UF_UNFORM$="UnForm"  
LET %UF_DOTSPERMM$="12"  
LET %UF_LABELSIZE$="3.25x5.5"
```

# 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, then launch UnForm.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: `|UnForm -f rulefile | lp -dprinter -oraw -s 2>/dev/null`

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

Before: `/dev/lp0`

After: `>UnForm -f rulefile -o /dev/lp0`

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("unform.exe -i work.txt -o LPT1 -f rulefile",6)=0 then
    end
else
    msgbox "UnForm failed to start."
end if
```

## FILES USED BY UNFORM

UnForm uses several text files and executable files, each described below. Unless otherwise noted, all files are located in the UnForm directory.

<code>/usr/bin/UnForm</code>	This is a shell script executable that acts as a UNIX pipe. Various options can be passed to this procedure, all of which are described below under UnForm Options.
<code>unform.exe</code>	This is a file that starts UnForm with user specified arguments under Windows. This file is located in the Windows directory.
<code>unform.ini</code>	This file is used on Windows installations to store the startup parameters for the run-time language. It is located in the Windows directory.
<code>reader.ini</code>	This file stores information about Acrobat Reader for use by the <code>-winx</code> drivers.
<code>ufsetup</code>	This is a UNIX shell script that creates <code>/usr/bin/UnForm</code> with proper parameters and directory pointers.
<code>ufsetup.exe</code>	This is a Windows program that prompts for the run-time Business Basic location and creates <code>unform.ini</code> .
<code>unform.cnf</code>	This is a simple BBx configuration file used by UnForm.
<code>unform.tpl</code>	This is a template UNIX shell script used by <code>ufsetup</code> to create <code>/usr/bin/UnForm</code> .
<code>ufparam.txt</code>	This contains font and symbol set information. A special version of this file, “ <code>ufparam.txc</code> ”, can be defined and customized for a particular site and not be affected by future updates.
<code>*.bb, *.pv</code>	These are program modules, which are executed by the <code>/usr/bin/UnForm</code> shell script or <code>unform.bat</code> file. Files ending in “ <code>bb</code> ” are BBx4 or PRO/5 programs; and files ending in “ <code>pv</code> ” are ProvideX programs.
<code>subst</code>	This is the default parameter substitution file. An UnForm command line argument can be used to specify a different file. See the <b>text</b> keyword for information about text substitution.
<code>*.rul</code>	These are rule files, one of which may be specified on the UnForm command line in order for any enhancements to take place to the reports being piped through UnForm. If no rule file is specified, a default rule file is used instead. The default rule file is specified in the <code>ufsetup</code> program during installation. Rule files can be



named in any manner; the .rul extension is simply a convention. Rule files are described in detail below, under Rule Files.

bbx4, pro5, vpro5, rt\* These are BB<sup>x</sup>PROGRESSION/4, PRO/5, or Visual PRO/5 executables, used as the run-time for UnForm. If you have rt\* files in the UnForm directory, you are running a bundled copy of UnForm.

mailcall.\* Bundled email utilities and support files.

# DEMO AND LIVE INSTALLATIONS

When first executed, UnForm activates itself as a demo version. A demo version will operate for 30 days, following which it will no longer add enhancements to output. It will, however, continue to pass data through unchanged. While UnForm is in the demonstration mode, a trailer page indicating that UnForm is in demonstration mode will follow every enhanced print job.

To convert a demo version of UnForm to a live version, you must obtain an activation key from your dealer or the publisher, and that key must be used to activate UnForm. If you are running a bundled version, then you will need not only an UnForm key, but a run-time serial number and key as well. UnForm's activation keys are based on the run-time serial number under which UnForm executes.

If you are running a bundled version, you must first activate the run-time. Once you receive your serial number and keys, you can activate the run-time by running **./activate** from the UnForm directory. You should be logged in as root to perform this step.

Once you have activated the bundled run-time, or if you are using your own bbx4, pro5, or pvx run-time, you can activate UnForm with the “-act” option. On Windows, just run the UnForm Activation option from the Start menu.

Activation keys in Version 4.1 are machine category based. For example, a RISC UNIX version of UnForm requires a different activation key than an Intel UNIX version. The UnForm -v (on Windows, the UnForm Information Start menu option) will show the machine category.

Some versions shipped as part of another software package may not require activation.

# USER COUNTS

UnForm uses a BB<sup>X</sup>PROGRESSION/4, PRO/5, or ProvideX run-time user slot while processing data from a report. This is the case even if no enhancements are added to the data. The user slot is returned to the user count pool once the output processing is complete, but some reports can print for a long time, and will occupy that user slot until the report is complete.

Under Windows, with the Visual PRO/5 or ProvideX run-time, each workstation can have multiple instances of the run-time running, so this user restriction is less likely to be important.

Under UNIX, if UnForm cannot start the run-time because there are no user slots available, then it will retry several times until it is successful. If, after several retries, it is unable to start a BB<sup>x</sup> task, then the report will be sent through without any processing at the UNIX shell level. If this occurs, or there is another problem with starting a BB<sup>x</sup> task, a page will be printed after the report describing the BB<sup>x</sup> error that occurred. The script that performs the retries (/usr/bin/UnForm) can be modified to support more retries and/or longer retry pauses. To modify the script, adjust the time= and retry= lines.

Most sites have several available user slots available at any given time, so this does not become an issue, but if your site often uses all available slots, then you may need to purchase a license for more users to accommodate UnForm. A separate license may be purchased for PRO/5 or Visual PRO/5 from your dealer or the publisher.

To minimize the impact that UnForm has on your user counts, it is advisable to only install UnForm on printer definitions for which it may be used. If your site has many printers, but only a few are used to print forms, then just install UnForm on those few.

# UNFORM OPTIONS

UnForm can be started with one or more options, which control various aspects of how it works. Without any options specified, UnForm does nothing more than pass data through as is. These options are described below.

Option	Description
-300	This option 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	This option will cause UnForm to prompt for a new activation key. This is used when you change from a demonstration copy of UnForm to a live copy. The driver being activated can be specified with the "-p <i>driver</i> " option placed before -act in the command line.
-c <i>copies</i>	This option causes UnForm to issue multiple copies of the entire report. This differs from the -pc option. If <i>copies</i> 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.
-e <i>error-file</i>	This option causes UnForm to output any errors to the file specified.
-exec " <i>launch-program</i> "	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 " <i>launch-program</i> " 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 <i>launch-program</i> value. If <i>launch-program</i> contains any spaces, it must be quoted.

<i>-f rule-file</i>	<p>This option is used to establish 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.</p> <p>UnForm will always search for the rule file first in the UnForm directory, then by the full pathname given.</p>
<i>-gw</i>	<p>Forces UnForm to pass through PCL image width and height escape sequences to the printer. This is sometimes necessary on color (RTL) 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.</p>
<i>-h</i>	<p>This will cause Help text to be printed to your screen.</p>
<i>-i input-file</i>	<p>This option allows UnForm to process an already existing text file. If not specified, then standard input (stdin) is read. Under Visual PRO/5, standard input cannot be used, so an input file must be supplied.</p>
<i>-ix input-file</i>	<p>This option is the same as the <i>-I</i> option except the input text file is removed upon completion of task.</p>
<i>-macros</i>	<p>This option turns on macros.</p>
<i>-macrocopy n</i>	<p>This option is used in conjunction with the <i>-makemacro</i> option. A macro will be created for the designated rule set copy.</p>
<i>-makemacro n</i>	<p>This option causes UnForm to simply create the appropriate macro for the designated rule set and designate it as the number <i>n</i>. It must be used jointly with the <i>-r</i> option and can be used in conjunction with the <i>-macrocopy</i> option. See special section discussing macros later in this documentation.</p>
<i>-o output-file</i>	<p>This option allows UnForm to send output to a text file. If not specified, then standard output (stdout) is written. Under Visual PRO/5, an output file must be supplied. In addition to files or devices, the output can be a printer alias named in the config.unf file, or (on UNIX) can be a pipe prefixed by a " " character: <b>-o " cat &gt;&gt;/usr/archive/file.txt"</b>. Output names that contain spaces should be quoted.</p>

<p><i>-p output-format</i></p>	<p>This option allows you to specify the output format. The default format is <b>laser</b>. Other formats are:</p> <p><b>zebran</b>, which produces ZPL II output at <i>n</i> dots per mm (6, 8, or 12 – default of 12) for Zebra label printers</p> <p><b>html</b>, which generates Web pages from reports, based on a special set of rule set keywords.</p> <p><b>pdf</b>, which generates files viewable by Adobe Acrobat. Adobe provides a free Acrobat Reader program, and Windows versions of UnForm include the reader.</p> <p><b>win</b>, <b>winpvw</b>, which automatically produce a pdf file and launch the Acrobat Reader. <b>win</b> will automatically print the document, after issuing a print dialog. <b>winpvw</b> will provide a print preview by launching the Acrobat Reader.</p> <p>The win and winpvw drivers are the only drivers available for UnForm Lite licenses.</p> <p>For special Zebra media handling, you can append the following to <b>zebran</b>:</p> <ul style="list-style-type: none"> <li>• 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.</li> <li>• Set print modes (T=tear-off, R=rewind, P=peel-off, C=cutter).</li> </ul> <p>The default values are YT. For continuous labels, 8 dpmm, you would specify <b>-p zebran8NC</b>.</p>
<p><i>-page lines</i></p>	<p>Use this option to specify 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.</p>

-paper <i>paper</i> -ps <i>paper</i>	<p>Specifies the paper size used by the printer. Valid values are letter, legal, ledger, executive, a3, and a4. The default is letter.</p> <p>For Zebra printers, the <i>paper</i> setting is generally required, and is in the format <i>widthxheight</i>, where <i>width</i> and <i>height</i> are decimal numbers indicating height and width in inches of each label. <b>3.25x5.5</b>, for example, would define a label size of 3.25 inches by 5.5 inches. The default size is 4x6.</p>
-pc <i>copies</i>	This option causes UnForm to issue multiple copies of the report, page by page. If <i>copies</i> are 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.
-printblanks -pb	This option causes UnForm to process blank pages the same as non-blank pages.
-prm " <i>parameters</i> "	<p>This option 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 <i>parameters</i> is "<i>parameter-1=value-1[;parameter-2=value-2;...]</i>" Any number of parameters can be specified within the limits imposed by the operating system for command line length. Each <i>parameter</i> becomes a global string in Business Basic (STBL or GBL), and each is set to the <i>value</i> specified. Multiple parameters need to be delimited by semicolons (;). <b>-prm "company=01;name=Acme Paint"</b>, for example, would establish two global strings: company and name. These could be referenced within code blocks (prepage, precopy, etc.) as STBL("company") (GBL on ProvideX) and STBL("name").</p>
-r <i>rule-set</i>	This option can be used to set a form name rule set to use within the rule file specified. 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 simply passes all the text of the report through the pipe. If the <i>rule-set</i> contains spaces, it should be quoted. Rule set names are not case sensitive.

<code>-s <i>sub-file</i></code>	This option 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 <b>text</b> keyword for more information. The default substitution file is called "subst".
<code>-testpr <i>font symset</i></code>	<p>When executed with this option, UnForm will generate a test print showing nearly all characters (ASCII 1 to 253) in the <i>font</i> and <i>symset</i> codes identified. For a list of font codes and symbol sets, see the ufparam.txt file, sections [fonts] and [symsets], respectively.</p> <p>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 out. Note that with the pdf driver, the only symbol set used is 9J.</p>
<code>-v</code>	This option will cause UnForm to print version information and exit.
<code>-x [<i>page[,page, ...]</i>]</code>	<p>This option causes the first page of output to be printed with a cross hair pattern. This is typically done once to assist in determining placement of text, then removed. Sometimes, a special printer definition is setup 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.</p> <p>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. If the input doesn't contain form-feed page delimiters, be sure to use the -page option as well.</p>

## VERSION 4 FEATURES

Version 4 of UnForm adds a great deal of power and flexibility to the product. Here are some of the key enhancements:

### New pdf driver to create Adobe Acrobat® files



The most important new feature in Version 4 is the development the pdf driver, implemented with the “-p pdf” command line option. This optional driver allows UnForm users to integrate electronic documents into their processing, and provide access to those documents via web browsers, email, or Adobe Acrobat Readers on many computer platforms.

In addition to these pdf driver benefits, UnForm offers dynamic integration with Acrobat Readers on Windows platforms to offer print preview and support for any printer that operates in that environment. A special driver set, “-p win” and “-p winpvw” will automatically launch the Reader to print or preview the document. These drivers are available independently of other drivers in a special low-cost “lite” version of UnForm for Windows. They are also included, with barcode support, in the laser and pdf drivers.

Note the following differences between the laser driver and the pdf driver:

- No printer control commands are supported (duplex, bin, tray).
- Only symbol set 9J (WinAnsiEncoding) is supported. To get special symbols to match between laser and pdf documents, set the default symbol set to 9J in ufparam.txt, or use symset 9J in text commands that contain special symbols.
- Laser fonts are mapped to one of three pdf fonts: courier, helvetica, and times.
- Output device names cannot change during the job. Output name changes in the rule set are ignored, except for a non-copy-specific **output** command used if no “-o” command line option is specified.
- MICR fonts are not supported.
- Bar coding is supported, but not in the “lite” version.
- Underlined text is not supported.
- Copies are supported, but always print collated, as in pcopies.
- A new **title** command (title “*title text*”) provides a title for the reader’s document info page.
- Image and attachment files may be named with any extension in the rule set, but will be converted to a .pdf extension when the pdf driver executes. This allows you to maintain parallel images, one for laser and one for pdf, using the same base name, and the rule set can identify a single name to support both laser and pdf drivers. Another option for different pcl and pdf images is to use the new “if driver pdf ... end if” or “if driver laser ... end if” structures.

### **Text justification**

UnForm calculates font metric approximations for center and right justifications. To use justification, add left, center, right, or decimal options to font or text commands. For the text command, add “cols *n*” to provide a printable region width for calculation of center and right alignment. The default decimal character is a period (.). To override this, add “decimal=*x*”, where *x* is the decimal character printed by your application, to [defaults] section of ufparam.txt or ufparam.txc, or add “decimal *x*” to the rule set. The font metrics UnForm uses are based on Windows print drivers and some font substitutions, so they may result in approximate rather than exact justification. The new pdf driver uses calculations based on close font families present in the pcl driver.

### **Logical pages on a single physical page**

Two new commands, `across` and `down`, enable scaling and fitting of multiple logical pages per physical page. The page is only ejected after `across*down` pages of input have been printed. All UnForm generated attributes except barcodes are scaled automatically. External items added via `image` and `attach` commands are not scaled and are likely not suitable for this type of report enhancement. `Across` and `down` are sensitive to copies but may not be suitable for `pcopies`. It is therefore possible, for example, to print customer invoices on full pages, and archive copies 2-up in landscape orientation, or even 2x2 if you have a good magnifying glass.

You can also use `across` and `down` to re-align labels to match laser label stock. A new option of the page command allows UnForm to parse any number of "up" labels coming in, and they can be realigned for laser stock with a new `rows/columns/across/down` combination.

The scaling algorithm for proportional text, when `across` or `down` is used, is  $\text{newpoints} = \text{oldpoints} * 6.3 / \text{newlpi}$ . Therefore, if you are printing a page that would normally be full size on a full page, the text will be scaled down; if you are printing labels that would normally be printed at standard text sizes, the scaling will have little effect. The minimum point size generated is 4 points.

Note if you switch from portrait to landscape, `pcl` images should be changed for the correct orientation.

`across` and `down` are supported only by the laser and pdf drivers.

### **New codeblocks provide more flexibility**

Two new codeblocks, `predevice{ }` and `postdevice{ }` provide the flexibility to work with output files just before and after UnForm works with them. Use this to interface with external applications, even where pipes are not available.

The `predevice` code block is executed just after a device is opened and initialized. Use it to gather information from the page that initiated the device change. The `postdevice` code block is executed just after a device is closed. Use it to execute tasks against the previous output file.

Note that the just the laser driver supports these code blocks.

### **Page command enhanced to support multi-up labels**

The page command can now take two numeric parameters. A single number, such as page *n*, still indicates a page is implied each *n* lines. `page 30,6` indicates a page is each 30 columns of each 6 lines. If lines are 90 characters wide, each set of 6 lines would represent 3 pages. This works for laser, pdf, and zebra rule sets, but not HTML.

### **Image scaling on pdf output**

The image command now accepts either two or four numbers. If four numbers are supplied, the third and fourth are used as the image size in columns and rows. `'image 1.5,1,20,12,'logo.pdf'` will scale the image to 20 columns and 12 rows, regardless of the original size. The `cols` and `rows` parameters are ignored on `pcl` and zebra images.

### **Relative positioning enhanced**

All relative positioning features, if the text ends with the string *@leftcol,toprow,rightcol,bottomrow*, then the search only works in the region identified. *rightcol,bottomrow* default to right edge and bottom of page if not present. Use \@ to specify a literal @ sign in the search string. Any phrase found to begin in the region is selected, even if it extends beyond the region.

### **Miscellaneous enhancements**

UnForm has relaxed the rules on output changes - now copy 1 can be changed. Results may be unpredictable if copy 1 is going to standard out (rather than the -o specification) and the device is changed, then changed back to null to return to standard out. This doesn't work for pdf, where only a single output file for the whole job is supported.

UnForm now allows rule files to be outside of the UnForm directory. This allows network clients running off a server to specify a local rule file. UnForm will attempt to open the named file in the UnForm directory first, then if it fails it will try the file as named.

The move command accepts an optional “retain” argument following the position parameters. This will copy the text to a new location without erasing the original text.

We have written a ‘Programming Fundamentals’ chapter for the manual, to aid programmers who are not experienced in Business Basic to write custom code blocks.

# UNFORM 4.1 FEATURES

UnForm 4.1 adds several important enhancements to UnForm 4, including:

- UnForm licensing is no longer limited to individual drivers. Instead, all drivers are included with a single activation key. In addition, the activation key is stored in a file rather than being stamped into a program module, so upgrades and patches to UnForm won't require re-activation as in prior releases.
- Both text and barcode commands now support relative positioning, like many other commands do. The first argument to text and PCL barcode commands can now be a literal or regular expression to search for, making it easy, for example, to barcode a whole column of numbers with a single barcode command.
- A special syntax of the text command will print the human readable barcode value or checksum value derived from the barcode command. This can be useful if you are using barcode symbologies which barcode a different value than is supplied, such as the zero-suppressed UPC-E symbology.
- A custom PCL barcode symbology, 126, has been developed that supports UPC-E 6-digit number input and produces a 1-series UPC-E code.
- Our new utility product, MailCall, is included for your use with UnForm, so if your system has access to a SMTP email server, UnForm output can be emailed from within the UnForm job.
- The cross hair pattern print has been enhanced to support selected pages of input and also to recognize the `-page n` command line option when form-feeds are not present in the input.
- A new set of variables is now recognized when set in code blocks, providing control over the printer and page from within the code block. See the `precopy{ }` command reference for more details.
- PDF support has been enhanced to support the creation of an outline (bookmark) structure in the PDF output file.

# 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 several 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 keywords 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).

Keywords 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.

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.

# ACROSS

## Syntax:

across *n*

## 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.

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 2
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.

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)

## BARCODE (PCL,PDF)

### Syntax:

1. barcode *col,row,"value" | {expr},symbolology,height,spc-pixels*
2. barcode “*searchtext/~regexpr*”, *col, row, "", symbolology, height, spc-pixels, getoffset cols, getcols cols, eraseoffset cols, erasecols cols*

### Description:

*col* and *row* determine upper left corner of the barcode.

*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.

*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 to allocate 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 *searchtext* 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*.

Drivers: laser, pdf (not available on the Lite version)

### 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:”,0,0,””,600,75,2, getoffset 5, getcols 10, erasecols 15** will search for data starting with “bcd:”, barcode the 10 characters following it, and erase the underlying text.

## BARCODE (ZEBRA)

### Syntax:

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

*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 number 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 to 3 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* sets 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* sets 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 one 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.

Driver: laser

## 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.

Drivers: all

## BOLD, ITALIC, LIGHT, UNDERLINE

## CBOLD, CITALIC, CLIGHT, CUNDERLINE

### Syntax

1. { bold | italic | light | underline } *column, row, columns, rows*
2. { bold | italic | light | underline } "*text*", *column, row, columns, rows*
3. { bold | italic | light | underline } "~*reg-exp*", *column, row, columns, rows*

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 *column, row, columns, 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 format 2 or 3 is used, then the region is defined relative to any occurrence of the *text*, or of text that matches the regular expression *reg-exp*. In these cases, there may be no affected regions, or several. *Column* and *row* are 0-based, in these formats. The search for *text* or *reg-exp* 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 *reg-exp*, it is necessary to specify "\@".

### 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 option.

# BOX, CBOX

## Syntax

1. box *column*, *row*, *columns*, *rows* [,*thickness*] [,*shade*] [,*color*] [,rgb *rrggbb*] [,dbl|double [*gap*]] [,left *l*] [,right *r*] [,top *t*] [,bottom *b*]
2. box "*text*", *column*, *row*, *columns*, *rows* [,*thickness*] [,*shade*] [,*color*] [,rgb *rrggbb*] [,dbl|double [*gap*]] [,left *l*] [,right *r*] [,top *t*] [,bottom *b*]
3. box "~*reg-exp*", *column*, *row*, *columns*, *rows* [,*thickness*] [,*shade*] [,*color*] [,rgb *rrggbb*] [,dbl|double [*gap*]] [,left *l*] [,right *r*] [,top *t*] [,bottom *b*]

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 will be drawn of the indicated dimensions. All dimensions can be specified to 2 decimal places, in the range of -255 to +255. Whole number *column* 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 *columns* or *rows* to 1. If both *columns* and *rows* are 1, then a vertical line is drawn one character high. To draw a box that is one column wide or one row deep, use 1.01 or .99.

If format 2 or 3 is used, then the box is drawn relative to any occurrence of the *text*, or of text that matches the regular expression *reg-exp*. 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 *reg-exp* 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 *reg-exp*, it is necessary to specify "\@".

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

The optional *shade* parameter may be used to specify a "percent gray" value of from one to 100. Most laser printers can only print about eight different shades of gray, so a value of 45, for example, may print the same pattern as 50.

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).



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.

For partial shading, partial color shading, or multiple color shading, see **shade** keyword,

### 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 vertical line, 55 lines high, at column 1, line 1.

**box "Customer Total",-1,-1,60,3** will draw a box around the text "Customer Total" and 44 columns beyond it.

Drivers: all

# 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, 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 that follow 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: Case does make a difference. “COLPOS” and “colpos” are different constants. Take care not to use constant names that may inadvertently cause replacements elsewhere than intended.

NOTE 2: Beware of constant names that are contained in other constant names. The longer name should be present in the rule file first. Otherwise, the shorter name will find and replace a portion of the longer name, resulting in unpredictable behavior. For example, if `const FONT=cgtimes,8` is followed by `const FONTB=cgtimes,8,bold`, any FONTB will become cgtimes,8B.

Drivers: all

# 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

# 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.

Driver: laser, pdf, zebra

# DETECT

## Syntax

1. detect *column,row,"text"*
2. detect *column,row,"~reg-exp"*

## 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 form must match all of them.

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

If format 2 is used, then detect is implemented to match the regular expression specified in *reg-exp*.

## Examples

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

**detect 10,4,"~.././.."** would match a date format at column 10, row 4.

Drivers: all

## DOWN

### Syntax:

down *n*

### 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.

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



## 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<sup>th</sup> 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.

Drivers: laser

# ERASE, CERASE

## Syntax

1. erase *column, row, columns, rows*
2. erase "*text*", *column, row, columns, rows*
3. erase "~*reg-exp*", *column, row, columns, rows*

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 format 2 or 3 is used, then the region is defined relative to any occurrence of the *text*, or of text that matches the regular expression *reg-exp*. In these cases, there may be no bolded regions, or several. *Column* and *row* are 0-based, in these formats. The search for *text* or *reg-exp* 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 *reg-exp*, it is necessary to specify "\\@".

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.

Driver: 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 fixed font 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

# FONT, CFONT

## Syntax

1. font *column, row, columns, rows* [*fontname*] [*font fontcode*] [*symset symset*] [*size*] [*bold*] [*italic*] [*underline*] [*light*] [*shade percent*] [*fixed | proportional*] [*color*] [*rgb rrggbb*] [*justification*]
2. font “*text*”, *column, row, columns, rows* [*fontname*] [*font fontcode*] [*symset symset*] [*size*] [*bold*] [*italic*] [*underline*] [*light*] [*shade percent*] [*fixed | proportional*] [*color*] [*rgb rrggbb*] [*justification*]
3. font “~*reg-exp*”, *column, row, columns, rows* [*fontname*] [*font fontcode*] [*symset symset*] [*size*] [*bold*] [*italic*] [*underline*] [*light*] [*shade percent*] [*fixed | proportional*] [*color*] [*rgb rrggbb*] [*justification*]

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 will apply font control to all text in the defined region of column, row, columns, and rows. The other parameters are all optional.

If format 2 or 3 is used, then font attributes are applied relative to the occurrence of *text* or the regular expression *reg-exp*. 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 *reg-exp* 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 *reg-exp*, it is necessary to specify “\@”.

*fontname* can be Courier (the default), cgtimes, or univers. These fonts are standard on virtually all PCL5 compatible printers. Alternately, its font number 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) may be specified, but proper use depends on the form 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* can also be specified from the “ufparam.txt” file.

*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. *symset* can also be a name from the “ufparam.txt” file.

*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.

**bold**, **italic**, **underline**, and **light** will apply the indicated attribute(s) to the text.

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

Specify fixed or proportional to override the default of fixed for Courier (or any *fontcode* below 4100), and proportional for all else.

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).

*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.

Note: 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 mis-aligned 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.

Drivers: all, but note the following:

pdf: maps pcl font names and numbers to courier, helvetica, or times. Symbol set 9J is the default and the only symbol set supported.

zebra: symset is 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). Fontnames are not mapped. Specify as font codes, which must be internal font identifiers, such as a-f, 0-9. See the ZPL documentation for font codes.

# 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 end-points to be at the center position of a character, which may impact how lines intersect. Lines are drawn one dot (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 “\@”.

## Examples

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

Drivers: all

## IF COPY ... END IF

### Syntax

```
if copy n,n,...  
...  
end if
```

### Description

The keyword "**if copy**" will cause any keywords 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 three 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 *column,row* [,*cols, rows*], {"*file*" | {*expr*}}

## 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.

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.

**Note:** The optional *cols* and *rows* parameters cause the image to be scaled to the rectangular region specified, and are only used by the pdf driver.

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.

An easy 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.

Another alternative is to use the publisher's web site image conversion utility, available from the UnForm page on <http://synergetic-data.com>. You can upload an image file and receive back images in PCL, RTL, or PDF format.

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 Image Alchemy. 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 (old) pdf file format, which the optimization does not produce.

## Examples

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

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

Drivers: all. Laser requires pcl raster format, pdf driver requires pdf format, zebra requires zpl format.

## ITALIC

See the **bold** keyword.

# LANDSCAPE

## Syntax

landscape

## Description

This keyword will ensure that UnForm produces output in landscape (horizontal) orientation. The default orientation is portrait, 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.

Also see the **portrait** keyword.

Drivers: laser, pdf

## 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. See section on creating macros later in this documentation.

Drivers: laser



# 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

# 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 four 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

# MICR

## Syntax

*micr col,row,account,check*

## Description

Prints MICR font for laser check printing. The account number is separated from the check number for performance reasons - the account number image is generated once per job, while the check number image is generated for each page. 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 by 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 become "transit symbols", and quotes become "on us" symbols. Account numbers can contain spaces or dashes, as well as 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 require a different format for the MICR.

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.

## Example

*micr 6,42.25,":123456789:9999-1234<34>",{trim(get(65,5,6))}*

Drivers: laser

# MOVE, CMOVE

## Syntax

1. `move col,row,cols,rows,newcol,newrow [,retain]`
2. `move "text",col,row,cols,rows,movecols,moverows [,retain]`
3. `move "~reg-exp",col,row,cols,rows,movecols,moverows [,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. Format 1 moves the region indicated by *col*, *row*, *cols*, and *rows* so the new upper left point is at *newcol*, *newrow*. Formats 2 and 3 search for occurrences of *text* or the regular expression *reg-exp*, respectively, and use 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 *reg-exp* 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 *reg-exp*, it is necessary to specify “\@”.

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

## 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/uniform/attachments/attach1.pcl”
    notext
end if
```

Drivers: all

# OUTLINE

## Syntax

outline

## 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.

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 (starting with |), a filename, or a printer alias defined in the config.unf file.

If the second format is used, *expression* is 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 a LET assignment and produce a string data type result, which must be a valid output device as noted above. See the **precopy** keyword for more information about programming expressions.

## 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, used if no -o command line option is specified. pdf output cannot be changed during printing.

# PAGE

## Syntax

1. page *rows*
2. page *cols, rows*

## Description

Format 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 format 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** will consider each 42 lines to be a full page.

**page 42**

**rows 66** would mean 42 lines input and 66 lines output.

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.

For PCL (Laserjet) printers, *size* can be any of the following:

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 zebran” 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**.

Drivers: all

# PORTRAIT

## Syntax

portrait

## 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.

See also the **landscape** keyword.

Drivers: laser, pdf

## 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 (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.
- **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 codeblock. Whenever a new device is opened for any given copy, this code block is executed. The programmer could then store information from the page that caused the device to be opened, such as a customer code and 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. The code is inserted into the module `ufmain.xx` at run-time. Note that BB<sup>X</sup>PROGRESSION/4 run-time environments are limited to 64K program sizes, so the amount of code added needs to be limited to 64K less the base size of `ufmain.bb`, if you are running UnForm under a BB<sup>X</sup>PROGRESSION/4 run-time.

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$[2]` is the second line of the page.
- **copy** contains the current copy number. Generally you shouldn't modify this value. If you need to skip printing of a copy, use the **skip** variable instead.
- **copies** is the number of copies. You can change this value to dynamically adjust the number of copies. If the number you specify is higher than the number specified by the rule set, then that highest defined copy's text and enhancements will be repeated until your specified copies are complete. This value is reset each page, so you can't set it in the prejob routine.
- **skip** may be set to a non-zero value in order to skip printing of any copy. Use this in a precopy routine.
- **output\$** can be changed for any copy other than 1. This is the device that the copy should print to. If it changes from one page to another, UnForm will close the prior output channel and reopen the new one. This can be used to send a copy to a different printer, or to a fax device. You can set the value to any printer alias known to UnForm (in the `uniform.cnf` file), any file, or a pipe, such as `"|fx - n "+faxnum$`
- **bin\$, tray\$, duplex\$, paper\$, cols\$, rows\$, and margin\$** can be set to values described in the `bin`, `tray`, `duplex`, `paper`, `cols`, `rows`, and `margin` commands. Note that as string variables, they must all be set to string values, but may be interpreted numerically. For example, to adjust the number of columns for a given page to 132, set `cols$="132"`.
- **orientation\$** can be set to "landscape" or "portrait".
- **outline\$** can be set to an outline string used when the PDF outline feature is turned on, by use of the `outline` command. Multiple levels of outlines can be defined by delimiting levels with vertical bars, such as `outline$="Customer type "+get(1,6,4)+"|Page "+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.
- **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.
- **trim(*expression*)** trims spaces from the left and right side of a text expression.
- **upper(*expression*)** converts text to UPPERCASE.
- **lower(*expression*)** converts text to lowercase.
- **proper(*expression*)** converts text to Proper Case.
- **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.
- **exec(*expression*)** may be used to execute **text**, **box**, **shade**, **dump**, and **barcode** keywords 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.

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.

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

## 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 cvs(get(10,2,30),3)="Acme Systems" comp$="01" else comp$="02"
dim cust$:fatrr(cust$)
readrecord (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$+cust.name$+"$ else skip=1
}
```

```
postjob {
if log$="" goto endjob
log=unt; open(log)"P1"
print (log)"Fax Verification Log"
print (log)log$,'ff',
close(log)
endjob:
}
```

Drivers: all

# 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. If no **page** keyword is present then UnForm will assume 66 input rows. If a document is created without formfeeds, then the **page** keyword must be used.

## Examples

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

Drivers: all

# SHADE, CSHADE

## Syntax

1. shade *column, row, columns, rows, percent* [,extend] [,color] [,rgb rrggbb]
2. shade *column, row, columns, rows, percent, skip, times* [,extend] [,color] [,rgb rrggbb]
3. shade "*text*", *column, row, columns, rows, percent* [,extend] [,color] [,rgb rrggbb]
4. shade "~*reg-exp*", *column, row, columns, rows, percent* [,extend] [,color] [,rgb rrggbb]

If cshade is used, then *columns* 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 *column, row, columns, 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%. PCL5 printers actually support only eight levels of gray, generally: 2%, 10%, 20%, 35%, 55%, 80%, 99%, and 100%. Values less than those are rounded up.

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.

Format 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 a number indicating 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 formats 3 and 4.

If format 3 or 4 is used, then the shading is drawn relative to any occurrence of the *text*, or of text that matches the regular expression *reg-exp*. 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 *reg-exp* 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 *reg-exp*, it is necessary to specify "\@".

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).

### 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 column 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.

## Examples

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

Drivers: all

# SYMSET

## Syntax

`symset symbolset`

## Description

The **symset** keyword overrides the default symbol set setting found in the [default] 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 digit(s) followed by an uppercase letter.

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 range.

Drivers: laser

# TEXT

## Syntax

1. text *column*, *row*, { "*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*]

2. text "*searchtext/~regexpr*", *column*, *row*, { "*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*], eraseoffset *cols*, erasecols *cols*

## Description

The *text* indicated in quotes will be printed at the column and row indicated by *column* 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 *text* is not quoted, but instead 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* is not quoted, but instead 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 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 a LET assignment and produce 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 **precopy** keyword for more information about programming expressions.

*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.

*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.

*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 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 are the opposite.

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

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

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

Specify fixed or proportional (or “prop”) to override the default of fixed for Courier (or any *fontcode* below 4100), and proportional for all else.

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>1997 Synergetic Data Systems Inc.”

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).

*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* with the cols option, so that UnForm can determine the right edge of the justification region.

If syntax 2 is used, then UnForm will search for occurrences of *searchtext* or the regular expression *regexpr*. In this case, *col* and *row* become 0-based offsets to the location of where matches are found. In addition, the erasecols *cols* and eraseoffset *cols* can be used to remove match text.

**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: “bcdsss|value” to print the human readable barcode value for symbology *sss* and barcode text *value*, “ck1sss|value” to print check digit 1, or “ck2sss|value” to print check digit 2. See the barcode command for symbology values.

## 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)}, cgimes, 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:",cgimes,12,bold,eraseoffset 0, erasecols 6** changes words TOTAL: to Total: in cgimes, 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.

Drivers: all. pdf driver fonts map to Courier, Helvetica, or Times-Roman, and support only symbol set 9J. 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.

# 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.

Drivers: laser

## 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 (1/300 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 specified, the lines are extended ½ character 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 “\@”.

## Examples

**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.

## Examples

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

Drivers: all

# CONCEPTS, PRIMER, AND TIPS

## How to Create a Rule Set

UnForm is a very powerful tool, with dozens of commands and features. It can be difficult to grasp the basics from such a large base, but the basics are really very simple. Once UnForm is installed by an administrator, the only skills required to develop typical business forms is 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, or on UNIX can go to UnForm'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.
- Each job has its own *geometry*, that is, the basic columns and rows that UnForm scales everything to. If you specify **cols 85**, then UnForm will scale each character and all the enhancement positions and sizes to 1/85<sup>th</sup> 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, bold, italic, and underline commands. You can also manipulate that text 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.

- 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 just starts with a line with the phrase “if copy *n*”, where *n* is the copy number, followed by any number of lines of enhancement commands, followed by a line with the phrase **end if**.

## Basic Ruleset Creation Steps

- Obtain sample output from your application for the form you want to enhance. This output can be printed to a text file, or you could 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: `UnForm -x -i input-file -o output-device` or `UnForm -x -i input-file | lp -dxxx`.
- 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: `UnForm -i input-file -f rule-file -o output-device` or `UnForm -i input-file -f rule-file | lp -dxxx`.
- 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 patterns, or use 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, \* 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 get an idea of what complete rule sets look like is the sample forms provided with UnForm, 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. The sample rule set is called *sample.rul*, and is found in the UnForm directory.

## Tips and Techniques

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

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

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

Set the environment variable “UFC” to *y* before starting your application. All UnForm printing will then generate a cross-hair pattern.

If your report doesn’t contain formfeed characters at the end of the page, then you should print just one page worth of data. 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. If user counts are a problem, then it may help to set up a single printer as the UnForm printer, and print selected reports through that.
- 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).
- To modify the font of text from the application, use the **font** command. 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.
- Text, such as headings or messages, can be added 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.
- Shading and box drawing can be added 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.
- Logos and other images are added with the **image** command. With this command, UnForm looks specifically for PCL raster images (or PDF images if the pdf driver is used) in the file. To produce a PCL raster image, use a Windows workstation to print your picture to a file, using a HP Laserjet III or higher print driver. PDF images can be produced using Adobe Acrobat® Distiller or Image Alchemy.
- To add overlays or attachments, use the **attach** command. 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 a separate copy: use the **copies** 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.

- If the application doesn't use form-feeds at the end of each page, then the **page** keyword must be used 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 Fundamentals chapter for more information.



# 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 timesavings 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.

In Version 3, macro support was introduced. To access permanent macros, simply add **macro *n*** (*n*=macro #) to the rule set. To cause UnForm to utilize temporary macros, simply 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 timesavings 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 timesavings, which is sometimes offset by the extra overhead incurred by UnForm to manage the macros in memory.

In Version 4, the macro functionality has been enhanced to allow you to use UnForm to generate permanent macros. Permanent macros can be downloaded once 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, nor 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:

```
UnForm -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 SETS

UnForm is supplied with three sample reports and associated rule sets. A description of each report and rule set follows. Each of the sample reports is in the UnForm directory, named “*samplen.txt*.” All rule sets can be found in the file “*sample.rul*” in the UnForm directory.

To produce these samples on your own laser printer, you can use the following command, substituting the proper sample text file:

```
UnForm -i sample-file -f sample.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:

```
UnForm -i sample1.txt -f sample.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.

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.

## INVOICE - INVOICE FOR PRE-PRINTED FORM

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.

UnForm -i sample1.txt -f sample.rul -o output-device

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

[Invoice]

*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 two space, 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,"~../../.. ....."
detect 9,11,"~....."
detect 10,21,"~../../.. ....."
```

*Set the printer to 600 dpi and 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. In addition, UnForm will generate two copies of the job: with all pages of copy 1, then all page of copy 2 (non-collated.)*

```
dpi 600
cols 80
rows 66
copies 2
```

*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. In addition, if the pdf driver is used, and no -o command line option specifies an output file, then UnForm will generate the file "invoice.pdf".*

```
title "Invoice Sample"
```

```
if driver pdf
output "invoice.pdf"
end if
```

*This set of commands places a black rectangle at column 61, row 6.8, and then places the phrase “Customer Copy” on copy 1, and “Remittance Copy” on copy 2. The text phrases are shaded at 0%, so they print white, on the black background.*

```
shade 62,6.8,18,1.3,100
if copy 1
text 62,7,"Customer Copy",shade 0,univers,13,center,cols 18
end if
if copy 2
text 62,7,"Remittance Copy",shade 0,univers,13,center,cols 18
end if
```

*Add the document title in 18 point, bold, cgtimes font. Set it at column 60, but right justify it in a 20 column region.*

```
text 60,1.7,"INVOICE",bold,18,cgtimes,cols 20,right
```

*Add the a logo and company information to the upper left of the form. If the driver is pdf, UnForm will look for an image file “sdsilogosm.pdf” automatically. Optionally, you could specify different image names within ‘if driver laser ... end if’ and ‘if driver pdf ... end if’ blocks.*

*The constant APOS is used to make it easy to adjust the text positioning should the logo size be modified later. Note that the 3<sup>rd</sup> and 4<sup>th</sup> arguments (12,6.5) of the image command are ignored in the laser driver. UnForm will only scale images in the pdf driver.*

```
image 1,1,12,6.5,"sdsilogosm.pcl"
const APOS=16.5
text APOS,3.0,"Synergetic Data Systems Inc.",univers,14,bold
text APOS,4.2,"907 Embarcadero Drive, Suite 4",univers
text APOS,5.2,"El Dorado Hills, CA 95762",univers
text APOS,6.2,"(916)-933-0120, Fax: 933-5793",univers
```

*The prepage subroutine 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 it finds it, 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.*

*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.*

```
prepage{
# find zip code in city,state,zip line
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$=get(x,15,5)
}
barcode 8,16,{shipzip$},900,60,2
```

*The next group of commands create headings and outlines with text, box, and shade commands. In addition, some of the text printed to UnForm is erased or bolded.*

```
text 61,4,"Date",cgtimes,italic
text 71,4,"Invoice",cgtimes,italic
text 79,4,"Pg",cgtimes,italic
box 60.5,3.5,20,2
shade 61,4,20,1,20

text 15,10,"SOLD TO",cgtimes,italic
text 59,10,"SHIP TO",cgtimes,italic
erase 9,11,6,1
erase 54,10,40,2
shade 2,9.9,34,1.1,20
shade 47,9.9,33,1.1,20
bold 15,10,7,1
bold 59,10,7,1

const L1=19
const L2=20
const BOXPOS=18.5
text 1,L1,"Order",cgtimes,italic
text 1,L2,"Number",cgtimes,italic
text 10,L1,"Order",cgtimes,italic
text 10,L2,"Date",cgtimes,italic
text 19,L1,"Cust.",cgtimes,italic
text 19,L2,"Number",cgtimes,italic
text 26,L1,"Sls",cgtimes,italic
text 26,L2,"Prs",cgtimes,italic
text 30,L1,"Purchase",cgtimes,italic
text 30,L2,"Order No.",cgtimes,italic
text 44,L2,"Ship Via",cgtimes,italic
text 57,L1,"Ship",cgtimes,italic
text 57,L2,"Date",cgtimes,italic
```

```

text 66,L2,"Terms",cgtimes,italic
shade 1,L1,80,2,20
box .5,BOXPOS,8.5,2
box 9,BOXPOS,9,2
box 18,BOXPOS,7,2
box 25,BOXPOS,4,2
box 29,BOXPOS,13,2
box 42,BOXPOS,14,2
box 56,BOXPOS,9,2
box 65,BOXPOS,15.5,2

const L1=23
const L2=24
const BOXPOS=22.5
text 1,L1,"Qty",cgtimes,italic
text 1,L2,"Ord",cgtimes,italic
text 6,L1,"Qty",cgtimes,italic
text 6,L2,"Ship",cgtimes,italic
text 11,L1,"Qty",cgtimes,italic
text 11,L2,"Bkord",cgtimes,italic
text 20,L2,"Item & Description",cgtimes,italic
text 52,L2,"U/M",cgtimes,italic
text 63,L1,"Unit",cgtimes,italic
text 62,L2,"Price",cgtimes,italic
text 72,L1,"Extended",cgtimes,italic
text 75,L2,"Price",cgtimes,italic
shade 1,L1,80,2,20
box .5,BOXPOS,4.5,2
box 5,BOXPOS,5,2
box 10,BOXPOS,6,2
box 16,BOXPOS,35,2
box 51,BOXPOS,4,2
box 55,BOXPOS,12,2
box 67,BOXPOS,13.5,2

```

*The following lines use UnForm's ability to locate text patterns on the page and anchor enhancements accordingly. In this sample, we use the pattern of a decimal point followed by 4 digits, which appears on each invoice line item. From each of those patterns, font commands are used to change the font and justification of the item line columns.*

*The font changes are followed by a shade command that looks for the same pattern, then draws a 5% shade rectangle over the line and its description.*

```

font "~\.[0-9][0-9][0-9][0-9]",-61,0,4,1,cgtimes,left
font "~\.[0-9][0-9][0-9][0-9]",-55,0,4,1,cgtimes,left
font "~\.[0-9][0-9][0-9][0-9]",-50,0,4,1,cgtimes,left
font "~\.[0-9][0-9][0-9][0-9]",-42,0,30,2,cgtimes,left
font "~\.[0-9][0-9][0-9][0-9]",-10,0,3,1,cgtimes,center

```



```
font "~\.[0-9][0-9][0-9][0-9]",-6,0,12,1,cgtimes,decimal
font "~\.[0-9][0-9][0-9][0-9]",7,0,12,1,cgtimes,decimal

# This looks for item detail lines and shades them
shade "~\.[0-9][0-9][0-9][0-9] ",-61,0,80,2,5
```

*Between the item detail lines may be comments, starting at column 28 and occurring between lines 25 and 56. The font command will look for 27 spaces followed by any number of characters, only searching in the region from column 1, line 25 through column 80, row 56.*

```
font "~^ (27 spaces) [A-Za-z0-9]+@1,25,80,56",27,0,53,1,cgtimes
```

*The form footer is drawn with some box commands, text commands, and font commands. Note again, the font command applies to the text input to UnForm. The <153> in one of the text command represents an ASCII 153 character, which in the symbol set 9J prints as a TM trademark symbol. The 9J symbol set shares characters with the pdf and laser drivers. The last text command uses an expression (indicated by the surrounding {}) characters. The expression concatenates a literal string "Your customer code is " with the result of a get function, which will retrieve text input from the page at column 9, row 11, for 6 characters, and finally appends a period.*

```
box 57,57,24,2,2,20
box 57,59,24,4,2,20
box 57,63,24,2,2,20

text 58,58,"Sales Amt",italic,cgtimes
text 58,61,"Sales Tax",italic,cgtimes
text 58,62,"Freight",italic,cgtimes
text 58,64.25,"TOTAL",italic,bold,cgtimes,14
font 58,60,10,1,italic,cgtimes
font 70,58,11,7,cgtimes,decimal,bold

text 1,64,"This sample was generated by UnForm<153> Version
3.",7,cgtimes,symset 9J,blue
text 1,65,{"Your customer code is "+get(9,11,6)+"."},7,cgtimes
```

## STATEMENT - PLAIN PAPER FORM, TWO PAGE FORMATS IN SAME JOB

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 statement 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.

```
UnForm -i sample2.txt -f sample.rul -o output-device
```

*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.*

```
detect 34,2,"STATEMENT"  
detect 65,7,"~../../.."
```

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 2 is used to tell UnForm to print two copies of each page, with copy 1 and 2 following each other in sequence for each (collated.) You will find later that UnForm doesn't actually print both copies of each page, but instead simply prints one copy or the other, depending on the format required.*

```
page 66  
cols 75  
# Copy 1 Statement with aging totals  
# Copy 2 Statement w/o aging totals  
pcopies=2
```

*This prepage code block gathers company name and address information in order to place it in new positions later. In addition, line 64 in the source text contains “!” characters to represent vertical lines. This code block will erase those characters. Elsewhere in the rule set, the vline command will be used to remove longer sequences of “!” characters.*

```

prepage {
  compname$=cvs(get(24,4,30),3)
  addr1$=cvs(get(24,5,30),3)
  addr2$=cvs(get(24,6,30),3)
  pageno$=cvs(get(71,5,4),3)
  date$=cvs(get(65,7,10),3)
# remove single ! from line draw regions
  x=1
  while x>0
    x=pos("!"=text$[64]); if x>0 text$[64](x,1)=" "
  wend
}

```

*This precopy block is critical, as it determines whether a given copy should print based on the content of each page. Copy 1 has been designated as the format to print on the final page of a customer's statement. This copy contains the aging data, and will not contain the "CONTINUED". Copy 2 is the format for the initial pages that contain "CONTINUED". The logic simply looks for the word. If UnForm is about to print copy 1, and the page contains the word CONTINUED, then the code sets the skip variable, and UnForm will skip the printing of this copy. If copy 2 is about to be printed, but it is NOT a copy to be continued, then UnForm will skip copy 2. As each page is read from the input, UnForm will therefore print one or the other copy.*

```

precopy {
  if copy=1 then if get(66,64,9)="CONTINUED" then skip=1
  if copy=2 then if get(66,64,9)<>"CONTINUED" then skip=1
}

```

*The balance of the rule set simply draws the form around the data, using many of the existing headings already printed. The hline and vline commands are used to erase the existing character outlines. The if copy blocks are used to define the two formats as required. Any commands outside of if copy blocks apply to both copies.*

```

# constants
const hdrattr=cgtimes,11,bold
const COMPPOS=3

# major boxes and lines
cbox 1,1,75,66
cbox 1,9,75,61
if copy 1,2
  cbox 1,15,75,17,1,10
endif
if copy 1
  cbox 1,17,75,56
  cbox 10,15,18,56
  cbox 27,15,39,56

```

```

        cbox 48,15,60,56
        cbox 63,15,63,56
endif
if copy 2
        cbox 10,15,18,61
        cbox 27,15,39,61
        cbox 48,15,60,61
        cbox 63,15,63,61
endif
if copy 1
        cbox 1,61,75,63,1,10
endif

#remove existing lines
hline "--",erase
hline "==",erase
vline "!!",erase
erase 24,4,30,3
erase 1,16,75,1

# header section
erase 34,2,9,1
text 33,2.5,"STATEMENT",cgtimes,18,bold
text COMPPPOS,3,"Please send payments to:",cgtimes,10,italic
text COMPPPOS,4.25,{compname$},cgtimes,bold
text COMPPPOS,5.35,{addr1$},cgtimes,bold
text COMPPPOS,6.45,{addr2$},cgtimes,bold
cerase 59,5,75,8
text 64,5,"Page",hdrattr
text 64,7,"Date",hdrattr
text 69,5,{pageno$},hdrattr
text 69,7,{date$},hdrattr
cbox 63,1,68,9,1,10
cbox 63,1,75,9
cbox 63,6,75,6

# customer section
erase 2,10,3,1
text 2,10,"To",hdrattr
font 12,11,30,4,cgtimes,bold
cbox 1,9,10,15,1,10

# detail header section
text 4.5,16,"Date",hdrattr
text 12.5,16,"Invoice",hdrattr
text 20,16,"Due Date",hdrattr
text 34,16,"Amount",hdrattr
text 41,16,"Pmt Date",hdrattr
text 52,16,"Pmt Amount",hdrattr
text 61,16,"TP",hdrattr

```

```

text 70,16,"Balance",hdrattr

# remarks section
if copy 1
    font 2,57,70,4,cgtimes
endif

# balance section
if copy 1
    box 14,61,1,5
    box 26,61,1,5
    box 38,61,1,5
    box 50,61,1,5
    box 62,61,1,5
    erase 2,62,74,1
    text 9,62,"Current",hdrattr
    text 19.75,62,"1-30 Days",hdrattr
    text 31,62,"31-60 Days",hdrattr
    text 43,62,"61-90 Days",hdrattr
    text 57,62,"Over 90",hdrattr
    text 68.5,62,"Total Due",hdrattr
endif
if copy 2
    erase 1,62,75,66
    text 65,65,"Continued . . .",hdrattr
endif

```

## AGINGREPORT - ENHANCED AGING REPORT

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.

```
UnForm -i sample3.txt -f sample.rul -o output-device
```

*This statement header identifies this rule set.*

```
[AgingReport]
```

*The only detect statement require is this one, looking for the report title at column 50, row 2.*

```
detect 50,2,"Detail Aging Report"
```

*This report should print in landscape orientation, rather than the default portrait. UnForm will scale the columns and rows to 132 by 66.*

```
landscape  
cols 132  
rows 66
```

*The report heading is modified to print in the univers font, and the title is printed in bold. A box is drawn from the upper left of the printable region to column 132.5, row 4.5. Since box drawing is based on the center of a character cell, so 132.5 is the right edge of column 132.*

```
font 1,1,25,4,10,univers  
font 26,1,80,4,10,bold,univers,center  
font 107,1,26,4,10,univers,right  
cbox .5,.5,132.5,4.5,1,10
```

*The next commands cause the column headings on rows 6 and 7 to be printed in reverse (white on black.) A 100% shade region is printed, and a 0% shade font change is made to the text.*

```
shade 1,6,132,2,100  
font 1,6,132,2,shade 0,bold
```

*The hline command is used to remove any dashed lines from the report. Any occurrence of three or more dashes is erased.*

```
hline "---",erase
```

*The following font command looks for a text pattern of a phone number surrounded by parentheses. The ( and ) characters are “escaped” with leading backslashes, since they would otherwise be meaningful as regular expression characters. From each occurrence of the phone numbers, back up 37 characters, remain on the same row, and for 80 columns and 1 row, change the font to bold, cgtimes. This causes each customer subheading line to be modified.*

```
font "~\(...-...-....\)",-37,0,80,1,bold,cgtimes
```

*The remaining commands draw boxes, lines, and other enhancements. Lines are drawn around the report and between the aging columns. A box is drawn around the customer subtotals, which are headed by a line of several dashes. The grand total line is shaded and its font changed when it occurs on the last page.*

```
const BOXY=57
box .5,6,1,BOXY
box 54,6,1,BOXY
box 132.5,6,1,BOXY
box "Future",7,-1,1,BOXY
box "Current",8,-1,1,BOXY
box "30-Days",8,-1,1,BOXY
box "60-Days",8,-1,1,BOXY
box "90-Days",8,-1,1,BOXY
box "120-Days",9,-1,1,BOXY

box .5,63,132,1

box "-----",0,.25,78.5,1.5,1,10
shade "Grand Total:",-9.5,-.25,131.5,1.25,20
font "Grand Total:",0,0,12,1,cgtimes,bold,13
bold "Grand Total:",13,0,120,1
```

## LABELS – TEXT LABELS TO LASER LABELS

UnForm 4 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 is 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.

UnForm -i sample4.txt -f sample.rul -r labels -o output-device

*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 to 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 codeblock 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

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.

```
UnForm -i sample3.txt -f sample.rul -r 132x4 -o output-device
```

The rule set header identifies the name.
--

[ 132x4 ]

<i>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.</i>
--

```
cols 132
rows 66
across 2
down 2
landscape
cbox .5,.5,132.5,66.5
hline "---"
```

## ZEBRA LABEL – ZEBRA® LABEL PRINTER EXAMPLE

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 as it does for laser printer. 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 a 8 dpmm printer and 3.25 by 5.5 inch label stock.

UnForm -i samplez.txt -f sample.rul -p zebra8 -paper 3.25x5.5

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 code 3 of 9 code 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

UnForm Version 4.1 adds support for 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.

```
UnForm -i sample5.txt -f sample.rul -r outline -p pdf -o test.pdf
```

```
[outline]
# UnForm -i sample5.txt -o test.pdf -p pdf -f sample.rul -r 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 FUNDAMENTALS

The prejob, predevice, prepage, and precopy subroutines (and their associated postxxx 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 interact with other applications, devices, or the operating system.

An experienced BBx or ProvideX programmer 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 SDSI web site. In this manual, we have provided some basic (no pun intended) information that will assist developers experienced in other programming environments.

## Statements

A statement consists of 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, and can hold any amount of text data, of any ASCII character value from 0 to 255. On ProvideX, strings can be up to 32767 bytes long; on BBx, they are limited only by available workspace memory. Workspace memory is controlled by the number of “pages” in the “-m” startup parameter found in /usr/bin/UnForm or in the MEM value of the uniform.ini file under Windows – a “page” is 256 bytes. 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 3 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 both BBx and ProvideX. These are called string templates or composite strings. Details about these constructs can be found in language manuals for those languages.

## 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 to perform tasks typical to its environment.

## 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 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, or the UnForm supplied mid() function can be used to avoid the errors.

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
- subtraction
- \* multiplication
- / division
- ^ exponentiation



- =        used for assignment or to test equality
- >        test for greater than
- >=      test for greater than or equal to
- <        test for less than
- <=      test for less than or equal to
- <>      test for inequality
- ()      control precedence
- and     boolean and in conditions
- or      boolean 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 false clause, if present, is executed.

Both the then clause and the else clause can contain any statements, including nested if statements. A closing FI 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") fi; 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, you can 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 text files and pipes can also be used.

If UnForm is working with an application written in Business Basic, then the intrinsic files used by the application are available to UnForm for native access. This can provide some important benefits, allowing an UnForm job to use data from an application that hasn't been provided in the print stream. For example, inventory cost could be found using part numbers as keys to an inventory file, and the invoice cost could be calculated and placed on an internal copy of an invoice. This could be accomplished without changing the invoice printing program, because UnForm could gather the data just with the part numbers supplied on the invoice detail lines.

If UnForm is working with a non-Business Basic application (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'"

labelprt=unt
open(labelprt)"|lp -dlabels"
```

## 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, 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 can't use a key= clause, so you just read sequentially through the file:

```
read(faxlist,end=done)cust$,faxnum$
```

## Writing files

You probably wouldn't want to write to your application files, but you could well want to write to external devices or log files. Writing is performed with these verbs: WRITE or WRITE RECORD and PRINT. Both use a channel number and arguments to print. Both 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$(31)=str(amount:"000000.00"); write record(log)block$
```

## **Common verbs and functions**

The following list is a summary of verbs and functions that are commonly used in UnForm applications. Note also that the section on the **precop**y command contains additional references that are useful for programming code blocks. Note that all functions accept a “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 {}. Note that the syntax presented is for BBx4 and PRO/5 (PRO/5 is the run-time supplied for bundled licenses of UnForm, so this table is applicable to those users.) ProvideX syntax for many functions and some verbs is different, so you should consult with a ProvideX reference manual for details.

ASC( <i>string</i> )	Returns the ASCII numeric value (0-255) of the first character of <i>string</i> .
ATH( <i>string</i> )	Returns a binary equivalent of a human readable hex string. ATH("1B") returns an escape character.
BIN( <i>integer,length</i> )	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 <i>linelabel</i> if <i>linelabel</i> is the line after the closing WEND or NEXT.
CHR( <i>integer</i> )	Returns a character string whose ASCII value is <i>integer</i> , between 0 and 255. CHR(27) returns an escape character.
CONTINUE	Executes the next iteration of a loop structure. Equivalent to GOTO <i>linelabel</i> , if <i>linelabel</i> is the closing WEND or NEXT.
CVS( <i>string,arg</i> )	Returns a converted string according to the cumulative value of the integer <i>arg</i> . 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( <i>julian</i> { <i>,time</i> } { <i>:mask</i> })	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( <i>string</i> )	Returns the decimal conversion of the binary integer in <i>string</i> . The counterpart to the BIN() function. To treat <i>string</i> as an unsigned integer, you should use the form DEC(\$00\$+ <i>string</i> ).
DIM <i>string</i> ( <i>length</i> { <i>,char</i> })	Returns a string of <i>length</i> size, of spaces or the specified <i>char</i> character.
DIM <i>name</i> [ <i>dim1</i> { <i>,dim2</i> { <i>,dim3</i> }}]	Create an numeric or string array variable. Dimensions can be simple integers, indicating an index range of 0.. <i>dim</i> , or two integers separated by a colon, like 1:12.
DIR("")	Returns the current disk directory. On Windows, DIR( <i>driveletter</i> ) will return the current directory for the specified disk drive.
EPT( <i>number</i> )	Returns the 10's exponent value of <i>number</i> . EPT(100)=3, EPT(12)=2.
ERASE <i>filename</i>	Erases a file. Obviously, care should be taken to only erase

	temporary work files.
EXITTO <i>linelabel</i>	Exits a loop structure (current level only, in nested structures) and jumps to the specified <i>linelabel</i> .
FBIN( <i>number</i> )	Returns a 64-bit IEEE number in natural left to right ordering.
FDEC( <i>string</i> )	Returns the decimal value of a 64-bit IEEE number.
FID( <i>channel</i> )	Returns a file identification string for the file opened on <i>channel</i> . 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( <i>integer</i> {, <i>string</i> })	Returns a string if <i>integer</i> length, made up of successive iterations of <i>string</i> , or spaces if no <i>string</i> is provided. FILL(7,"abc") will return "abcabca".
FIN( <i>channel</i> )	Returns additional file information not found in the FID() function. A common use of this function is to determine file size, which 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 <i>numvar</i> = <i>start</i> TO <i>end</i> {STEP <i>increment</i> }	Initiates a loop, using a numeric variable initialized to <i>start</i> the first pass through the loop, incrementing by 1 or the specified <i>increment</i> , which can be negative, until the variable exceeds (or goes below in the case of a negative <i>increment</i> ) <i>end</i> . The statements following this command, until a NEXT <i>numvar</i> statement, are executed. The loop can be broken from with the BREAK or EXITTO verbs.
FPT( <i>number</i> )	Returns the fractional portion of a number. FPT(100.66) returns .66.
GOSUB <i>linelabel</i>	Jumps to the specified <i>linelabel</i> . Statements will be executed until a RETURN verb is encountered, and execution will return to the statement after the GOSUB.
GOTO <i>linelabel</i>	Jumps to the specified <i>linelabel</i> .
HTA( <i>hexstring</i> )	Returns a human readable hex string of <i>hexstring</i> . HTA(CHR(2)) returns "02". HTA("0") returns "30".
IF <i>test</i> THEN <i>statement(s)</i> {ELSE <i>statement(s)</i> } {FI}	Conditional execution of statements. <i>test</i> 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 FI are executed without regard to the condition of the last IF test. Nested IF statements are accepted without practical limit.

INFO( <i>num1,num2</i> )	Returns text information, sometimes in binary integer form, of various system or run-time elements. Common uses: INFO(0,0)=operating system name, INFO(0,1)=operating system version, DEC(INFO(3,0))=task ID, INFO(3,2)=user ID, INFO(3,3)=user name. Many other information strings are available. See the language reference for a complete list.
INITFILE <i>filename</i>	Initializes an existing file. Note that on UNIX, permissions will be set based on the user's umask setting. You can use the SCALL() function to execute a chmod command if necessary. Use caution: this verb should only be used on temporary work files.
INT( <i>number</i> )	Returns the integer portion of a number. INT(99.645)=99.
JUL( <i>year,month,day</i> )	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( <i>string</i> )	Returns the length of the string.
LET <i>var=value</i> {, <i>var=value...</i> }	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 an implied verb the statement uses the assignment syntax.
MASK( <i>string</i> {, <i>regexpr</i> })	Returns the position where a regular expression pattern was found in the <i>string</i> , or 0 if not found. If <i>regexpr</i> is not specified, then the last <i>regexpr</i> used is re-used. This provides a performance benefit for repeated uses of the same <i>regexpr</i> . The length of the string matched is returned by the TCB(16) function.
MAX( <i>num</i> {, <i>num...</i> })	Returns the largest number found in the list of <i>nums</i> .
MIN( <i>num</i> {, <i>num...</i> })	Returns the smallest number found in the list of <i>nums</i> .
MOD( <i>num1,num2</i> )	Returns the remainder of dividing <i>num1</i> by <i>num2</i> . MOD(4,3)=1, MOD(6,3)=0.
NUM( <i>string</i> )	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 <i>integer</i> GOTO GOSUB <i>linelabel</i> {, <i>linelabel...</i> }	Branches to one of the indicated line labels based on the value of <i>integer</i> . If <i>integer</i> 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 <i>integer</i> values greater than that of the last label.
OPEN( <i>integer</i> {, <i>err=linelabel</i>   <i>next</i> }{, <i>isz=i</i>   <i>integer</i> }) <i>string</i>	Opens the file named in <i>string</i> on channel <i>integer</i> . To open a file in binary mode regardless of the file type, you may optionally (required on ProvideX) specify a block size with

	the “,isz= <i>integer</i> ” option.
POS( <i>string1 relation string2 {,increment {,occurrence}}</i> )	Scans <i>string2</i> for a substring having the specified <i>relation</i> to <i>string1</i> . 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( <i>channel</i> ) <i>value {,value...}</i> {,}	Prints a series of values, numeric and/or string, to the file channel specified. A linefeed character is added to the channel unless the last character of the statement is a comma.
READ{ RECORD}( <i>channel {,options}</i> ) <i>variable {,variable...}</i>	Reads data from the specified channel into the specified variables, looking for field terminator characters to delimit variables. Field terminators include linefeeds, carriage returns, and nulls. Valid <i>options</i> include “err= <i>linelabel</i> ”, “end= <i>linelabel</i> ”, “siz= <i>blocksize</i> ”. “key= <i>keystring</i> ”, “ind= <i>index</i> ”, and “dom= <i>linelabel</i> ”. For intrinsic keyed files, use the key= or ind= options to read specific records. For text files, use READ to process linefeed 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	Place a non-executing remark line in the code. In UnForm, you can also use a # character.
RETRY	Retry the statement that caused the last error branch to be taken.
RETURN	Return from a GOSUB branch.
RND( <i>integer</i> )	Return a pseudo-random number. The random number sequence can be reseeded 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 <i>integer</i> can be a number from –32767 to +32767. Positive numbers return a random integer from 0 to <i>integer</i> -1. If <i>integer</i> is 0, a random number between 0 and 1 is returned.
ROUND( <i>number,precision</i> )	Return <i>number</i> , rounded to <i>precision</i> . ROUND(1.566,2)=1.57. ROUND(100.83,0) returns 101.
SCALL( <i>string</i> )	Executes the operating system command in <i>string</i> . Returns the result returned 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 <i>linelabel</i>	To provide a generic error handler to catch errors not trapped by <i>err=linelabel</i> branches in functions and verbs. UnForm also adds error handling code to code blocks, but will report errors on a warning page following the print job.
SGN( <i>number</i> )	Returns a 1, 0, or -1, depending on the sign of <i>number</i> .
STBL( <i>string1</i> {, <i>string2</i> })	Returns and/or sets the global string table value named <i>string1</i> . If <i>string2</i> is present, then the string table is set to <i>string2</i> . In both cases, the value is returned. If <i>string1</i> has not been set, STBL( <i>string1</i> ) will result in an error (trappable with <i>err=linelabel</i> , of course).
STR( <i>number</i> {: <i>mask</i> })	Convert a number to a string, optionally formatted with a <i>mask</i> . 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 <i>filename</i> {, <i>err=label</i> }	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", <i>err=next</i> .
TCB( <i>integer</i> )	Returns task control information. Commonly used <i>integer</i> values include: 10=last operating system error code and 16=length of MASK() function match.
TIM	Numeric variable that holds the decimal time of day, from 0.0 to 23.99.
UNT	Numeric variable that holds the next available file channel number.
WHILE <i>condition</i> ...WEND	A looping construct that performs statements between WHILE and WEND statements as long as <i>condition</i> is true or non-zero.
WRITE	Same syntax as PRINT.



# EMAIL INTEGRATION

UnForm Version 4.1 includes a copy of the MailCall utility that enables emailing of attachments from within UnForm. This is most often used to send PDF files, but 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 stick with 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.

## Configuration

On UNIX, we have supplied the perl program mailcall.pl, which is a SMTP client program designed to accept a submission file and interface with a SMTP server. In order for this program to operate, you must have a perl interpreter. If your system is missing this free scripting tool, you can probably find a binary distribution on <http://www.cpan.org>. MailCall can also interface with sendmail or mmdf if one of those products is configured and operational.

On Windows, we have supplied a simple Win32 executable called “mailcall.exe”, which accepts the submission file, extracts To: and CC: addresses, and communicates with the SMTP server configured in the mailcall.ini file.

Perhaps 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 pretty 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 a MS-DOS Command Window). If you get a non-error response, MailCall should work.

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 mailcall.ini. Note the following configuration options found in the file:

### **server=smtp-server**

This contains a reference to the IP address or domain name of the SMTP email server. This is used by the mailcall.exe and mailcall.pl programs. If your mailer= setting uses sendmail or mmdf, this value is not used.

### **from=email-address**

If no from\$ address is provided during the CALL to mailcall, this address is used instead.

### **mailer=commandline**

This line configures how MailCall actually sends the mail. It executes this command line via a BBx SCALL() or ProvideX SYS() function. There must be a % character in the command line, which MailCall substitutes with the email submission file at run-time.

On Windows, this 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 command line should be '/usr/mailcall/mailcall.pl % >/dev/null' (adjust the directory path as necessary, of course).

If you use sendmail, the command line '/usr/lib/sendmail -t <% ' should work, as it instructs sendmail to scan stdin for addresses.

If you use mmdf, then the command line '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.

### **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 timezone is determined from the date command.

## **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 "/usr/unform/mailcall.bb",1,x$,""
x.to$="someone@somewhere.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 "/usr/unform/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 "/usr/unform/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$="sdsi@synergetic-data.com"
call "/usr/unform/mailcall.bb",0,x$,""
erase output$
}
```

## MailCall Reference

CALL "mailcall.bb", mode, dat\$, errmsg\$

For ProvideX, use "mailcall.pv". Note that on Windows, do not call mailcall.exe directly. The mailcall.bb and mailcall.pv programs invoke mailcall.exe when required.

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\$**

**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*=0), msgtxt:c(1*=0),  
attach:c(1*=0),status: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\$**.

**dat.from\$** contains the sender's email address. On UNIX, the `sendmail` or `submit` programs automatically add Sender information based on the user's login name. The From value still appears on the recipient's email, however. On Windows, the `mailcall.exe` program will use this value when negotiating with the mail server.

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 angle 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.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 (`$0D0A$`). This will normally be null.

**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.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 mailcall.exe to display a status window displaying progress as it communicates with the SMTP server. This is only used on Windows installations.

**errmsg\$** will contain the text of an error message, if one occurs.

# NESTED RULESET EXECUTION

There are cases when it is helpful or necessary to use nested executions of UnForm from within a rule set. For example, since the PDF driver can only create a single output file per input stream, a nested rule set can be used to split a print job into multiple documents. Another example might be to combine collated and non-collated copy sets from the same job, since a single rule set must be set to one or the other mode.

In order to execute a nested rule set, first design the rule set(s) needed for the final output, just as you would any other rule set. After that, you need to design a rule set that does nothing more than track and store the input stream and execute UnForm jobs as needed. The input stream is stored for each page in the text\$[] array.

For example, this rule set will execute a secondary copy of UnForm for each page of the print job, and store a pdf file based on the document number on that page. The example prints one page at a time through the “invoice” rule set in the imaginary rule file “acme.rul”.

```
[primary]
prepage{
# create PDF file from document number
document_id$=trim(get(70,6,8))
secondary=unt
open(secondary)“|UnForm -f acme.rul -r invoice -p pdf -o /archive/invoices/”+document_id$+”.pdf”
for line=1 to 66
    print(secondary)text$[line]
next line
close(secondary)

# don't print anything from this ruleset
skip=1
}
```

On Windows, or when running with a ProvideX run-time, you can create text files and then use the SCALL function (SYS on ProvideX) to execute the secondary task, rather than a pipe. Use the STRING command (SERIAL on ProvideX) to create text files.

Note that when you run a secondary copy of UnForm, a second user-slot is used by the run-time engine, so you may need to license some extra users to account for this.

# HTML OUTPUT

UnForm Version 4 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 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. Or 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 (aging.htm in the above example) file 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 one of several styles. The rule set options used to trigger the style are shown in parentheses:

- The simplest form is simply 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 document, or in individual documents.

Note that all the 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 won't 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. This 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= <i>filename</i> toc= <i>filename</i> both= <i>filename</i> frame= <i>filename</i>	<p>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.</p> <p>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.</p> <p>See "OUTPUT TEMPLATES", below, for more information.</p>
colwidth= <i>text</i>	<p>The default column cell width is <i>text</i>. This can be a pixel value, such as "colwidth=9", or any other value accepted by a &lt;td width=<i>value</i>&gt; tag in HTML. If no value is specified, UnForm uses "2em", which indicates 2 <i>half-characters</i>, based on the average width of a character in the default font. This value can also be specified for individual reports using the <b>colwidth</b> keyword in a rule set.</p>
pagenum= <i>text</i>	<p>This text is used to generate the default table of content's values. A space and the page number follow</p>

Element	Description
	the text.
<code>imagelib=directory</code>	This points to a directory where image files are physically stored on disk. If any column definition is defined to contain 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 the page rendering speed by the browser.
<code>imageurl=url-prefix</code>	When image tags are generated in a column, the <i>url-prefix</i> is placed in front of the file name. This allows the Web server to map the name to a physical location on the server.
<code>complete=text</code> <code>incomplete=text</code>	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*  
    { *codeblock* }
2. coldef “*text* / ~*reg-exp*”, *coloffset, cols, options*  
    { *codeblock* }
3. coldef “*text* | ~*reg-exp*”, *coloffset, “to-text* | ~*to-reg-exp*”, *to-coloffset options*  
    { *codeblock* }

Format 1 defines an absolute column region. **coldef 30,21** for example, would define column region from column 30 for 21 columns (30-50.) If the “ccoldef” format is used, then *col* is the starting column, and *cols* is the ending column. **ccoldef 30,50** would define the same region as above.

Format 2 defines a region based on a search for a starting point. For each *text* value or *reg-exp* (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.

Format 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 (<br>). 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= <i>value</i> attribute to control the background color. The color can be expressed as a #rrgbbb 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= <i>value</i> ” attribute to control vertical justification. The default is “top”.
center, left, right	Cell gets “align= <i>value</i> ” attribute to control horizontal justification.
color=#rgb, color=color	Text gets <font color= <i>value</i> > attribute. The color can be expressed as a #rrgbbb hexadecimal value or as a color name supported by the target browser, such as red, blue, white, etc.
font=font	Text gets <font face= <i>font</i> > attribute. Several modern browsers support this, though the <i>font</i> typeface selected may not be available on all clients.
hdr=html text	The top of the column gets the <i>html text</i> , 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, is placed in a cell with <td> attributes specified <i>hdrtd text</i> , and text attributes <i>hdron text</i> and <i>hdroff text</i> . Be sure to turn off any <i>hdron text</i> HTML tags in <i>hdroff text</i> .
italic	Text gets <i> attribute.
image	Text is assumed to be file names that are image files. 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 .
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.
otheropt=options	The table cell gets additional attributes not otherwise

Option	How it gets applied
	specified by the other options.
size= <i>n</i>	Text gets <font size= <i>n</i> > attribute. Size ranges from 1 to 7, with 3 being considered a “normal” size.
suppress	If this word is present, then column data will be 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 increment 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:

<b>Function</b>	<b>Description</b>
<code>get(<i>col,row,cols</i>)</code>	Returns text from the page, given the column, row, and cols parameters.
<code>htmlencode(<i>text</i>\$)</code>	Returns <i>text</i> \$ after converting HTML entities into displayable versions.
<code>set(<i>col,row,cols,text</i> <i>\$</i>)</code>	Sets <i>text</i> \$ into the page at the given column, row, and columns.
<code>urlencode(<i>text</i>\$)</code>	Returns <i>text</i> \$ 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 a 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 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><b>` and **hdroff=**`</small></b>` 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 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 ruleset 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. The code is inserted into the module *ufmain.pn* at run-time, and BB<sup>X</sup>PROGRESSION/4 run-time environments are limited to 64K program sizes, so the amount of code added needs to be limited to 64K less the base size of *ufmain.pn* if you are running UnForm under a BB<sup>X</sup>PROGRESSION/4 run-time.

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  
    { codeblock }
2. rowdef “text | ~reg-exp”, rowoffset, rows, options  
    { codeblock }
3. rowdef “text | ~reg-exp”, rowoffset, “to-text | ~to-reg-exp”, to-rowoffset options  
    { codeblock }

Format 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**.

Format 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 *reg-exp* 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).

Format 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 a #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.
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. 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, is 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

Option	How it gets applied
	column definitions, even if they contain data values of 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 a HTML unordered list (<ul>... </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 by 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 what keywords perform what tasks.

## AGING REPORT SAMPLE

To produce this aging report sample to a file, execute the following command:

**unform -i sample3.txt -o aging -p html**

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 if 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 a HTML frame construct.*

```
multipage=y  
toc=y  
frame=y
```

*Between each page will be a 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 one for five 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, 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 with the ccoldef keyword with 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

- Acrobat, 22, 31, 58
- across, 30, 72
- Activation, 20
- Adobe Acrobat, 22
- alias**
  - Unix, 10
  - Windows, 11
- align, 146, 161
- attachments, 31, 44
- attr\$ variable, 147
- Attributes
  - bold, italic, light, underline, 39
- barcode, 32, 34
- BBx
  - user counts, 19
- BBx applications, 10
- bgcolor, 146, 161
- bin, 37, 76
- blink, 146, 161
- bold, 39, 51, 146, 161
- bookmarks, 70, 117
- border. *See*
- box, 40
- Business Basic code, 75, 158
- cgtimes, 53, 84
- check printing, 67
- coldef, 145
- color, 40, 58, 80
- color (RTL) images, 21
- cols, 43, 76
- colwidth, 149
- config.unf, 12
- constant values, 42
- copies, 20, 31, 44, 76
- copy**, 76
- Courier, 53, 84
- cpi, 45
- Cross hair pattern, 24
- detect, 46
- Device support, 6
- dots per inch**, 48
- double lined boxes, 40
- down, 47, 72
- dpi**, 48, 66, 89
- dump, 58
- DUMP**, 11
- duplex, 50, 76
- encode, 146
- end if, 56, 57
- environment variables, 84
- err=next**, 77, 159
- Examples, 100
- exec function, 77
- fixed, 85
- fixed font, 52
- fixed spacing, 54
- font, 16, 53, 84, 146, 161
- fontname*, 53, 84
- frame, 140, 150
- get(col,row,length)**, 77, 159
- hdr, 146, 151
- hdroff, 146, 151, 161
- hdron, 146, 151, 161
- hdrtd, 146, 151, 161
- hline, 55
- How to Create a Rule Set**, 92
- HTML output, 22
- HTML rule sets, 143
- HTML template files, 140
- if copy, 56, 57
- image, 58, 146
- images, 11
- Input file, 21
- Installation**
  - Unix, 7
  - Windows, 8
- international character sets, 83
- italic, 39, 146
- labels, 72
- landscape, 61, 76
- light, 39
- line drawing, 40
- line labels, 77, 159
- load**, 152
- logos, 11, 58
- lpi, 63
- macro, 64
- macros, 65, 97
- mailing labels, 72
- margin, 76
- margins, 66
- micr, 67
- mid(arg1\$,arg2,arg3)**, 77, 159
- move, 68
- MS-DOS, 6
- multipage**, 150, 153
- multi-up printing, 30, 47
- noencode, 146
- notext, 69
- Novell, 12
- nullrow, 154
- orientation, 76

- Orientation
  - landscape, 61
  - portrait, 74
- outlines, 70, 117
- output, 21, 71, 155
- output\$, 76**
- page, 72
- Page dimensions
  - cols, 43
  - cpi, 45
  - lpi, 63
  - margins, 66
  - rows, 79
- page size, 73
- pagesep, 157
- paper size, 73, 76
- Paper size, 23
- PCL5, 6
- pcopies, 44
- pdf files, 22
- PDF outlines, 70, 117
- percent gray*, 54, 80, 85
- pictures, 58
- point size*, 53, 85
- portrait, 74, 76
- post processing, 11
- postcopy, 75
- postjob, 75, 158
- postpage, 75, 158
- precopy, 75, 158
- prejob, 75, 158
- prepage, 75, 158
- programming*, 75, 84, 158
- proportional, 54, 85
- ProvideX, 13
- regular expression, 99
- rowdef, 160
- rows, 76, 79
- rule file, 21, 29
- rule set, 23, 29
- sample HTML rule set, 166
- samples, 100
- set(col,row,length,value\$)**, 77, 159
- shade, 80
- shift, 82
- skip**, 76
- substitution file, 84
- Substitution file, 24
- Supported devices, 6
- suppress, 161
- symbol set, 16, 53, 85
- symset, 83
- table of contents, 140, 164
- text, 84
  - movement, 82, 91
  - suppressing, 69
- Text
  - expressions, 84
- Text analyzer, 46
- text\$[all]**, 76, 158
- tips and techniques, 92
- Tips and Techniques**, 94
- title, 163
- toc**, 150, 164
- tr, 162
- tray, 76, 87
- trim, 146
- ufparam.txt, 16, 53, 84
- underline, 39, 147, 162
- UnForm
  - about, 6
  - command line options, 20
  - cross hair pattern, 24
  - in BB<sup>x</sup> or PRO/5, 10
  - MS-DOS, 6
  - programming, 75, 158
  - text analyzer, 46
  - Unix, 6
  - version 2 features, 24
  - Windows, 6
- units, 89
- univers, 53, 84
- Unix, 6
- user counts, 19
- valign, 146, 161
- variables, 77, 159
- Version 2, 24
- Visual PRO/5**, 12
- vline, 90
- vshift, 91
- width, 165
- wildcards*, 99
- Windows, 6
- Windows printing, 22
- Zebra label printers, 22
- zebra printers, 34
- ZPL II, 22