



UnForm 10.1 Manual

© 2024 Synergetic Data Systems Inc.

Table of Contents

Foreword	0
Part I OVERVIEW	11
1 INTRODUCTION.....	11
2 UNIFORM COMPONENTS.....	12
3 CLIENT-SERVER ARCHITECTURE.....	12
4 FLOW OF PROCESSING.....	13
5 CONCEPTS, PRIMER, AND TIPS.....	15
6 10.0 ENHANCEMENTS.....	19
7 10.1 ENHANCEMENTS.....	21
8 LEGAL NOTICES.....	22
Part II INSTALLATION	24
1 LINUX INSTALLATION.....	24
Server Installation	24
Standalone Client	26
Uninstall	27
2 WINDOWS INSTALLATION.....	27
Server Installation	27
UnForm 10.1 Manager	28
Standalone Client	29
Uninstall	30
3 LICENSING.....	30
4 HTTP SERVER PORTAL.....	34
5 FIREWALL CONFIGURATION.....	36
6 UPDATING AN EXISTING INSTALL.....	36
7 UPGRADING FROM 10.0.....	37
Part III CONFIGURATION	38
1 CONFIGURING THE SERVER.....	38
2 CONFIGURING THE CLIENT.....	48
3 CONFIGURING APACHE.....	49
4 CONFIGURING IMAGE MAGICK.....	51
5 CONFIGURING GHOSTSCRIPT.....	51
6 CONFIGURING TESSERACT.....	53
7 TCP/IP MONITOR.....	54
8 DELIVER CONFIGURATION.....	56
9 MESSAGE TRANSLATIONS.....	63
10 RPQ CONFIGURATION.....	63
Part IV INTEGRATING UNIFORM WITH APPLICATIONS	64

1 General Integration Concepts.....	64
2 Integrating UnForm with BBx.....	65
3 Integrating UnForm with ProvideX.....	67

Part V UNIFORM COMMAND LINE OPTIONS 69

1 Server Options.....	69
2 Command Line Admin.....	70
3 Client Options.....	70
Archive Options	89
Job List Options	94

Part VI DOCUMENT ARCHIVING AND MANAGEMENT 94

1 Overview.....	94
2 Logical Structure Details.....	96
3 Physical Structure Details.....	96
4 Document-level Identification.....	97
5 Image-level Identification.....	99
6 Document Data.....	99
7 Adding UnForm-Generated Documents.....	100
8 Using the Web Browser Interface.....	101
9 Direct Browser Access to Documents.....	102
10 REST Interface.....	102
11 Customizing the Web Interface.....	104
12 Custom Web Forms.....	104
HTML Form Structure	105
CGI-Driven Rule Sets	106
Javascript Execution of Rule Sets	107
Form Access Configuration	108
13 Using the UnForm Client.....	111
Triggering Archiving of UnForm Jobs	111
Adding External Documents	112
Document Image Retrieval	113
Document Deletion	113
Document and Index Listings	113
Searching for Documents	116
Testing Existence of Documents	116
Importing 7.x/8.0 Libraries	116
Importing Documents from sdStor	117
14 Functions Related To Archiving.....	118
15 Transferring Archives to A New System.....	119
16 Migrating Libraries From Previous Versions.....	120
Migrate to 10.1 on a new system	120
Migrate to 10.1 on the same system	121

Part VII ARCHIVE BROWSER INTERFACE 122

1 Login Form.....	122
2 Main Window.....	123

3	UnForm 10.1 Manual	
	OneDoc.....	124
4	Browse Library.....	124
5	Searches.....	125
6	Search Form.....	126
7	Search Results.....	128
8	Marked Images.....	130
9	Marked Documents.....	130
10	Document Viewer.....	131
11	Image Viewer.....	132
12	Multiview.....	133
13	Custom Forms.....	134
14	Settings.....	134
15	Address Books.....	135
16	Administrator Options.....	136
	Libraries	136
	Library Repair.....	137
	Users	139
	External Users	140
	Groups	141
	Build Demo Libraries	142
	Access Tables	143
	Active Directory/LDAP Synchronization	143
	Inbound Sources	145
	Web Extension Deployment	147

Part VIII DESIGN TOOL 148

1	About Rule Files.....	149
2	Sample Rule Files.....	149
3	Windows.....	150
	Main Window	150
	Open/Save File Window	151
	Select Rule Set	151
	Select Line	152
	Search and Replace	153
	Test Printing	153
	Preview	154
	Grid Printing	155
	Preview Options	156
	Samples	157
	Watch	158

Part IX IMAGE MANAGER 159

1	Overview.....	159
2	Document Sources.....	160
3	Image Manager Users.....	162
4	Browser Interface.....	162
	Document Selection	164
	Running Jobs	165

	Contents	5	165
	Manually Set Values		165
	Editing and Validation		166
	Transferring to Libraries		168
5	Job Definitions.....		168
	Overview		168
	Name/Info		169
	Parameters		170
	Zones		171
	Detection		173
	Data Fields		174
	Identification		175
	Custom Scripting		176
6	Script Libraries.....		179
	Filters		180
	Validations		180
	Lookups		181
	Set Values		181
	Global Code		182
7	Standard Filters.....		183
8	XML Document.....		192
9	Bulk Job Update.....		193
10	External OCR Processing.....		194

Part X DOCFLOW AND ANNOTATION 195

1	Overview.....		195
2	Browser Interface.....		196
	Attachments		198
	Actions		199
	Notes		200
	Log		201
	Annotations		202
	Settings		203
3	Flow Definitions.....		204
	Parameters		206
	Steps		207
	Fields		208
	Scripting		210
4	Roles and Users.....		211
5	XML Document.....		212
6	Annotated SubID.....		214
7	Stamps.....		215

Part XI SERVER MANAGER 215

1	Jobs		216
2	Connections.....		216
3	Log Browser.....		217
4	Log Monitor.....		218
5	Configuration.....		219
6	Scheduler.....		221

Part XII OTHER FEATURES**223**

1	APPLICATION FORMATTED OUTPUT (AFO).....	223
2	WINDOWS SUPPORT SERVER.....	225
3	DATABASE ACCESS.....	228
4	ADDRESS BOOKS.....	230
5	DYNAMIC RULE FILE TRANSLATIONS.....	230
6	HTML5 OUTPUT.....	232
7	MAILCALL.....	234
8	LOAD BALANCING/FAIL OVER.....	234
9	FAIL RECOVERY.....	235
10	UNIFORM WEB EXTENSION.....	236
	Configuration	237
	Rule Files	238
	Code Block Response	241
11	SDSI WEB EXTENSION.....	242
	Configuration	243
	Rule Files	245
	Code Block Response	248
12	DESKTOP DELIVERY AND FORMS.....	248
	Desktop Delivery	249
	Desktop Forms	250
13	DESKTOP CLIENT.....	252
	Deployment	253
	DTC Rule Sets	253
	Detect	253
	Title	254
	DTCPanel	254
	DTCHelpfile.....	254
	DTCButton.....	254
	Code Block Response For Buttons	255
	Code Block Response For ParseValue Requests.....	256

Part XIII RULE FILES**256**

1	Content-based Rule Sets.....	257
2	ACROSS.....	257
3	ANNOTATE, CANNOTATE.....	258
4	ARCHIVE.....	259
5	ATTACH.....	262
6	AUTHOR.....	263
7	BARCODE (PCL,PDF, PS).....	263
8	BARCODE (ZEBRA).....	269
9	BIN	271
10	BOJ, BOP, EOJ, EOP.....	272
11	BOLD, ITALIC, LIGHT, UNDERLINE, CBOLD,.....	272

	Contents	7	273
12	BOX, CBOX.....		273
13	BOXR, CBOXR.....		276
14	CIRCLE.....		277
15	COLS.....		278
16	COMPRESS, NOCOMPRESS.....		279
17	CONST, GLOBAL, LOCAL.....		279
18	COPIES, PCOPIES.....		280
19	COVER.....		281
20	CPI		281
21	CROSSHAIR.....		282
22	DELIVER.....		282
23	DETECT.....		284
24	DOWN.....		285
25	DPI		286
26	DSN_SAMPLE.....		286
27	DTCBUTTON.....		286
28	DTCHELPPFILE.....		287
29	DTCPANEL.....		287
30	DUMP.....		288
31	DUPLEX.....		288
32	EMAIL.....		289
33	ERASE, CERASE.....		290
34	FIXEDFONT.....		291
35	FONT, CFONT.....		291
36	GS		294
37	HLINE.....		294
38	HSHIFT.....		295
39	IF COPY ... END IF.....		295
40	IF DRIVER ... END IF.....		296
41	IF EXPRESSION ... END IF.....		296
42	IMAGE.....		297
43	IMAGES.....		300
44	ITALIC.....		301
45	JAVASCRIPT.....		301
46	KEYWORDS.....		302
47	LANDSCAPE, RLANDSCAPE.....		302
48	LCOPIES.....		302
49	LDARKNESS.....		303
50	LIGHT.....		303
51	LINE		303
52	LOAD.....		304

53	LOCKCOLS.....	305
54	LPI	305
55	LSPEED.....	305
56	MACRO.....	306
57	MACROS.....	306
58	MARGIN.....	306
59	MERGE.....	307
60	MICR.....	307
61	MOVE, CMOVE.....	308
62	NOTEXT, NOOVERLAY.....	309
63	OUTLINE.....	309
64	OUTPUT.....	310
65	OVERLAY.....	311
66	PAGE.....	311
67	PAPER.....	312
68	PORTRAIT, RPORTRAIT.....	313
69	PRECOPY, PREDEVICE, PREJOB, PREPAGE, POSTCOPY,.....	313
70	PROTECT.....	314
71	ROWS.....	315
72	SHADE, CSHADE.....	316
73	SHIFT.....	317
74	SUBJECT.....	318
75	SYMSET.....	318
76	TEXT.....	319
77	TITLE.....	324
78	TRANSPARENCY.....	324
79	TRAY.....	325
80	UNDERLINE.....	325
81	UNITS.....	326
82	VLINE.....	326
83	VSHIFT.....	327
84	WEBACTION.....	327
85	WEBFIELD.....	328
86	WEBITEM.....	329
87	WEBPANEL.....	329
88	ZCOPIES, ZDARKNESS, ZSPEED.....	330

Part XIV SAMPLE RULE FILES

330

1	simple.rul.....	330
	SIMPLE1 - invoice rule set	331
	SIMPLE2 – add font control and text	332
	SIMPLE3 – add copies, barcode, watermark	335

SIMPLE4 – constants, color, expressions	338
2 advanced.rul.....	342
INVOICE - from preprinted to UnForm	342
STATEMENT - two page formats in same job	348
AGINGREPORT - enhanced Aging report	354
LABELS – text labels to laser labels	361
132x4 – multi-up and scaled reporting	362
ZEBRA LABEL – Zebra label printer example	363
OUTLINE - PDF outline	364
3 Additional Sample Rule Files.....	365

Part XV PROGRAMMING CODE BLOCKS AND JOBS 367

1 Basic Syntax.....	368
2 Object Oriented Programming.....	374
Object Instantiation	374
Object Access	375
Object Destruction	375
3 Built In Objects.....	375
addrbook	376
binfile	377
cirrusprint	378
collection	378
compare	379
date	380
docflow	382
docflow doc	384
doclist	385
filters	387
grid	388
http	392
inbound	393
inbounddoc	397
infile	399
jobdef	400
jobdefs	402
json	402
keyfile	404
libdoc	406
libraries	407
library	408
lookups	415
mailattachment	416
mailmessage	416
mailreader	418
mailsender	419
marked	420
memcollection	421
notify	421
objcollection	422
rac	422
search	423
system	425
textfile	426
validations	427

webapi	428
xmlreader	430
4 Internal Variables.....	433
5 Internal Functions.....	438
6 Runtime verbs and functions.....	462
7 Error Codes.....	468

Part XVI LEGACY HTML RULE FILES 470

1 Creating HTML.....	470
2 HTML Configuration.....	471
3 HTML Output Templates.....	472
4 HTML Rule Sets.....	474
5 BORDER.....	475
6 COLDEF.....	475
7 COLWIDTH.....	478
8 FRAME.....	478
9 HDRON, HDROFF, HDRTD.....	479
10 LOAD.....	479
11 MULTIPAGE.....	480
12 NULLROW.....	480
13 OUTPUT.....	481
14 OTHEROPT.....	481
15 PAGESEP.....	481
16 PREJOB, PREPAGE, POSTJOB, POSTPAGE.....	482
17 ROWDEF.....	483
18 TITLE.....	485
19 TOC	485
20 WIDTH.....	486
21 Sample HTML Rule Set.....	486
22 Aging Report Sample.....	486

Index 0

1 OVERVIEW

1.1 INTRODUCTION

UnForm is a software product designed to work as a filter between an application and an output device or file, such as a laser printer or a PDF document, 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.

UnForm is unique in its ability to analyze report output to determine what, if any, customization to apply. When a report is detected that requires enhancements, UnForm can add line drawing, images, barcodes, 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 programming environment to add true programmed intelligence to any print job.

The enhanced output can be used to simulate pre-printed forms, or to change the look of plain-paper forms 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 PCL5, PostScript, or PDF print languages. Most laser printers, some inkjet printers, and many software products support at least one of these forms of output.

UnForm can also produce virtually identical output in Adobe's Portable Document Format (PDF), and HTML5, and similar output in Zebra's ZPL II language, supported on many Zebra thermal label printers. With proper configuration, UnForm can automatically convert its PDF output to any format supported by Ghostscript, including PCL3, PCL6, Windows print drivers, tiff, jpeg, or png images, and more.

While UnForm has traditionally accepted plain text print streams and constructed documents from this basic text, it can also accept PostScript or PDF print streams that contain application-formatted documents. In conjunction with GhostScript, this pre-formatted data is translated to printable format, with optional enhancements such as images, barcodes, text, and drawing features added by UnForm. Further, the text elements provided in the input are available to UnForm jobs for designing full-featured document management applications.

UnForm can also accept XML data, and other text formats like CSV, to produce documents based on the content.

UnForm has built in document delivery capability, so that any document would normally be printed can be emailed, faxed, sent to disk, or sent to another program. The destination information for these deliveries can come from the print data itself, from UnForm address books, or from external sources such as ODBC or web services.

UnForm can also archive whatever it produces in a powerful and secure document repository, and offers an extensive document management capability, including browser-based browsing, searching, and viewing, REST-based or command line programmatic interface, and scanning and importing of external images and files into the repository.

1.2 UNIFORM COMPONENTS

UnForm Document Management incorporates several features and components, all designed to work seamlessly together.

- **Print, Electronic Delivery**, is the basic functionality provided with any UnForm server. Convert plain and structured text, or PDF files, into professional and useful documents in many formats. Deliver those documents to printers, email addresses, and fax gateways. A powerful built-in scripting language provides extensive capability for customizing the processing and production of all your documents.
- **Document Archiving**, which provides a secure, extensively indexed, document repository, viewable with a web browser. Document libraries are encrypted and compressed storage files, protecting the documents themselves from file system navigation. The document files can be virtually any format, most often PDF and image formats, but also text, XML, Office documents, drawings, and so on. The only limitation is a 2GB file size limit and the ability of a viewing system to display the file when viewed or downloaded via a web browser.
- **Image Manager** provides automated handling of inbound documents, those that are not printed by the user's business application. Documents can be retrieved from email, file system locations, or user uploads, enabling management of both PDF and scanned images, as well as other document formats. This component includes a browser-based interface plus extensive server-side scripting capabilities to provide highly automated handling of the inbound documents. Note that in 10.1, this tool replaces the Windows-based *legacy Image Manager*. The older Image Manager is still available for customers who have existing scanning work flows, but it is deprecated.
- **DocFlow** enables document-based work flow scenarios to be created, for routing, approval, and more. Capture data, attachments, notes, and draw annotations, as designated users manage documents through one or more steps. Scripting provides logical flows based on conditions like document value or age. Design proof of delivery processing with signature capture, photo capture, and user-entered data.
- **Design Tool** is a browser-based component that provides developers with an integrated editing and test environment for rule files. Using the Design Tool can save many hours of effort in rule set development.

1.3 CLIENT-SERVER ARCHITECTURE

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

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

The UnForm clients can be installed anywhere on the network, on Windows or UNIX systems. On Windows, the client is a native Windows executable. On UNIX, the client is a Perl program, so UNIX systems require Perl level 5.6 or above. Clients perform the application interface work, taking input from

the application, submitting it to the server, and in many cases, returning the result back to the client for processing.

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

For complete information about how to operate the client and server programs, read the [Command Line Options](#) chapter. In general, on Windows the server is operated from the Server Manager option or as a Windows service, and on UNIX the server is operated like this:

```
uf101d start  
uf101d stop
```

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

```
cat sample1.txt | uf101c -f simple.rul | lp -dhp -oraw  
uf101c -i sample1.txt -f simple.rul -o ">lp -dhp -oraw"  
uf101c -i sample1.txt -f simple.rul -p pdf -o client:sample1.pdf
```

In the first example, uf101c submits the job and returns the result to its spooler. In the second example, uf101c submits the job and the server prints the result to its spooler. In the third example, uf101c submits the job requesting PDF output, and returns the result to its file sample1.pdf.

1.4 FLOW OF PROCESSING

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

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

```
uf101c -i sample1.txt -o ">lp -dlaser -oraw" -f acme.rul
```

```
cat sample1.txt | uf101c | lp -dmylaser -T pcl
```

```
cat sample1.txt | uf101c -p pdf >/home/mypdfs/xyz.pdf
```

```
uf101c -i sample1.txt -o client:myfile.pdf -p pdf
```

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

For performance reasons, it is normally desirable to specify a server-based output designation with the `-o` option. In that circumstance, the client only runs long enough to submit the job and ensure the command line arguments are acceptable to the server, then returns to the application. If the client will be receiving

the output, it must wait for the job to finish and retrieve it, which can be time consuming (though certainly less so if the client and server are on the same machine).

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

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

Note that if the input is found to be PostScript, then UnForm processes the job using [UnForm AFO](#).

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

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

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

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

- The prepage code block is executed.
- Any command expression values are resolved.
- For each copy:
 - The precopy code block is executed.
 - Command expressions are resolved.
 - Any `hshift` or `vshift` commands are executed (if `shiftfirst=1` in `ufparam.txt` [defaults]).
 - Move commands are executed.
 - Font, bold, italic, underline, and light commands are executed.
 - Shade commands are executed.
 - Box commands are executed.
 - Text commands are executed.
 - Hline and vline commands are executed.
 - Erase commands are executed.
 - Any `hshift` or `vshift` commands are executed (if `shiftfirst=0` in `ufparam.txt` [defaults]).
 - Attach commands are executed.
 - Image commands are executed.
 - Barcode commands are executed.
 - The application text, with any font attributes applied, is added.
 - Micr commands are executed.
 - The postcopy code block is executed.

- The postpage code block is executed.
- When all pages have been processed, the postjob code block is executed.
- As the job is processed, the output designation for each copy is checked, and if the output is changed, predevice and postdevice code blocks are executed. When running a PDF job, the only time the output can be changed is in the prejob code block, or with an output command that is non-copy specific. The postdevice code block is executed after the output is complete and closed, making it suitable for handling the output file itself (for emailing, faxing, etc.).

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

So the following scenario will conflict:

- **uf101c -i sample1.txt -o client:/tmp/invoice.pdf -f advanced.rul -r invoice**
- In the invoice ruleset is this: **output "/home/pdfs/invoice.pdf"**
- The server will send output to its /home/pdfs/invoice.pdf file, leaving the temporary output for the client empty. The client /tmp/invoice.pdf file will be an invalid, empty file.

1.5 CONCEPTS, PRIMER, AND TIPS

UnForm is a very powerful tool, with dozens of commands and features. It can be difficult to grasp the basics from such a large toolset, but the basics are really very simple. Once UnForm is installed by an administrator, the only skills required to develop typical business forms are an ability to edit text files on your system, and an ability to execute UnForm as needed to test your changes. The Windows-based UnForm Design Tool can provide an efficient environment for rule file development, if desired.

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

- UnForm processes text input and produces formatted output. The input can come from a file or, on UNIX, can come from UnForm's standard input. The output can go to a file or a device on either the server or the client, or on UNIX can go to the client's standard output.
- UnForm can also process pre-formatted application output, if that output comes to UnForm as PostScript. See the [UnForm AFQ](#) chapter for more information.
- UnForm uses a *rule file* to define all the form and print jobs it might process. In that rule file are one or more *rule sets*, each of which represents one form or print job. Rule files and the rule sets they contain are simply text files with command lines, which you can edit with any text editor. The rule file should be stored in the UnForm directory, and specified with the "-f *rulefile*" command line argument. If you don't specify the rule file on the command line, then the default rule file named at installation is used.
- Unless the "-r *ruleset*" command line option is used, UnForm reads the first page of input and compares that first page with all the **detect** statements found in each rule set. These statements instruct UnForm to look for text or patterns at specified locations or lines (or anywhere on the page). If all the detect statements for a given rule set match the contents of the first page, then UnForm selects that rule set and begins to produce output. If a match is not found, then the next rule set is tested, and so on until all the rule sets have been tested. If no match is found, then UnForm will pass the job through without any changes or enhancements, or in the case when a pdf or pcl driver is specified with a **-p driver**

command line option, will produce a text job scaled to fit each page.

- Each job has its own *geometry*, that is, the basic columns and rows to which UnForm scales everything. If you specify **cols 85**, then UnForm will scale each character and all the enhancement positions and sizes to 1/85th of the printed space between the margins. In a sense, the job wraps enhancements around the text input as it is sent to the output.
- The commands in the rule set determine what enhancements are applied. These can be text additions, font changes, boxes, shade regions, barcodes, images, and more. Each change is controlled by a command line in the rule set, such as **box 5.5,2,20,4**.

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

- Some commands control the printer. For example, the **tray** command can select the input tray on a laser printer, and the **bin** command can select an output bin.
- You can have UnForm generate multiple copies of each page of input. Each copy can have unique characteristics by using **if copy n** blocks. This is a simple structure that starts with a line "**if copy n**", where *n* is the copy number, followed by any number of lines of enhancement commands, followed by a line "**end if**".

Creating Rule Files with the UnForm Design Tool

- Obtain sample output from your application for the form you want to design. Most applications provide the means to print to a text file. If no other means exists, you can define a printer that prints to UnForm with a **-debug** command line option, in which case UnForm will leave a copy of the input stream on the server, under the UnForm directory, in temp/jobno.in. You can find job numbers and their print times and size with the **uf101c -myjobs** command.

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

- Access the [UnForm Design Tool](#) from a web browser, and login. Create a new rule file, then a new rule set, then set the sample to the file created above. The UnForm Designer is a rule file editor with on line help, command editors, and drawing and preview capabilities.

Manual Rule Set Creation Steps

- Obtain sample output from your application for the form you want to design. This output can be printed to a text file, or you can simply use two printers defined with UnForm, one with the crosshair option (-x), the other with normal output. If you are working on a Windows system or have network access from a Windows system to the server where UnForm operates, you can use the pdf driver and an Acrobat Reader to save paper while developing the design.
- Print your sample through UnForm with the crosshair option turned on. This will provide you with a grid of text positions printed by your application. If you have a file printed by your application, the command line for a grid would look like this: **uf101c -x 1-99 -i input-file -o output-device** or **uf101c -x 1-99 -i input-file | lp -dxxx**. If your sample does not contain form-feeds, you can add a **-page n** option to tell

UnForm how many lines are to be read per page.

- Since you will be printing this sample many times, you may wish to create a script or batch file to automate the command line, which will be something like: `uf101c -i input-file -f rule-file -o output-device` or `uf101c -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 multiple patterns by using more than one detect statement. Patterns are specified by starting the detect string argument with a `~` character. The balance of the string is a regular expression. Common syntax elements for regular expressions include `.` to match any character, `[0-9]` to match any digit, `[A-Z]` to match any capital letter, and `*` to match any number of repetitions of the prior match character. A more complete description of regular expressions is in the Regular Expressions chapter.

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

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

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

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

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

Tips and Techniques

- * Always start with a crosshair pattern, so the basic text provided by the application, and its exact placement, can be seen. As the crosshair mode prints just the first page, use short versions of the reports or forms. There are several ways to create a crosshair version of a report:
 - Print the report to a file, then process that file with UnForm's command line, such as **`uf101c -i filename -o output-device -x`**
 - Add a printer configured with the `"-x"` option, and print to that printer.

If your report doesn't contain form-feed characters at the end of the page, then you should print just one page worth of data, or add a `-page n` option to the command line. Otherwise, UnForm will assume the page is made up of as many lines as are printed, up to 255 lines.

- * Use **detect** statements to identify each form. UnForm is designed to process all your reports and just enhance those it can identify; all others are passed through unchanged. This is easier to set up than forcing a given printer device to be named for every form or report, as is required of most form packages.
- * Specify the columns and rows for the form or report using the **cols** and **rows** commands. If this isn't done, then UnForm will assume 80 columns by 66 rows. An exception to this assumption is that if a **page** keyword is used, then the rows will be taken to be that value unless a rows command is also present.
- * Remove unwanted text with the **erase** command, or move it with the **move** command. In programming code, such as in the prepage or precopy routines, you can modify the text\$[] array directly or via the set() function.
- * Apply attributes to the text with the **bold**, **italic**, **light**, or **underline** commands. These apply to the text generated by the application (not to text you add with the **text** keyword). Or use the **font** command, which can apply any of these attributes as well as apply other characteristics to the application text data.
- * Use the **font** command to modify the font of text from the application. All text printed by the application will print in Courier unless changed with the font command. When changing to a proportional font, be sure to make the changes to specific logical regions, such as a column of prices. If you change the font for the entire page, then columns will not align properly.
- * Add text, such as headings or messages, with the **text** command. Text can be literals enclosed in quotes, named values from a substitution file if prefixed with "@", environment variables prefixed by \$, or an expression enclosed in { } characters. Text can be rendered at any size and in any font supported by the printer or device. Remember that fixed pitch fonts, such as Courier, are sized in characters per inch, while proportional fonts are sized in points. The larger the cpi, the smaller the font. The larger the point size, the larger the font.
- * Add shading and box drawing with the **shade** and **box** commands. Reverse shading is accomplished by shading a region with 100% (black) shading, and using a **font** or **text** command to modify the text to shading of 0% gray (white). Simply using a row or column value of 1 will draw lines. To draw a box and shade the interior, use the shade option of the **box** keyword.
- * Add logos and other images with the **image** command. With this command, UnForm normally looks specifically for PCL raster images (or PDF images if the pdf driver is used) in the file. UnForm can also be configured to use Image Magick or Image Alchemy for on-the-fly conversion of traditional image formats to native PCL, EPS, or PDF.
- * Use the **attach** command to add overlays or attachments. This command does not search only for image data. It does, however, search for and remove initialization and form-feed codes.

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

To create an overlay, use the **attach** command, but allow the text and enhancements to also be applied on the same copy. Attachment documents for PCL output can be created using a PCL5 printer on Windows, selecting the Print to File option or setting it up to use a FILE: port. For PDF attachments, use Adobe Distiller, choosing PDF 1.5 or lower compatibility to avoid some features that are not compatible with UnForm's PDF parsing engine.

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

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

1.6 10.0 ENHANCEMENTS

UnForm 10.0 offers many significant enhancements to its already powerful predecessors. A list of major enhancements is provided below.

New Image Manager Component

Image Manager is a browser-based tool with extensive capabilities for identifying incoming documents, enabling custom data capture and automatic or manual property assignment. It includes extensive scripting facilities for handling of the most complex document automation challenges. Inbound documents are collected from folder, email, and user sources.

This tool replaces the previous Windows-based Image Manager, which is now included as the Legacy Image Manager, primarily to facilitate migration of existing job definitions so they can run with Version 10 servers.

New DocFlow Component

DocFlow And Annotation is a browser-based tool that enables document routing and workflow applications to be developed. It supports step-based routing, with role-based security. Custom scripting enables flexible rules to be enforced, or ERP integration to be designed. Annotation capability enables signature capture and proof of delivery applications to be designed as well.

Print Management Enhancements

Enhanced Security with the addition of a shared-secret *authkey* value, so a client cannot connect to the server without a matching *authkey* value.

Proxy Connections, allowing a client to connect to an UnForm server via a properly configured web server with the `-proxy` option. The internal Apache server is automatically configured to handle such connections. This feature enables UnForm to reside behind a firewall, but enable print jobs to be submitted via a public http/https connection.

New Functions and Objects:

- `logdeliver()` function enables rule set additions to the deliver logs

- grid object to manage tab-separated-value data
- notification object for sending template-based email notifications to addresses, users, and DocFlow role groups
- lookups, filters, and validations objects to run script code built with a simple editor included with the new Image Manager
- inbound, inbounddoc objects for Image Manager operations
- docflow, docflowdoc objects for DocFlow operations
- libdoc object for simplified library document management
- runjobon() function enables execution of a job on a specific server
- skip can be set to true (non-zero) in prejob to exit the job without processing, such as when a job has been sent to another server

Inbound Sources enable running of print jobs from any monitored folder, or via email.

Archive Command changes to support archiving on a separate server, with spooling to ensure fault tolerance.

Performance Improvements when using archive and deliver commands, through a timing change in when PDF processing is performed.

History of print job input files is maintained at the server, with an automatic purge cycle controlled by the histage=*days* setting in uf101d.ini.

Load Balancing support is available, by specifying multiple servers in the -server option of uf101c. Note also the archive command changes as it is usually important to have archive libraries maintained on a single server. Part of load balancing is maintaining duplicate files across multiple systems. This is automated through a syncing process described in the load balancing section.

Subjob Enhancement allows subjob ID values of thousands of characters (the former limit was 25). Beware, however, of using large deliver docid's as those values are used to generate file names.

Fail Recovery enables client-side storing and resubmission of [failed job](#) submissions, controlled by the failhist setting in uf101c.ini, and the -rerun command line options.

The **margin** command is now used by AFO jobs to adjust the position and size of the page overlay.

Improved support for commands with expressions inside code blocks. Expressions are resolved in-line so looping constructs are now supported.

Email passwords now support encrypted password storage maintained with the admin browser interface. Use "store:storename" in place of the plain text password anywhere it is configured or used.

Document Archiving Enhancements

Doc Data feature added, allowing user-defined name, value pairs, with indexing. This feature is used extensively by the new Image Manager and DocFlow components, but is also available for use in custom archiving requirements. Both Browse and Search have been enhanced to allow document viewing by Doc Data names and values, and the uf101c command line has also been enhanced for its use.

Mark Images has been improved, with a toolbar option to mark all images of selected documents, in both Browse and Search.

Embedded PDF Viewer, a customized version of Mozilla Foundation's PDF.js toolkit, is included and optionally turned on by user. This restores hyperlink support that was lost when browser vendors dropped support for the Acrobat PDF plug-in.

Search Results include a library column, rather than a library selector, making multi-library search results more intuitive.

LDAP Sync now supports multiple profiles, enabling support for multiple LDAP/AD servers in a single installation.

Design Tool Enhancements

Code Folding improves navigation through rule sets and provides a clear visual indicator of logical sections such as loops.

Print Rule Set feature provides complete rules et printing rather than the default browser visual space-only printing.

Browser View of Test Print supports PCL, Postscript, and ZPL previewing at the browser, through the use of GhostPCL, GhostScript, and the labelary.com web service, respectively.

Server Manager Enhancements

Several minor enhancements, including a Test Email capability and the ability to configure GhostPCL and MuPDF.

Evaluation Mode Changes

When UnForm 10 is installed on a new system, it is automatically activated as a 21-day evaluation version. This mode supports all the features of UnForm, with 3 print jobs, 3 image manager users, 10 docflow users, design tool, and archiving. All documents produced by UnForm have a watermark, and data files are limited to 5,000 records. **Some limitations are different than previous versions of UnForm.**

1.7 10.1 ENHANCEMENTS

Version 10.1 offers many enhancements, including:

Priority Levels for Printing

When jobs are submitted with `-async` or `-asyncripq` options, they are added to a queue directory "rpq". This also applies to jobs submitted via print-type inbound sources. Version 10.1 processes this queue in three priorities: high, medium, and low. You can specify a priority with the new `-priority` option in the `uf101c` command line, and can also configure content-based prioritization in the new [rpq.ini file](#).

Multi-level Inbound Source Folders

An inbound path source can have subfolders that get processed in addition to documents in the top level folder. The subfolder is referenced as the meta data value "path", so is available for logic in image manager scripting.

Batch Rotation in Image Manager

You can select a group of documents in Image Manager and rotate them all the same direction. In addition, if mutool is configured, PDF pages are rotated using it, to retain their vector formatting.

Global Code Merge

A new [globalcode.custom.ini](#) file is maintainable in Image Manager script libraries to establish code that is inserted into every Image Manager and DocFlow job. The file contains sections for each of the custom code sections found in the jobs of these two modules. Code added to this file is automatically inserted into jobs following custom code sections.

Command Line LDAP/AD Sync

New -ldap* [command line options](#) are provided to backup the user/group databases and synchronize with a LDAP or Active Directory server. Previously this feature was only available through the browser interface.

Improved Images Command PDF Handling

When an images command specifies PDF files and no formatting options and the job output format is PDF, the PDF pages are inserted in vector format rather than first being converted to raster images.

Additional Enhancements

- An admin user can now move selected documents to a new inbound source from the Image Manager menu.
- A non-admin user can be assigned the right to assign documents to other users in Image Manager
- A -nodeliver command line, and the nodeliver variable, can be used to turn off deliver functions
- The archive search interface provides a button to clear previously selected libraries
- The filename metadata item is now defined for manually uploaded documents in Image Manager
- XML files are not picked up from inbound source paths until they are complete
- Dropped the confirm password prompt in user editing, as it was not needed with the Show Me button to display the password for the admin user
- Updated the DocFlow open in new window button to open in a frame window initially, to allow the primary image to be displayed while other tabs are viewed. The frame window contains a button to open the image in a new top level window to provide the 10.0 functionality.

1.8 LEGAL NOTICES

UnForm is published under license by:

Synergetic Data Systems, Inc.
1040 Camerado Drive, Suite 100
Cameron Park, CA 95682
USA

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

UnForm is Copyright ©1994-2014 by Allen D. Miglore. All rights reserved.
UnForm is distributed under license by Synergetic Data Systems Inc.
UnForm is a registered trademark of Synergetic Data Systems, Inc.
Other product names used herein may be trademarks or registered trademarks of their respective owners.

In order to install UnForm, the installer must accept a license agreement, the text of which is below:

**UnForm® Document Management Software
License Agreement**

NOTICE: INSTALLING THIS PACKAGE INDICATES YOUR ACCEPTANCE OF THE FOLLOWING TERMS AND CONDITIONS. PLEASE READ THEM. IF YOU DO NOT AGREE WITH THEM, DO NOT INSTALL THE PACKAGE, AND REMOVE IT FROM ANY SYSTEM YOU HAVE PLACED IT ON.

"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.

2 INSTALLATION

2.1 LINUX INSTALLATION

2.1.1 Server Installation

UnForm 10.1 for Linux requires C libraries 2.5 or higher. You can check the version on your system with the command **ldd --version**. Note that the numbering convention does not use zero-padding, so 2.10 is higher than 2.5.

Another thing to note is that UnForm can be configured to use third party packages, and it is usually best to be on a recent version of Linux in order to have access to recent versions of those packages. The publisher recommends using a well supported (commercial or community) Linux with readily available packages, such as Redhat, CentOS, Ubuntu, Debian, or OpenSUSE.

- Login as root, or use "sudo -i" to become an interactive super user.
- Create a directory to hold the UnForm files, and change to that directory.

```
Example:  
umask 0  
mkdir /opt/uf101  
cd /opt/uf101
```

- Uncompress and extract UnForm from the download file. If the uncompress program is not available, you can use gunzip instead.

```
gunzip uf101_xxx_tar.gz  
tar xvf uf101_xxx_tar
```

- Execute the UnForm set up script.

```
./ufsetup.sh
```

The ufsetup.sh script will create two scripts, called /usr/bin/uf101c and /usr/bin/uf101d. The uf101c program is the client, while uf101d manages the server.

The setup script will offer to create the user 'unform', or if found, to run UnForm under that user. If you choose not to run UnForm under that user, it will default to starting up as 'root', and you will need to edit the uf101d.ini file [apache] section to specify a different user under which to run the private Apache HTTP server instance (since that cannot run as root).

Note that earlier versions of UnForm defaulted to running as 'root', so if you migrate rule files and other files to version 9, you might need to adjust permissions to allow the 'unform' user to read and write those files.

The setup script will also look for an Apache web server in several common locations. If it isn't found, you will be prompted for a location, both of the path to the httpd executable and also a directory where Apache modules reside.

If Apache is not installed on your Unix system, you must install it before installing UnForm. Apache is normally present on any Linux or OSX system, but may or may not be present on AIX systems, though is readily available from IBM.

The setup script will test to see if the chkconfig program exists on the system (many modern Linux systems have this), and if so and the /etc/init.d/uf101d script does not exist, it will offer to install it. This will enable automatic startup of the UnForm server at boot time, and the server can be controlled via the 'service' command line typically used to manage other services: 'service uf101d start', for example.

- Activate evaluation mode, or activate permanently, using **./license.sh**.
- Start the server: **uf101d start**
- Run **uf101c -v** command to ensure UnForm is installed and set up correctly. The output from this command will display information about the installation. Note that uf101c requires Perl version 5 or higher.

See the [licensing](#) chapter for activation information.

Automatic Startup

The install script will automatically attempt to set up UnForm as a service that starts at boot time. It recognizes several standard Linux methods for doing this. However, if this fails due to having an unknown or older Linux distribution, you can review these methods:

- Add a line to /etc/inittab to run uf101d start one time at all regular run levels

```
uf101:2345:once:/usr/bin/uf101d start
```

- Create a script S99uf101d, and place this script in any /etc/rc#.d directory associated with normal run levels (typically /etc/rc2.d, /etc/rc3.d, or /etc/rc5.d). It doesn't hurt to place the script in levels 2, 3, and 5. The script can be as simple as:

```
#!/bin/sh
uf101d start
```

On some systems there is a pre-installed script called "/etc/init.d/rc.local" that runs at boot time. You can place the line 'uf101d start' in this script.

Shared Library Dependencies

The install script will attempt to create symlinks to shared libraries that it depends on. If this fails for some reason, you can follow these steps:

Two optional features require access to shared libraries on Linux, to support SSL and Zlib compression. These are loaded as libssl.so, libcrypto.so, and libz.so at runtime whenever they are required. In some

cases, these generic names might not be defined in the system's lib directories. If that is the case, then you can define symbolic links with these names to the correct local paths.

For example, if the installed and available name for libssl.so is /lib/libssl.so.1.0.1, you can create a symbolic link like this:

```
ln -s /lib/libssl.so.1.0.1 /usr/lib/libssl.so
```

You will likely have the same requirement for the related libcrypto.so file:

```
ln -s /lib/libcrypto.so.1.0.1 /usr/lib/libcrypto.so
```

Verify each of the three .so files are present or links to local versions: libssl.so, libcrypto.so, and libz.so. Note that if UnForm is a 64-bit version, you will need to perform this step for 64-bit libraries, typically found in /usr/lib64.

Custom Launch Settings

The /usr/bin/uf101d script will look for the file uf101d.include in the UnForm home directory. If found, its lines will be included in the script with ". uf101d.include". Use this feature to set up custom environment variables or perform other initialization tasks. The \$n variables contain the arguments passed to the script, such as 'start' or 'stop'.

2.1.2 Standalone Client

The uf101c client software can be used to submit jobs to UnForm from anywhere on your network after the server is installed and operating. The client software is automatically installed on the same machine as the server, so jobs can be submitted locally. However, you can install the client software on any network computer. Any client can talk to any server, so you can mix and match different operating systems as you need. For example, you could install the Windows server, and have both Windows and UNIX clients submit jobs to it.

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

The UNIX client is installed from the file uf101c_tar.Z, while the Windows client installer is called uf101c_setup.exe.

The UNIX install steps are as follows:

- Ensure the system has Perl level 5.6 or higher: **perl -v**
If not, Perl can be obtained from <http://perl.com> or <http://cpan.org>.
- Create a directory for the client, such as **mkdir /opt/uf101c**
- Set permissions on that directory: **chmod 777 /opt/uf101c**
- Copy the uf101c_tar.Z file to that directory and **cd** to that directory
- Uncompress the file: **uncompress uf101c_tar.Z**. If you have gzip, then the gunzip utility can also uncompress the file.
- Extract the files: **tar xvf uf101c_tar**
- Run the setup script: **./ufcsetup.sh**
- Edit the uf101c.ini file located in the install path to set up the [client configuration](#).

2.1.3 Uninstall

Before uninstalling, be sure to copy any important files, such as rule files, archive libraries, and images. Every installation is unique, so be sure to identify and copy all such files before removing the installation directory.

To uninstall UnForm on a Unix or Linux system:

You can run `./uninstall.sh` as root from the UnForm installation directory, or perform these steps:

- Stop the server if it is running (`uf101d stop`)
- Remove the install directory (`rm -rf /usr/local/uniform` for example)
- Remove the `/usr/bin/uf101c` and `/usr/bin/uf101d` scripts
- If the server is being started automatically by an entry in `/etc/init.d`, `/etc/rc*.d`, `/etc/inittab`, or `systemd`, remove such entries.

Note the original install process may have installed third party tools, such as Image Magick. These tools are not automatically removed by the `uninstall.sh` script.

2.2 WINDOWS INSTALLATION

2.2.1 Server Installation

The method used to install the server on Windows depends on the type of system.

- **Desktop systems (i.e. Windows 7)**

Simply execute the downloaded setup executable.

- **2008 and higher, *without* Terminal Services**

Right-click the setup executable, and 'Run As Administrator'.

- **2008 and higher *with* Terminal Services**

Use Control Panel, Install Application on Terminal Server, and execute the setup executable. Later versions refer to "Terminal Server" and "Remote Desktop Services".

Follow the on-screen prompts from the installer to install UnForm to your system. This will install both the server program and a local client (`uf101c.exe`) program. The client program and its associated support files will be installed in both the server directory and in the Windows System directory, enabling a command line launch without a full path, as the Windows System directory is generally included in the PATH environment variable.

This will also install shortcuts to the server manager and documentation, under the Start, Programs, UnForm 10.1 Server menu. Run the [server manager](#) to activate and operate the server.

Permissions

Often, a local UnForm client (`uf101c.exe`) is used to submit jobs to the UnForm server. When the client connects to the server "localhost", it knows it is running on the same system, and will copy submission files, and when needed retrieve output files, directly to and from the server's "temp" sub-directory. The

installer will attempt to grant the special user "Everyone" full rights to that directory, but if this fails, and administrator can add this access manually.

Client Path Access

The local client is installed in the server folder, and an attempt is also made to copy the uf101c*. * files to the System32 and SysWow64 directories to enable path-free execution. However, on some systems this does not work for a setup script, so you can take one of several approaches:

- Use full paths to the uf101c.exe executable
- Copy uf101c*. * from an existing path to the Windows\System32 directory
- Edit the system PATH environment variable to include a directory where the uf101c*. * files reside

Bundled Apache HTTPD Server

The Windows installation includes Apache. Apache for Windows is compiled using a recent version of Microsoft Visual Studio, and the Apache binaries require a VC runtime of a compatible level. That runtime redistributable installer is included in the UnForm server's httpd directory. If Apache (httpd.exe) fails to start on your system, it is likely that this redistributable needs to be installed by an Administrator.

2.2.2 UnForm 10.1 Manager

The Windows Server Manager (distinct from the browser-based [Server Manager](#)) is used to activate and operate the UnForm 10.1 server on a Windows machine. It has a tabs for activation and to see a view of server tasks that are running, and a toolbar to operate the server.

The first two toolbar buttons are used to start and stop the server. If it is installed as a service, and you are not running as an administrator, these will be disabled.

The second two buttons are used to install or uninstall the UnForm server as a service. You must be an administrator or these buttons will be disabled. It is generally preferable to install the server as a service, as it will then not display tasks on the desktop, and will remain running when no user is logged in. However, you should not install as a service until you've confirmed the server starts successfully in application mode, as startup error messages are then visible.

The last two buttons provide access to the browser interface and the documentation.

Process	Started	Information
5244	5/25/2020 7:42:47 AM	uf100drun.exe ufrp.pv (00:00:00.0937500)
1616	5/25/2020 7:42:47 AM	uf100drun.exe ufpvmon.pv (00:00:00.1093750)
7132	5/25/2020 7:42:47 AM	uf100drun.exe ufinbmon.pv (00:00:00.1406250)
7776	5/25/2020 7:42:45 AM	uf100drun.exe sdsi\uf100\server\uf100drun.exe -s unifom_10.0 (00:00:00.4062500)
2784	5/25/2020 7:42:47 AM	uf100drun.exe ufinbmon.pv (00:00:00.1250000)
1664	5/25/2020 7:42:47 AM	httpd.exe -f "c:\sdsi\uf100\server\apache.conf" -e "c:\sdsi\uf100\server\temp\tmp\w...
2892	5/25/2020 7:42:49 AM	httpd.exe -d c:\sdsi\uf100\server\httpd -f c:\sdsi\uf100\server\apache.conf -e c:\sdsi...
636	5/25/2020 7:42:47 AM	uf100ss.exe (00:00:02.9531250)

2.2.3 Standalone Client

On Windows, the installation steps are:

- Run the **uf101c_setup.exe** installer program.
- Run the **Configure UnForm Client** option from the Start menu, which starts Notepad with the uf101c.ini file. Enter the appropriate values for the server and port, and optionally the log file.

On Windows, the uf101c.ini file is stored in the %PROGRAMDATA%\SDSI folder. Typically, this would be C:\ProgramData\SDSI\uf101c.ini. This file can be [edited directly](#).

The client is composed of several files, one of which is uf101c.exe. This program is installed both in the installation path and the system Windows path (typically C:\Windows) to enable execution without a full path. The uf101c.exe program will look for the install path setting in the uf101c.ini file, in order to locate the addition files.

Note that uf101c.exe is a small .NET program that operates the uf101cc.exe console client behind the scenes. You can also run the console client directly from a Windows command line. The client is located in the install path as uf101cc\uf101cc.exe.

Unbundled Perl Client

Included with the Windows client install is the standard uf101c.pl script included with UnForm on Linux. This script can be run on Windows if you install a Perl interpreter, such as ActiveState community edition, or Strawberry Perl. The uf101c.exe program will automatically look for one of these in standard install

paths, or you can update the uf101c.ini file to include the perl=*path\to\perl.exe*. You can also run a perl command line from console mode:

path_to_perl.exe path_to_uf101c.pl followed by standard client command line arguments.

2.2.4 Uninstall

Use Control Panel, Programs and Features, select the UnForm 10.1 Server, and click Uninstall.

This will attempt to stop the server, uninstall the service, remove Start menu entries, and finally remove most directories and files from the install directory. Certain data files, such as archive libraries stored in the "arc" folder, are left behind to avoid accidental deletion of potentially valuable data. These files can be removed manually when ready.

2.3 LICENSING

UnForm is licensed based on the number of concurrent jobs it can process, with counts available as 1, 3, 5, 10, and on up to unlimited.

Additional licensed components include archiving, design tool, image manager, and docflow.

Licensing is controlled entirely by the server process, uf101d. You can install the uf101c client programs freely anywhere on your network.

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

There are up to three activation keys that must be entered for full operation of UnForm: two system keys, and an UnForm key. The system keys enable UnForm to operate on a specific computer. The UnForm key enables counters and features that have been purchased. For demo mode operation, just system keys are required; demo mode operation automatically enables 3 print jobs, 1 Image Manager, 3 work flow users, and both archiving and the design tool.

There are three types of system activations:

21-Day Evaluation

This license has a fixed serial number (UF0799999) and can run on any machine for 21 days. While running under this serial number, UnForm will print "UnForm Evaluation Version" phrases on any enhanced output. This is the first mode activated after an installation, as it enables the retrieval of a System ID and Machine Class needed for permanent licensing later, as well as allowing UnForm to operate in evaluation mode.

This mode also restricts data files, such as archive libraries and job history, to 5,000 records. Larger files can be read, but not written to, until a non-demo serial number has been activated in either permanent or emergency temp mode.

If you need to extend this 21 day period, you can re-license it as an evaluation one time, or uninstall and remove the software and reinstall it. If you remove it, be sure to save copies of any custom rule files, and any files they depend on, like images.

Permanent

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

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

Emergency Temporary

This license is assigned a serial number, like a permanent license, but it does not require a System ID or Machine Class to activate. This allows you to re-install UnForm on a different machine than originally licensed, and operate it for 21 days.

Rental Licensing

If UnForm is rented, it will regularly contact the synergetic-data.com website for updates to the UnForm key for your serial number, which includes an expiration date. The expiration date can be renewed to allow UnForm to continue running. If the expiration date expires, there is a 10 day grace period during which the system will continue to run, and after that, the system will stop running. To ensure continuous operation, the system will attempt to email two addresses during the grace period. One address is defined as "`expnotify=address`" in `uf101d.ini`. The other is sales@synergetic-data.com. The expiration messages, plus any email errors, are logged to the server log file(s). It is critical that the system running a rental version of UnForm can connect to the internet to ensure uninterrupted operation.

A rental license requires a permanent or emergency temporary system activation to enable the runtime engine to operate.

UNIX Licensing

To activate UnForm on UNIX, perform the following steps:

- Login as root.
- **cd** to the UnForm directory (i.e. `cd /usr/lib/sdsi/uf101`).
- Execute **./license.sh**.

The `license.sh` script prompts for the following options:

```
UNIFORM LICENSING OPTIONS
```

```
Use the following options if this machine is connected to the
Internet:
```

```
-----
-
1 - Permanent Activation (requires serial number and PIN code)
2 - Emergency Temporary Activation (also requires SN and PIN)
3 - 21-Day Eval Mode Activation (does not require Internet access)
```

```
Use the following options for manual activation.  Activation keys
```

can be obtained from <http://unform.com/uf10lic.cgi>.

```
-----  
4 - Display System ID and machine class (needed for option 5)  
5 - Enter Permanent Activation  
6 - Enter Emergency Temporary Activation  
  
i - Initialize activation file (recover from dead/expired activation)  
  
q - quit  
Enter selection:
```

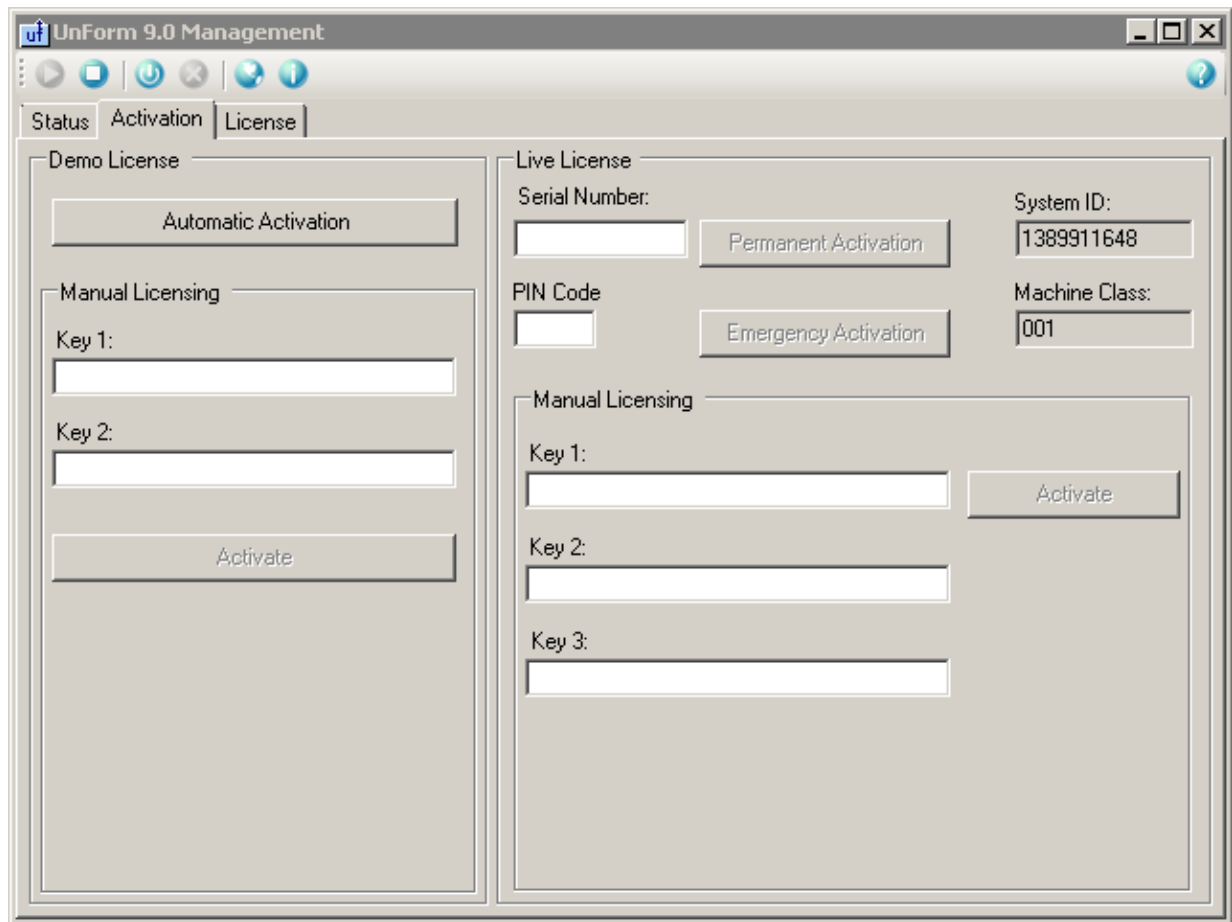
To obtain either a permanent or emergency temporary activation, you will need to know your serial number and PIN code previously assigned by SDSI. These values are not necessary to obtain 21-day eval mode activation.

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

If you had a temporary activation that expired, you might need to initialize the activation to enable access to the other license steps, some of which themselves require an operational runtime engine. In this case, you can choose the "i" option, which initializes the activation file to a limited mode, followed by the desired standard licensing step.

If the Internet is not available from the install machine, then you can perform activation manually by using another machine to visit <https://unform.com/uf10lic.cgi>. Use option 4 to display the System ID and Machine Class, which will be required to obtain a permanent activation key from this web site. Options 5 and 6 will prompt for a serial number and activation keys.

Windows Licensing



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

To activate evaluation mode:

Evaluation mode activation is automatic when UnForm is first installed. If you need to extend the evaluation period one time, you can click the Automatic Activation button to re-activate the evaluation period.

Once activated, you should be able to click the Start Server button on the toolbar, and the Status tab should show process activity.

To activate permanent mode:

To activate automatically over the Internet, verify the System ID and Machine Class fields contain values. If not, activate the product in eval mode first. Then fill in your serial number and PIN code, and click the Permanent Activation button. This will use your information to obtain a permanent activation key for the system and activate all the features of your license.

To activate UnForm manually, note your System ID and Machine Class, then go to <https://unform.com/uf10lic.cgi>. Enter your serial number and PIN code, then click the button to get a permanent license. When prompted, enter the System ID and Machine Class exactly as noted on this screen. Note the three activation keys returned, and enter them exactly as provided in the three entry fields, then click the Activate button.

To activate in emergency temporary mode:

To obtain a temporary activation over the Internet or manually, follow the steps for a permanent license, but click the Emergency Activation button. The System ID and Machine Class are not used for temporary activations.

Activation Errors

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

In the meantime, you can obtain an emergency temporary activation to allow your serial number to be used on a new installation for 21 days.

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

2.4 HTTP SERVER PORTAL

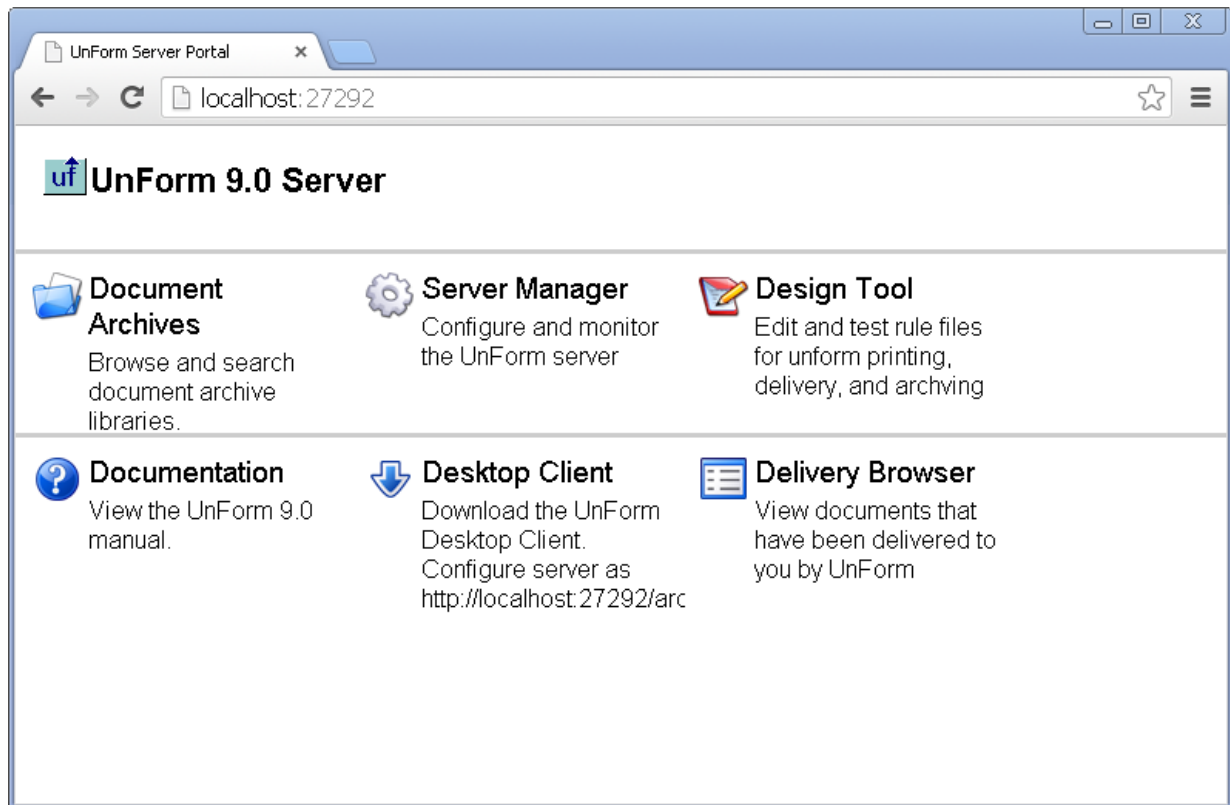
The UnForm server manages the operation of an Apache web server, which provides browser access for server management and document libraries when archiving is enabled. On Windows, this server is bundled with the server install. On Unix, there must be an Apache server available on the system. Apache is generally pre-installed on any Unix or Linux system, and is readily available from the operating system vendor if it is not already installed.

When installing on Unix systems, the ufsetup.sh script will look for the Apache server at several standard locations. If not found, it will prompt for a path to the httpd executable and also the Apache home directory where its 'modules' directory is located.

When the UnForm server is started, it in turn creates an Apache configuration file (apache.conf) based on configuration settings defined in uf101d.ini, and starts a private instance of the Apache web server, by default listening on the port 27402. To access this server, open a browser to a URL like this:

<http://192.168.1.10:27402>

This will open the UnForm 10.1 portal page on the address specified, which provides links to other browser-based features of the UnForm server.



You can access certain features directly, without the portal page:

- <http://192.168.1.10:27402/arc> for archive library access
- <http://192.168.1.10:27402/arc?sm=1> for server manager access
- <http://192.168.1.10:27402/arc?dsn=1> for design tool access

Login Access

Most options require a login. Logins consist of a user ID and password, and are defined in the [Administrator options](#) of the archive browser interface, which are available when logged in as an administrative user. When the system is initially installed, there will be one administrative user pre-defined, user ID of "admin", password of "admin". Use this login to manage this and other users.

External Access

Often, the UnForm server is behind a router in a private address space. To access it from the internet requires configuring the router to forward a router port to the UnForm server address and port. The router's firewall may also require configuration to allow connections to that port.

It is also possible to configure an external web server to provide access via a reverse proxy mechanism. For example, an Apache web server located in the cloud could be configured to router a certain subdomain to an UnForm server behind a router, using the Apache ProxyPass directive. Here is a sample httpd.conf section:

```
<VirtualHost 11.11.11.11:80>
    ServerName archives.mycompany.com
    ProxyPass / http://99.99.99.99:27402/
</VirtualHost>
```

The above would create a virtual host accessed by the server name specified, archives.mycompany.com. The ProxyPass directive causes Apache to retrieve all files from the 99.99.99.99 site, which represents the public address of the network where the UnForm server is installed, where port 27402 is forwarded by the router to the UnForm server machine in its local network.

Under an IIS web server, you can create a new site that uses a universal URL Rewrite feature that defines a reverse proxy inbound rule to the public address and port of the UnForm web server. Access to the IIS site retrieves content from the UnForm server behind the router.

See your web server documentation for reverse proxy configuration.

2.5 FIREWALL CONFIGURATION

It is common for a firewall to block access to any of the ports that the UnForm server or Apache HTTP server listen on. For ports, all that is needed is to open the two listening ports, which default to 27400 and 27402. If you use a standalone Windows Support Server, that by default listens on port 27401, so that would need to be configured to allow the UnForm server to connect to that port on the Support Server machine.

In addition, some firewalls block programs as well as ports. The programs are uf101drun and httpd. On Windows, both programs will be in the UnForm installation path under "rt", and be named "PxPlus" for the uf101drun.exe program, and "Apache HTTP Server" for the httpd.exe program. On Unix, the local http server is used, rather than one inside the UnForm path, but the uf101drun program is found in the UnForm path under "rt".

2.6 UPDATING AN EXISTING INSTALL

The publisher releases frequent updates to UnForm. To install an update (i.e. updating from 10.1.08 to 10.1.09), the process is fast and simple: just install the latest version into the current installation location. This will overwrite the UnForm program files and samples, but will *not* overwrite any rule files or configuration or data files, and will *not* change the system ID, which would otherwise require re-licensing.

After downloading the update install file, stop the UnForm server, install the software over the current installation, and start the UnForm server.

Be sure to follow the normal installation flow for your platform, as described in [Linux Installation](#) or [Windows Installation](#). The following is a summary.

Linux

You must be super user (root), and must change to the UnForm directory before extracting the tar file. Run 'uf101d stop', extract the tar file 'tar xvf /path/to/tarfile', and start the server with 'uf101d start'.

There is no need to re-run ufsetup.sh or license.sh.

Windows

First check if you are running remote desktop services by looking for an entry in Control Panel "Install Application on Remote Desktop Services". If that entry exists, use it to run the uf101d_setup.exe. If it does not, then right-click the setup program and Run as Administrator. Stop the UnForm service, and

close the UnForm 10.1 Manager, if open. Make sure no uf101drun.exe processes are running (note that stopping the service does not stop running print jobs automatically). Be sure to identify the correct current installation folder, and install the update to that same folder. The default is C:\SDS\uf10\server, but that is sometimes changed to place UnForm on a different disk.

There is no need to re-activate.

Client Updates

The UnForm client is rarely updated, and if it is modified there will be a mention of that in the release notes. If the client is updated, you do not need to update the client included with the UnForm server. It gets updated automatically by the server install. However, if you have a client installed on a remote system, that can be updated in similar fashion, by running the client installer to the same install location.

Please note, this procedure is NOT for upgrading the main version you are running, such as from 10.0 to 10.1. That requires a new installation rather than an in-place update.

2.7 UPGRADING FROM 10.0

Here are the things to think about when upgrading from 10.0 to 10.1.

- The two versions cannot share inbound sources, which are monitored paths and IMAP email accounts
- The two versions cannot share TCP ports that can be used to accept raw print jobs as configured in [tcpports] in uf100d.ini/uf101d.ini
- The two versions cannot share the same service listening ports for client submissions or http access

In summary, it is usually not possible to run 10.0 and 10.1 at the same time on the same system, or at all if you use an email-based inbound source, so in general you will need to perform the upgrade with 10.0 stopped and disabled. Here are the steps.

Obtain an UnForm 10.1 Serial Number

Unless you are treating this as a new test install, you should have your 10.1 license available to minimize down time. Obtain this license by contacting sales@synergetic-data.com.

Disable UnForm 10.0

- On Windows, stop and disable the UnForm 10.0 service
- On Linux, run `uf100d stopall` and check for the system startup configuration to disable auto startup:

if 'which systemctl' returns a name, run 'systemctl disable uf100d'

if 'which chkconfig' returns a name, run 'chkconfig --del uf100d'

Optionally, you can uninstall 10.0, using `uninstall.sh` on Linux, or with Settings or Control Panel on Windows. This process removes the UnForm software, but not the rule files or configuration files. Any files you will need to copy to 10.1 are left in place.

Install 10.1

Follow the standard installation steps for [Linux](#) or [Windows](#). This can be to a new directory, or on top of the 10.0 directory. If using the same directory, you won't need to copy many of the required files, but

there may be confusion over the uf100* and uf101* files both in the path. You can choose to remove the uf100* files after the new install is fully configured.

If a new directory, copy these files from the 10.0 directory:

- ufarc*.dat
- ufcgi*.dat
- ufjobdef.dat
- ufoauth.dat
- ufsched.dat
- rule files
- any files used by the rule files, unless they are full paths, such as image files

Copy uf100d.ini to uf101d.ini, and uf100c.ini to uf101c.ini. Note on Windows uf101c.ini is found in C:\ProgramData\SDSI. This will maintain any custom settings you have from 10.0, but also note this will cause 10.1 to use the 10.0 listening ports for client submission and http access. If you would rather use standard 10.1 ports, edit uf101.ini and set port=27400 in [defaults] and port=27402 in [apache], and edit uf101c.ini and set port=27400 in [defaults]. Also note a standalone client on another system also uses uf101c.ini and should be edited as necessary to match the [defaults] port in use by the server.

If using the default 10.1 ports, be sure to update any http access links, and add firewall rules to allow access to the new ports. Also adjust any application functions that use the uf100c client to use uf101c.

Archive Libraries

If a new directory and you use UnForm archiving, and use non-full path library names in rule files or image manager jobs, the 10.0 libraries will need to be moved. Libraries from the 10.0 "arc" directory should be moved to the 10.1 "arc" directory. Since these are referenced as full paths in the library system, you will need to rename them in the archive browser interface Admin, Libraries option once UnForm 10.1 is started.

Activate 10.1

Using your 10.1 serial number and pin, activate this installation. This is important if you've copied live files to 10.1, as there are record count limits in the demo license and it is very likely those will have been exceeded in some of the 10.0 files copied over.

Start up 10.1, and run uf101c -v to verify local access, and verify access to the http port with your browser.

If a new directory was used, now you can rename the library paths from the 10.0 arc path to the 10.1 arc path, using the archive browser Admin, Libraries option.

3 CONFIGURATION

3.1 CONFIGURING THE SERVER

The server is configured via the uf101d.ini file, which can be edited with any text editor. In addition, the browser-based [Server Manager](#) offers configuration access to most configuration parameters.

In addition to these items, you can also configure access to [Ghostscript](#), [Image Magick](#), or [Image Alchemy](#) elsewhere in the uf101d.ini file.

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

[defaults] section	
port= <i>n</i>	Sets the primary listing TCP/IP port to <i>n</i> . The default is 27400. Note that if you use NAT translation or if you have a firewall between the clients and server, then this port (along with the proports defined below) must be configured to allow clients access.
logfile= <i>path</i>	Sets the name of the server's log file to <i>path</i> . By default, it is stored in the UnForm directory. Standard log entries include connection information. Detailed logging includes verbose data transactions.
logdetail= <i>n</i>	Set <i>n</i> to 0 for standard logging, 1 for detailed logging. You should not leave detailed logging enabled for normal use, as the log file can grow very large.
timeout= <i>n</i>	Set <i>n</i> to the number of seconds that a connection can remain idle before closing. The default value is 3600, or one hour. Setting this value to 0 will avoid timeout-based disconnects. This value primarily affects designer connections, which can remain active for long periods.
age= <i>days</i>	<p>This value sets the maximum age, in days, of job log entries. When jobs are submitted, basic job information is kept in a log file. If errors were recorded, the error file also remains in the temp directory under the UnForm server. After this many days, the files and log entries are automatically removed. A fraction of a day can be supplied, such as age=.25 for 6 hours.</p> <p>This value is also used for purging unprocessed jobs in the 'rpq' directory, which receives incoming TCP/IP print streams as well as asynchronous client submissions.</p> <p>This value is also used for purging search result sets generated by the archive browser interface.</p>
agetmp= <i>days</i>	This value sets the maximum age, in days, of ./temp/tmp files, which is the default directory for work files. A fraction of a day can be supplied, such as age=.25 for 6 hours.
rulefile= <i>path</i>	Sets the default rule file to <i>path</i> , used for jobs that do not specify a rule file on the command line.
bbpath= <i>path</i>	If the bbxread() function is used, this value points to the BBx executable that is invoked when required, such as /usr/lib/basis/pro5/pro5.
library= <i>path1;path2;...</i>	Sets directory paths that are automatically searched for rule files, images, and attachments. By default, UnForm searches the UnForm directory and also supports full paths.
sshhost= <i>host</i>	Sets the default host IP address or name of the Windows Support Server. In addition, the sshhost() function can be used in a code block to specify the host and port at run-time.

<code>ssport=port</code>	Sets the default port to connect to the Windows Support Server on the host specified by <code>sshost</code> .
<code>sstimeout=seconds</code>	Sets the Support Server timeout value, which determines how long an UnForm job will wait for a response from the Support Server before logging an error and continuing the job. Set to -1 for an infinite timeout.
<code>imageage=days</code>	Images that are converted by an external conversion program or by the Windows Support Server are cached by default. The last date an image is used is also stored, and images that have not been used in <i>days</i> days are removed automatically.
<code>stylesheet=name</code>	Sets the name of the style sheet used by the archive browser interface programs. A file called "default.css" is provided with the server installation (found in the <code>web/en-us</code> directory). This style sheet is also used when archives are exported to static HTML structures.
<code>bufsize=bytes</code>	An initial block is tested for each job in order to determine if the job contains binary data or text data. The size of this block defaults to 8196 bytes, but you can adjust it to any integer value with this entry. The minimum value is 1024.
<code>cr=0 1 2 3</code>	Controls default handling for embedded carriage return (<code>chr(13)</code>) characters in lines read from the input stream. This value may be overridden with the <code>-cr</code> command line option. <ul style="list-style-type: none"> • 0 will truncate lines at the first CR. • 1 will strip CR character, so the line continues as if the character were not present. • 2 will fold lines, and non-space characters are placed in the line buffer, simulating an overstrike. • 3 will fold lines and insert an extra space, which accommodates Windows Generic/Text Only printers that overstrike conflicting characters.
<code>rebuild=0 1</code>	If set to 1, the next start of the <code>uf101d</code> server will attempt to repair certain control files, such as the job history database and user table. Use this feature if you suspect corruption in one of these files. It should normally be set to 0.
<code>tcpportretry=n</code>	<p>The number of times a job received on a direct TCP/IP port will be retried if a non-license (998) error occurs. As an example, if a network printer goes down and UnForm returns errors trying to open the output device (<code>-o devicename</code>), this sets the maximum number of times the job will be submitted by the port sweeper. The sweeper runs each time a job is submitted and every 5 seconds when idle.</p> <p>Setting this value to a reasonable number allows for temporary problems to be self-corrected without causing an unlimited number of log and error files to build up due to a configuration issue.</p> <p>To release jobs once a problem is corrected, manually remove the <code>*.rty</code> file(s) from the <code>rpq</code> directory.</p>

pdftrans=0 1	Sets the default PDF transparency setting. If 1, then PDF files will use transparency.
textjob=0 1	Sets the default behavior on generation of the textjob\$[all] array, which is a collection of all print lines for the job. This array can be useful when performing report mining operations, or parsing a full job into pages in a prejob code block, but when large print streams are processed, a significant amount of memory and CPU resources are consumed generating the array. The -textjob, -notextjob command line options override this setting.
errnotify= <i>email address</i> errnotifysubject= <i>subject</i>	<p>If an error occurs while running a job, messages are written to a job number error file (temp/jobno.err). This file remains on disk for a configured amount of time, typically seven days. You can configure an email address (or multiple emails separated by commas) to have the server send the contents of these error files to an administrator, reducing the need to proactively monitor for errors.</p> <p>Only jobs that encounter runtime errors trigger mailing. Errors in jobs being tested in the design tool are not sent. Also, some errors can occur only on the client side, such as an invalid server address. In those cases, the server is unaware of the error and no message will be sent. Such errors need to be captured via client error handling.</p> <p>The <i>subject</i> specified may include a tag "@jobid" to reference the job number, which can be helpful to cross reference back to a rule file and rule set by locating the "Job complete" message in the server log file.</p> <p>If an error occurs while emailing, a message is logged in the server's log file (logs/server.date.csv). This feature depends on having email sending properly configured.</p>
expnotify= <i>email address</i>	If UnForm is licensed as a rental, then this address is used to notify an administrator if the license expires. Rental licenses have a 10 day grace period, and during that period expiration messages are logged, plus an email is sent to this address and sales@synergetic-data.com, as long as an email server is configured in the server manager.
ssl=0 1	<p>If set to 1, the server listens on a secure (SSL) socket. All client connections, including Image Manager connections, must be configured to connect using SSL. The uf101c command line can also contain a -ssl option to connect using SSL.</p> <p>If this is set to 1, and the [apache] section has sslcert and sslkey values defined, then the certificate configuration used for Apache is also used by the client listener. Otherwise, a self-signed certificate is used.</p>
dtrest=0 1	The desktop delivery browser normally monitors for new activity using a low-overhead HTTP request to the regular printing port, rather than the HTTP port. If ssl is turned on and the default self-signed certificate is used, then desktop deliver can encounter problems due

	<p>to the delivery browser interface not accepting the certificate for the printing port.</p> <p>In such a case, you can set <code>dtrest=1</code> to force the delivery browser to use the REST interface with the standard HTTP port. This can help in cases where the browser interface runs without ssl. The REST interface is not as efficient as the above method, but is suitable for modest numbers of delivery browsers actively polling the server.</p>
<code>afo2=0 1</code>	Default to the alternate AFO word parsing algorithm.
<code>afogsorder=0 1 2</code>	<p>If set to 1, AFO jobs and Image Manager OCR parsing relies on the word order provided by Ghostscript or Mutool, rather than attempting to sort words into top-down, left-to-right order.</p> <p>If set to 2, AFO jobs and Image Manager OCR parsing uses a center-range algorithm to allocate rows, rather than the default bottom-rounding algorithm.</p>
<code>rpqinterval=seconds</code>	Sets the interval used by the server to check for rpq job submissions, which come in via the tcp/ip monitor or jobs submitted by uf101c with the <code>-async</code> option. At each interval, the server checks to see if there are rpq jobs and runs the sweeper if any jobs are present and the sweeper is not currently running. The minimum and default value is 1.
<code>rpqmaxjobs=count</code>	When releasing jobs from rpq, the sweeper stops after this many jobs have been successfully submitted. Other jobs wait in the queue for the next sweep interval. This setting, along with <code>rpqinterval</code> , can be used to adjust how large numbers of jobs are released from the queue without overwhelming a system. It is particularly useful with large or unlimited licenses. If not set, all available jobs are processed.
<code>fitpage=0 1</code>	Sets the default fitpage mode for auto-scaling of AFO input streams. See <code>-fitpage</code> in command line options.
<code>imjobage=days</code>	Days to retain image manager job history logs.
<code>histage=days</code>	Days to retain history/* data, which stores inbound documents received by email and path monitors, and print files. If set to 0, no history is created.
<code>syncfrom=server[:port[:authkey[:proxy]]]</code>	<p>If set, this instance of UnForm will attempt to sync configured files from the server specified to the local system. The purpose is to enable a primary server to provide rule files, image files, etc. to child servers in a load balancing or auto-failover setup, avoiding manual file management. The process uses a uf101c client, so the server and port should specify the print client port rather than the http port (default is 27400).</p> <p>On the primary server, there is a companion section in its uf101d.ini file, <code>[syncfiles]</code>, that lists files or file wildcards.</p>

<code>syncinterval=minutes</code>	Determines how often files are pulled from the 'syncfrom' server.
<code>syncdelto=server[:port[:authkey[:proxy]]]</code>	If set, this instance of UnForm will attempt to sync its delivery logs with the target server. The logs will be named based on this machine's host name: <code>deliver.yyyymmdd.hostname.csv</code> .
<code>syncdelinterval=minutes</code>	Determines how often deliver log files are updated on the target server.
<code>inboundmon=count</code>	The number of inbound monitor tasks to start. Each task monitors all inbound sources for new documents to parse.
<code>syslibage=days</code>	This setting, if <i>days</i> is positive, will force purging of syslib library recover files older than this many days are removed. This helps to keep disk usage down. Syslib libraries are those used by Image Manager and DocFlow, for documents being managed by those modules. Since these libraries are generally used to manage transient documents, there is no need to keep recover files older than the oldest active document, and those are purged automatically. This setting goes further: even if the library contains documents older than this, the recover files are still purged, so a library rebuild, if ever necessary, would not include such documents.
<code>groupid=name</code>	On Linux, UnForm is usually configured to run as a user other than root. This user is shown in the server log, and can be found as the UFUSER setting in <code>/usr/bin/uf101d</code> . When UnForm is launched by root, such as at system boot time, it will attempt to switch both its user and group to this UFUSER name. If you require a different group, and the user is a member of that group, you can define this setting, and UnForm will attempt to set the group to this value before switching the user.
<code>debug=0 1</code>	Set <code>debug=1</code> to turn on print job debug mode by default (the same as the <code>-debug uf101c</code> option). To override this at a job level, use the <code>-nodebug uf101c</code> option.
<code>noirs=0 1</code>	Set <code>noirs=1</code> to globally turn off inline rule sets.
<code>deadproc=seconds</code>	When a client connection is initiated for job submission, a process is started by the server to handle that connection. If for any reason that process fails to start, the client may hang waiting for a server response. Set this value to some number of seconds that allows for the process to start, no less than 10 seconds. The client will receive an error 1058 and an error will be logged at the server if this situation is encountered. It normally means the server is out of some resource.
[security] section	
<code>allow=list</code>	This is a semi-colon delimited list of valid IP addresses or wildcards that are allowed to connect to the server. Note that the loopback address 127.0.0.1 is always allowed to connect. The default list is <code>192.*.*;10.*.*</code> , which allows the two standard non-routable LAN spaces to work.
<code>encryptrul=0 1</code>	If set to 1, the Design Tool will will encrypt rule files when saved. The login is restricted to a user who is either an administrator or has

	<p>been granted Design Tool access via the user maintenance features of the archive web browser interface.</p> <p>Rule files saved or published in encrypted format can be read by any UnForm 10.1 server, regardless of its designer security setting, but can only be edited by the design tool.</p> <p>If set to 0, rule files are maintained as text files, which can be edited using any text editor as well as the design tool.</p> <p>Note this setting replaces the 'designer=<i>n</i>' option of previous releases, which also forced a login from the design tool. The design tool in version 10.1 is browser-based and always requires a login.</p>
<code>authkey=string</code>	<p>This establishes a shared-secret value between the server and client(s) on the standard (non-http) port. Client connects must be configured in uf101c.ini or via the uf101c command line to specify a matching authkey value. If connections arrive with a missing or mismatched value, an error 996 results.</p>
<code>password=0 1 2</code>	<p>Sets a minimum password complexity level when updating user records. This does not apply to LDAP/AD provided user records, and is enforced only when user records are saved. Existing user records are not affected.</p> <p>0=No complexity requirement, and passwords are entirely up to the user defining them.</p> <p>1=Minimum 8 characters, with at least one upper-case letter, lower-case letter, digit, and special character present.</p> <p>2=Minimum 14 characters, with at least one upper-case letter, lower-case letter, digit, and special character present.</p> <p>Special characters are one of: ! @ # \$ % ^ & * () - + = , . : ; /</p> <p>A complex password can be generated from the user maintenance window.</p>
[tcpports] section	
<code>port=options</code>	<p>Each line defines a port on which the server listens for raw print job deliveries, such as from Windows TCP/IP ports. Each job submission is then processed using a uf101c command line configured with a pre-defined -ix option plus any other <i>options</i> defined. For more information, see the TCP/IP Monitor chapter.</p>
[webprinters] section	
<code>name=options [;description]</code>	<p>Each line defines a printer available to the Web Extension, which can submit PDF files to the printer. This enables AFO printing directly from a compatible web browser, without the need to set up special printers at each browser machine.</p>

	When PDF files are submitted, a uf101c command line is run with the options specified, such as <code>-f rulefile -o "device"</code> . An automatic <code>-ix</code> option is generated using the submitted file. The web extension displays the device using the name and optional description.
[archive] section	
<code>deflib=defaultlib</code>	Sets the default library name, for use when archive commands do not specify a library name. This library will be placed under the default "arc" subdirectory below the UnForm server.
<code>keywords=n</code>	Specifies the maximum number of default keywords generated for UnForm job-based archives. Default keywords are unique words generated from the job input stream that do not match patterns defined in the <code>nonwords=</code> file. If this value is set to -1, then all unique words become keywords. The benefit of this is that more words of job print streams are available for searching. The cost is greater time spent parsing reports for words and additional disk space utilization.
<code>nonwords=file</code>	Specifies a file which contains lines of regular expressions for "words" that should not become keywords. See <code>ufnonwords.txt</code> for examples.
<code>nonchars=charlist</code>	This is a list of characters that are removed from keywords. The default list provided with UnForm is: <code><>{}[]()*~`"' </code>
<code>endchars=charlist</code>	This is a list of characters that are removed from the end of keywords. For example, you may want to remove periods from the ends of words as a period typically ends a sentence. The default list provided with UnForm is: <code>?!.,;</code>
<code>searchage=days</code>	When archive searches are performed in the Web browser interface, work files are generated. This sets the maximum number of days these files will remain on disk.
<code>webdirs=dir1;dir2;...</code>	If you need to support multiple languages, or you wish to offer a customized user interface for archive browser users, you can copy the <code>./web/en-us</code> directory to other <code>./web/*</code> directories and customize them. In particular, the <code>messages.txt</code> file and various html templates or style sheets can be customized. This directory list (and associated <code>name=title</code> values in each <code>messages.txt</code> file) are presented in the browser login screen.
<code>sesage=hours</code>	Set the number of hours a browser session can last before a login is required again. By setting this to 0, browser users must login each time their web browser is re-started and the web interface is accessed. Set it to a large number to allow users to login once per workstation and have that login remembered.
<code>defperm=perms</code>	Sets the default permissions on new libraries. Set to zero or more semi-colon delimited letters, <code>r</code> , <code>w</code> , and <code>d</code> for read, write, and delete. For example, <code>defperm=r;w</code> for default read and write, or <code>defperm=r</code> for just read only, or <code>defperm=</code> for no default permission, meaning only administrator logins can access the library initially.
<code>defseq=0 or 1</code>	Sets the default Force Sequence on Sub ID flag value for new libraries. If set to 1, then sub ID's are auto-sequenced to prevent overwriting.

<code>pdfname=name.*.pdf</code>	In the browser interface, when images are consolidated into a single PDF, a file name is suggested when the PDF file is saved or an attachment is emailed. This value forms a pattern, with an asterisk positioned to indicate where unique sequencing can be applied. Use this to identify a company, such as <code>SDSI.Document.*.pdf</code> , so when an email recipient receives an email, the attachment name will be readily identifiable.
<code>dtddl=http://unformserver:port</code>	If this line is enabled, then the browser interface will display a menu entry to launch the desktop delivery browser client.
<code>selfmanage=0 1</code>	If set to 1, the browser interface will allow users to email their login and password to themselves, and can change their own password.
<code>ses_notext=0 1</code> <code>ses_mailfrom=email</code> <code>ses_tiftpdf=0 1</code> <code>ses_imgtopdf=0 1</code> <code>ses_hidexml=0 1</code> <code>ses_pdfjs=0 1</code>	These entries provide session defaults for the browser interface. The <code>ses_notext</code> value can be used to turn off @text images when browsing. <code>ses_mailfrom</code> provides a default From address when emailing consolidated documents. <code>ses_tiftpdf</code> and <code>ses_imgtopdf</code> enable TIF or all images to be converted to PDF before viewing in the browser. This capability requires that Image Magick be configured on the UnForm server or in the Windows Support Server. The <code>ses_hidexml</code> value will suppress xml images when browsing. The <code>ses_pdfjs</code> value can turn on the PDF.js viewing engine, a customized version of that tool that supports PDF hyperlinks that are embedded in UnForm generated PDF files in some integrations.
<code>logo=filename</code>	Sets the logo to be used as an icon in the browser interface. This should be a small, square image. The default is "unform.gif", which is found in the web/en-us directory.
<code>title=value</code>	Sets the title for the browser window, which most browsers display as a tab title. The default title is "UnForm Document Archiving".
<code>enablefax=0 1</code>	Enable fax tabs in the browser interface. Fax submissions rely on the <code>deliver()</code> function, so faxing must be configured in the <code>deliver.ini</code> file for this functionality to work.
<code>faxcover=name1[,name2...]</code>	When faxing is enabled, this provides a list of one or more cover page names that are acceptable for use with the <code>deliver</code> command's configured faxing product. For example, <code>msfax</code> users could rely on the "generic" cover definition by specifying <code>faxcover=generic</code> . The browser interface offers a choice of no cover, plus any cover names defined here. Multiple names are delimited with commas.
<code>nulltospace=0 1</code>	If set to 1, any updates to document categories will set any mid-segment null values to a space. Null value segments are considered the end of a segment list by the browser interface and library object, so doing this enables "null" segments to be accessed.
<code>server=nameOrIP[:port]</code>	The default archive server, where archive libraries reside. This is useful in cases where a disaster recovery or load balancing server is used. Documents are generated by the local server, but sent to the archive server for storage. In most cases, this should be set to the same value as the [default] section's 'syncfrom' value.
<code>sweep=minutes</code>	Each time a job with an archive command runs, it launches a sweep task to update libraries (or remote servers) with library files and data.

	In addition, this setting controls how often an automated sweep process executes, in case there are errors in the job-specific execution.
<code>map=url</code>	This is used as a mapping URL in DocFlow related documents. Geolocation data can be captured when an image is signed/annotated, and the saved information can include a link to the location. It will use url value, substituting %lat and %long tags with latitude and longitude values.
<code>showice=0 1</code>	Enables inclusion of the Workflow ICE tool on the browser interface Admin menu. This tool is deprecated in favor of DocFlow, but available for existing Workflow users.
[mailcall] section	
<code>name=value</code>	<p>The mailcall section can be used to define mailcall values not available in the email command and email() code block function, such as timeout or bodymime, or to provide a default setting if the command or function doesn't supply a value (or supplies a null value). Any options not set in an email command or function will be filled in with values in this section.</p> <p>For example, if you want a bcc sent to a local support account by default, add a line that says <code>bcc=email address</code>. Or, if you find that the default timeout of 30 seconds isn't enough time for a slow internet connection, add a line like <code>timeout=60</code>.</p>
[syncfiles]	
<p>This section can contain a list of files or wildcards, one per line, that are sent to child servers that have a [defaults] section syncfrom= setting. The child servers connect to this server and request a list of files. The list includes full pathnames and content hashes. Any changed files are then retrieved by the client from this server.</p> <p>The intent of this section is to provide a list of files that are necessary for printing, generally rule files and their dependencies, such as merge files, image files, overlay and attachment files, and so on. These files can be maintained on a primary server, and the child servers can automatically retrieve new versions of those files on a regular schedule.</p>	
[httpd] section - no longer used (see Configuring Apache)	

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

Of particular interest in `ufparam.txc` is the font configuration. All fonts are assigned a numeric ID. Those that are common in pcl5 printers have pre-assigned values from HP, while soft fonts can be given user-defined numeric IDs. These `name=number` pairs are defined in the [fonts] section. ID numbers can then be assigned to soft font names in the [psmap] and [ttmap] sections, or mapped to PDF base fonts in the [pdfmap] section. See the standard `ufparam.txt` file for examples and notes.

When UnForm processes a text or font command, it attempts to match a named option with a font name in the [fonts] section, and it then uses the associated font number. Alternatively, the text or font command can identify the font number directly, with a font *n* option.

Once a number is identified, UnForm then looks for a native soft font definition, depending on the output format. It looks in the [softfont] section for pcl5 output, or the [psmap] section for postscript output, or the [pdfmap] section for mapping to an internal PDF base font. If no match is found, then the [ttmap] section is scanned for a match, and the associated TrueType soft font is embedded in the output and used.

An example of mapping a True Type font would look like this:

```
[fonts]
...
vera=19200

[ttmap]
19200=Vera,VeraBd,Veralt,VeraBI
```

In the [fonts] section is a *name=number* pair. The *number* is user-defined and must not conflict with the various fixed PCL font numbers found in the section. The [ttmap] section contains a *number=font(s)*, where a list of font file names (without the .ttf extension) is provided for normal, bold, italic, and bold-italic versions of the font. Note that not all fonts provide all these versions. True Type font files are found in the ./ttfont directory, or on Windows in the %windir%\fonts directory, or can be specified as full path names.

3.2 CONFIGURING THE CLIENT

The uf101c.ini file can be used to configure the client, both standalone and the client included with the server. Most entries can be overridden by command line options, so these values are only used as defaults. On Linux, the uf101c.ini file is located in the client install path (where uf101c.pl is). On Windows, it is found in %PROGRAMDATA%\SDSI, typically C:\ProgramData\SDSI.

Alternatively on Windows, if the uf101c.ini file is not found in the standard local path, the client will look instead in the executable path. This enables configurations where the client is run across a network, and is not installed on local PC's.

The uf101c.ini file looks like this:

```
[defaults]
server=localhost
port=27400
#logfile=uf101c.log
#mailto=root
retry=30
wait=2
ssl=0
cmptrans=0
authkey=
minavail=1
failhist=1
home=install_folder
perl=perl_executable
```

Change the server= line to point to the server host name or IP address, and the port line to the proper listening port configured in the server's uf101d.ini file. The port default is 27400 and will not normally be changed. Note that the server and port can also be specified on the uf101c command line. The values entered here serve as defaults.

If you want uf101c to log submission errors, uncomment the logfile= line, setting the value to a log file name. This log will accumulate over time, so should be periodically cleared. Note that individual job errors can be captured with the -e *filename* command line option. This log is simply an accumulation of messages that would normally go to the error file or the standard error handle.

If you want uf101c to email (using the Linux mail command) error messages to an administrator, uncomment the mailto= line, setting the value to an email address available from the client computer. Note that the Windows client does not support emailing of error messages.

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

The ssl setting, if 1, will cause the client to connect to the server using ssl. The server must be configured to listen using ssl. Both server and client must have matching settings.

The cmprans setting controls whether or not the client will communicate to the server using zlib compression. This can improve performance over low bandwidth connections.

The authkey setting must match whatever the server's authkey setting is, even if null.

The minavail setting is used when specifying multiple servers when submitting a job, in a load balancing environment. The first server with at least minavail job slots available is chosen. If this setting is 0, then if all servers have active jobs, the one with the least active jobs as a proportion of their license count is chosen.

The failhist setting sets the number of days that [job failures](#) are stored at the client. This number must be 1 or higher for failures to be stored, and the fail history queue is purged of directories over this many days old.

On Windows, the home= setting is created during installation. It is used by the uf101c.exe client to locate the uf101cc.exe console client, which performs the actual communication with the UnForm server.

Also on Windows, if you have ActiveState or Strawberry Perl installed, you can instruct uf101c.exe to run the uf101c.pl perl script as a console client, by creating a perl=*path* line that points to the perl interpreter executable. This setting is optional, as the uf101cc\uf101cc.exe program contains perl, but there may be a small launch performance benefit to running the script via an interpreter directly.

3.3 CONFIGURING APACHE

The UnForm 10.1 server launches a private instance of the Apache web server to provide access to all browser-based functions and the REST interface. Note this differs from previous versions, which included an internal HTTP server and optionally offered cgi scripts to run on an external web server.

On Windows, UnForm includes a bundled version of Apache, so there is no need to configure it. On Linux and Unix systems, Apache must be installed on the system where UnForm is installed. It is generally included with any Linux or Unix system. All that is necessary for UnForm to operate its instance is to know where the Apache httpd executable is, and the Apache "home" directory where its modules are located.

The Unix ufsetup.sh setup script looks for Apache in several common locations, and will ask if Apache can't be found. It will then create two variables in the /usr/bin/uf101d script called HTTPD and

HTTPDHOME. These are the default locations used when starting the Apache server. Additional configuration is performed in the [apache] section of the uf101d.ini file.

<code>httpd=path</code>	<p>Path to the Apache httpd executable (i.e. /usr/sbin/httpd).</p> <p>Automatically configured under Windows, and defaults to the HTTPD variable defined in /usr/bin/uf101d.</p>
<code>serverhome=directory path</code>	<p>Home directory of Apache, where modules subdirectory is located (i.e. /etc/httpd).</p> <p>Automatically configured under Windows, and defaults to the HTTPDHOME variable defined in /usr/bin/uf101d.</p>
<code>port=port number</code>	<p>The HTTP port the Apache server listens for UnForm requests. Defaults to 27402.</p>
<code>apacheuser=user</code> <code>apachegroup=group</code>	<p>Normally the UnForm server runs under the 'unform' user on Unix systems, but it can be configured to run under any user, including root. If it runs as root, then the Apache server must start under a different user and group.</p> <p>Enable these lines and set them to the proper values for your system if the UnForm server runs as root. Note the ufsetup.sh script will attempt to enable these lines if it configures UnForm to run as root and finds one of several standard Apache users.</p> <p>Under Windows, the Apache server always runs under the same user as the UnForm server service.</p>
<code>adminemail=address</code>	<p>Configures an admin email address for the Apache server to use.</p>
<code>ssl=0 1</code>	<p>If set to 1, Apache will listen on an SSL port, requiring that browsers connect using the https:// protocol rather than http.</p>
<code>sslcert=path.pem</code>	<p>If configured to listen using SSL, by default UnForm will supply a self-signed certificate for Apache to use. Browsers will warn users about the certificate not being trusted, which the user can ignore to continue, or he/she may install locally to avoid the warning message again.</p> <p>If desired, a signed certificate can be supplied that matches the domain used to connect to the server. Certificates for Apache are available from many vendors, such as LetsEncrypt, Verisign, Thawte, or GlobalSign. See the mod_ssl documentation, specifically the SSLCertificateFile directive, on http://httpd.apache.org for details.</p> <p>A combined certificate, containing a private key followed by the certificate, and possibly a certificate chain if required by the certificate authority, may be used. More commonly, a vendor will supply separate files for the certificate, chain, and key. You can enter those additional files in the sslchain and sslkey settings.</p>
<code>sslchain=path</code>	<p>If needed, a chain file can be supplied. This is used for the SSLCertificateChainFile in the Apache configuration, if supplied.</p>
<code>sslkey=path</code>	<p>If needed (not in the sslcert file), a private key file can be supplied. This is used for the SSLCertificateKeyFile in the Apache configuration, if supplied.</p>

Apache Configuration File

When UnForm launches the apache server instance, it does so with an apache.conf configuration file that is derived from the apache.tpl file in the UnForm server install location. That file is a template with tags that are replaced at runtime to generate apache.conf. The apache.tpl file is included in the UnForm installers and is overwritten by an update, so can't be customized. If there is a need for customization of this file, you can copy it to apache.custom.tpl and modify that file. Be sure to do this under controlled conditions and ensure that with an UnForm restart, Apache is started successfully by UnForm. Apache configuration commands are very precise and an error will prevent Apache from starting.

Alternate Main Portal Page

The web/index.html file is the standard main portal page, displayed when a browser views the top level url, such as `http://hostname:27402`. An alternate page can be displayed by creating the web/index.custom.html page. Two examples are provided, index.list.html and index.tree.html, which provide different styles of indexes. To make either of these active, copy the desired file to index.custom.html.

3.4 CONFIGURING IMAGE MAGICK

Image Magick is used to convert images between various formats. For example, if a logo image is provided in jpeg format, when UnForm needs to print the image to a PCL printer, it needs to convert and scale the image to the proper format and size. It uses Image Magick to do this. Otherwise, the image must be provided in all the various formats that might be needed.

Most Linux systems include a copy of Image Magick, or it can be installed using the system's package manager (ufsetup.sh will offer this option). Windows and other versions can be downloaded from <http://imagemagick.org>. On Linux, you can easily verify if Magick is installed by running 'convert -version'.

Once installed, all that is needed is to set the line 'converter=*path*' to point to the Magick executable "convert" (convert.exe on Windows). The Server Manager provides a form for this setting, or you can manually edit the uf101d.ini file. This line is in the [images] section.

Note that in Image Magick 7, the convert program name changed to 'magick', so link to 'magick' or 'magick.exe'.

3.5 CONFIGURING GHOSTSCRIPT

Ghostscript is a program used to convert Postscript and PDF documents to image formats. Modern versions (starting with 9.06) can also be used to extract text data from PDF documents. Whenever image output is needed, or text extraction is used such as with [AFO](#), UnForm can internally execute Ghostscript to perform the required function.

If Ghostscript is not already installed on your system, you can use your system's package management tools to install it (ufsetup.sh will offer this option), or you can obtain it from <http://ghostscript.com>. You can easily verify if Ghostscript is on a Linux system by running 'gs -version'.

Ghostscript is configured in the [drivers] section of uf101d.ini. You can edit this file manually, or use the Server Manager's [configuration](#) option to configure the Ghostscript executable path.

The [drivers] section includes several options, plus a series of "drivers" that associate a driver name with a Ghostscript output format.

- **gs=path** to define the executable name for the Ghostscript executable on the server. On Linux systems, this is often just set to "gs" (i.e. gs=gs), because the "gs" command is found in a universal path like /usr/bin/gs. On Windows, the path will be found in Program Files or Program Files (x86), and the executable name is gswin32.exe or gswin64.exe (or you can use the "console" versions gswin32c.exe or gswin64c.exe).
- **pdffitpage=0|1** is used to specify if Ghostscript is of a level that supports the PDFFitPage option, which can be used to improve performance of some UnForm jobs. This feature was added to GhostScript at 8.10, so only very old versions will not support it.
- **gpcl=path** defines a path to a GhostPCL executable, such as /usr/bin/ghostpcl, or c:\ghostpcl\gpcl6win64.exe. This is used by the Design Tool when performing test printing to a PCL output format, when viewed in the browser window. GhostPCL is used to convert UnForm's PCL output to PDF.
- **labelary=1** enables use of the labelary.com web service when previewing Zebra (ZPL) labels in the Design Tool. See that site for information about usage limits.
- Other entries are simply *name=device,multipage,dpi*, where *name* is the UnForm driver name, *device* is the `-sDEVICE` name used by Ghostscript, *multipage* is a 0 or 1, where 1 means the output is multi-page=multi-file and 0 means all pages go to a single file, and *dpi* is the dots-per-inch resolution.

Note that the use of multi- or single-page output is often dependent on the image format. For example, bmp files do not support multiple pages per file, while tiff files do. Below is a sample [drivers] section. In most cases, the only configuration required is setting the gs= path.

Note also that if you can't locate a suitable Ghostscript version for your Unix system, you can use the Windows Support Server to execute a Windows Ghostscript for jobs that require it.

[drivers]

```
# enable Ghostscript drivers by uncommenting the gs= line
gs=gs
# windows would typically need a full path
# gs=c:\gs\gs9.xx\bin\gswin32c.exe
pdffitpage=1
# driver lines are structured as name=gsdevice,multipage,density
# gsdevice is the Ghostscript sDEVICE value
# multipage is Boolean 0 or 1, 1 means -o file is file<page>.ext
# Many formats require a 1, as the image format supports only a
# single image per file.
# density is output density, as hhh[xww] (horizontalxvertical) dpi
```

```
bmp=bmp256,1,300
bpmmono=bpmmono,1,300
tif=tifccrle,0,300
tifmono=tiffig3,0,300
png=png256,1,300
pngmono=pngmono,1,300
jpeg=jpeg,1,300
ps=pswrite,0,300
eps=epswrite,1,300
deskjet=deskjet,0,300
pcl6=pxlmono,0,300
```


pcl6c=pxlcolor,0,300

3.6 CONFIGURING TESSERACT

If your installation uses the [Image Manager](#), you might wish to install Tesseract, an open source OCR engine. If configured, and UnForm receives images or PDF files without a text layer, it will process those images through Tesseract to build a text layer. Note that if your implementation relies heavily on OCR processing, better results are typically achieved with a commercial OCR tool that is able to produce PDF files with a text layer that UnForm can read.

Windows

There are several sources for tesseract on Windows. One such source for Windows, recommended by the Tesseract project, is:

<https://github.com/UB-Mannheim/tesseract/wiki>

Linux

Tesseract can be installed on Linux using standard package management tools. Use the following commands:

- Redhat/Fedora/CentOS: yum install tesseract
- Ubuntu/Debian: apt-get install tesseract-ocr
- SUSE Linux: visit <https://software.opensuse.org> to locate rpm install packages for your version.

Redhat Enterprise users may need to first install EPEL repository extensions (Fedora installs should already have these available). The following web page provides copy/paste command lines and information:

<https://fedoraproject.org/wiki/EPEL>

As of this writing, there are relevant command lines:

- CentOS 6: yum install <https://dl.fedoraproject.org/pub/epel/epel-release-latest-6.noarch.rpm>
- CentOS 7: yum install <https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm>
- RedHat: subscription-manager repos --enable "rhel-*-optional-rpms" --enable "rhel-*-extras-rpms"

Alternate Repository

Alternatively, if your package manager installs an older version and you wish to have a more recent version, you can find instructions on installing or updating from a custom repository here:

<https://github.com/tesseract-ocr/tesseract/wiki>

Note that Tesseract 4 and higher has proven to produce generally better results than version 3.

Once installed, restart the UnForm server and it should locate Tesseract automatically, assuming you choose a default installation path. On Linux, it will locate it in /usr/local/bin or /usr/bin. If necessary, you

can manually configure the path to the tesseract executable in uf101d.ini, the [tesseract] section, setting tesseract=*path*.

Preprocess Filter

When Tesseract is called by UnForm, it is provided an image or set of image pages. The format of these pages is either the native image format imported by the inbound source, or JPEG files if the incoming file is a PDF. UnForm uses GhostScript to convert the PDF file to JPEG page files. If desired, you can configure an Image Magick command line as a pre-process filter that is run against these image files. This is configured in two lines in uf101d.ini. First, define an Image Magick command line fragment in the [images] section. The name of this entry can contain "jpg", "jpeg", or "tif" to determine the resulting format of the command. The default format is "png", so any other name results in PNG images being given to Tesseract. An example might be this:

```
tesspng="%i" -blur 1x1 -monochrome "%o" >/dev/null 2>&1
```

Then instruct UnForm to use this filter before running the file through Tesseract, by adding this line in the [tesseract] section:

```
filter=tesspng
```

3.7 TCP/IP MONITOR

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

The monitor is automatically started if there are one or more port configuration lines defined in the [tcpports] section of uf101d.ini. For example:

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

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

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

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

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

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

The following substitutions are made in the command line definition:

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

When jobs are submitted to the UnForm server in this manner, it is important to realize that the submission is one-way, and once printed the job resides entirely on the server. It is therefore not possible to print a job and have data returned to the client (i.e. `-o client:device`), or to have PDF previews generated on the submitting workstation (`-p winpw`). Once the job is submitted to the TCP/IP monitor, it becomes local to the UnForm server, as if `uf101c` is physically run on the server (which, in fact, is what happens).

When jobs are submitted, they are dropped into the `rpq/` subdirectory under the UnForm server installation. All submission files are given a unique name with a `.in` extension, and a companion file with a `.cmd` extension is also created that contains the command line options. As jobs are received, and also at least once every 5 seconds, a sweep is made of newly submitted jobs, each submitted to the server via the server's local `"uf101c"` program. As a byproduct, you can drop jobs into this directory independently of the server, being careful to create the `.cmd` file first, then the associated, complete `.in` file, using your own unique naming algorithm. Note that the sweep assumes that any `*.in` file is a complete file and will have an associated `.cmd` file, so it is incorrect to open a `.in` file and begin writing to it, as the sweep may attempt to process an incomplete file. Instead, create the file with a different extension and then rename it when it is ready for processing.

When there are launch errors processing the job, such as an invalid output device, the `rpq` spool files remain in the queue. Non-license errors are retried a maximum number of times (based on the `tcpportretry=n` setting in `uf101d.ini`), and then go idle to await administrator disposition. Job license count exceeded errors are retried indefinitely, until the queue files are purged via normal aging cycles. There is a `.rty` file for each job that counts the retries, so if a problem is transient, an administrator can remove `*.rty` files from `rpq` and the jobs will start processing again. The `*.cmd` files can also be edited, if needed, to fix a command line option that is causing an error, before removing the `*.rty` files. Note that errors that occur after a job is started, such as subjob errors or syntax errors, do not trigger retry processing.

To configure Windows printers to submit jobs to this monitor, you can use the built-in Windows support for TCP/IP printers. When configuring a printer, you can choose to Add a Port, selecting Standard TCP/IP Port. The Printer name or IP address of the "printer" will be the UnForm server, the Protocol is "Raw", and the Port Number is the number of the configured port line defined in `uf101d.ini`. UnForm can accept two types of input: plain text and PostScript, so you can choose either the Generic / Text Only print driver, or a PostScript driver, such as the Generic MS Publisher Imagesetter or one of the many printer vendor PostScript drivers. Note there are significant differences in the way UnForm handles the two different types of input. See the [UnForm AFO](#) chapter for more information.

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

Configure Standard TCP/IP Port Monitor

Port Settings

Port Name:

Printer Name or IP Address:

Protocol: ☒ Raw ☐ LPR

Raw Settings

Port Number:

LPR Settings

Queue Name:

☐ LPR Byte Counting Enabled

☐ SNMP Status Enabled

Community Name:

SNMP Device Index:

OK Cancel

Note that other operating systems also support methods of printing to raw TCP/IP printers. For example, Linux CUPS supports the socket://address:port device setting. The configuration is identical to that of configuring a network printer, except the "printer's" IP address and port are the UnForm server and raw port number.

3.8 DELIVER CONFIGURATION

The deliver.ini file contains configuration information for the deliver command and deliver() code block function. The file contains a [default] section, an [email] section, one or more fax definition sections, and custom sections for specialized document delivery. The [default] section contains a fax= line that specifies which fax definition section is used. It also contains logging parameters.

Any section can contain a line 'async=1' if deliveries using that section should be queued rather than immediately delivered. This can be helpful in cases where delivery might be time consuming and would slow down jobs, or where automatic retries are desired. Email and fax deliveries are good candidates for async delivery.

Default Section

The [default] section defines standard settings used by the deliver command and deliver() function. The lines are

fax=section	Identifies the deliver.ini section that defines the fax gateway configuration.
logage=days	The number of days deliver logs are maintained before automatic purging.

logtags=0 1	Determines whether or not to log tags in the log files (0=don't log tags, 1=log tags). This is often helpful information, but does add to the verbosity of the log files.
queue= <i>minutes</i>	When asynchronous delivery is used, this defines how many minutes between queue sweeps are performed. For example, queue=10 would sweep on the 10 minute increments (00, 10, 20, 30, 40, and 50).
retryage= <i>days</i>	The number of days that async deliveries are retried before being purged. If this is 0 or not set, the logage value is used.

Email Section

The [email] section defines standard tags used for emailing. These tags supplement the configuration found in prog/mailcall.ini and also in the [mailcall] section of uf101d.ini. Generally there is no need to adjust this section unless you choose to implement the async flag.

Fax Sections

A fax definition indicates the type of interaction that is required, command line, email (SMTP), or the internal msfax option, which utilizes the Windows Support Server to send faxes via Microsoft Fax. Each configuration line can contain optional and required tags, which are substituted at runtime with tag values specified in the deliver command or deliver() code block function. Tags are used to specify fax numbers, subject lines, recipient names, and other values, and the values from the commands are mapped to the configuration each time a delivery is performed.

There are five pre-defined tag names that can be used in a fax configuration.

Tag	Usage
%to	Fax number or email address.
%faxfile	File to be sent (the output from UnForm, typically of a subjob). The deliver command manages this value automatically. The deliver() function has a file name argument that this tag references.
%errfile	An error file, which if used and contains content will be returned in the errmsg\$ argument of the deliver() function. UnForm generates the file name automatically, and then expects to find error messages, if any, in the file when the fax or email is submitted.
%rspfile	A response file, which if used and contains content will be returned in the response\$ argument of the deliver() function. UnForm generates the file name automatically, and then expects to find messages, if any, in the file when the fax or email is submitted. For example, this might be the standard output of a vfx command line, which can contain a fax job number.
%attach	Used for additional documents beyond the %f file. All attach tags are accumulated, with space delimiters for command methods, and comma delimiters for email methods, then the entire list is substituted for the %attach marker. The msfax method does not support attachments.

Additional tags are referenced with a %*name* syntax, and are substituted from the tag options provided by the deliver command or deliver() function. Optional tags are specified using a { %*name* } syntax, where a tag and any text can be specified inside the curly braces. After all %*name* substitutions have occurred, any remaining {...} sequences that contain %*name* tags are removed before the delivery is actually executed.

When the deliver command generates a document for delivery, it does so with an UnForm subjob. When delivering to an email address, the subjob always produces a pdf file. When delivering to a fax number, the type of file is configured with the format=pcl|pdf|ps line. When executing the subjob, several command

line options are automatically managed. The fax configuration section can specify additional options, as can the deliver command itself.

Other Sections

The deliver command can also "*deliver*" documents to other destinations. If the [deliver](#) command's target address matches a section name in deliver.ini, then the configuration in that section is used, rather than a fax or email configuration.

Each of these sections must have a `method=` line which determines how the balance of the section is interpreted, and a `format=` line to specify what type of file the deliver command produces.

method=file	<p>Delivers a document to the file system, using the path specification. The path may contain tags, as there should be some dynamic values in the file. One common tag would simply be %file, which represents the base name of the work file generated, which itself is based on the "<i>docid</i>" value provided in the deliver command. This might, for example, be an invoice number or purchase order number, though it can be any value required. Other tags will be substituted from the tags supplied in the deliver command.</p> <p>The <code>extension=</code> value can be used to override the default extension provided by the driver.</p> <p>A <code>sequence=number</code> can be used to prevent overwriting of a file. The value of <i>number</i> specifies the default number of digits used in the sequencer. For example, <code>sequencer=3</code> would produce unique sequences like -001, -002, and so on. Sequences are always appended to the end of the file, before the file's extension.</p> <p>If sequence is 0, you can specify <code>overwrite=1</code> to allow overwriting of existing files. Otherwise, if the file already exists, an error is generated.</p>
method=command	<p>This executes the system command specified by the <code>command=value</code> specification. You can optionally use special tags %rspfile and %errfile to capture response and error data from the command for logging purposes.</p>
method=desktop	<p>This method uses Desktop Delivery to send a document to a user running the Desktop Delivery browser interface. A user and optional ipaddr value must be specified in deliver command tags so the system knows where to send the document. Optionally, specify a title and style tag. The style tag can be 0 for standard 'notification' style delivery, or 1 for a popup delivery, which is automatically displayed on the user's desktop if the user is currently connected.</p>
method=code	<p>This method executes runtime code (native pvx) found between 'code on' and 'code off' lines within the section. This code can return a value in response\$, or an error message in errmsg\$. Lines in the code may contain references to standard and user-supplied tag values, such as %file or %path. This method provides a great deal of flexibility to allow delivery of documents to cloud services, external document management systems, and other external features.</p>

In addition to tags provided in the deliver command and deliver() functions, the following tags can be used to reference the file to be delivered:

%file	<p>The base file name of the document file to be delivered (no path information). This is derived by the system from the document ID value supplied in the deliver command. Useful in file, command, or</p>
-------	---

	code methods when the derived document ID has value.
%workfile	The full path of the document file to be delivered. Useful in code or command methods to reference the document data file.

ASYNCHRONOUS DELIVERY

When the deliver command runs, it generates a document file based upon a document ID value and normally will attempt to deliver that file as the job runs. If the delivery section contains an 'async=1' configuration, then the delivery file and information are queued in the server's "deliver" subdirectory, and a separate process is launched to perform the delivery. In addition, if the delivery of any given file gets an error, the file is retained and another process will attempt the delivery later. This retry mechanism is logged as a status of RTY in the deliver log file, indicating there was an error and the delivery will be retried later.

The frequency of retries is determined by the queue=*minutes* setting in the [default] section.

The retries will continue as long as the delivery .spl file remains in the queue. They are automatically purged based on the same aging process as the log files (by default 30 days). Set the retryage=*days* parameter to change this retry time frame. All errors are logged both in the deliver logs and in the main server log.

Queued delivery files are not editable, and it is possible that tags embedded in the files produce permanent errors. For example, if a mail server tag is included with an invalid address, the delivery will always fail. It is possible to create a file named "tagsubs.txt" in the deliver subdirectory. If this file exists, any *oldvalue=newvalue* lines in the file cause literal text substitution in the tags of each delivery file. As an example, if delivery tags included a bad server specification of 'server=ssl:smtp.gnail.com', you could add a line "smtp.gnail.com=smtp.gmail.com" to dynamically correct the domain name. Note that if you need to specify an "=" sign in either value, use "\=".

Deliver Errors

You can customize handling of permanent errors in a [permerrs] section. See the example below. If a particular delivery file will always fail, you can delete the associated deliver/*.spl file found in the log message manually to stop the repeated retries.

EXAMPLES

Default section

This example [default] section specifies that the vsifax fax definition will be used for faxing. Daily delivery log files will be placed in the deliver subdirectory under the UnForm server, tags will be logged in addition to basic delivery data, and log files will be retained for 30 days.

```
[default]
fax=vsifax
#fax=msfax
logtags=1
logage=30
queue=5
```

Email section

This [email] section defines a To: header format with an optional quoted name and the to address. The name can be supplied via a name="value" tag in the deliver command or function. If not supplied, then the To: header will just have the email address, automatically supplied as the "to" tag.

Other values can be supplied via additional tags. The email will always include a file attachment, as generated by the deliver command or supplied to the deliver function. In addition, if there are any "attach filename" tags, those files will be attached to the email as well.

```
[email]
to={"%name" }%to
subject={"%subject}
msgtxt={"%note}
attach=%f{,%attach}
logfile={"%logfile}
#otherhead=
#login=
#password=
#server=
# Note, additional options come from [mailcall] section in uf101d.ini,
# or the mailcall.ini file.
async=1
```

Vsifax section

The vsifax section defines a fax gateway that uses the vfx command to send a fax. The method is set to "command", indicating that a command line is created and executed. The command is specified over several lines in the deliver.ini file by using \ at the ends of lines to indicate that the line continues on the next physical line.

There are several optional elements, such as name and subject. If they are supplied as tags in the deliver command or deliver() function, then -t options will be added to the vfx command line. If not, then those elements are suppressed from the command line. For example, if one tag was 'name="John Smith"', then there would be a '-t tnm="John Smith"' added to the command line.

Standard output is captured in a response file, and error output is capture in an error file. The use of %r and %e in the command allows the deliver() function to return the command's response or error message.

When the deliver command executes the subjob to produce the file to be faxed, it will produce it in pcl format (using a -p pcl option). In addition, a -nohpgl option will be used.

```
[vsifax]
method=command
command=vfx -n '%to' -F pcl \
{-t tnm="%toname"} \
{-t tco="%tocompany"} \
{-t sub="%subject"} \
{-t fnm="%fromname"} \
{-t fco="%fromcompany"} \
{-t ntx="%note"} \
"%faxfile" { %attach} \
>%rspfile 2>%errfile
format=pcl
options=-nohpgl
```


Note that for Windows, since vfx.exe is a console command, the command should start with "cmd.exe /c vfx.exe ...".

Rapidfax section

The rapidfax section shows an example of using a third-party Internet faxing service for sending faxes. Such services accept email submissions via email. PDF format is generally supported, and many services offer support for additional document types, meaning you can add additional attachments, such as text files, image files, or Microsoft Office files.

The method is specified as "email" to indicate that UnForm will submit faxes by emailing them to the fax service. The To address for most Internet services is *faxnum@service.com*. In the case of Rapidfax, the domain is rapidfax.com. Rapidfax expects fax submissions from a specific email address, and with a user-configured subject line. In this example, the From address and subject lines are hardcoded. There can be a message body, specified by a note tag, and both the submitted file and any other files specified by attach=*filename* tags are submitted for faxing.

```
[rapidfax]
method=email
to=%to@rapidfax.com
from=you@yourcompany.com
subject=yourpassword
msgtxt=%note
format=pdf
attach=%faxfile{,%attach}
```

Msfax Section

The msfax section specifies the "msfax" method, which is an internal UnForm method that works in conjunction with the Windows Support Server. The command value specifies tag names documented in the Windows Support Server chapter and the msfax() function.

```
[msfax]
method=msfax
format=pdf
command={toname="%toname %tocompany"}\
{,fromname="%fromname"}{,fromcompany="%fromcompany"}\
{,subject="%subject"}{,note="%note"}{,cover="%coverpage"}
```

CirrusPrint section

The cirrusprint section is designed to enable a deliver command to target a CirrusPrint output device. It is a code-based section authored by the publisher, using the cirrusprint object to submit a file to CirrusPrint.

Set the server defaults for logging in, or optionally specify them with server, login, and password tags in the deliver command. The command must also specify locid and devid tags to specify the CirrusPrint output device. For example:

```
deliver "cirrusprint",{docid$}, locid "warehouse1", devid {printrname$}
```

If the password has the format "store:*name*", the password is retrieved from the secure password store maintained in the admin browser interface.

```
[cirrusprint]
method=code
# below settings establish defaults
server=https://someserver:port
login=someuser@companyid
```

```
password=thepassword
```

```
code on
d=new("infile","deliver.ini")
o=new("cirrusprint")
o'server$="{%server}"
if o'server$="" o'server$=d'getitem$("cirrusprint","server")
o'login$="{%login}"
if o'login$="" o'login$=d'getitem$("cirrusprint","login")
o'password$="{%password}"
if o'password$="" o'password$=d'getitem$("cirrusprint","password")

success=o'sendfile("%workfile","%locid","%devid","%file")
if not(success) errmsg$o'lasterrmsg$
drop object o
drop object d

code off
```

Deliver to File Section

```
[testfiles]
method=file
format=html5
extension=html
path=deliver_files/{%subdir}/%file
sequence=2
#overwrite=0
```

Deliver to Command Line

```
[testcmd]
method=command
format=pdf
command=cp "%workfile" "/tmp/%file"
```

Deliver to Desktop (using desktop delivery)

```
[testdtel]
method=desktop
format=pdf
title=%title
user=%user
{ip=%ipaddr}
style=%style
```

Deliver to Code

```
[testcode]
method=code
code on
```

```
# sample code that copies the deliver file to the testcode_files directory
workfile$="%workfile"
directory "./testcode_files",err=*next
# uncomment to allow overwriting
#erase "./testcode_files/%file",err=*next
```

```
o=new("system")
o'copyfile(workfile$,"./testcode_files/%file",errmsg$)
drop object o
```

code off

Permerrs Section

This section can be created to designate certain error message patterns as permanent errors that should not be retried when async (retry) mode is turned on. Internally a matching line will switch a RTY status to ERR and that delivery will no longer be retried. Use this feature to prevent unwanted retries for errors outside the default settings. By default, bad to or from address formats and Error 5xx response codes are considered permanent errors.

```
[permerrs]
# Enter lines as either wildcards or ~regex values, matches are case-insensitive, or #comments
*operating system command failed*
~error 43\d
```

3.9 MESSAGE TRANSLATIONS

UnForm 10.1 supports the ability to use user-selected message files for language translations of user interfaces in the following tools: Windows Server Manager, Windows Support Server, Design Tool, and Image Manager.

Each of these programs has a configuration .ini file that can contain a [languages] section. This section contains lines in the format of *ext=title*, where *ext* is a file extension to use when opening the messages file. UnForm is always supplied with English messages files with an "eng" extension.

The interface for each program offers a drop-down box to select a current language from those configured. The selected language is saved for future executions. The following table shows the configuration files that can contain the [languages] section, and the base names of the language files.

Program	Configuration File	Language Files
Windows Server Manager	uf101dx.ini	uf90dl.*
Windows Support Server	uf101ss.ini	ufssmsg.*
Design Tool	ufdsn.ini	ufdsnmsg.*
Image Manager	uf101scn.ini	ufscnmsg.*

There is also a Windows client messages file, uf101cmsg.*. The default file is uf101cmsg.eng, but there is a uf101c.exe command line option to specify an extension (-lang *ext*).

3.10 RPQ CONFIGURATION

The rpq.ini file can be used to configure handling of the rpq directory, which is a spool of jobs that have been submitted for print processing. Jobs can arrive in this queue via [TCP/IP printing](#), or -async/-asynrcpq uf101c options, or through custom processing.

Queue Processing

The rpq directory contains file sets with common base names. There are always at least two files per submission, with .in and .cmd extensions. The .in file is the input for the job, used as a -i filename option when submitting, and the .cmd contains uf101c command line arguments, other than -i and -e options, which are handled internally by the queue processor.

As jobs are submitted for processing, if an error occurs a .err file is created containing an error message. Each time a non-license count error occurs, a .rty file is also maintained to track how many times the error occurred. Once this file reaches a threshold defined in uf101d.ini as "tcpportretry", the error is assumed to be permanent with no further attempts to submit the job. The files remain in queue for up to 30 days, and if the error message found in the .err file can be corrected, removing the .rty file enables submission to occur again.

Job Priority

It is possible to set some jobs to high or low priority, in addition to the default medium priority. Jobs with high priority are processed before other jobs, and those set with low priority are processed after other jobs. This can be done explicitly via uf101c command line options, or implicitly via three sections in the rpq.ini file.

Explicit priority is set with the -priority 1|2|3|high|medium|low command line option.

- -priority 1 or -priority high will set the submitted job to the highest priority.
- -priority 2 or -priority medium, or no -priority option, will set the submitted job to medium priority
- -priority 3 or -priority low will set the submitted job to the lowest priority.

You can also define content matching for priorities in the rpq.ini file. Add lines to the [high], [medium], and [low] sections as required. If the initial block of the input file (defined by blocksize=n in the rpq.ini [default] section) matches any line in a section, the job is given the associated priority. When PDF files are submitted, text content from the first page is used, as a sequence of space-separated words, in place of a raw data block.

For example, this section would process jobs with PICK TICKET or an xml element <docid> with content of "counter sale " and digits as high priority:

```
[high]
PICK TICKET
~<docid>counter sale \d+</docid>
```

Regular expression matches are defined with a ~ prefix. Regular expression patterns are not case-sensitive.

4 INTEGRATING UNFORM WITH APPLICATIONS

4.1 General Integration Concepts

UnForm is capable of interfacing with any application that can provide it with text input or PostScript. On UNIX, this integration is generally performed via pipes. On Windows, your application can use [TCP/IP printing](#), or can print to a file, and then launch uf101c.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: `|uf101c -f rulefile | lp -dprinter -oraw -s 2>/dev/null`

`|uf101c -f rulefile -o '>lp -dprinter -oraw'`

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

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

Before: `/dev/lp0`

After: `|uf101c -f rulefile -o /dev/lp0`

Note the use of the `-oraw` option in the above spooler examples. It is important for UnForm's output to be handled as binary data by the spooler. The `-oraw` option is used in the Linux/OSX CUPS spooler. Other spoolers require different options, such as `-o-dp` for AIX, or `-l` (lowercase L) for the lpr spooler. Check your lp configuration tools or man pages for the appropriate settings for options such as "binary", "raw", or "pass-thru" printing.

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

open "work.txt" for output as #1

print #1,tab(35); "INVOICE"
... *more printing* ...

close #1

if shell("uf101c.exe -i work.txt -o //server/hplaser -f rulefile",6)=0 then
 end
else
 msgbox "UnForm failed to start."
end if

4.2 Integrating UnForm with BBx

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

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

UNIX Aliases

A printer alias line on UNIX generally pipes to a program, such as the uf101c client program. This client program in turn can pipe its output to the spooler, or to a file, or it can instruct the server to handle the output from its end, by specifying the `-o` option.

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

```
alias P1 "|uf101c -f my.rul | lp -dxyz -oraw -s 2>/dev/null" "Printer Name" ... various modes ...
```

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

```
alias P1 "|uf101c -f my.rul -o \">lp -dxyz -oraw\" \"Printer Name\" ... various modes ...
```

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

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

```
alias P1 "|uf101c -f my.rul -o /dev/lp0" "Printer Name" ... various modes ...
```

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

Windows Alias Lines

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

A Windows alias line will look similar to this:

```
alias P1 C:/TEMP/P1.TXT "UnForm Printer" CR, LOCK=C:/TEMP/P1.LCK, O_CREATE,  
SPCOLS=132, SP=1B451B287331362E3636481B266B3247, EXECOFF="uf101c.exe -ix  
C:/TEMP/P1.TXT -o device -f my.rul"
```

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

Note that pathnames containing backslashes will need double backslashes, due to the way BBx parses the command line. For example, to refer to "uf101c.exe -i c:\data\p1.txt ...", you would need to specify "uf101c.exe -i c:\\data\\p1.txt ...". You can also use forward slashes in place of backslashes, and you don't need to double them.

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

- An LPTn port, which can be mapped to a UNC device name with the Windows "net use" command.
- A UNC device name, defined by sharing a printer, so the name becomes `//system/printer`, where *system* is the system with the shared printer, and *printer* is the "share name" of that printer.

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

```
alias PUNF C:/TEMP/PUNF.TXT "UnForm Printer"
CR,LOCK=C:/TEMP/PUNF.LCK,O_CREATE,SPCOLS=132,EXECOFF="uf101c.exe -ix
C:/TEMP/PUNF.TXT -p winpw -f my.rul"
```

Note that **the uf101c client software must be installed locally on any workstation that will execute it to submit jobs.**

BBj Client and Server Modes

BBj can create/open files and run EXECOFF programs on either the server or the client. To force one or the other use "client,clientexec" or "server,serverexec" options in the alias line, per the Basis documentation for BBj. Be sure to use both options together so the file that the execoff program needs is where the execoff program runs. Remember that where the EXECOFF runs, the UnForm client must be installed or accessed via a network path.

4.3 Integrating UnForm with ProvideX

Simple UNIX Integration

On UNIX systems, you can integrate UnForm within the link file as the output device, and use a standard LaserJet or plain text print driver. The device used in the link file would be simply a re-direct to the uf101c program (if using ProvideX 6.0 features, a pipe (| rather than >) can be used as well), such as ">uf101c -f acme.rul -o '>lp -dhp -oraw'".

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

Integration using the ProvideX Print Driver uf101ptr

This method works for both UNIX and Windows environments, and provides more program control over the UnForm options when executing the uf101c client.

UnForm installation includes a ProvideX print driver **uf101ptr** which should be copied to your ProvideX lib/_dev directory. This driver provides platform-independent support for UnForm, along with additional capabilities for managing UnForm command lines from the ProvideX application. In addition, it supports

WindX-based output. Once copied to your ProvideX lib/_dev directory, this driver is available to use when defining ProvideX link files, which are used as printers in ProvideX.

To use the uf101ptr print driver:

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

The uf9ptr driver determines a default output device based upon the link file's specified output, and then re-routes the printer output to a temporary work file.

It then looks for a configuration file for additional uf101c command line parameters. This file is simply a text file named *linkfile.unf*. For example, for a link file named P1, uf9ptr will look for a file called P1.unf for additional parameters. In this text file can be one or more lines with uf101c command line options.

Once the file-based parameters have been loaded, uf101ptr then looks for the OPT value that was used in the OPEN directive, if any, for additional parameters. Any parameters named in the OPT value will override those found in the configuration file.

When all parameters have been resolved, a uf101c command line is built for execution at the end of the job. In cases where the output needs to be returned to a WindX client, the driver handles uf101c appropriately to create local output and copy that output back to the WindX PC.

Example 1:

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

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

The result will be an uf101c command like this, which executes when the printer is closed:

```
uf101c -i workfile -p pdf -o /archive/00015.pdf
```

Example 2:

P1 is a link file pointing to device ">lp -dhp4000 -oraw".
No P1.unf file is defined
OPEN(1,OPT="-f acme.rul")"P1"

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

Example 3:

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

Example 4:

P3 is a link file pointing to device NUL.

P3.unf contains `-p winpww`.

Opening P3 will cause production of a temporary PDF file. This file will automatically be viewed on a WindX client or in a Windows ProvideX session.

5 UNFORM COMMAND LINE OPTIONS

5.1 Server Options

The server program can be started with the following options:

UNIX command lines	
uf101d start	Starts the server daemon.
uf101d stop	Stops the server daemon.
uf101d stopall	Stops the server, then kills any hung UnForm tasks.
uf101d restart	Stops, then starts the server daemon.
uf101d admin <i>options</i>	Runs command line administration tasks. Must be root.
Windows command lines	
uf101d.exe	Displays the Server Manager window.
uf101c.exe -v	Executes a local UnForm client (uf101c.exe) to show the server version.
uf101d.exe -start	Manually starts the server (run as administrator if installed as a service)
uf101d.exe -stop	Manually stops the server (run as administrator if installed as a service)
uf101d.exe -stopall	Manually stops the server, and then terminates any hung UnForm tasks (run as administrator if installed as a service)
uf101d.exe -install	Manually install as a service (must be administrator)
uf101d.exe -uninstall	Manually uninstall the service (must be administrator)
uf101d.exe -admin <i>options</i>	Runs command line administration tasks. Must be an administrator. Use <code>-o filename</code> to prevent results showing in a message box.

Apache HTTPD Server Control

The UnForm server will automatically start and stop the Apache web server instance controlled by it, and will restart it once per day in order to ensure daily log files are managed. You can control the Apache HTTPD server independently of the main server as well, by creating transient files in the UnForm server directory. The file names are `httpd.stop`, `httpd.start`, and `httpd.restart`. The UnForm server monitors for these files and will signal the Apache server to perform the requested operation, then remove the file. You can easily create these files from a command prompt, or your own method, though you need to ensure that the UnForm server can erase the file when its action is complete to prevent continuous attempts.

On Linux systems, use the touch command: `'touch httpd.restart'`, for example.

On Windows systems, use the echo command: `'@echo XX>httpd.restart'`.

5.2 Command Line Admin

An administrator user can use the uf101d admin command line options. Such a user is either root on Linux, or a member of an administrators group on Windows. The following options are supported.

All commands are structured as **uf101d admin *options***

Note that square brackets [] indicate optional arguments and are not part of the syntax.

The *options* that are available are:

- **-setpass "*userid*" -password "*password*"**
Sets the user's password to the specified value. If *password* is "", then a random password is generated. The new password is displayed if successful.
- **-getpass "*userid*"**
Displays the specified user's password.
- **-setemail "*userid*" -email "*email-address*"**
Sets the user's email address to the specified value. The new email is displayed if successful. Note if the user's current password doesn't meet complexity requirements, a message about the password will appear due to the inability to update the record using the existing password.
- **-getemail "*userid*"**
Displays the user's current email.
- **-setconfig [-section "*sectionname*"] -item "*itemname*" -value "*value*" [-force]**
Updates a line in the uf101d.ini file. If no section is provided, defaults is used. If there is an existing item in the named section, its value is updated. If the item doesn't exist in the section, and -force is used, the item is created. Further, if values affecting the local client are updated, such as the server port, the corresponding entry in uf101c.ini is updated.
- **-getconfig [-section "*sectionname*"] -item "*itemname*"**
Displays the specified item's value from the uf101d.ini file. If no section is provided, defaults is used.
- **-o *filename*** writes any response to a file. Without this option, results are displayed on screen.

Examples

uf101d admin -getpass amber -o output.txt writes user amber's password to output.txt

uf101d admin -setconfig -item agetmp -value .25 sets agetmp=.25 in the [defaults] section of uf101d.ini

uf101d admin -setconfig -section security -item authkey -value abc123 sets authkey=abc123 in the [security] section of uf101d.ini, and also in the [defaults] section of uf101c.ini.

5.3 Client Options

The uf101c client program offers many options, which control various aspects of how it communicates with the server and how the server is told to execute the job. Note that if the command line becomes too long

for the operating system, you can use the `-z` or `-zx` options, which cause command line options to be read from a text file.

On Windows, there are two clients:

- `uf101cc.exe` is a console application that supports options like the Unix client does.
- `uf101c.exe` is a graphical application that interprets some command line options and then executes a hidden console client.

Standard Options	
Option	Description
<code>-300</code>	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.
<code>-about</code>	A Windows-only option that displays information about the client, including the location of the active <code>uf101c.ini</code> file.
<code>-afo2</code> <code>-noafo2</code>	<p>Enables or disables a newer algorithm for "word" parsing of AFO print streams. This algorithm, which is new in version 9, works hard to produce words rather than relying on text sequences produced by the printing application.</p> <p>This can be made the default for all jobs by setting <code>afo2=1</code> in <code>uf101d.ini</code>, in which case the <code>-noafo2</code> option turns it off for the job.</p>
<code>-async</code> <code>-asyncrpq /path/to/rpq</code>	<p>Submits a print job for asynchronous processing. In this mode, the input file and the command line are submitted to the server in the <code>rpq</code> directory, where subsequent processing is handled by the server, using the same mechanism used for TCP/IP Monitor printing. Normal job processing, including job license management, is delegated to the server, allowing client processing to complete as long as it can connect to the server.</p> <p>Async submissions are not compatible with client-side output and report an error if the job utilizes that. Therefore, jobs require a <code>-o</code> option, and that value cannot specify a "client:" prefix. Any client-side error file will only log problems connecting to the server.</p> <p>The <code>-asyncrpq</code> option enables lower-overhead submission if the client and server are on the same machine, and the client connects to the server named 'localhost'. Instead of transferring the input and command files by socket, it simply creates</p>

Standard Options	
Option	Description
	<p>them in the rpq directory specified. Specify the full path to the rpq directory, without a trailing slash. The <code>-asyncripq</code> option implicitly turns on <code>-async</code>.</p> <p>See also the <code>-priority</code> option.</p>
<code>-authkey keyvalue</code>	Specifies an authkey for the connection, which must match the authkey setting configured in the server's <code>uf101d.ini</code> file (in the [security] section). Note this value is often configured in the <code>uf101c.ini</code> file.
<code>-c copies</code>	Causes UnForm to issue multiple copies of the entire report. This differs from the <code>-pc</code> option. If <i>copies</i> is set to less than 2, this option is ignored. This option and the <code>"-pc"</code> option are mutually exclusive; also, rule sets can specify copy options that will override command line options.
<code>-ce copies-enabled</code>	Performs implicit skips of any rule set copy NOT found in the <i>copies-enabled</i> list. For example <code>-ce "1,2"</code> would force copies other than 1 and 2 to be suppressed. This option is useful in sub-jobs executed with the <code>jobexec()</code> function or via the archive command to suppress certain copies.
<code>-ci</code> or <code>-color</code>	Forces pcl image conversions to retain color rather than force black and white. See the image command for more information about automated image conversion and scaling. This also implicitly sets the <code>-gw</code> option.
<code>-cols n</code>	Sets the default columns per page when a job is using default scaling, as when the <code>-p pdf</code> or <code>-p laser</code> options are used and no rule set is detected or specified. See also the <code>-rows</code> option.
<code>-compress</code> or <code>-cmp</code> <code>-nocompress</code>	<p>The <code>-compress</code> or <code>-cmp</code> options will force compression of PDF files, even if the best compression available is RLE. If the operating system supports zlib compression, then Flate compression is turned on by default and this option is redundant.</p> <p>If you want to disable the automatic Flate compression, use the <code>-nocompress</code> option.</p>
<code>-cmptrans</code>	Attempts to compress portions of the transmission to a remote (non-localhost) server. If the client and server both support zlib compression, then uploaded and downloaded files are compressed to improve performance over slow network connections.
<code>-config</code>	A Windows-only option that displays a configuration form and updates the client's <code>uf101c.ini</code> file. See the

Standard Options	
Option	Description
	-about option for the location of this file.
-cover "ruleset [,rulefile [,args]]"	Generates a cover page for the job using the rule set specified. If a rule file is specified, the rule set is read from that rule file. If arguments are specific, the subjob used to generate the cover page will include the specified command line arguments. The arguments can be used to pass parameters and other processing options, but should not include -f, -r, or -p arguments.
-cr 0 1 2 3	<p>Sets handling for embedded carriage returns (chr(13)) in lines read from the input stream. The default value is defined in uf101d.ini, in the [defaults] section, cr=<i>n</i> entry.</p> <ul style="list-style-type: none"> • 0 will truncate lines at the first CR. • 1 will strip CR character, so the line continues as if the character were not present. • 2 will fold lines, and non-space characters are placed in the line buffer, simulating an overstrike. • 3 will fold lines and insert an extra space, which accommodates Windows Generic/Text Only printers that overstrike conflicting characters. <p>Note if UnForm detects line-terminators are CR characters rather than LF or CRLF sequences, this option is not operational.</p>
-debug -nodebug	<p>Causes job-based submission <i>serverpath/temp/</i> files *.in, *.out, and *.err to be retained rather than deleted after job completion. It also causes generation of the following job-based files: *.eml log file for email operations, *.gs.log for ghostscript operation error and standard output (Unix or console version Windows only).</p> <p>Use -nodebug to turn off default debug mode defined in uf101d.ini.</p>
-e <i>error-file</i>	Causes UnForm to output any errors to the file specified. Error files reside on the client system, not the server.
-emattach " <i>value</i> " -embcc " <i>value</i> " -emcc " <i>value</i> " -emfrom " <i>value</i> " -emlogin " <i>value</i> " -emmsgtxt " <i>value</i> " -emoh " <i>value</i> " -empswd " <i>value</i> " -emsubject " <i>value</i> "	These options supply values for an automatic email command. See the email command documentation for descriptions of each option. The -emto option is required, all others are optional, though certainly the -emsubject and -emmsgtxt are likely required for a given application. For emailing to work, the job must be a PDF job, and the server's mailcall.ini file must be properly configured with a server= line defining the SMTP server.

Standard Options	
Option	Description
-emto " <i>value</i> " -emlogfile " <i>value</i> "	
-f <i>rule-file</i> -f " <i>file1;file2;...</i> "	<p>Establishes a different rule file than the default specified during the installation. Rule files are text files that contain descriptions of the form enhancements for one or more forms. The enhancement options are described in detail under Rule Files, below.</p> <p>UnForm will always search for the rule file first in the UnForm server directory, then by the full pathname given. Rule files must reside on the server machine, not the client.</p> <p>If multiple rule files are specified, delimited by a semicolon, they are merged, with the global regions merged in reverse sequence.</p> <p>By convention, rule files have a .rul suffix, though this is not a requirement, and the <i>rule-file</i> value can be any file name. The UnForm Designer tool maintains a .rud suffix for working rule files and a .rul suffix for published rule files.</p> <p>Note that if you want to ensure that no rule file is used, such as when using -i and -o options to transfer a file, you can specify -f /dev/null (Linux) or -f nul (Windows).</p>
-fitpage	Turns on auto-scaling of AFO input streams, by requesting Ghostscript to scale incoming pages to the paper size, which defaults to Letter and can be changed with the -paper option. This feature is particularly useful when incoming print streams are A4 and output should be Letter, or visa versa.
-gb -greenbar [<i>options</i>]	Adds alternating shade patterns to simulate green bar paper. If the <i>options</i> parameter is supplied, it should be in the form defined by the shade command for repeating shade values. If no option value is supplied, the default is 3 lines shaded at 10%, 3 lines skipped, repeated until the end of the page.
-gs	Causes UnForm to generate laser driver shade regions graphically, rather than using internal PCL shade commands. The result is finer shading detail, especially at 600 dpi. Using this option will add between 2K and 4K per job.

Standard Options	
Option	Description
	The gs command can also be used in rule sets to control graphical shading at a copy level.
-gw	Forces UnForm to pass through PCL image width and height escape sequences to the printer. This is generally necessary on color laser images to avoid a black stripe from the right image edge to the right margin. However, if you are using PCL images, then it is important that all images on a form contain width and height values so they won't conflict with one another. Some image generating programs don't store the width and height values.
-i <i>input-file</i>	Names an input text file for UnForm to process as input. If not specified, or if it is a dash (-i -), then standard input (std input) is read. Under Windows, standard input cannot be used, so an input file must be supplied. Note that the input file must reside on the client's computer, not the server.
-ix <i>input-file</i>	Same as the -i option except the input text file is removed upon completion of task. Note that the input file must reside on the client's computer, not the server.
-land	Turns on landscape print mode as the default. A portrait command in a rule set will override this option. Note that landscape printing usually requires a reduction in the number of rows per page, as compared with portrait printing, in order to produce usable results.
-lang <i>languageext</i>	Render messages and window titles from the file <i>uf101cmsg.ext</i> . The file <i>uf101cmsg.eng</i> is always supplied with UnForm, and the default language is English.
-ldapsync -ldapserver [ssl:] <i>name:port</i> -ldapdomain <i>name</i> -ldaptype <i>name</i> -ldaplogin <i>login</i> -ldappassword <i>password</i> -ldaprofile <i>profilename</i> -ldappageing <i>pagesize</i>	See the archive options table for more details.
-lib " <i>dir[;dir,...]</i> "	Add directory (or directories delimited by semicolons) to the library of search paths used for locating external files, such as images, attachments, or merge rule files. Note that the library= line of <i>uf101d.ini</i> provides another method of doing this, and the rule file's path is also automatically added to this search list.

Standard Options	
Option	Description
-lockcols	When the label dimensions and cols setting are established, UnForm scans mono-spaced internal fonts for the closest match that will not exceed the cols specified, then recalculates the cols to agree with the font selected. This allows print stream text and all other enhancements to scale together. However, it also causes labels to shrink in printable area width, sometimes very noticeably, resulting in graphical commands not being placed where expected. This option was added to prevent this recalculation from occurring, at the expense of losing the print stream scale matching. With this option, graphical commands will print where expected on the label, but may not align with print stream output. Zebra-only command.
-m	Sets the printer model to a name that can be found in the ppd directory (without the ".ppd" suffix, e.g. -m hp4000 will load the ppd/hp4000.ppd file). This is useful when producing PostScript jobs that use printer features such as duplex or tray selection, as the code for those features is defined in PPD files provided by printer manufacturers. If no -m is provided, then UnForm will select a default PPD file based on the driver. Custom PPD files can be obtained from a printer vendor or from various Internet sources, or can be written from scratch or based upon one of the generic files.
-macros	Turns on macros.
-macrocopy <i>n</i>	Used in conjunction with the -makemacro option. A macro will be created for the designated rule set copy.
-makemacro <i>n</i>	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 -r option and can be used in conjunction with the -macrocopy option. See special section discussing macros later in this documentation.
-nn	Indicates that an error message should be issued if the input stream is empty. The value used for the error message is in the [defaults] section of uform.txt, in the entry nullmsg= <i>message text</i> .
-noafo	Suppresses the automatic assumption that Postscript/PDF input should initiate an AFO job . This flag can also be useful when using jobexec() to generate non-AFO subjobs when run from an AFO job.

Standard Options	
Option	Description
-noarc	Turns off any archive commands for the job.
-nocompress	See the <code>--compress</code> option.
-nodebug	See the <code>--debug</code> option.
-nodeliver -nodel	Turns off any deliver commands for the job.
-nofitpage	Turns off auto-scaling of AFO input streams.
-nohpgl	Reverts to full PCL, rather than a mixture of PCL and HP/GL output. A number of laser printed features use HP/GL, which is a standard feature of the PCL5 language. Some PCL interpreters, such as those that may be included in some fax or viewing software, may not support HP/GL, so this option can be used to force standard PCL5 coding for many options, such as box drawing and text alignment. A few features, such as rounded corner boxes, require HP/GL and are not supported if this option is specified.
-nointr	With this flag set, the Unix/Linux client will ignore interrupt signals once the connection to the server is established. This allows it to keep running even if a parent process receives an interrupt signal on platforms that propagate the signal to child tasks.
-noirs	Suppress possible interpretation of some binary data input files as containing an input rule set. Useful in particular when copying data files with the uniform client that happen to contain headers with values in square brackets.
-notextjob	With this flag set, UnForm will not construct the <code>textjob\$[]</code> array, saving time during parsing of the input stream.
-o <i>output-file</i>	<p>Specifies an output file or device. If not specified, then standard output (stdout) is used. Under Windows, an output file must be supplied unless one of the special drivers, <code>win</code> or <code>winpw</code>, is used. On UNIX, the output can be a redirect or pipe to another program, such as <code>lp</code> or <code>lpr</code>.</p> <p>The output device can be specified in the form <code>"tcp:nameorIP:port"</code> to direct output to a network printer at the name or address specified. If the optional <code>:port</code> is not supplied, port 9100 is used, which is the default network printing port. Example: <code>-o "tcp:192.168.1.45"</code>.</p> <p>The output device can be specified in the form <code>"cp:locid:devid:title"</code> to send output to a CirrusPrint</p>

Standard Options	
Option	Description
	<p>output device. The [cirrusprint] section of deliver.ini must be configured with server, login, and password values so the UnForm server can submit the job. The title value is optional. If not supplied, the title provided to CirrusPrint is "UnForm job <i>n</i>".</p> <p>Output names that contain spaces or characters that are meaningful to the operating system must be quoted.</p> <p>The output file or device will, by default, be opened on the server machine. If the name is prefixed with the phrase "client:", then it is returned to the client for local handling. Here are some examples:</p> <p>Server output: -o ">lp -dhplaser -oraw -s 2>/dev/null" -o "/tmp/archive/12345.pdf"</p> <p>Client output: -o client://prntrv/laser -o client:c:\archive\12345.pdf</p> <p>Note that if a Unix pipe or redirect (or >) is not quoted, output is actually to the client's standard output handle, so it is implicitly client-based. To print to a server-based spooler or program, be sure to quote the argument, as shown above.</p> <p>Server-based output can contain characters that will be substituted at run-time. These character replacements are:</p> <ul style="list-style-type: none"> • %d for the date in YYYYMMDD format • %t for the time in HHMMSS (24-hour clock) format • %p for the process ID of the UnForm task generating the file • %j for the UnForm job number, a sequential counter • %k for a millisecond value, generally appended to a %t tag <p>Note that on UNIX, if there is no -o specified, or if the output is simply a dash (-o -), then output goes to the client's standard out. A special output of /dev/tty is also recognized as client-side output to the /dev/tty device, often used for slave printing (see the -slon/-sloff options).</p>

Standard Options	
Option	Description
	<p>If the output will be handled by the server, the client will generally exit as soon as the job has been successfully started on the server. If the output is to be returned to the client (or the <code>--wait</code> option is specified), then the client will wait for the server to finish.</p>
<p><code>-o "*windev*;name"</code> <code>-o "*winprt*;name"</code></p>	<p>When UnForm is installed on a Windows system, two special print devices are available: <code>*windev*</code> and <code>*winprt*</code>. Note that the <code>*</code> characters are optional in version 10.1, and either a semicolon or colon can be used.</p> <p>Windows print queues can be referenced in raw mode (<code>*windev*</code>) or via a Windows print driver (<code>*winprt*</code>). When in raw mode, a PCL5 or PostScript driver must be used, specified with the <code>-p</code> option (some printers support direct printing of PDF output as well). When using a Windows print driver, Ghostscript must be installed and configured, which allows UnForm to create temporary PDF output, convert that to image output, and print the images using any Windows print driver (the <code>-p</code> option is ignored). This Ghostscript-driven method is suitable for local printers, but not for remote printers accessed over a slow network connection, due to the size of output generated. Each page is a raster bitmap, and such full page images are large files (i.e. letter-size, 300 dpi, black and white is about 1MB, color about 8MB).</p> <p>The <i>name</i> value specified must match a Windows printer name (not a share name). If not specified, the default printer for the UnForm server process will be used (not that of the submitting user). No selection window can be presented for dynamic selection, as UnForm jobs run in background on a server.</p> <p>When using <code>*winprt*</code>, limited printer control is offered on a job-wide basis, for tray, duplex, and orientation. Color output is generated if a <code>--color</code> command line option is used. As the output is produced initially in PDF format, it is not possible to change the output device in mid-job. You can set output dynamically at the start of a job in order to override the <code>-o</code> command line argument, using an output command or setting <code>output\$</code> in a prejob code block, using the same <code>"*winprt*;name"</code> syntax.</p>

Standard Options	
Option	Description
	Tray numbers differ from PCL trays, often being manufacturer-defined values above 256. A list of tray numbers for a given Windows printer can be obtained using the system object's <code>winprttrays\$(printer\$)</code> method.
<code>-o unc:\\server\share</code>	This special Windows UNC printer syntax is designed to print indirectly to the device via a work file and a Windows copy command. This technique works around a limitation of Windows 2008 that prevents printing to UNC printer shares. If UnForm detects it is running on 2008 or higher, it automatically implements this technique, but if needed, the <code>unc:</code> prefix can be specified to force this processing method.
<code>-p output-format</code>	<p>Specifies the output format for the job. It may be one of the following values:</p> <p>laser (or pcl), which produces PCL5 or PCL5c (color) output. The default format is PCL5, but if this option is specified, and no rule set is detected or specified, then the output is scaled to fit the page in conjunction with the <code>–cols</code> and <code>–rows</code> options, or the content itself. Without any <code>–p</code> option, and without a rule set, the job is passed through unmodified.</p> <p>pdf, which generates files viewable by Adobe Acrobat Reader or PDF viewers. If no rule set is detected or specified, then a scaled text job is created, based on the <code>–cols</code> and <code>–rows</code> options, or the content itself.</p> <p>ps or ps3, which generates PostScript output. If no rule set is detected or specified, then a scaled text job in PostScript format is created, based on the <code>–cols</code> and <code>–rows</code> options, or the content itself. The ps3 version leverages the zlib compression support of PostScript level 3, which is supported on many printers, for monochrome images.</p> <p>html5, which generates HTML5 output, generating page oriented documents similar to PDF or PCL (this differs from the legacy "html" driver).</p> <p>eps, which generates a PostScript EPS image from the first page of output.</p> <p>zebran, which produces ZPL II output at <i>n</i> dots per mm (6, 8, or 12 – default of 12) for Zebra label printers.</p>

Standard Options	
Option	Description
	<p>For special Zebra media handling, you can append the following to zebran:</p> <ul style="list-style-type: none"> Media tracking (Y=standard, N=non-standard label stock). Standard label stock is non-continuous, meaning the media has some type of physical characteristic (web, notch, perforation, mark, etc.) to separate the labels. NOTE: changing between standard and non-standard requires recalibrating the printer. Set print modes (T=tear-off, R=rewind, P=peel-off, C=cutter). <p>The default values are YT. For continuous labels, 8 dpmm, with a cutter, you would specify -p zeбра8NC.</p> <p>html, which generates Web pages from reports, based on a special set of rule set keywords.</p>
-p <i>output-format</i> (continued)	<p>win, winpw, which automatically produces a PDF file and launches the Acrobat PDF viewer on the Windows client. win (note win5 is a synonym for win) will print the document using the Acrobat /p command line option. This generally provides a printer selection dialog before printing. On some versions of Acrobat a window is left open after printing. A second format, "win:printer", uses the Acrobat /t option to print directly to the printer named <i>printer</i>. Printer names generally match the names in the Windows printer select dialogs, but sometimes can be UNC names. To print to the default printer, specify win:default or win:dflt. These options only work in Windows clients. The winpw driver works on Windows and OS/X (Mac) systems, and also in Unix/Linux systems that have a graphical viewer that is specified in the PDF_VIEWER environment variable.</p> <p>Image drivers Special Ghostscript-driven drivers are also available if Ghostscript is available on the server machine, and if you have configured the uf101d.ini file [drivers] section. The configuration specifies the path to Ghostscript and a set of driver names with Ghostscript -sDEVICE names, a multi-page flag, and a resolution. For example:</p> <pre>[drivers] gs=gs bmp=bmp256,1,300</pre>

Standard Options	
Option	Description
	<p>If the command line contains <code>-p bmp -o imagefile.bmp</code>, then UnForm will generate an interim PDF file, and execute the <code>gs</code> command to convert that to the format <code>bmp256</code>, with output files <code>imagefile-1.bmp</code>, <code>imagefile-2.bmp</code>, and so on. The images will be produced at 300 dpi resolution.</p> <p>Many standard drivers are configured, and you can add more as needed and as supported by Ghostscript.</p>
<code>-page lines</code>	<p>Specifies the number of lines per page that UnForm should read from the input. Normally, UnForm will find form-feed characters to delimit pages. However, if the application simply prints even numbers of lines per page, this can be used to define that value so UnForm can properly parse the input stream. The rule file <code>page</code> 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 <code>-x</code> option) to properly parse individual pages.</p>
<code>-paper paper</code> <code>-ps paper</code>	<p>Specifies the paper size used by the printer. Valid values include <code>letter</code>, <code>legal</code>, <code>ledger</code>, <code>executive</code>, <code>a3</code>, and <code>a4</code>. The default is <code>letter</code>. For a complete list, see the <code>[paper]</code> section of <code>ufparam.txt</code>.</p> <p>The paper size can also be specified with a <i>widthxheight</i> setting, with an optional suffix of <code>"cm"</code> or <code>"mm"</code> to specify the value in centimeters or millimeters (i.e. <code>20x30cm</code>).</p>
<code>-pc copies</code>	<p>Causes UnForm to issue multiple copies of the report, page by page. If <i>copies</i> is less than 2, this option is ignored. This option and the <code>"-c"</code> option are mutually exclusive; also, rule sets can specify copy options that will override command line options.</p>
<code>-pdfauthor "value"</code> <code>-pdfkeywords "value"</code> <code>-pdfprotect "value"</code> <code>-pdfsubject "value"</code> <code>-pdftitle "value"</code> <code>-pdftrans -pdfnotrans</code>	<p>These options supply default values for the author, keywords, protect, subject, title, and transparency commands, respectively. All options are used exclusively with PDF output.</p>
<code>-ping</code>	<p>Queries the server, returning a 10-digit string (plus a newline) , made up of 5 digits of total job licenses and 5 digits of job licenses in use at the time the connection is made. The <code>-server</code> and <code>-port</code> options are honored. On Windows, using <code>uf101c.exe</code>, the <code>-o</code> option and <code>-e</code> options are honored.</p>

Standard Options	
Option	Description
-port <i>n</i>	Specifies the port that the server is listening on, if other than the default of 27400. The <code>-server</code> line can also be used to specify the port, in the format <code>server:port</code> . The <code>uf101c.ini</code> file also can contain the default port to use in the absence of this option.
-printblanks -pb	Causes UnForm to process blank pages the same as non-blank pages. Normally, blank pages are suppressed.
-priority " <i>value</i> "	Sets the priority of <code>-async</code> job submissions. Valid values are "high" or 1, "medium" or 2, or "low" or 3.
-prm " <i>parameters</i> "	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 (use the <code>GBL()</code> function to retrieve), and each is set to the <i>value</i> specified. Multiple parameters need to be delimited by semi-colons (;). -prm "company=01;name=Acme Paint" , for example, would establish two global strings: <code>company</code> and <code>name</code> . These could be referenced within code blocks (prepage, precopy, etc.) as <code>GBL("company")</code> and <code>GBL("name")</code> .
-proxy <i>http-proxy</i>	This option enables connection via an http server that has been configured as a forward proxy supporting the <code>CONNECT</code> verb. If used, the connection to the UnForm server is accomplished by first connecting to the proxy server over <code>http/https</code> , then issuing a <code>CONNECT</code> request to the standard server setting found in <code>uf101c.ini</code> or the <code>-server</code> option. The http server establishes a transparent connection to the UnForm server. The <i>http-proxy</i> value must be a <code>http</code> or <code>https</code> url.
-quiet	Forces the Windows version of <code>uf101c</code> to route any errors to the log file defined in <code>uf101c.ini</code> , or " <code>uf101c.log</code> " by default, and to any <code>-e</code> file named on the command line. Without this option, errors are reported in message boxes.
-r <i>rule-set</i>	Used to specify a rule set name to use for the job. The rule set specified must exist in the rule file used for the job (see the <code>-f</code> option). If this option is not used, UnForm will attempt to automatically detect what form is being processed based on

Standard Options	
Option	Description
	specifications contained in the rule file. If no form is detected, then UnForm creates a simple text job or may pass the job through to the output unmodified. If the <i>rule-set</i> contains spaces, it should be quoted. Rule set names are not case sensitive.
-rd <i>n</i> , -rdelay <i>n</i>	Introduce a delay of 1 to <i>n</i> seconds (a random value) to slow down the pace of jobs submitted to the server when large numbers of jobs are sent by an application. The delay is only imposed if the number of active jobs on the server exceeds a threshold defined by the -rdt option. For example, the following options would implement a random delay of from 1 to 10 seconds whenever more than 5 jobs are active on the server: -rd 10 -rdt 5
-rdt <i>n</i> , -rthresh <i>n</i>	The minimum number of active jobs on the server before the -rd or -rdelay option will be active. If active jobs exceeds this value, a random delay is imposed.
-rerun [<i>files</i>]	Used in job recovery to rerun specified jobs associated with a list of files found in the fail history client-side directory. See Fail Recovery for more information.
-retry <i>count</i>	Sets the number of times the client will retry a connection if it receives an error 998 (out of licenses) response from the server. This is related to the -sleep option, which determines how long to pause between retries.
-rland -rport	Turn on reverse landscape or reverse portrait orientation. These options are only valid on laser output.
-rows <i>n</i>	Sets the default rows per page when a job is using default scaling, as when the -p pdf or -p laser options are used and no rule set is detected or specified. See also the -cols option.
-s <i>sub-file</i>	Specifies a text file to be used as a substitution file. Substitutions are used by UnForm when placing text in the form output. If the text can vary from one form to another, such as company names and addresses, then multiple substitution files can be defined, each containing different names and addresses, and the proper one identified with this command line option. See the text keyword for more information. The default substitution file is called "subst". If <i>sub-file</i> is not a full path, UnForm will look for it in the UnForm directory. UnForm will

Standard Options	
Option	Description
	automatically generate stbl("@name") definitions for each line in the substitution file. Code blocks and expressions can use the stbl() function (gbl() on ProvideX) to return these values.
-serialize	Causes the client to block other client processes until the server has acknowledged the connection, effectively spooling client processes until the server has resources to support additional connections. This can help to overcome server resource issues when a swarm of jobs is sent to the server very quickly. The blocking is accomplished with a lock file, in the client install path on Unix, or in the % ProgramData%\SDSI path on Windows.
-server <i>server</i>	<p>Specifies the server, if the default server found in uf101c.ini is incorrect. The <i>server</i> value can be a hostname or IP address of the system running the UnForm server.</p> <p>Optionally, the server can be specified with up to 5 segments, delimited by colons. Unspecified segments take values from uf101c.ini or specific command line arguments. The format order is:</p> <p><i>servernameOrIP</i> <i>:port:authkey:proxynameOrIP:proxyport</i></p> <p>The <i>server</i> argument can contain a semicolon delimited list of server values (be sure to quote it). If it is a list, then when submitting jobs the client will check each for availability, depending on the uf101c.ini setting for minavail=<i>n</i>. If <i>n</i> is 0, the client looks for the server with the most availability. The algorithm is based on the ratio of print licenses to licenses in use. If <i>n</i> is not 0, then the first server with at least this many job slots available is chosen. No matter which method is configured, the first server with no jobs in use is chosen.</p> <p>When not submitting jobs, such as with a -v option, only the first server in the list is used.</p> <p>Note that if you use archiving, it is important to set the archive server, generally with the [archive] section server= option in uf101d.ini, to ensure that archive libraries are updated on a primary archive server no matter which server is chosen to handle the print job.</p>

Standard Options	
Option	Description
	If multiple servers are specified, they all must use the same authkey value.
-shift <i>n</i>	Causes all input text to shift <i>n</i> columns to the right, similar to the action of the shift command. This can be useful in conjunction with the -x crosshair option to force text to match the alignment it would have with a shift <i>n</i> command in a rule set.
-sleep <i>seconds</i>	This sets the delay between retries when reconnecting after an error 998 (out of licenses) from the server. This is related to the -retry option.
-slon " <i>codes</i> " -sloff " <i>codes</i> "	<p>Causes local (client side) output to be started with the slon <i>code</i> and ended with the sloff <i>code</i>. This option is only supported in the UNIX client. The <i>code</i> can contain text and special escaped characters:</p> <p>\e Escape \n Newline \r Carriage return \0<i>nn</i> Octal character <i>nn</i> (i.e. \033 is an escape) \x<i>hh</i> Hex character <i>hh</i> (i.e. \x1b is an escape)</p> <p>These values are typically set in conjunction with a -o /dev/tty option, in order to send a job back to the client-side terminal device for slave printing. Use of these options also causes the UNIX client to attempt to change the stty setting of the -o device to "raw" for the duration of the output.</p> <p>A typical slave print client command line might look like this:</p> <pre>cat sample1.txt uf101c -f simple.rul -slon "\e[5i" -sloff "\e[4i" -o /dev/tty</pre>
-ssl -nossl	Turns SSL on or off when connecting to the server. The default value is set in uf101c.ini, and these options override the value. This setting must match the server's configuration.
-status -nostatus	Overrides the default behavior of the status window when submitting jobs in the Windows client uf101c.exe. The default behavior is to show the window for jobs that will be returned to the client, and not show the window for jobs that will be printed by the server.
-sshost " <i>server.port</i> "	Sets the support server host and port, overriding the sshost= and ssport= lines in uf101d.ini for the job. See also the sshost() code block function.

Standard Options	
Option	Description
-sstimeout secs	Overrides the Support Server timeout value specified in the uf101d.ini file. Set to -1 to set an infinite timeout.
-textjob	If this is specified, then the textjob\$[all] array is built even if that is not the default setting, as defined in textjob=x in uf101d.ini.
-testpr <i>font symset</i>	<p>Generates a test print showing nearly all characters (ASCII 1 to 254) 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 output. Note that with the pdf driver, the only symbol set used is 9J.</p>
-timeout <i>n</i>	Sets the socket timeout, for connecting to the server, to <i>n</i> seconds. If the server takes more than this amount of time to accept the connection, the client produces an error. The default value is 10 seconds.
-trans " <i>filename</i> "	Specifies a translation file, used to substitute text, barcode, and search values that are dynamically translated as the job runs. The active file can also be specified with the settrans(" <i>filename</i> ") code block function.
-usess	Forces the use of the Windows Support Server for image conversions and Ghostscript execution, even if server-based configuration is enabled.
-v	Causes UnForm to print version information and exit.
-vshift <i>n</i>	Causes all input text to shift <i>n</i> rows down, similar to the action of the vshift command. This can be useful in conjunction with the -x crosshair option to force text to match the alignment it would have with a vshift <i>n</i> command in a rule set.
-wait	Causes the client to wait for job completion, even if the server is printing the job. Normally, when the client submits a job to the server, it will exit as quickly as the server acknowledges the job has started (not, of course, if the output needs to come back to the client). By including the -wait option, the client will wait until the server job is complete, even if the output will be handled by the server. The purpose of this option is to allow client reporting of

Standard Options	
Option	Description
	any errors the server might encounter once the job starts.
-x [<i>page[,page, ...]</i>] -xl [<i>page[,page, ...]</i>]	<p>Causes the first page of input, or the pages specified, to be printed with a cross hair pattern. This is typically done once to assist in determining placement of text, and then removed. Sometimes, a special printer definition is set up within an application, using the -x option, so that any form can be printed to that printer for layout purposes. Note that setting the environment variable UFC to "y" will cause this option to be automatically implemented.</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. For example, '-x 1,3-5' would produce cross hair patterns on pages 1, 3, 4, and 5, suppressing all others. If the input doesn't contain form-feed page delimiters, be sure to use the -page option as well.</p> <p>When the -x option is used, no rule set is applied to the job. See the crosshair command if you want to apply a grid to enhanced output.</p> <p>The -xl option will produce a landscape version of the crosshair printing.</p>
-z <i>filename</i> -zx <i>filename</i>	<p>Adds command line options contained in the text file <i>filename</i> to the command line as if they were part of the command line itself. This option is helpful if a command line length exceeds the operating system limit. If the -zx option is used, then <i>filename</i> is erased once it has been read.</p> <p>The file is simply a text file with arguments separated by white space or new lines. Lines beginning with a # character are not included.</p>

5.3.1 Archive Options

Option	Description
-arcargs "args"	Sets the default archiving subjob options. When archives are written via UnForm jobs (as opposed to the <code>-arcput</code> option), a PDF file is generated as a subjob while the UnForm job is processed. The subjob may require UnForm command line options to run properly. This option sets those options. The archive command can also set options.
-arccats "cats" -arccategories "cats"	Sets the archive document category indexes for document writing. There can be any number of semi-colon delimited categories, with each category supporting up to three pipe-delimited levels. Each level can be up to 20 characters long. A different character than the pipe can be use by specifying a <code>-arcsep</code> option. Generally this option must be enclosed in quotes to protect semi-colons, pipes, and spaces in the command line. The format structure is this: "cat1.1 cat1.2 cat1.3;cat2;cat3.1 cat3.2;..."
-arccount <i>n</i>	Specifies the maximum number of records to return in some <code>-arclist*</code> options. Useful along with <code>-arcfirst</code> to return pages of information to scripts.
-arcdel	Triggers removal of an archive image or document, honoring the options for <code>-arclib</code> , <code>-arcdotype</code> , <code>-arcdocid</code> , <code>-arcsbid</code> . Delete access to the library is required for the <code>-arclogin</code> user. If a <code>-arcsbid</code> is supplied, then just that image is removed. If no <code>-arcsbid</code> is supplied, then the document record and all sub-images are removed.
-arcdocdata "name=value"	Sets a name-value pair for updating doc data values, or for searching. Multiple pairs can be separated by semicolons, or multiple <code>-arcdocdata</code> options can be specified.
-arcdocid "docID"	Sets the archive document ID for document writing or retrieval.
-arcdotype "doctype"	Sets the archive document type for document writing or retrieval.
-arcdtm "ymddate" -arcdate "ymddate"	Sets the archive document date for document writing to the date specified. Normally this value is calculated automatically. The date supplied must be in the format <code>yyyymmddhhmmss</code> , with the first 8 characters required.
-arcdtmupdated "ymddate" -arcdateupdated "ymddate"	Used with the <code>-arcsearch</code> command to filter the date/time updated. Setting this does not affect the date updated stamp when writing archive records.
-arcend "end"	Specifies an ending point for <code>-arclist</code> and <code>-arclistdocs</code> options, allowing for a range a range of documents to be returned. The ending point should logically be associated with the order specified in the <code>-arcorder</code> option. If the order is "category", then <i>end</i> can contain pipe separators to indicate category level breaks. If the order is "docdata", then a docdata name can be supplied.
-arcentityid "entityid" -arcentid "entityid"	Sets the entity ID for document writing, either the default for UnForm jobs or the value for direct (<code>-arcput</code>) document writing.
-arcexists	Returns a 1 (found) or 0 (not found) plus a CRLF to the output device, typically <code>-o client:filename</code> , based on testing the existence of <code>-arclib</code> , <code>-arcdotype</code> , <code>-arcdocid</code> , and <code>-arcsbid</code> options. This option does not require a login. The number of options supplied determines the test performed, from simply testing

Option	Description
-arcargs " <i>args</i> "	Sets the default archiving subjob options. When archives are written via UnForm jobs (as opposed to the <code>-arcput</code> option), a PDF file is generated as a subjob while the UnForm job is processed. The subjob may require UnForm command line options to run properly. This option sets those options. The archive command can also set options.
	for library existence, to looking for any documents of a specified document type, to testing for a specific document ID, then finally image sub ID.
-arcfilter " <i>filter</i> "	Specifies a filter string to apply to the <code>-arclist</code> and <code>-arclistdocs</code> options. This filter applies to all elements of the document, including title, keywords, and categories. If the filter starts with a tilde (~), the remaining characters are interpreted as a regular expression. Otherwise, wildcard characters * and ? are supported for matching any characters or any single character.
-arcfirst <i>n</i>	Used by certain <code>-arclist*</code> options, typically along with <code>-arccount</code> , to return a portion of results. Useful when designing scripts to return pages of information.
-arcflowid " <i>flowid</i> "	Can be specified during a <code>-arcput</code> operation, so that after the image has been added to the library, a DocFlow is started in the specified flow ID.
-arcget	Triggers retrieval of an archive image, honoring the options for <code>-arclib</code> , <code>-arcdoctype</code> , <code>-arcdocid</code> , <code>-arcsubid</code> , and <code>-o</code> . Read access to the library is required for the <code>-arclogin</code> user.
-arcimport " <i>from lib</i> "	Used to import a library from a previous version of UnForm, or an sdStor library.
-arckeywords " <i>kws</i> "	<p>Sets the archive document keywords for document writing. There can be any number of semi-colon delimited list of words or phrases. If none is set, then documents inserted by UnForm jobs (as opposed to those written via a <code>-arcput</code> option) will have a configurable number of unique keywords generated automatically from input file content. Generally this option must be enclosed in quotes to protect semi-colons and spaces in the command line. The format structure is this:</p> <p>"kw1;kw2;..."</p>
-arclib " <i>libpath</i> "	Sets the archive library path for document writing or retrieval.
-arclink " <i>links</i> " -arclinks " <i>links</i> "	<p>Sets the archive document links, which is a semi-colon delimited list of links to web documents, either external or other archive documents. Each link in the list can be in one of the following formats:</p> <ul style="list-style-type: none"> • A full URL, optionally matching a URL used to load a document or image from a library, or a URL to an outside page or document. This structure, if it begins with <code>http://</code> or <code>ftp://</code>, can be prefixed with a title in the format of <code>title=URL</code>. If the title is specified, that becomes the visible link in the browser. • A simplified pipe-delimited structure of <code>library doctype docid[subid]</code>, which is displayed in the browser interface as a URL link to the document or image named by library, document type, document ID, and optionally image sub ID. <p>There can be any number of links in the list.</p>

Option	Description
-arcargs "args"	Sets the default archiving subjob options. When archives are written via UnForm jobs (as opposed to the <code>-arcput</code> option), a PDF file is generated as a subjob while the UnForm job is processed. The subjob may require UnForm command line options to run properly. This option sets those options. The archive command can also set options.
-arclist	Triggers a listing of archives, honoring the options for <code>-arclib</code> , <code>-arclistfmt</code> , <code>-arcorder</code> , <code>-arcstart</code> , <code>-arcend</code> , and <code>-arcfilter</code> . This listing displays the document archives, but not the image sub ID's. Use the <code>-arclistdocs</code> for this additional information. The <code>-o</code> option can be used to send the results to a file.
-arclistcats	List category segments in a library, honoring the <code>-arcstart</code> option to provide segment seeds, and <code>-arcfirst</code> and <code>-arccount</code> options to limit results.
-arclistdocs	Triggers a listing of archives and image sub ID's, honoring the options for <code>-arclib</code> , <code>-arclistfmt</code> , <code>-arcorder</code> , <code>-arcstart</code> , <code>-arcend</code> , and <code>-arcfilter</code> . This listing displays the document archives plus their associated images. The <code>-o</code> option can be used to send the results to a file.
-arclistkeywords	Lists keywords in the library, honoring <code>-arcstart</code> to start the list at a certain position, and the <code>-arcfirst</code> and <code>-arccount</code> options to limit results.
-arclistkeywordseeds	Lists keyword segments, such as those starting with "a" or "ab", honoring the <code>-arcstart</code> option to specify the starting prefix, and the <code>-arcfirst</code> and <code>-arccount</code> options to limit results.
-arclisttypes	Triggers a listing of document types in the library specified with an associated <code>-arclib</code> option. The <code>-o</code> option can be used to send the results to a file. A <code>-arclogin</code> is typically required. The listing is a single text column of values, one document type per line.
-arclistfmt <i>fmt</i>	<p>Sets the format for <code>-arclist</code> and <code>-arclistdocs</code>. It must be one of the following: <code>tab</code>, <code>csv</code>, <code>pipe</code>, <code>html[:stylesheet]</code>, or <code>xml xmlf[:stylesheet]</code>.</p> <p>If the format is <code>xml</code>, a XSLT stylesheet can be referenced by URL with a colon suffix: <code>"xml:http://somewhere.com/stylesheet/mystyle.xml"</code>. Likewise, if the format is <code>html</code>, a CSS stylesheet URL can be referenced with a colon suffix: <code>"html:http://somewhere.com/default.css"</code>. An alternate format, <code>"xmlw3c:stylesheet"</code> can be used to generate the header <code>type= tag</code> as <code>"application/xsl"</code> rather than <code>"text/xsl"</code>.</p> <p>If stylesheets are specified, the application that views are parses the document must have access to the specified stylesheet.</p> <p>If the format is <code>xmlf</code>, then a base64-encoded version of the image file is included with the xml output.</p>
-arclistlibs -arclistwlibs -arclistwlibs	Triggers a listing of libraries, including path, description, creation date, and default permissions. This list returns libraries that are readable by the user specified with the <code>-arclogin</code> option. Alternate options return libraries with read/write access, or full access.
-arclogin "userid/pswd" ask	Sets the login user ID and password (they must be separated by a slash - /) for the command. Logins are required for many archive command line operations. Under some circumstances, login information can be read from files. See the Document Archiving and Management chapter for more details.

Option	Description
-arcargs " <i>args</i> "	Sets the default archiving subjob options. When archives are written via UnForm jobs (as opposed to the <code>-arcput</code> option), a PDF file is generated as a subjob while the UnForm job is processed. The subjob may require UnForm command line options to run properly. This option sets those options. The archive command can also set options.
	Optionally, enter the word "ask" and the client will prompt for the login information. Note the Windows console client will not suppress user input when prompting. The Unix client does suppress display of input.
-arcnotes " <i>notes</i> "	Sets the archive document notes for document writing. To force line breaks, insert <code>\n</code> mnemonic character sequences.
-arcorder <i>order</i>	Sets the order of the lists retrieved by the <code>-arclist</code> and <code>-arclistdocs</code> options. The order must be one of the following: id, date, title, docdata, or category. The default order is id.
-arcput	Triggers writing of a file directly to a library, bypassing UnForm processing of the input file. The option requires values for <code>-arclib</code> , <code>-arcdotype</code> , <code>-arcdocid</code> , <code>-arcsbid</code> , and <code>-i</code> , and supports the archive property setting options such as <code>-arctitle</code> , <code>-arckeyword</code> , etc. Review the archiving chapter for more details.
-arcsep <i>char</i>	Sets the separator character for category levels to <i>char</i> . The default is <code> </code> .
-arcsearch	Triggers a search to be performed, in conjunction with the <code>-arclib</code> , <code>-arcdotype</code> , <code>-arcdocid</code> , <code>-arctitle</code> , <code>-arcentityid</code> , <code>-arcddate</code> , <code>-arckeywords</code> , <code>-arcnotes</code> , <code>-arccats</code> , <code>-arclinks</code> , <code>-arctext</code> , <code>-arcsbid</code> , and <code>-arcdocdata</code> option(s). The search results are sent to the <code>-o</code> output file. The format of the results is controlled by the <code>-arclistfmt</code> option.
-arcsearchdocs	Identical in function to <code>-arcsearch</code> , except that image sub ID information is additionally returned.
-arcserver " <i>name[:port]</i> "	Sets an archive server name and optional port for archive commands. This overrides the <code>server=</code> entry the [archive] section in <code>uf101d.ini</code> . An archive command can also specify a <code>,server</code> option, which has the highest priority.
-arcstart " <i>start</i> "	Specifies a starting point for <code>-arclist</code> and <code>-arclistdocs</code> options, allowing for a range a range of documents to be returned. The starting point should logically be associated with the order specified in the <code>-arcorder</code> option. If the order is "category", then <i>start</i> can contain pipe separators to indicate category level breaks. If the order is "docdata", then a docdata name can be supplied.
-arcsubdtm " <i>ymddate</i> "	Sets the archive document image sub ID date for document writing to the date specified. Normally this value is calculated automatically. The date supplied must be in the format <code>yyyymmddhhmmss</code> , with the first 8 characters required.
-arcsubid " <i>sub ID</i> "	Sets the archive document image sub ID for document writing or retrieval. When used in conjunction with <code>-arcget</code> , you can use two special suffixes, <code>"-<"</code> and <code>"->"</code> , to return the first or last sub ID that matches the sub ID string. For example, a value of <code>"@UnForm->"</code> would return the last sequential <code>@UnForm</code> sub ID in an auto-sequenced library. Escape the suffix as <code>"\->"</code> to look for the literal value.
-arcsubtitle " <i>title</i> "	Sets the archive document image sub ID title for document writing.

Option	Description
-arcargs " <i>args</i> "	Sets the default archiving subjob options. When archives are written via UnForm jobs (as opposed to the <code>-arcput</code> option), a PDF file is generated as a subjob while the UnForm job is processed. The subjob may require UnForm command line options to run properly. This option sets those options. The archive command can also set options.
-arctextdata " <i>value</i> "	Used in conjunction with the <code>-arcsearch</code> option to search text image data, archived with <code>@text</code> sub IDs, for the value supplied.
-arctitle " <i>title</i> "	Sets the archive document title for document writing.
-ldapsync	<p>Triggers a command line LDAP/AD sync process, which update user and group databases based on additional <code>-ldap*</code> options. This process mirrors that provided by the archive browser interface Admin, Users, Sync option.</p> <p>The command line sync always backs up the user and group database files, which are <code>ufarcusr.dat</code>, <code>ufarcacc.dat</code>, <code>ufarcgrp.dat</code>, and <code>ufarcgac.dat</code>. These files are copied to <code>.bak</code> versions. After the sync, if there are issues with user logins, you can copy the <code>.bak</code> files back, or use the Restore function found in the archive browser interface Admin, Users, Sync window.</p> <p>The additional <code>-ldap*</code> options will update the <code>ldap.ini</code> or <code>ldap.profile.ini</code> file and run the sync process. Check the server logs for messages prefixed with "LDAP sync". Only the <code>-ldaplogin</code> and <code>-ldappassword</code> are required if the <code>ldap.ini</code> or <code>ldap.profile.ini</code> have been configured to work using the browser interface sync process.</p>
-ldaprofile <i>profile</i>	Specifies the profile to use, and which <code>ldap.profile.ini</code> file to work with. If not specified, the <code>ldap.ini</code> file is used.
-ldapsrvr <i>server</i>	<p>The <i>server</i> format specifies the hostname of the LDAP/AD server. Optional <code>:port</code> suffix, and optional <code>ssl:</code> or <code>tls:</code> prefix indicating how to communicate with the LDAP/AD server. Examples:</p> <ul style="list-style-type: none"> • <code>ssl:myserver:699</code> (uses <code>ldaps</code>, non-standard port) • <code>192.168.1.130:389</code> (open, standard but specified port) • <code>tls:10.10.1.100</code> (standard port, uses STARTTLS)
-ldaplogin <i>login</i> -ldappassword <i>password</i>	Specifies the authentication values used when connecting to the LDAP/AD server. Note this is different from the <code>-arclogin</code> values used to authenticate the UnForm user that will perform the action. This must be a login that can list users and groups.
-ldaptype <i>type</i>	Specifies the type of sync process. A <i>type</i> refers to a section in the <code>ldap ini</code> file that defines how to interpret the server responses. When using the browser sync interface, types are presented as a Server Type radio selection option.
-ldappaging <i>count</i>	If there are a large number of users and groups to sync, this option enables paged processing to avoid memory errors. A common option is <code>-ldappaging 100</code> .

5.3.2 Job List Options

Option	Description
-jobs -myjobs	<p>These options trigger the viewing of jobs submitted to the server. The <code>-jobs</code> option shows all jobs submitted to the server, while <code>-myjobs</code> shows just those jobs submitted by the current user. Job records are kept for a configurable amount of time, determined by the <code>age=</code> setting in the <code>uf101d.ini</code> file on the server.</p> <p>By default, the data displayed includes the job number, date/time, user, input size, pages complete, percentage complete, and status. The <code>-detail</code> option, below, adds the rule set, driver, and error message columns.</p> <p>The output can be sent to standard output on Unix, and filtered, like this:</p> <pre>uf101c -jobs grep 'Errored' more</pre> <p>On Windows, either a <code>-o</code> option or a <code>-p winpw</code> option is required, and the output goes to the named file or to a temporary file and viewed with that file types native viewing application.</p>
-active	This option will limit the job display to jobs that are currently processing on the server.
-detail	This option will cause the job listing to include additional data, including the ruleset, driver, and any ending error message.
-jobsfmt <i>format</i>	<p>Controls the format of the jobs output. Two options for <i>format</i> are supported: <code>txt</code> and <code>csv</code>. The <code>txt</code> option produces a tab-delimited list, while the <code>csv</code> option produces a comma-separated-values list. The default format is <code>txt</code>.</p> <p>On Windows, when used in conjunction with <code>-p winpw</code>, the native application for the format (or output file specified) is used. On systems where Microsoft Excel is the default application for <code>.csv</code> files, specifying <code>-jobsfmt csv -p winpw</code> will result in a display of jobs in Excel.</p>

6 DOCUMENT ARCHIVING AND MANAGEMENT

6.1 Overview

The UnForm document archiving and management component provides a suite of archiving functions which are seamlessly added to UnForm's library of commands and tools for document enhancement and delivery.

Existing UnForm integrators and designers familiar with UnForm's unique text filter technology will find it simple, intuitive and hassle-free to add archiving commands and arguments into UnForm's rule-file oriented flow of processing. Or rule-files can be bypassed altogether to archive non-UnForm-generated documents using the familiar command-argument interface to the UnForm client software. And UnForm's separate Image Manager component can add scanned image files to the archive and match them with existing documents previously stored using manual ID matching or barcode/OCR-based image ID capture.

With an internal web server and a browser-based document retrieval interface, UnForm makes it easy to browse, search, list, view, administer, and secure archive libraries. Libraries can scale up to a theoretical capacity of 4-billion documents. Context-sensitive help links include sample page images, and help guide

the user or administrator through the browse, search and administration functions. Sample archives are included and are referenced in the help pages. They can aid in the design of a logical custom archiving library and identification structure suited to the needs of sophisticated end-users.

Flexible pre-defined and user-defined document index structures are designed to make document identification and retrieval practical, fast and easy. Pre-defined index structures exist for a two-segment type-ID index, and a date-time index. A user-defined up to ten-segment pipe-delimited category key structure is also provided for indexing. The browser-based document retrieval interface provides an intuitively sensible drill-down browse function through the levels of the multi-segmented indexes. Libraries are file-system-based locations. A three-tiered library-document-image hierarchy is employed which allows multiple versions of a document, e.g. text and pdf, to be stored together, uniquely identified by a Sub-ID index, and further allows multiple text or non-text image or data files to be attached as sub-documents to a parent. When archiving from an UnForm job, both text and pdf versions are stored automatically.

Subject to access-rights, document and images being listed and/or viewed in the browser interface can have properties modified by users to update document status, correct indexes, and maintain associated notes and keywords at a document level. Files on the network can be browsed and added as sub-documents from within the browser.

Security is managed by library and by user and/or group. All documents are encrypted and compressed when stored in the library. To access documents, a user login is required, and each login can be granted read, write, or delete access to a given library, or can be allowed to access the library based on the library's default access profile.

Groups may be defined by the administrator. Users can be assigned to one or more groups, and library access can be granted to a group rather than individual users. If a user is granted specific rights to a library, those rights are used. Otherwise, the list of groups that the user is a member of is scanned and all rights offered to each of those groups are granted to the user. If a user is assigned to one or more groups, default library permissions are not applied.

A user can also be assigned an Entity ID. This classifies the user as an *external user*. External users are given very limited access in the web browser. They can only browse records by document type and ID, or by date. Access is limited to documents that have a matching Entity ID value.

The browser-based, multi-library search function creates disk-based query-lists of documents which can be further manipulated independent of other documents in the library. The query lists can be the basis for what are known as bulk actions, which include copying to new or existing libraries, transferring to new or existing libraries, and exporting to HTML. The HTML export produces a completely self-contained, pure-HTML directory structure suitable for loading on other storage media, such as a CD/DVD, a zip file, a web site directory, etc.

Imagine, for example, exporting all of a customer's invoices from a date range to a zip archive and emailing it to them. Another example would be to off-load old documents to external storage, then purging them to free up disk space.

The separate Image Manager client provides image management and uploading into a library. Images can be extracted from file paths or email, or uploaded directly by users. Both barcode recognition and OCR recognition assist in automating document identification, and extensive scripting capabilities ensures that any inbound document processing requirements can be met.

6.2 Logical Structure Details

The structure of UnForm archiving is a hierarchical one, where an image of a document is at the bottom of a nested structure. In fact, what you may consider a "document", such as an invoice or a purchase order, is at the middle of this structure, as there can be many versions of a given document. Here is a description of this structure:

Library	<p>A <i>library</i> is a folder or directory path to the location where archived documents are stored.</p> <p>There can be one or many libraries to store documents. Access and security is controlled by library, so the process of designing a single or multiple library structure, and determining which documents will be stored in which library, needs to take into account the access rights of groups and users.</p> <p>Beneath a library path UnForm uses a file storage algorithm with a theoretical capacity for over 4 billion distinct documents, each of which can contain multiple images. All data about documents and images, as well as the images themselves, is encrypted and compressed.</p>
Document	<p>A <i>document</i> is one or more related files which share a unique <i>Document Type</i> and <i>Document ID</i> combination, different from any other document in the archive.</p> <p>The document unit in UnForm can be seen as an "envelope" or "wrapper" which surrounds one or more associated files, stored as "image" records.</p>
Image	<p>An <i>image</i> is a single file with a distinct ID (called a sub ID) which distinguishes it from other files associated with the same document. An image can be any type of file, not limited to image files. For example, when UnForm adds an archive from an UnForm print job, it adds both text and PDF images, with ID values of @text and @unform.</p> <p>Think of a "text image" of an invoice, versus a "PDF image" of the same invoice, versus a signed delivery slip "image file" pertaining to the invoice, versus a Word document "image" of the order quotation preceding the invoice, all associated with the same document ID.</p> <p>Each image within a document is identified with a unique <i>sub ID</i>.</p>

6.3 Physical Structure Details

A library is a disk directory on the machine where the UnForm server runs. This can be a full path, or it can be a simple name, in which case its path becomes "*serverpath/arc/name*", a directory in the "arc" subdirectory of the UnForm server location.

Within this directory are several data files and subdirectories, described below.

cats.dat	Category indexes
control.dat	Control file used for internal management
docdata.dat	Document user-defined data, for future use
docs.dat	Indexed document property file
docs2.dat	Additional document property data
keywords.dat	Keyword indexes
subdocs.dat	Indexed sub-document properties

images/	Directory structure that contains encrypted/compressed image data files
recover/	Directory that contains transaction data used for library recovery

Differences from 7.x/8.0

- All file names are different, so the same library directory can host both new and old formats without conflict
- Keyword indexing has been added
- The design allows for adding indexed, user-defined data in the future
- The image store does not contain shadow records, so will be smaller
- The recover directory stores transaction data by day, and provides a faster recovery facility should the library ever need to be rebuilt. It is also possible to copy the recover directory to a new system and generate a duplicate library.

If disk space becomes an issue, you can backup old recover files and remove them. In the event a rebuild is necessary, you will have to restore the old files before running the rebuild. If you don't need all the records rebuilt, such as records you'd like purged anyway, you only need to restore data files required.

6.4 Document-level Identification

The table below briefly describes the EIGHT main data elements which UnForm uses to identify documents at the document level. With the exception of the date/time stamp, and some character-separator rules enforced by UnForm on some of the fields, the data format for each of these text elements is user-configurable. A significant part of the administrator's implementation process is to design a document identification structure for the archive which will meet the enterprise's needs over a meaningful period of time.

Document Type	<ul style="list-style-type: none"> • First segment of the primary document identifier-key. • Maximum 20 characters • Null value allowed <p>Example document types from our sample libraries:</p>		
	demo_sales	demo_accounting	demo_purchasing
	"ArStatement" "Invoice" "Order" "Quote"	"ApAging" "ApCheck" "ArAging" "GLDailyDetail" "OpSalesRegister"	"PurchaseOrder"
Document ID	<ul style="list-style-type: none"> • Second segment of the primary document identifier-key. • Maximum 20 characters • Null value NOT allowed. However, the null is trapped by UnForm and replaced with a unique, date-oriented sequence number. <p>The combination of Document Type and Document ID form a unique identifier-key to a document within a library.</p>		

	Note that libraries are distinct units to each other so document identifier-keys are only unique to a library, in other words, identical document identifier-keys can exist in two distinct libraries without over-writing.
Categories	<p>Category indexes provide segmented keys to help navigate to specific document collections within a library. For example, a category index could be based on customer name and customer PO. If orders, packing lists, and invoices were all keyed with the same customer PO category, they would provide a convenient method of collecting these related documents into a single, browsable or searchable index.</p> <p>There can be any number of category indexes assigned to each document, and each index can contain up to ten segments, allowing much flexibility in custom indexing. When editing or creating categories, the system uses pipe symbols () to delimit segments.</p> <p>If a given category segment will potentially contain many thousands of items, it may be desirable to divide the segment into two tiers. For example, if a customer name is used as a segment, and there are thousands of customers, a two-tiered customer name could be designed, such as <code>left(custname\$,2) + " " + custname\$</code>. During browsing, the user would first locate the customer alphabetical group based on the first two characters of the name, then access just that sub-group of customers.</p>
Document Title	A broad general description of the document, sometimes composed of several major data values that help distinguish the document from other similar documents.
Keywords	Additional document identifiers that can help narrow and limit searches to locate documents and groups of documents, improving search efficiency. Keywords are semi-colon delimited words or phrases. Often they are auto-generated from the content of the job submitted for UnForm processing. When keywords are auto-generated, the generation is subject to configuration rules found in the [archive] section if the uf101d.ini file. Keywords are indexed, to provide fast searching and browsing.
Links	<p>A list of links to other documents, either in the archive system or external to it. This list is displayed in the web browser interface when viewing the document. The list is semi-colon delimited, with each link being one of these formats:</p> <ul style="list-style-type: none"> • A URL, such as <code>http://acme.com</code>, or a complete link to an UnForm browser page. If it starts with http or ftp, it can be prefixed with <code>title=</code> to specify a title for the browser to display. • A pipe-delimited structure that identifies the library, document type, document ID, and optionally image sub ID. The structure is <code>library type docid subid</code>, with the <code> subid</code> portion being optional.
Entity ID	<p>A security data element which can be included with a document and/or user account to filter access to specific documents or groups of documents to login user accounts which carry access authorization referencing the same entity ID.</p> <p>The concept of the Entity ID is one of ownership, designed for situations where external web access to documents in a shared library needs to be restricted to the entity specified. For example, where customer XYZ can login and browse, list and view invoices for customer XYZ without ever seeing documents for other entities listed.</p> <p>If documents are written to the archive with an empty entity ID field, then any user account with an empty entity ID will pass the entity test for access to a record. In an environment with empty entity ID field on documents in the archive, simply assigning a user any non-blank entity ID value can be used to restrict access to all documents in the archive.</p>

Date / Time	<p>The date/time stamp of a document is used as a secondary sort-key in the library to allow a by-date browse-method drill-down to locate documents. The date/time value can be maintained by the UnForm rule set used to archive the document, or via a command line option. It defaults to the initial date and time the document was added.</p> <p>Additional dateupdated and timeupdated fields are maintained each time the document is updated.</p>
Notes	Free-form text notes can be stored with a document and can be edited in the document properties box of a located document.

6.5 Image-level Identification

The elements discussed above apply to a document at a document "envelope" level, and serve to identify, group, and sort documents, and store additional useful information about a document, including free-form notes.

Because there can be multiple "images" associated with a document, each separate image file is assigned a unique identifier-key known as a Sub-ID. Note that the term "image" is used loosely here, and simply refers to a different version of the document. There can be a text image, a PDF image, and/or a tiff image of the same document.

When text-based documents are stored in the archive by an UnForm ruleset command, UnForm will default the Sub-ID value to `@text` for the text version of a document, and `@unform` for the pdf version of a document.

Documents stored in the archive via UnForm's command-line syntax will not have a default sub-ID assigned, so the user or the application must create a Sub-ID using the `-arcsbid` command line option.

In addition to the document file and the sub ID, UnForm also stores the date and time a particular image was last updated, plus a description field called a sub-title.

6.6 Document Data

Libraries can store custom user-defined fields called docdata values. These are named items stored with a document, identified by the document type, document id, and a name. Names can be up to 20 characters, and values can be any length. Names and values are both indexed (value indexes are only unique up to 40 characters), enabling the library object to provide indexed searches for name-value pairs.

The values are stored as url-encoded values to prevent delimiter issues. Note that this affects sorting of docdata lists returned by the library object, as some non-alpha-numeric data is converted to encoded hexadecimal and will sort out of sequence when compared with other non-alpha-numeric data.

Some document data is editable in the archive browser interface, as described below.

There are three types of docdata elements, standard elements, entry descriptors, and metadata elements.

Standard elements have alpha-numeric names and store any value.

Entry descriptors have names beginning with an asterisk (*), with the balance of the name used as a entry category, and the value storing a tab-separated values list composed of standard element names, an input type, and a validation name. The input type is one of: text, longtext, note, number, date, checkbox, `radio=item1;item2...`, `list=item1;item2...`, `lookup=lookupname`, grid, or `label=label text`. Lookups and validations are defined in the Image Manager's Custom Code tool. This descriptor definition provides the browser interface with the data needed to present an input form for the associated docdata elements.

Metadata elements have names prefixed with @, and are read-only in the browser interface. Note also that any docdata field that is not defined in an entry descriptor definition is also read-only.

6.7 Adding UnForm-Generated Documents

UnForm document archiving supports several methods for adding documents to libraries. One of the most useful methods is via UnForm jobs themselves, through the use of command line arguments or an **archive** command in a rule set. The benefit of this is that as jobs print and are formatted by UnForm, they can be automatically archived, eliminating the need to scan and archive reports using an external system.

Note: in order to be archived properly, jobs must be designed to successfully produce PDF output. In particular, jobs that use a pcl attachment or pcl images, but do not provide for PDF versions of these, will not be formatted properly in the archive.

If any `-arcxxx` command line arguments pertaining to the archive command are used, such as `-arclib` or `-arcdotype`, the options establish defaults for any archive command found, or initiate job archiving as if an archive command were included in the rule set.

For example, assume the `uf101c` command line includes these options:

```
-arclib "/archives/reports" -arcdotype "Reports"
```

If a rule set contains an archive command, the above defaults will be overridden by the command's options. However, if a rule set does not contain an archive command, or if no rule set is selected, the job will be archived regardless, using the above library and document type (in this case using an auto-generated document ID). This capability makes it easy to set up default archiving, with the ability to control archives on selected jobs with the addition of an archive command.

When UnForm archives a job, it evaluates the library, doctype, and docid elements of the archive command (or the values from the command line) page by page. Whenever an element changes, a new document is generated. In some cases, such as with hard coded command line options, these elements don't change during the job, and the whole job is archived as a single document. In other cases, an element such as a document ID might change as pages are processed, and a job can result in several documents being added to the archive.

UnForm documents can contain multiple versions or images, each identified with a sub ID. When UnForm archives one of its jobs, it archives two versions of the document. The first format is a PDF version of the document, which by default is given a sub ID of "@UnForm". The second format is a text version, derived from the incoming text stream. This is given a sub ID of "@text".

Archives generated from UnForm jobs receive automatic title and keywords if these values are not otherwise specified. If no title is specified and no **title** command is used, then the default title is derived from the content of the incoming text. Keyword generation is controlled with several parameters in the

uf101d.ini file, including a maximum keyword count (keywords=*n*), a list of patterns to not archive (nonwords=*file*), and a list of characters to eliminate (nonchars=*list*).

If no document type is provided, then if a rule set is used for a job, its name is used as the document type.

Starting in version 10, a separate archive server can be specified, enabling print jobs to run on one server, and archives to be stored on another server. This is important in sites that manage multiple UnForm servers for disaster recovery or load balancing purposes. While the command line and archive command options can specify an archive server, it is generally recommended that this value be configured in uf101d.ini ([archive] section, server= value) to ensure it remains consistent for all jobs.

The following command line arguments enable archiving and provide defaults for **archive** commands.

```
-arclib "libpath"
-arcdoctype "doctype"
-arcdocid "docid"
-arcsubid "subdoc ID"
-arcsubtitle "subdoc title"
-arctitle "title"
-arccats "cat1.1|cat1.2|cat1.3;cat2;cat3.1|cat3.2;..."
-arckeywords "kw1;kw2;..."
-arcnotes "notes (\n?)"
-arcargs "uf101c args for subjob"
-arcsep char (separator for category segments - default is |)
-arcdtm yyyymmddhhmmss
-arcsubdtm yyyymmddhhmmss
-arcserver "server:port"
```

6.8 Using the Web Browser Interface

The web browser interface is used to browse, search, and view archives and associated images. The web server used is an instance of the Apache HTTP server, managed by the UnForm server. This web server is generally configured to listen on port 27402, so the initial portal page is accessed like this:

`http://192.168.1.10:27402`

The IP address above is an example. Use the hostname or address of the UnForm server. You can also access the archive interface directly, adding "/arc" to the URL (`http://myserver:27402/arc`).

The user is presented a login form first. The first time it is used, an administrator can login with the name "admin" and password "admin", and can then add additional users and change the admin password using the browser.

The OneDoc feature provides direct access to a single, known document.

The Browse feature provides drill down capabilities through the library, document type, document ID structure, date indexes, keyword indexes, or category indexes.

The Search feature provides cross-library searching for generic text patterns or by specific field attributes or ranges. Selected documents can be viewed, exported, transferred, or copied.

Administrative features, such as user login management and library security set up, are available when an administrative user logs in.

A session cookie is used by the web browser interface. If cookies are not enabled in the web browser, then the browser interface will not function correctly. The lifetime of a session is controlled by the `sesage=hours` value in the [archive] section of `uf101d.ini`. If set to 0, a login is required each time the user starts their web browser. If set to some other value, then sessions last the specified number of hours before a new login is required.

For more details, view the [Archive Browser Interface](#) chapter.

6.9 Direct Browser Access to Documents

It is possible to view documents and document images directly, bypassing the full user interface, by using one of the following URL structures:

```
http://server:port/arc?a=ww&lb=library&doctype=doctype&docid=docid
```

```
http://server:port/arc?a=ww&lb=library&doctype=doctype&docid=docid&subid=subid
```

```
http://server:port/arc?a=ww&lb=library&doctype=doctype&docid=docid&subid=subid&i=1
```

The first form will load a document-level view page, which includes links to images. The second will load an image viewer for the specified *subid*. In each case, the server and script path must point to the UnForm web interface path, and the library, document type, document ID, and sub ID values must be URL-encoded (certain characters are converted to %*hex*. Information about URL-encoding can be found in any HTML guide). To extract the raw image, without the viewer framework, use the third syntax, which adds *i=1*.

If the *subid* specified is simply "@", as in "...&subid=@", then if the document only has an @unform subid (ignoring the @text subid), that PDF document will be viewed. If there is no @unform subid or there are additional images, such as scanned documents, then the document view page will be shown.

If the browser interface session has expired, then a login screen will be presented before the document is shown. It may be preferable to configure sessions to last longer, several hours or a day or more, to avoid this requirement.

Alternatively, it is also possible to use an UnForm client (`uf101c`) to retrieve and optionally view a document image from an archive, or the REST interface to retrieve a document programmatically (see [getimage] in `ufrest.ini`).

6.10 REST Interface

REST stands for Representational State Transfer, a technique of programatically accessing a web server with defined URL patterns to retrieve information. Designing a REST interface allows a server to provide data to client programs written in any language that can work with TCP/IP sockets, using the HTTP protocol of web servers. Many languages have easy to use bindings to such interfaces. There are even command line tools, such as CURL, to do so. Much of the browser interface provided with UnForm uses REST access from Javascript to access data from the server.

UnForm's REST interface is provided in two files. One, `web/en-us/ufrest.ini`, defines interfaces managed by the publisher and used by the browser interface. This file should not be edited, and will be replaced by

updates to the UnForm server. A second file can be created, named 'ufrest.custom.ini', if you desire to create your own REST interfaces.

Interfaces

The structure of ufrest.ini is similar to an INI file. There are sections defined with names in square brackets. The name identifies the interface, and the URL specified by the client must include the syntax 'rest=*name*' in the URL structure. Within the section are lines of code (in pure PxPlus Business Basic, with code block style block 'if ... else ... end if' in addition to native curly brace syntax) that are executed whenever that interface is executed. The code can utilize built in objects, such as the library object, but not rule set commands or custom UnForm functions. This code is expected to set two variables: response\$ and mimetype\$, to define the response data and the MIME type of that data.

The REST call from the client can specify variables in the URL, using standard HTTP query string syntax. In addition, variables can be supplied via a POST transaction, using the standard application/x-www-form-urlencoded encoding protocol, or multipart/form-data encoding. These variables are available to the code as cgi.*name*\$.

A simple example that returns a string containing a URL value might look like this:

```
[echotest]
response$="You sent the test value "+cgi.test$
mimetype$="text/plain"
```

To run this interface, you would use a URL like this:

`http://192.168.1.10/arc?rest=echotest&test=My%20Name`

As with any HTTP-based transaction, URL values must be URL-encoded, as shown above (the %20 represents a space character).

The standard ufrest.ini file contains many examples, both simple and complex, for reference.

Authentication

REST URL calls made from Javascript in a browser-session that is already logged in will already contain a session ID header, and can operate without special authentication. However, if you wish to access an interface outside of the bounds of a browser session, you need to provide user and password authentication. There are two ways to do this.

If you are just doing one access, you can simply add authentication to that transaction, in the form of two URL variables:

- s_userid=*login*
- s_password=*password*

This will create a new session for that request, and return the requested data. Each time these variables are specified, a new session record is created.

If you will be doing multiple requests, a better method is to request an authenticated session ID and supply that with subsequent requests. To do this, use the following URL query string:

`rest=auth&userid=login&password=password`

If successful, this will respond with two text lines: a session ID and the number of hours the session is valid. On subsequent requests, you can supply this session ID as a query string variable "ufsid=*sessionID*".

If an error occurs, the response will begin with an ASCII NAK character (hex 15 or character 21): `\x15error message`, such as `\x15"Invalid password"`.

6.11 Customizing the Web Interface

The web interface is a very complex product that utilizes third party tools and UnForm-specific code. Much of the interface is made up of html templates, which can be customized to adjust wording or add special coding. There is also a messages file, `messages.txt`, that contains short messages used by the server. A special directory "`web/en-us/custom`" (or other language) directory is available for the purpose of customizing the interface. Any file that is copied to the 'custom' directory will be used in preference to the standard version of that file. Files in this directory are not overwritten by updates to UnForm.

Note: this method differs from previous versions of UnForm, which required an entire new directory at the "`web/`" level and additional configuration. This method provides for simpler customization of specific files.

Due to the integrated nature of the web interface components, and the potential impact of changes the publisher might implement, customizations to the web interface are not encouraged.

While it is possible to modify the cosmetic appearance of the browser interface, it is not possible to modify the underlying structure or navigation, so be sure that if you modify templates all bracketed tags and links are maintained.

Caution: Any modifications performed to the templates should only be performed by experienced and knowledgeable HTML, Javascript, and CSS programmers. SDSI cannot warrant the performance of customized templates.

6.12 Custom Web Forms

The archive browser interface supports custom forms that can be accessed from specific points. These forms, when submitted, execute an UnForm job and provide that job with information from the form as well as the user's browser session and the CGI environment. The UnForm job can perform tasks that might involve updating document properties on a selected or related document, or execute some other task related to a document.

When the job is run, it can be designed to return data to the browser, in the form of a PDF file or HTML text. Optionally, the job can be run asynchronously and a configurable response message is returned instead.

The job must be designed to operate with no usable input stream. If the job's output is to be returned to the browser, the first page of data will be replaced or suppressed by the job. Variables are provided to access information the user completed in the form, so typically the job will be designed with code blocks that interact with this data.

There are several related configuration steps for setting up browser forms:

- HTML forms must be defined and stored in the server's active `web/en-us` or `web/language` directory. The forms must have a `.html` extension.

- A rule file and rule set must be created, designed to act upon the data from the form. This rule set will have access to some template variables: `cgi$`, `env$`, and `session$`, which are populated from the form and the user's session.
- The `forms.ini` file must be defined to enable users or groups to access forms designed for a given library and document type.

6.12.1 HTML Form Structure

The HTML form may have any valid HTML structure for a form, but must contain a few mandatory elements. The `form_sample.html` file found in `web/en-us` provides this example form code:

```
{include stdhead_arc.html}
<body>

<h2>Approve or Deny Action on this document</h2>

<form name="approval" action="[SCRIPT_NAME]" method="post">
<input type="hidden" name="a" value="uf">
<input type="hidden" name="args" value="-f cgiforms.rul -r approval -p pdf">
<input type="hidden" name="run" value="">
<input type="hidden" name="async" value="0">
<input type="hidden" name="lb" value="[lb]">
<input type="hidden" name="doctype" value="[doctype]">
<input type="hidden" name="docid" value="[docid]">

<table class="form">
  <tr>
    <th>Library</th>
    <td>[lb]</td>
  </tr>
  <tr>
    <th>Doc Type</th>
    <td>[doctype]</td>
  </tr>
  <tr>
    <th>Doc ID</th>
    <td>[docid]</td>
  </tr>
  <tr>
    <th>Categories</th>
    <td>
      <script>
        // present categories in lines
        var cat='[categories]';
        cat=cat.replace(/;/g,"<br>");
        cat=cat.replace(/\\|/g," &gt; ");
        document.write(cat);
      </script>
    </td>
  </tr>
  <tr>
    <th>Approval</th>
    <td>
      <input type="radio" name="approval" value="0" checked>Denied<br>
      <input type="radio" name="approval" value="1">Approved<br>
    </td>
  </tr>
</table>
```



```
[approval]
prejob{
    approved=num(cgi.approval$)
    cgiresponse$="Approved"
    if not(approved) then cgiresponse$="Not approved"
}
```

In this very simple rule set, a value is sent from the HTML form field "approval". This field is a radio button with a value of 0 or 1, depending on which button is checked (see the HTML sample above for the HTML form coding used). The variable cgiresponse\$ is set to a text value that is returned to the browser.

Alternatively, if cgiresponse\$ is not set, then the job's output is returned to the browser. In this case, the HTML form would specify a "-r approval2" and "-p pdf" in the args variable. The result will be a simple PDF document with text indicating the value of the approval field. Note that the print stream data contains data communicated to the rule set that is normally of no use to the user, hence the use of the notext command to suppress print stream text. Other techniques might be code block resets of the text\$[] array, or an erase command.

```
[approval2]
prejob{
    approved=num(cgi.approval$)
}
notext
text 10,10,{"Approval: "+str(approved)},cgtimes,12
```

There are three template strings provided to code blocks in a CGI-driven rule set:

- cgi\$ contains all the form fields that are present in the HTML form. All cgi\$ fields are string fields, so all are accessed as above: `cgi.field$`.
- cgienv\$ contains the CGI environment variables, if needed, such as `cgienv.script_name$` and `cgienv.remote_host$`. CGI environment variables are documented in many places on the Internet and in numerous books on web scripting.
- cgisession\$ contains fields related to the user's session. Of particular note might be the user login ID, which is found in `cgisession.s_userid$`.

6.12.3 Javascript Execution of Rule Sets

A function is supplied in the ufarc.js Javascript library that is loaded into web forms. This function executes a rule set on the server, and the rule set's cgiresponse\$ value is returned to Javascript.

The syntax of this function is:

```
var txt=runRuleSet("rulefile","ruleset",flds[,args[,callback]])
```

The rulefile and ruleset arguments are self-explanatory, as they simply become -f and -r arguments to an UnForm job. The flds argument can be either a URL-encoded string, such as "name=Jim%20Smith", or

an JavaScript object containing name:value pairs, in which case the function will URL-encode all the names and values. The fields defined in this argument are available in the rule set as `cgi.name$`.

If additional job command line arguments are required, they can be supplied in the optional args string argument.

If a *callback* function is supplied, it is executed when the request completes, and is supplied with the request response as its only argument. If no callback function is supplied, the request runs synchronously, and will return the server's response to the function. Note that running the function synchronously is not always supported by modern browsers. The preferred method of executing this function is with a callback function rather than expecting a text response.

Note if there is data in delimited or INI format available on the server and needed in Javascript, there is a json object that can be used in the rule set to convert such data into JSON format, enabling easier and faster access in the script code.

6.12.4 Form Access Configuration

The forms.ini file contains configuration information that the browser interface uses to provide automatic links to forms under certain circumstances. A sample forms.ini.sds file is provided with the UnForm installation for reference.

The system will merge other forms.*.ini files found in the UnForm directory at runtime, to facilitate easier management of more complex form configuration scenerios. The merge utilizes forms.ini as the initial file, then appends or adds sections found in other forms.*.ini files. The accumulated definition is stored in the file forms.ini.all, which serves as a cache and is rebuilt whenever any source files are modified or added. This speeds delivery of the forms configuration at runtime.

Document-Oriented Forms

When a user is logged in and viewing document properties or a document image, a Forms tab is offered if any forms are configured that meet the session criteria, and links to all available forms are provided in that panel.

Here is a sample forms.ini file:

```
#sections by library|doctype
# user:<name>=form name[,description[,option]]
# group:<name>=form name[,description[,option]]
# *=form name[,description]

[demo_sales|OpInvoice]
user:theboss=form_approval,Approval Form,boss=1
group:sales=form_rep_action,Sales Rep Action
*=form_site,Site Form Info
```

Sections in this file are identified by a `[library|doctype]` heading. When a document of that document type is viewed from that library, forms configured on the following lines will be available, depending on the user and user group membership. Any given form is only presented once, even if the circumstances would cause multiple configuration lines to present the same form.

In the above example, the user "theboss" can access the form_approval.html form, any user in the group "sales" can access the "form_rep_action.html" form, and any user can access the "form_site.html" form.

All forms that meet the security criteria are available. For example, user "theboss" will have access to form_approval and form_site, and any user in the group "sales" will have access to form_rep_action and form_site.

In this example, "option" element is supplied to the web form rule set as cgi.option\$="boss=1".

Sections identified as [*library*]*doctype* specify forms to be presented when an document viewer is displayed. *Doctype* may be * to indicate any document type in the library.

Sections identified as [*library*]*doctype**subid* specify forms to be presented when an image viewer is displayed. *Subid* may be * to indicate any image of the specified doctype.

Library-oriented Forms

Library-oriented forms are available in the Browse interface when a library has been selected. Sections for these forms are identified with the header [*library-name*]* or [*library-name*]*doctype*.

When library forms are invoked, the doctype and docid fields are set to "".

Search- and Marked-Related Forms

In addition to forms selected when viewing documents of a configured library and document type, forms can also be executed from search results and marked record lists. These are configured in a similar manner, with sections in the forms.ini file.

A [search] section denotes user or group form access when viewing search results. If the results of a saved search are being viewed, then a section named [search]*saved search name* can be defined, also with user:*userid* and/or group:*groupid* access. Search-based rule sets can use a search object's doclist object to navigate and react to the search results.

Finally, a [marked] section can be defined to allow forms driven from a marked records list. The marked object can be used to navigate and manipulate the marked record list for a user's session.

```
[search]
group:managers=form_transfer,"Transfer, Documents Form",nonadmin
user:admin=form_transfer,"Transfer, Documents Form",admin
```

```
[search|CurrentInvoices]
group:AR=form_CurrentInvReport,Report of Current Invoices
```

```
[marked]
user:jsmith=form_jsmith_purge,Purge Marked Documents
```

In the above search example, the web form rule set will have cgi.option\$="admin" or cgi.option\$="nonadmin".

Menu-Related Forms

The forms.ini structure can be used to add custom panels to the main menu of the browser interface. Panels are added to the bottom of the menu, and link to web forms in the same window or a new window. A URL is constructed based on configuration entries, so that when the user clicks the panel, the custom web form is presented. The code of this form can perform whatever tasks are necessary, even to the point of loading a new page if desired.

Menu panel configuration requires multiple sections. First, a [menu] section must be defined that specifies form names and default titles for different users or groups. This configuration is almost identical to other web form sections, and the active forms are selected based on the session user. One exception

to this format is that a *~value* suffix can be added to the form name to allow multiple sections to be configured for a given form. The suffix is used only to locate a related configuration section, and is not considered part of the form's related HTML file name.

For each menu web form name, an additional section must be configured in the file to describe the attributes of that panel and how it launches the web form. The section header is [menu]*formname*, where *formname* is the name portion (including any *~value* suffix) of the name,title assignment in the selection lines of [menu]. Within each section, the following *name=value* pairs can be used:

- desc=*description* - provides panel descriptive text.
- title=*title* - overrides the default title from the [menu] selection line
- icon=*file* - names an icon file for the panel. Standard icons can be found in web/en-us/icons. A full path is not necessary if the file is in that or another standard UnForm directory, such as the install location and the web/en-us paths.
- newwin=0|1- if set to 1, the panel opens a new window for the form.
- lb=*library* - names the library the form is associated with (this value is required).
- doctype=*doctype* - names the document type the form is associated with.
- All other *name=value* pairs are added to the URL that is used to run the form. Values can be retrieved from the form documents search property using JavaScript, or more conveniently, if the form loads the common.js script library, use the queryValue(*name*) function.

The following menu section describes the menu panels that will be presented to various users and groups. For example, the group AP will see two panels, My Docs Status Report and Capture to ERP Batch Processing, which will run the md7-v2 and md8-v2 forms, respectively.

A webform name of "break[,*title*]" displays a line break and title bar, rather than a menu panel.

```
[menu]
group:AP=break,AP Workflow
group:AP=md7-v2,My Docs Status Report
group:AP=md8-v2,Capture to ERP Batch Processing
user:admin=md9-v2,All Users Doc Status Report
user:collins=md9-v2,All Users Doc Status Report
user:mje=md9-v2,All Users Doc Status Report
user:heyman=md9-v2,All Users Doc Status Report
```

The configuration for the md9-v2 form is shown below. The panel title will be the default, All Users Doc Status Report. The panel description and icon are shown. The web form will use library AP-MDF and will be shown in a new window. When the form is displayed, two additional URL fields will be available, RunType=0 and SortType=1, which code in md9-v2.html can reference.

```
[menu|md9-v2]
desc=Status report for my AP documents
icon=Play_24.gif
lb=AP-MDF
newwin=1
RunType=0
SortType=1
```

Image Manager Forms

A drop-down toolbar option appears in Image Manager when forms are configured. Forms can be configured for all sources, or one source, using sections named [imagemanager] or [imagemanager|*sourceid*]. Each section can contain lines prefixed with user:*userid*, group:*groupid*, or * for all users.

These sample sections enable admin access to the form_unassigned.html form, and any IM user access to the pod_summary.html form when viewing the POD inbound source.

```
[imagemanager]
user:admin=form_unassigned,Unassigned Documents Report
```

```
[imagemanager|POD]
*=pod_summary,Proof Summary Report
```

DocFlow Forms

A drop-down toolbar option appears in DocFlow when forms are configured. Forms can be configured for all sources, or one source, using sections named [docflow] or [docflow|*flowid*]. Each section can contain lines prefixed with user:*userid*, group:*groupid*, or * for all users. This sample section would allow members of the AR group access to the AR Valuation Report form, and the admin user the Late Documents in AR form, when working in the AcctsRec flow.

```
[docflow|AcctsRec]
group:AR=form_values,AR Valuation Report
user:admin=late_ar_docs,Late Documents in AR
```

6.13 Using the UnForm Client

The UnForm clients, which include the Perl-based Unix client (uf101c), and the Windows client (uf101c.exe), can each perform document management functions via command line options. In many cases, a login is required. This can be supplied via the -arclogin option in the form "*userid/password*". To secure this information so it does not appear on the command line, which may be visible to other users, it can be provided in a file referenced by the -z command line option (or -zx to immediately erase the file).

In addition, the Unix clients can prompt for login information by supplying the special syntax **-arclogin ask**, and/or this information can be stored. If login information is not supplied, the Unix client will look in the files \$HOME/.ufarc or /etc/.ufarc for two lines, login=*userid* and pswd=*password*. These are stored in clear text, so the only effective security for this is to use user-specific .ufarc files (in \$HOME) and make sure they are readable only by the user.

The windows client requires that the login information be supplied via the -arclogin "*user/pass*" option, directly or via the -z/-zx file options.

6.13.1 Triggering Archiving of UnForm Jobs

In addition to using archive commands in rule sets, you can use command line options to establish default values and to trigger archiving even in cases there rule sets do not contain archive commands, or if no rule set is invoked and jobs are passed through to output.

At a minimum, you should specify a library, using the -arclib option. Other options that set archive properties can be used, but care needs to be taken as these are generic options that will apply to all documents that are not archived via specific archive commands.

```
-arclib "libpath"
-arcdoctype "doctype"
-arcdocid "docid" (if not supplied, an auto-generated number is created)
-arcsubid "subdoc ID"
```

```
-arcsubtitle "subdoc title"
-arctitle "title"
-arccats "cat1.1|cat1.2|cat1.3;cat2;cat3.1|cat3.2;..."
-arckeywords "kw1;kw2;..."
-arcnodes "notes" (use \n character sequence to embed line breaks)
-arcentityid "entityid"
-arcargs "uf101c args for subjob"
-arcsep char (separator for category segments - default is |)
-arcdtm yyyymmddhhmmss (normally automatically calculated)
-arcsubdtm yyyymmddhhmmss (normally automatically calculated)
```

The keywords setting can be a semi-colon delimited list, or a number indicating how many keywords to generate from content (-1 or "all" for all). The default number of keywords is found in uf101d.ini. When keywords are scanned, there is a file (ufnonwr.txt by default) that contains words and patterns to ignore.

Use \; or \| (or \<sep char>) to embed delimiters in keywords or categories. Use \\ to embed a backslash.

Note ; and | characters (and spaces) must be protected from the shell, so keywords and categories in particular should be quoted, as well as any argument with spaces in it.

-arcsubid "subid*" will sequence the key to prevent overwrites. Using "subid*" will force "subid*".

6.13.2 Adding External Documents

To add external documents to a library, bypassing any UnForm processing of the input, use the `-arcput` command line option, in conjunction with the `-i` option to name the file to add. Note this differs from archiving of UnForm jobs, as the input file is not processed through a rule set, but rather written directly into the archive. In addition, further options may (and probably should) be used:

```
-arclib "libpath"
-arcdoctype "doctype"
-arcdocid "docid" (if not supplied, an auto-generated number is created)
-arcsubid "subdoc ID"
-arcsubtitle "subdoc title"
-arctitle "title"
-arccats "cat1.1|cat1.2|cat1.3;cat2;cat3.1|cat3.2;..."
-arckeywords "kw1;kw2;..."
-arcnodes "notes" (use \n character sequence to embed line breaks)
-arcentityid "entityid"
-arcargs "uf101c args for subjob"
-arcsep char (separator for category segments - default is |)
-arcdtm yyyymmddhhmmss (normally automatically calculated)
-arcsubdtm yyyymmddhhmmss (normally automatically calculated)
-arclogin "userid/pswd"
-arcflowid "flowid"
```

The keywords setting can be a semi-colon delimited list, or a number indicating how many keywords to generate from content (-1 or "all" for all). The default number of keywords is found in uf101d.ini. When keywords are scanned, there is a file (ufnonwr.txt by default) that contains words and patterns to ignore.

Use \; or \| (or \<sep char>) to embed delimiters in keywords or categories. Use \\ to embed a backslash.

Note ; and | characters (and spaces) must be protected from the shell, so keywords and categories in particular should be quoted, as well as any argument with spaces in it.

-arcsbid "subid*" will sequence the key to prevent overwrites. Using "subid*" will force "subid*".

Write access to a library is required to add a document to it.

6.13.3 Document Image Retrieval

The command line can be used to extract a document image from a library. To do so, you must supply the `-argget` command line option, along with identifying options to indicate the library, document type, document ID, and image sub ID. The `-o` option is used to specify where the document should be placed, typically with a "client:" prefix, like `-o client:/tmp/myfile.pdf`.

```
-arcget  
-o filename  
-arclib "libpath"  
-arcdoctype "doctype"  
-arcdocid "docid"  
-arcsbid "subdoc ID"  
-arclogin "userid/pswd"
```

Read access to the library is required.

When used in conjunction with the `-p winpw` driver, a local view of the document is presented on the workstation running the UnForm client.

When used in conjunction with `-arcget`, the `-arcsbid` option supports two special suffixes, "`-<`" and "`->`", to return the first or last sub ID that matches the sub ID string. For example, a value of "`@UnForm->`" would return the last sequential `@UnForm` sub ID in an auto-sequenced library. Escape the suffix as "`\->`" to look for the literal value.

6.13.4 Document Deletion

The command line can be used to delete a document image from a library. To do so, you must supply the `-arcdel` command line option, along with identifying options to indicate the library, document type, document ID, and optionally an image sub ID.

```
-arcdel  
-arclib "libpath"  
-arcdoctype "doctype"  
-arcdocid "docid"  
-arcsbid "subdoc ID"  
-arclogin "userid/pswd"
```

If subid is supplied, just that subid is deleted. If no subid is supplied, the full document record is deleted, including category indexes and all subid images.

Delete access to the library is required.

6.13.5 Document and Index Listings

There are many listing options, specified by using one of the `-arclist*` options. Below is information about the command line options required to run each type of listing.

List Libraries

-arclistlibs or -arclistrwlibs or -arclistrwdlibs
-arclistfmt tab|csv|pipe
-o "output file"
-arclogin "userid/pswd"

Lists all libraries that the userid has read access to. Alternate forms list read/write libraries and full access libraries.

List Document Types

-arclisttypes
-arclib "library"
-o "output file"
-arclogin "userid/pswd"

Lists the document types in the library specified.

List Category Segments

-arlistcats
-arclib "library"
-o "outputfile"
-arclogin "userid/pswd"
-arcstart "seg1|seg2|..."
-arcfirst index
-arccount count

Lists category segments of a certain level in the library specified. With no -arcstart option, the top level categories are listed. Using -arcstart "segment" will list second level segments within the top level segment specified. Categories nest up to ten levels, so you can continue to add segments delimited with pipe characters. The -arcfirst and -arccount option can be used to return *count* records starting at *index* position.

List Keywords

-arlistkeywords
-arclib "library"
-o "filename"
-arclogin "userid/pswd"
-arcstart "prefix"
-arcfirst index
-arccount count

List *count* keywords starting with *prefix*, optionally starting at the *index* position in the list.

List Keyword Seeds

-arlistkeywordseeds
-arclib "library"
-o "filename"
-arclogin "userid/pswd"

-arcstart "prefix"
 -arcfirst index
 -arccount count

List keyword *count* seed values, optionally starting at the *index* position in the list. A keyword seed is the initial portion of a keyword, of some limited length. You can set a prefix for the seeds with the -arcstart option, and the result will be a list of keywords seeds starting with the *prefix*, each with a length of 1 more than the prefix. With no prefix, the list will be of all first characters in the file. With a one-character prefix, the seeds will be all two-character values starting with that prefix. This feature can be useful for developing drill-down keyword lists.

List Documents, List Documents and Images

-arclist (or -arclistdocs)
 -arclib "libname"
 -arclistfmt tab|csv|pipe|html|xml|xmlf
 -arcorder id|date|title|category
 -arcstart "starting point in order specified"
 -arcend "ending point in order specified"
 -arcfilter "filter string"
 -arcsep char (used to parse start/end values for category segments)
 -arclogin "userid/pswd"

You must have read access to a library to list documents in it.

The columns listed include document type, document ID, date, time, title, the document storage file path, and entity ID. If you specify the category order, then the category's segments are added as a single column. Otherwise, no category information is shown. If the list format is html or xml, then notes, keywords, links, and categories are added to the list.

The start and end range values relate to the order. For example, if the order is "id", then the start and end ranges refer to document type and ID sequences. For segmented ranges, such as for type and ID values, or category segments when listing in category order, separate them with the -arcsep value (a pipe (|) by default). For example:

-arcorder id -arcstart "Invoices|000152"

Be sure to quote the range if it contains characters, such as pipes or spaces, which are significant to the operating system. For date ranges, enter dates in the structure *yyymmddhhmmss* (as many characters are significant to your request). For example, -arcstart 200612011800 -arcend 200612020800 for the range from 6:00 PM on December 1, 2006 through 8:00AM on December 2, 2006.

To filter documents returned in the list, use the -arcfilter option. The filter can be a simple word or phrase, a simple wildcard containing * and/or ? characters such as "Acme*", or a regular expression prefixed with a tilde, such as "~[0-9][0-9]\.[0-9][0-9]". Filters are applied to a concatenation of the document type, document ID, date, time, and title values, and in the case of category order, the categories are filtered as well. If the list format is html or xml, then the notes, keywords, and categories are added to the list. Keywords and categories, though not category segments, are delimited with spaces rather than semicolons when filtered. If the list format is xmlf, then a base64-encoded version of the image file is also added to the xml document.

To list documents and their associated image information, use the -arclistdocs command line option rather than the -arclist option. The same options as shown above for -arclist apply.

When used in conjunction with the `-p winpw` driver, a local view of the listing is presented on the workstation running the UnForm client.

6.13.6 Searching for Documents

To search a library, use the `-arcsearch` command line option:

```
-arcsearch  
-arclib "library"  
-arclistfmt tab|csv|pipe|html|xml|xmlf  
-o "output file"  
-arclogin "userid/pswd"
```

Refine the search by adding any appropriate arguments from this list: `-arcdotype`, `-arcdocid`, `-arctitle`, `-arcentityid`, `-arcdate`, `-arcdateupdated`, `-arckeywords`, `-arcnotes`, `-arccats`, `-arclinks`, `-arctext`, `-arcdocdata`, or `-arcsbid`. Each of these arguments can be a wildcard (**value** or *value**), an exact value "12345", a range "12/1/2007–12/31/2007" (double-hyphen range delimiter), or a regular expression ("~[0-9][A-Z]"). You can use "not" to look for archives that do not match a criteria, and "and" or "or" to search for multiple values or alternate values.

Searches are optimized when possible. The best optimizations are document IDs, entity IDs, small date ranges, and multi-level categories.

Note that document types and document IDs are case-sensitive when optimized.

6.13.7 Testing Existence of Documents

To test the existence of a library, document type, document ID, or image sub ID:

```
-arcexists  
-arclib "library"  
-arcdotype "type"  
-arcdocid "docID"  
-arcsbid "subid"  
-o "[client:]outputfile"
```

No login is required. You can test for just the library, a library and document type, a library and document type and document ID, or all four elements. The system prints a 0 or 1 to the output device, plus a CRLF (0=not found, 1=found).

6.13.8 Importing 7.x/8.0 Libraries

UnForm 10.1 uses a different library structure, and different naming conventions, than previous UnForm versions. A simple command line is used to import an old library into a new, version 10.1 library. By using different naming conventions, it is actually possible for version 8 and 10 libraries to coexist in the same physical location, though it is important to realize that the two libraries do not interact and are not updated in tandem by either version. They are functionally two different libraries, even if they share the same disk path.

Care should be taken when importing, as UnForm 10.1 libraries require more disk space due to new indexing and recovery transaction logs. How much more space required is unpredictable, but it isn't uncommon for disk use to approximately double from previous versions. It may be preferable to create the

10.1 libraries on a different file system, or a different machine altogether, though the import process needs access to both the old and new libraries, so a copy could be made, or network file system access used, to read the old library.

Imports are incremental, in that if a give doctype/docid or doctype/docid/subid already exists in the new library, it will not be overwritten by a subsequent import. This allows importing multiple times, incrementally adding new documents and images. If you want to ensure that an import captures all documents, including those that may have been imported previously and later updated in the older system, you must import into a new, empty library.

The command line arguments used are:

```
uf101c -arclogin "user/password" -arclib "new library" -arcimport "old library" -o "logfile.txt" [-arcdotype "doctype" | -arcdtm "yyyymmdd[-yyyymmdd]" ]
```

An administrative login is required to perform the import. The import will run in background, writing logging details to the server file specified in the -o option. Be sure not to specify a client-side output file, or else the client will continue running for as long as the import runs.

Libraries are not locked during the import, so it is possible to use them while the import is running. There are two caveats, however, related to potential duplicate ID values.

- Any documents that have auto-generated document ID values could cause conflicts, as new documents being added would likely receive an ID that will be duplicated by the import, resulting in skipping the import of the older document.
- Any sequenced sub ID value is potentially in conflict with a subsequent imported sub ID, resulting in skipping of importing the older image.

If a library contains documents with auto sequencing, either of doc ID's or sub ID's, it is best to not print or upload such documents until the import is complete.

When importing, in addition to the output log file, the system logs entries to the server log at the start and end, and every 15 minutes of the job, to indicate status.

Importing ranges of documents

You can specify either a document type or a date/date range by adding -arcdotype or -arcdtm options to the command line. If a document type is specified, just documents of that type are imported into the new library. Likewise, if you specify a date or date range, just documents in that range of document date are imported. The date can be specified as a year (i.e. 2014), or a month (201401), or a full day (20140115). If you require a more specific range, use a hyphen to specify it (i.e. 20140115-20140215).

By using ranges, you can run multiple import jobs concurrently to speed import processing, within the resource limits of the system.

6.13.9 Importing Documents from sdStor

UnForm can import images from an sdStor library. It performs this by extracting the text documents from the library and running them as UnForm jobs. The import is performed for a single library, specified using the -arcimport option, which also allows specification of the sdStor login and password:

```
-arcimport "sdstor_libname;login/pswd"
```

The default UnForm library will match the path used for the sdStor library. To override this default, add: -arclib "libname".

A few additional command line options are added automatically in order to:

- Retain the original date and time of the sdStor document
- Retain the title and keywords from sdStor
- Add an additional keyword "sdstor *sdStorID*"
- Add a category "sdstor|*stStorID*"

The keyword and category additions are provided to help link documents in sdStor with documents added via the import to an UnForm library.

For enhanced processing, specify a rule file using the `-f rulefile` option, and use **archive** commands in the rule file to provide the document settings desired. When using a rule file, be sure to not specify a date and time, so the command line options that capture the date and time from sdStor will not be overridden.

The import is processed by extracting all the documents from the sdStor library and placing them in the rpq directory (the direct TCP/IP printing queue), where they are automatically processed sequentially. As each document is added to the queue, a log line is printed to the command line's output, so it is recommended that a `-o` option be used to send log output to a server file (don't use the client: prefix on the output file), as imports can be time consuming. The amount of time spent extracting to the queue, and the amount of time it takes for the queue to be processed, depends on the number and size of documents in the sdStor library.

6.14 Functions Related To Archiving

When processing a job, UnForm can not only add documents to an archive, but it can also extract documents from a library for use in processing. For example, a statement job could be designed to extract a list of invoice PDF images and attach them to the statement using the **images** command.

The [library object](#) provides a large suite of functions and properties related to a library and is often used for archive retrieval and manipulation from a rule set. In addition, the following functions are provided for functional programming work:

To retrieve a document during an UnForm job, use this function in a code block:

```
getarc(library$,doctype$,docid$,subid$,filename$[,errmsg$])
```

If filename\$="", it will return a temporary file containing the document. This temporary file will be erased at the end of the job. If a filename is supplied, that file will be created and will not be erased at the end of the job.

If errmsg\$ is present, it will return any error if the document can't be found or if another unexpected error occurs.

To convert a PDF file into an image, using Ghostscript, use this function:

```
pdftoimage(fromfile$,tofile$,format$ [,resolution [,errmsg$]])
```

This function will invoke Ghostscript, either on the UnForm server or on the UnForm Support Server, to convert the PDF file in `fromfile$` to an image file in `tofile$`. The format of the output will be named in `format$`, and it must match one of the [driver] names found in `uf101d.ini`. The image is created at the dpi specified in resolution, or 300 dpi if not.

If `tofile$` is null, it will be returned with a temporary file name that will be erased when the job is complete.

Any error message is returned in `errmsg$`.

Ghostscript must be configured in the [drivers] section of the `uf101d.ini` file, or an UnForm Support Server with Ghostscript configured must be available. Note that the **images** command automatically performs this step function if it encounters a PDF file name.

To test if a library exists, use the `libexists()` function:

`libexists(lib$)` returns 0 if library `lib$` doesn't exist, or 1 if it does.

To obtain a list of image subids associated with a document, use the `getsubids()` function:

`getsubids(lib$,doctype$,docid$[,dlim$])`

Returns a list of document sub IDs, such as the @text and @UnForm subids automatically generated by the archive command. The list is returned as a delimited string, with the default delimiter being a semicolon. If the delimiter occurs in any subid, it is escaped with a \. The returned value may be used by the parse functions.

When a series of document images should be attached to a page produced by an UnForm job, you can extract the desired documents to work files, using the `getarc()` function, and append them to the page, optionally tiled, using the **images** command.

To determine if a document or sub document exists, use these functions, which return 0 if the specified entity does not exist, or 1 if it does:

`docidexists(lib$,doctype$,docid$)`
`subidexists(lib$,doctype$,docid$,subid$)`

6.15 Transferring Archives to A New System

The archive libraries are subdirectory structures that are binary compatible between operating systems, so it is possible to move them between Windows and Linux servers if necessary. In addition to the library directories, there are several files in the UnForm server directory that reference the full paths of the various library directories, or are used for library security. These files are:

<code>ufarcacc.dat</code>	User-library access rights
<code>ufarcgac.dat</code>	Group-library access rights
<code>ufarcgrp.dat</code>	Groups table
<code>ufarclib.dat</code>	Libraries, identified by their full path
<code>ufarcusr.dat</code>	Users table

If the libraries are moved to the same path names on the new system, then all that is required is to copy these files to the new UnForm server directory. Note that simple library names are converted to use the full path of the UnForm server directory, plus the "arc" subdirectory, so the UnForm server should also be installed in the same path on the new system to take advantage of this capability.

Note that the UnForm server runs under a specific user account, and that account must have full permission to the library directories and files. Typically, the files you copy will be owned by you, possibly with limited permissions for the UnForm user. Modify ownership and permissions accordingly.

If the libraries are to be moved to a new path on the new system, then the library, user-library access rights, and group-library access rights tables will need to be created using the browser administrator interface. In addition, any rule file archive commands and external scripts that reference library paths will also need to be updated before executed. The users table and groups table can be copied "as is", since they do not reference library path names.

If there are external links to libraries that now reference the wrong library path, you can edit the library properties of the new path, and add any number of aliases that match old paths. When a request for an old, invalid path is received, the system will locate the path as an alias and use the new library name.

Image Manager and DocFlow Jobs

These are stored in platform-independent files: `ufjobdef.dat`, `ufarcdfr.dat`, and `ufarcdfw.dat`.

Inbound Sources

These records are stored in the platform-independent file: `ufarcinb.dat`. Any path sources are of course file system dependent.

6.16 Migrating Libraries From Previous Versions

UnForm 10 and Unform 9 share the library format, so there is no conversion required for UnForm 9.0 libraries to be migrated to an UnForm 10.1 installation. However, only 10.1 is capable of maintaining DocData values, so if records are updated from within Unform 9.0, data stored by UnForm 10 may be lost. For this reason, sites that use the UnForm 10 Image Manager or DocFlow modules, which rely heavily on DocData values, should not share libraries between the two versions.

UnForm 10.1 uses a different library format from versions previous to UnForm 9.0. Older libraries must be converted for use with the 10.1 system using an [importing](#) process. This conversion can be done "in place", where both 10.1 and previous versions share the same library directory. The two versions of libraries use different file naming conventions, so there is no conflict and both can reside in the same physical directory, though space usage will be approximately doubled

You can also import into a new directory, or to a new machine by first copying old libraries to some location on that new machine. Both old and new library directories must be available to the UnForm server.

6.16.1 Migrate to 10.1 on a new system

To migrate to 10.1 on a new server, install the 10.1 server into a new directory. Next, copy all library directories (including those in the default "arc" subdirectory) to the same paths on the new system.

For version 8.0 and previous libraries, run a command line import of any libraries to convert them to 10.1 format. This procedure is described in [Importing 7.x/8.0 Libraries](#).

Then copy these files to the UnForm server directory:

- ufarcacc.dat – library/user access table
- ufarcgrp.dat – groups table
- ufarcgac.dat – library/group access table
- ufarcclib.dat – library full paths and properties
- ufarcusr.dat – user table

If you wish to use new paths for any libraries, you may do so, but will need to use the browser interface Library maintenance function to change library path names before any archiving activity or browser access to the libraries takes place. Any library location changes require that the library table be updated to reflect the new physical library locations.

In order to continue to print from the old system to the new one, install an UnForm 10.1 client on the old system, and add a `–server name-or-IP` option to the uf101c command lines to submit print jobs to the new server.

Group Permissions

If you are migrating from 7.0/7.1, then you need to be aware that group permission interpretation changed.

As always, if a user is assigned specific permissions to a library, those take precedence. However, if a user is assigned to any group, then the permissions of the group are used for all libraries, and no library default permissions are applied. If a user is not a member of any group, then default permissions are applied by library. In 7.0/7.1, default permissions applied to any library that was not specifically assigned group permissions. In general, this means that if you use groups and have users assigned to those groups, then you must explicitly define group permissions for each library.

Windows Drive Letters

In 10.1, all Windows libraries include the drive letter in their path. If you copy ufarcclib.dat from a previous system that does not have drive letters, you could end up with duplicate library names pointing to the same physical library. You can delete the non-drive letter version of the library using the admin browser interface Libraries tool.

6.16.2 Migrate to 10.1 on the same system

To migrate to 10.1 on the same server as the previous installation, install the 10.1 server into a new directory

For version 8.0 and previous libraries, run a command line import of any libraries to convert them to 10.1 format. This procedure is described in [Importing 7.x/8.0 Libraries](#). Note this process can be run multiple times if needed, to assist in migration of large libraries. Only new or updated documents and images are imported.

Take steps to ensure no UnForm printing will take place to the previous server, then copy these files to the UnForm server directory:

- ufarcacc.dat – library/user access table
- ufarcgrp.dat – groups table
- ufarcgac.dat – library/group access table
- ufarcclib.dat – library full paths and properties
- ufarcusr.dat – user table

If the full paths to libraries is changed, such as if they were stored under the "arc" directory of another UnForm installation, use the browser interface Admin, Libraries page to modify the library paths to each of your new library locations. As you do so, the custom user and group access records will be updated with the new library name. If there are existing links to now-obsolete paths, add the old paths as aliases in the new library record. Aliases enable the access to current library names using old names.

Be sure to update any rule files that have full paths to library names in archive commands, so printing archives will update the new library locations. If there are external links to old library names, add the old names as aliases in Library maintenance.

At this point, you can begin submitting print jobs to the UnForm 10.1 server.

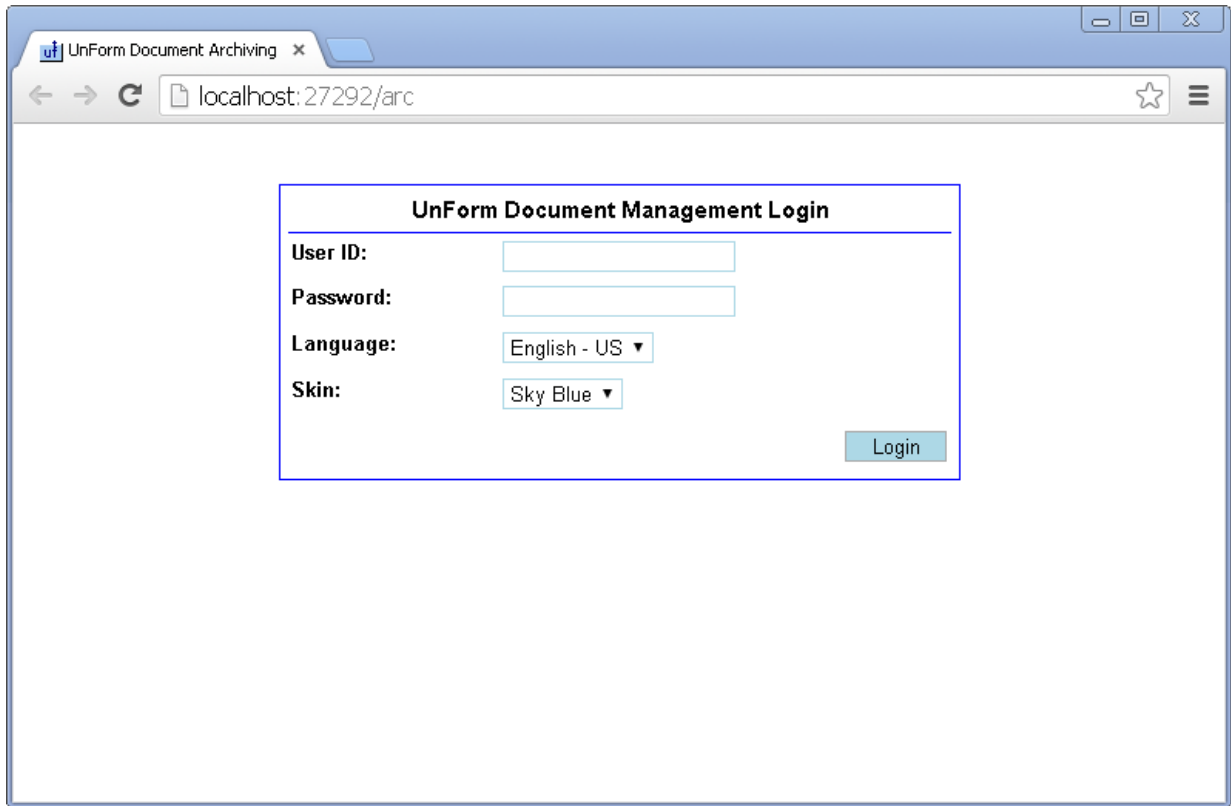
7 ARCHIVE BROWSER INTERFACE

The archive browser interface provides users and administrators a flexible, browser-based solution to document retrieval and management. UnForm libraries are available for browsing and searching, and documents can be edited, emailed, faxed, and exported individually or in batches. Administrators can update libraries, users, and groups, and can manage security profiles for users.

7.1 Login Form

The login form requests login, password, and session setting information. If user self-management is enabled, a Help button is provided to allow the user to request their login details be sent to the email address on file for the user. The login detail provides a short-term recover password link to display the password for the user.

Administrators can define logins and associated permissions using the administrator's [Users](#) or [External Users](#) forms. If no users have yet been configured, the default login is admin/admin, which should be changed by the implementer to use a secure password.



7.2 Main Window

All archive browsing activity works within a main window. As options are selected, windows open in the browser workspace to perform selected tasks. Most tasks are remembered across logins, so when a user logs in, previously opened windows are restored.

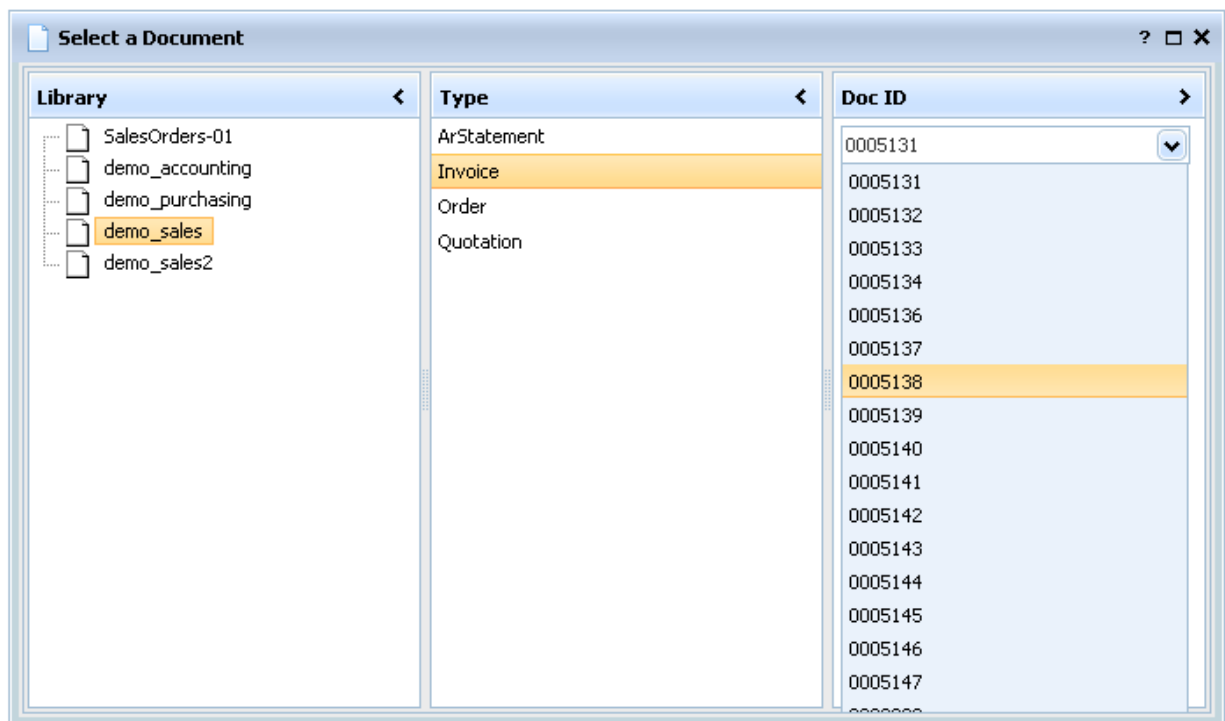
The menu bar organizes the available features under several top level options:

- Documents
 - [One](#) provides quick selection and viewing of a single document.
 - [Browse](#) provides indexed navigation, keyword searching, and category browsing in a library.
 - [Search](#) provides searching across one or more libraries for documents that match certain criteria, and provides tools to manage those documents, such as transfer and export.
- Marked
 - [Images](#) provides tools that work on a collection of images that have been selected while browsing or viewing.
 - [Documents](#) provides tools that work with a collection of documents that have been selected while browsing or viewing.
- Windows
 - Provides window management functionality, in addition to that provided by window title bar icons. This menu group can provide improved navigation in small screen environments.
- Settings
 - [Options](#) is a form for configuring some session settings.
 - [Address Book Edit](#) enables users who have address book editing permission to manage address books and their entries.

- Admin (appears for administrator users)
 - Libraries provides library management, including property settings and corruption recovery
 - Users provides user maintenance, including user-specific permission settings and group membership
 - External Users provides external user maintenance (external users are those with an entity ID)
 - Groups provides group maintenance, including group-specific permission settings
 - Build Demo Libraries generates demo libraries from sample data
 - Server Manager opens the Server Manager, for server monitoring and configuration

7.3 OneDoc

The One document option is used to access a single document quickly, in three steps. First, select a library, then select a document type, then enter the document ID. As you enter the ID, documents starting with your input are displayed for faster selection. Press Enter when the desired ID value is highlighted or in the input field.



7.4 Browse Library

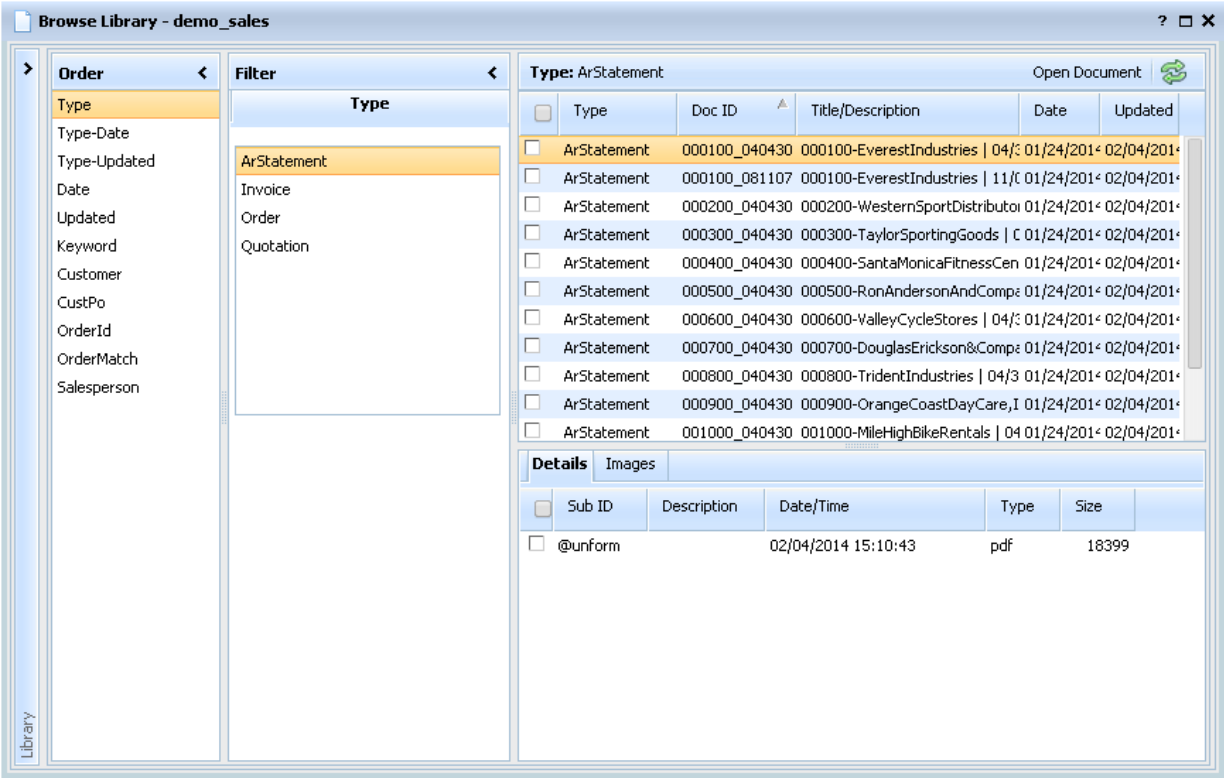
The Browse function is designed to provide access to sections of a library, such as all invoices, or all orders for a given day, or all documents that contain a specific keyword. It uses indexes, both standard and custom, to access specific sections of a library quickly.

Browsing is presented in five panels. The leftmost panel is used to select a library, and is normally minimized to save screen space. To select a library, click the open arrow to open the library panel. The Order panel is used to select the order of documents, and the Filter panel is used to filter the order. For example, if the Type-Date order is selected, the filter panel will contain both type and date selectors.

Once a filter is specified, the filtered list of documents is presented in the top right panel. Selecting a document displays its sub-documents in the lower right panel. The sub-documents can be presented as a list, or as thumbnail images, by selecting the appropriate tab. In addition, if the document type chosen supports custom forms, links to those forms are available in a third tab.

You can mark documents or images with the check boxes in the left column of each panel. [Marked documents](#) can be exported, copied, or transferred. [Marked images](#) can be emailed, faxed, or viewed together or side by side.

To view a document, with its properties and sub-documents, double click it or select it and press the Open Document button. To view a sub-document image, just click it.

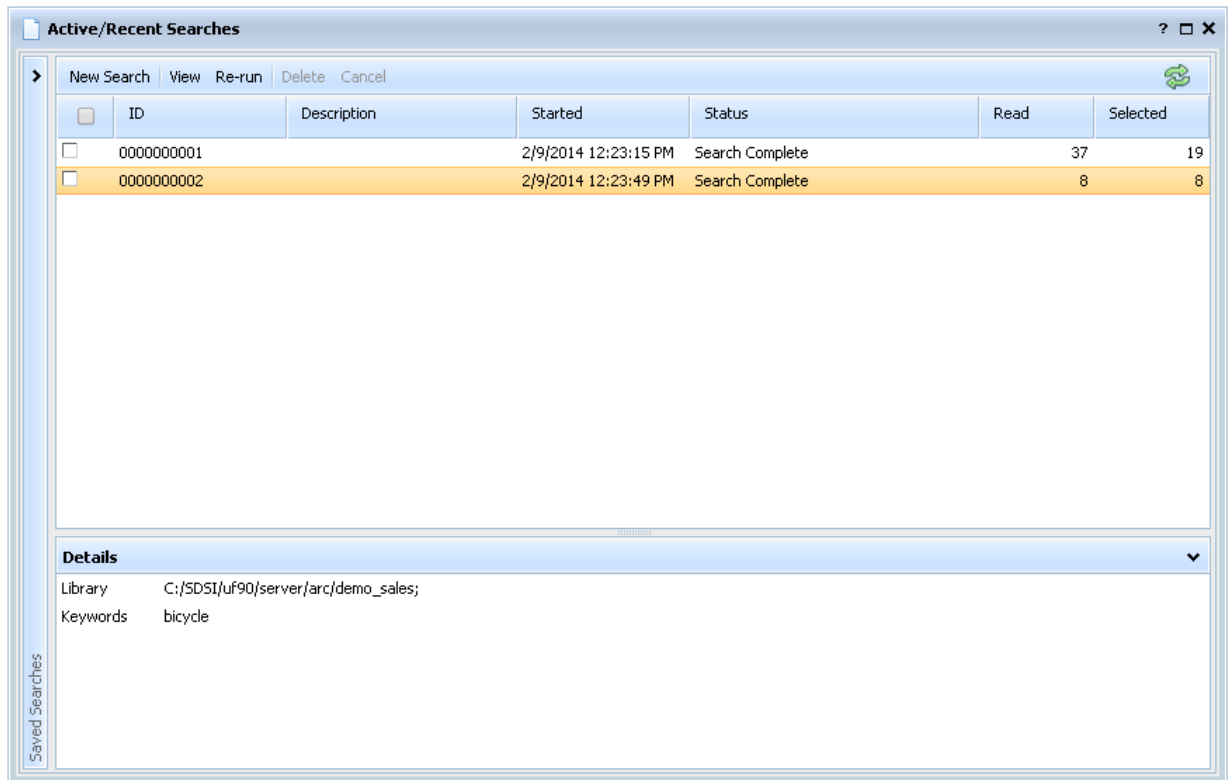


7.5 Searches

Searches scan one or more libraries for documents that match certain criteria. The result of a search is a collection of documents that can be viewed, exported, copied, or transferred to another library. Searches run in background on the server, and access to the result is provided through the Active/Recent Searches window.

The window displays searches that are running or have run recently. Click a search to see its status, or double-click it to see its results. The toolbar provides access to features, such as creating a new search, or viewing the results.

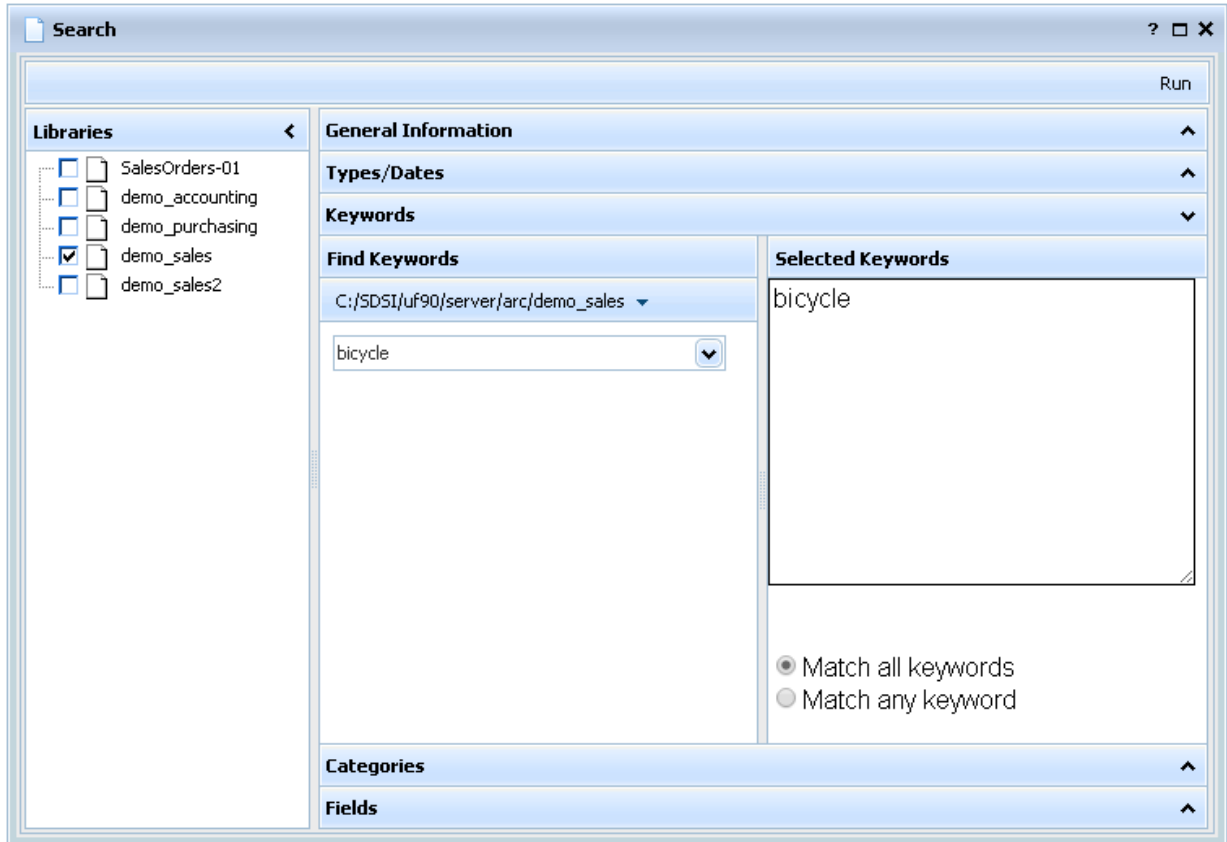
The left panel can be opened to access Saved Searches, which are search definitions that have been predefined. Saved searches can be defined for all users by an administrator, or a user can define their own, saving time for searches that are run many times. The Saved Searches panel also provides access to the form used to create a new [Saved Search](#).



7.6 Search Form

The Search Form is used to select one or more libraries and enter criteria for a search. Panels are provided for different types of criteria, including types, date ranges, keywords, categories, and other general fields, such as the document ID, title, entity ID, or notes. Once all criteria has been entered, press the Run button to start the search, which will run in background on the server. While the search is running, a status window displays its progress, and when it is complete, a view window opens.

This same form is used to edit Saved Searches, but instead of a Run button, there is a Save button, and the General Information tab provides fields for search name and, for administrators, the option to make the search available to all users.



Panels

General Information

- Enter a title to describe this search. This appears in the search list.
- Filter is a generic search field that matches against most properties of a document. A customer name, for example, might be a good filter to locate all documents for a given customer, though you might find it finds more documents than you expect.

Types/Dates

- Select one or more document types to limit which types will be searched.
- Enter date ranges for document or last-updated dates.

Keywords

- Select keywords by choosing a library and entering keywords in the left panel. They are collected in the right panel.
- Choose to match any or all keywords selected.

Categories

- Select categories by choosing a library and entering category fields in the left panel. They are collected in the right panel.
- Choose to match any or all categories selected.

Fields

- Enter criteria in any field. Simple text is matched against the field content, but additional features include:
- Use commas or the word " or " to search for any of a list of items
- Use semicolons or the word "and" to require all of a list of items
- Use double-dashes (--) to indicate a range (start -- end)
- Use asterisks to indicate a wildcard (*name*)
- Use ~expression to indicate a regular expression pattern match
- Use quotes to hide any of the above from interpretation, treating it as simple text

You can enter a run-time prompt at any text entry field, in the format `{{prompt text}}`. Prompt expressions will be substituted with a user-entered value at runtime.

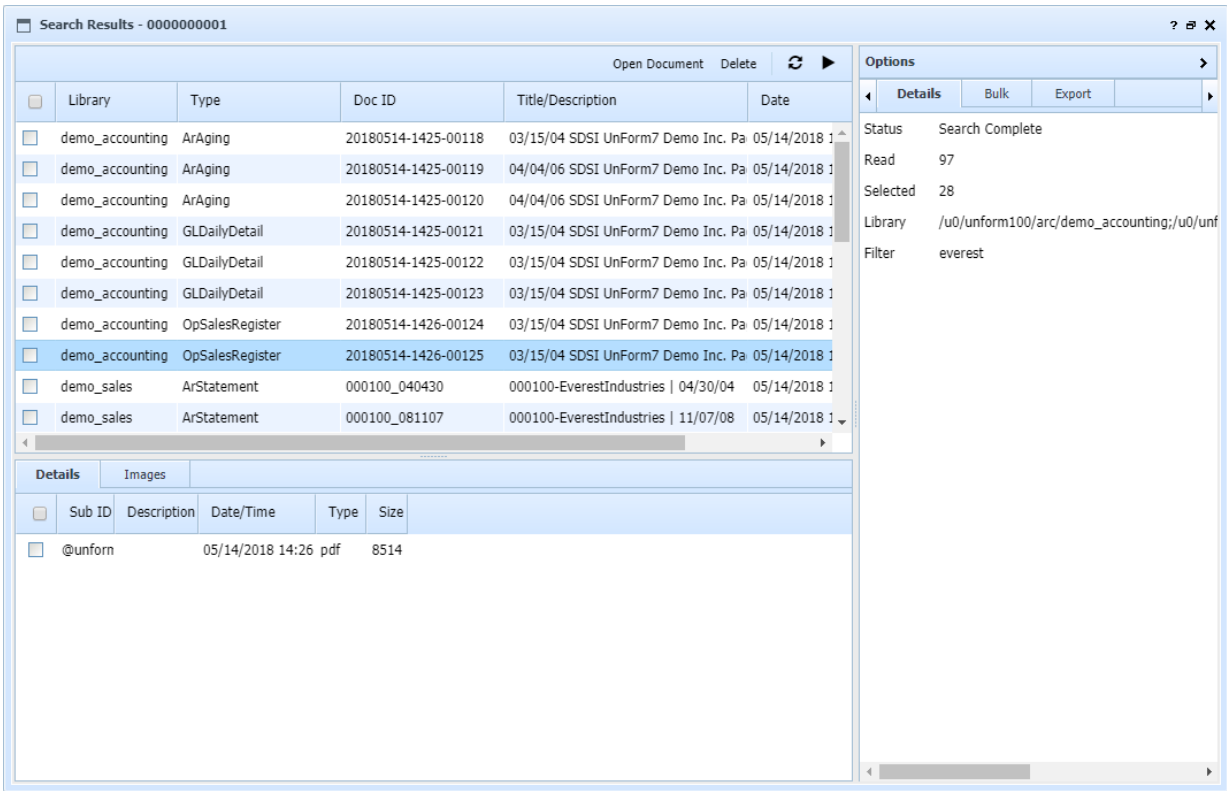
7.7 Search Results

The result of a search is a collection of documents, viewable with the Search Results window. The left panels show the documents for a given library, and selecting a document displays its sub-documents in list or thumbnail format. To view a document, double click it or select it and press the Open Document button. To view a sub-document image, click it in the lower left panel. There are also toolbar buttons to refresh the list or re-run the search.

You can mark documents or sub-images using the grid checkboxes. You can also select one or more documents, and then click the Delete button to remove the selected items from the search results. Note this does not delete the documents from the library itself, just from the search results list.

The most recently selected document can be opened by double-clicking or clicking the Open Document button.

There are several actions that can be performed on the collection of documents, available in the right panel, documented below.



Bulk

The Bulk tab provides functions to copy or transfer documents to a different library, or to export documents and sub-document images to a stand-alone HTML structure, or to list the documents to a server text file for potential use in external functions.

- **Copy/Transfer** Enter a target library path to send documents to, and when using copy or transfer, select a method to handle duplicates when the source and target libraries contain the same document ID. You can skip or overwrite duplicates and all their associated sub ID records (target sub ID's are removed before the write), or tell the system to choose the document with the most recent updated date and skip or overwrite conditionally based on the latest date updated (the date condition applies to sub ID's as well as document ID's, and finally, to perform a full merge, where document properties are merged, and sub ID's are merged, using sequencing to ensure no sub ID's are overwritten. If the target library does not exist, it will be created.
- **HTML Export** Enter a target directory to create the structure in. The file 'index.html' will be a root document to be loaded by a browser. If the directory is entered as a full path, it must be empty or new to export into it. If it is a simple, unpathed name, it is created in the ./htmlexports path in the UnForm server directory, after first being cleared.

Export

The Export tab provides links to export the document list, with or without sub-document details, to a browser-side CSV file. Browser handling of a CSV file is system-dependent, but typically results in a spreadsheet view.

Forms

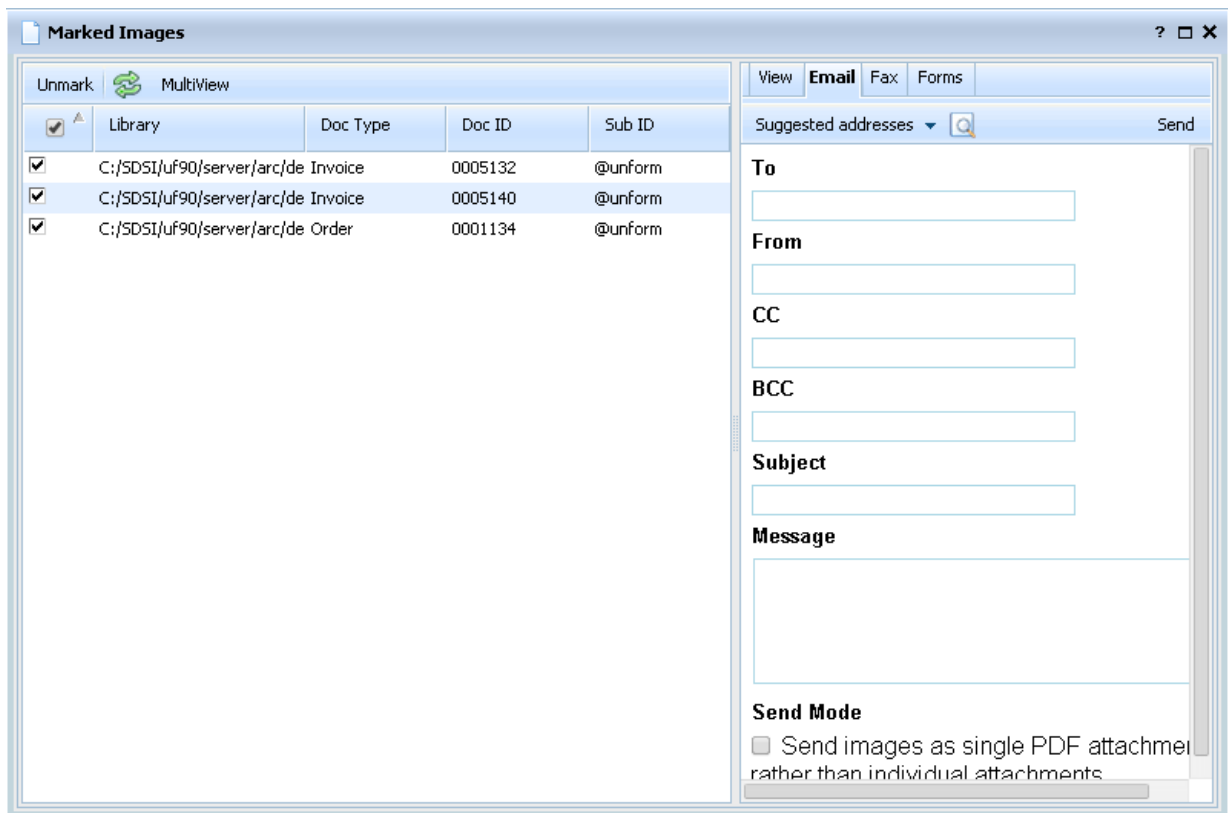
If configured, custom forms can be designed to operate on the search results. Links to these forms are presented in a Forms tab.

7.8 Marked Images

As users browse or otherwise view document images, they can select them using a checkbox. The selected images form a collection called Marked Images. The Marked Images browser window is used to work with this collection in several ways.

- View them in a single PDF file, with options for resolution, color, and tiling.
- Email them, either as multiple attachments or as a single PDF file (note that a single PDF file can be quite large).
- Fax them, if faxing is configured in the UnForm server.
- Use [MultiView](#), which displays images side by side for quick analysis and comparison
- If custom web forms have been configured for marked images, links to them are provided in the Forms tab.

Note this feature differs from [Marked Documents](#), which works with a collection of documents rather than document images.



7.9 Marked Documents

As users browse or otherwise view documents, they can select them using a checkbox. The selected documents form a collection called Marked Documents. The Marked Documents browser window is used to work with this collection in several ways.

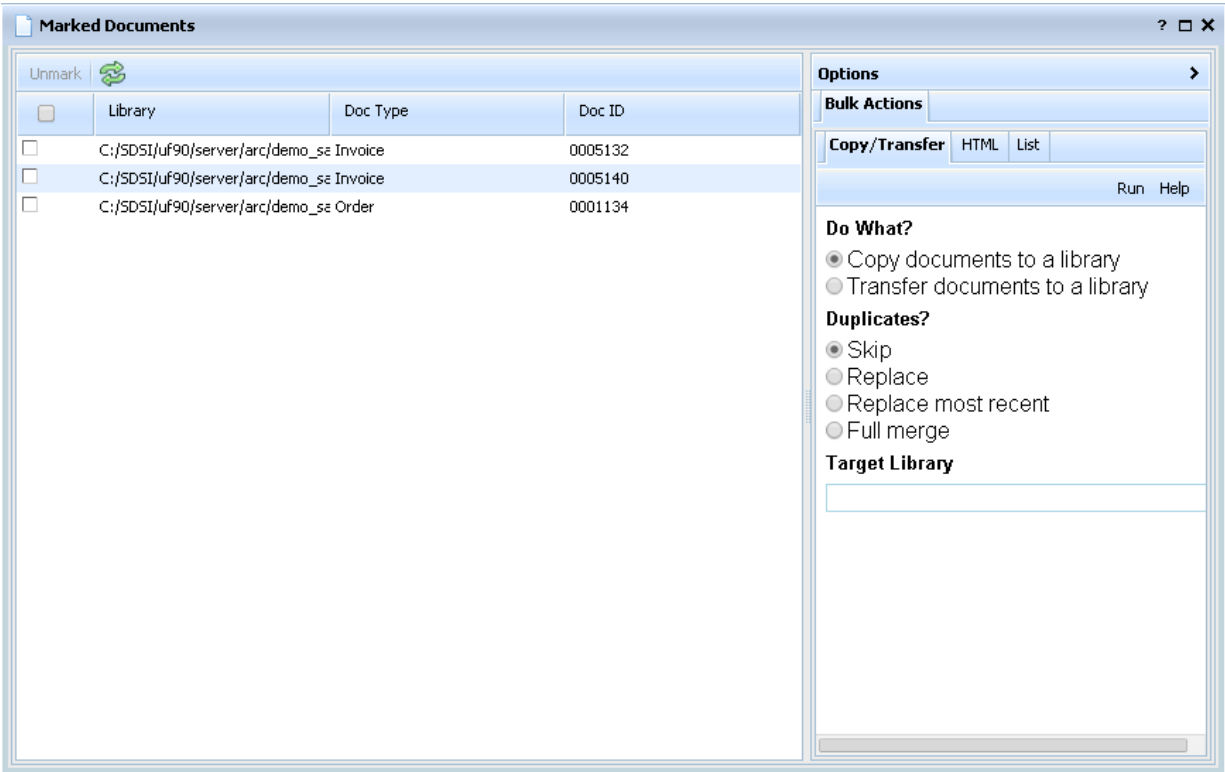
- Documents can be copied or transferred to another library, one that exists or will be new. Use this feature to merge libraries, or split them, or to purge documents by transferring them to a different library

that will later be removed from the system. When using this feature, there are several ways that duplicates can be handled (where the same document type and ID exists in the target library).

- Duplicates can be skipped (nothing transferred)
- Duplicates can be replaced (target documents and images are removed, then replaced)
- If the target document is older, based on last updated date and time, it is removed and replaced
- All images, keywords, categories, and links are merged

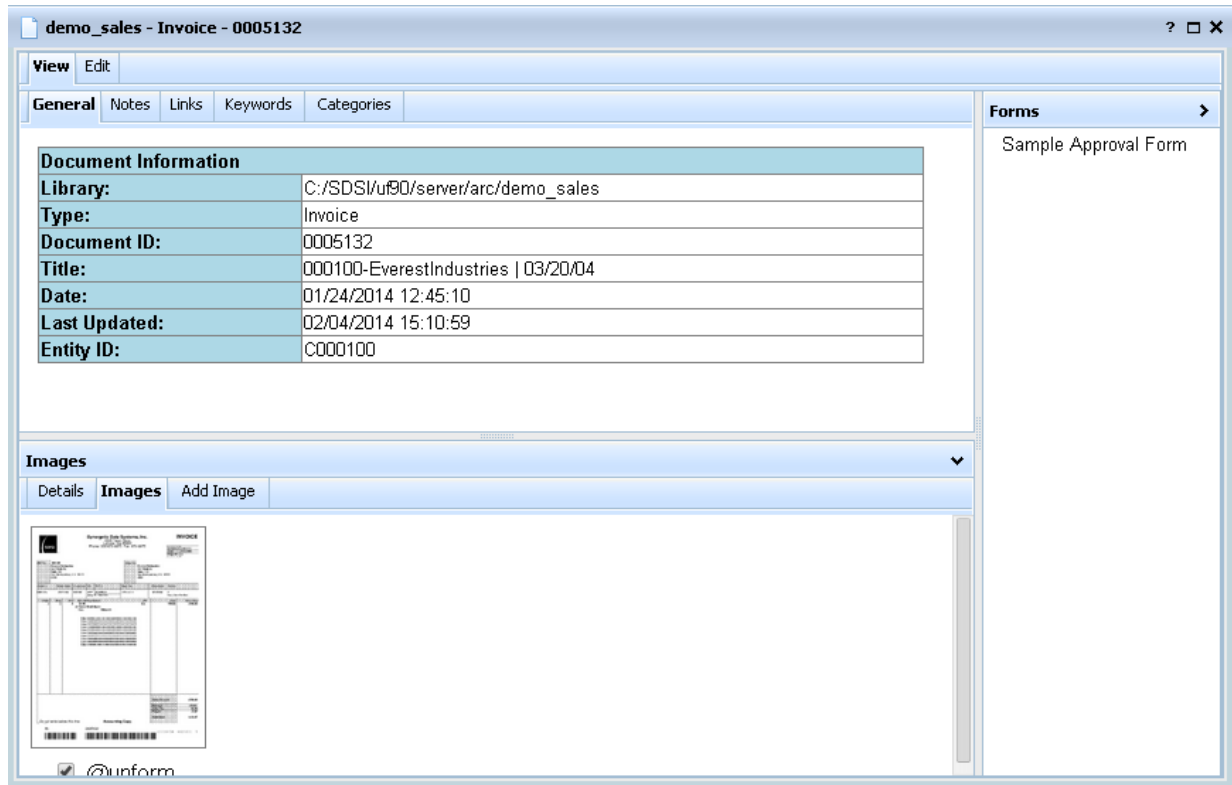
- Documents can be exported to a standalone HTML directory structure. This directory can be copied anywhere, and viewed using a web browser, without the use of the full document archiving interface. Use this feature to make document collections available to third parties, or for historical archival purposes.
- The list of documents can be exported to a server file for use in other processing.

Note this feature differs from [Marked Images](#), which works with a collection of document images rather than parent documents.



7.10 Document Viewer

The document viewer displays document properties and details or thumbnail images of sub-documents. With proper permissions, the properties can be edited, and addition images can be uploaded. If custom web forms have been configured for the current document type, a forms panel provides links to those web forms. To view a sub-document image, simply click it, and the window will be replaced with the [Image Viewer](#).



7.11 Image Viewer

The Image Viewer displays a sub-document image, along with tabs to view image properties and related documents and images. There are toolbar buttons to view the parent image in the Document Viewer, or to move the Image Viewer to an external window for a larger view, and also to refresh the image or view or download it as a standalone file.

The right side panel provides features to email or fax the image (if faxing is configured in the server), or with proper permissions, to edit its properties. The email and fax features support address books, and can suggest addresses if address book entries are found that match the entity ID of the document.

[illegible]

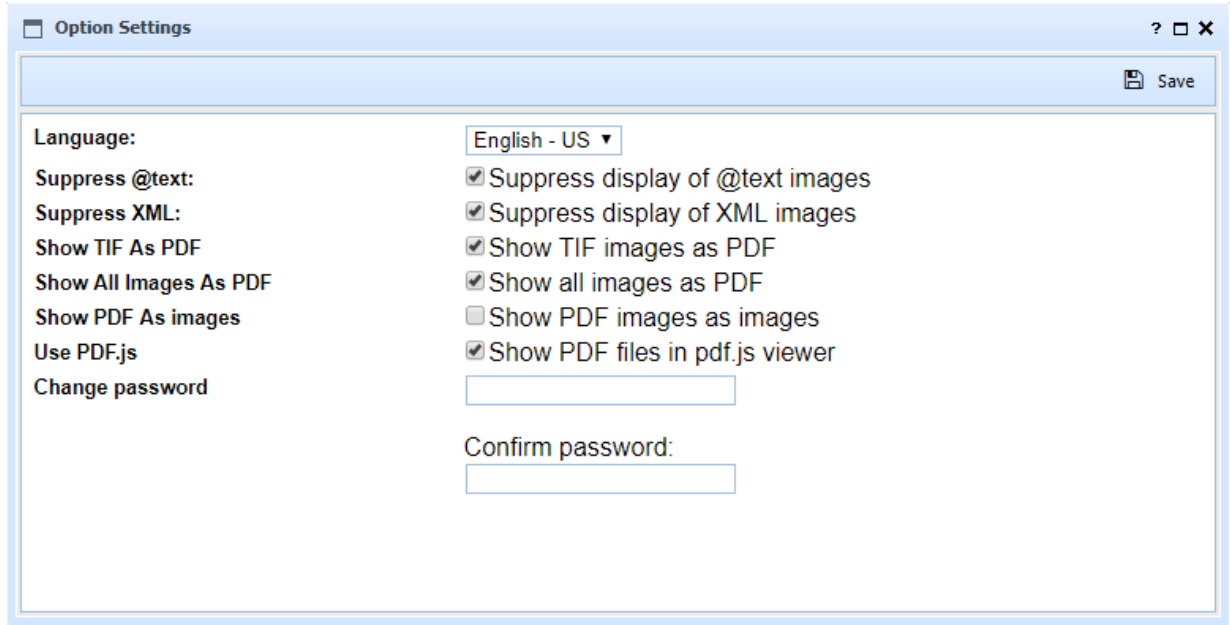
7.13 Custom Forms

Custom web forms are HTML documents that offer custom features that have been designed for a particular site or application. They are used extensively, for example, in UnForm's document workflow application. Access to web forms is based on section of the browser interface that is active, and a configuration file on the server. For example, generic web forms can be created and visible on the main menu, or document-type specific forms can be visible with that type of document is being viewed.

Configuration details are available in the [Custom Web Forms](#) chapter.

7.14 Settings

Use Settings to modify session settings that control how some portions of the browser interface work. For example, if you would like to see @text versions of sub-documents, uncheck the Suppress display of @text images option. If you are using a device that does not display PDF files as part of a web page, you can ask the server to convert such files to images for you.

A screenshot of a software window titled "Option Settings". The window has a light blue header bar with a "Save" button on the right. The main area contains a list of settings on the left and their corresponding controls on the right. The settings include: "Language:" with a dropdown menu showing "English - US"; "Suppress @text:" with a checked checkbox; "Suppress XML:" with a checked checkbox; "Show TIF As PDF" with a checked checkbox; "Show All Images As PDF" with a checked checkbox; "Show PDF As images" with an unchecked checkbox; "Use PDF.js" with a checked checkbox; and "Change password" with a text input field. Below the "Change password" field is a "Confirm password:" label and another text input field.

Option Settings

Save

Language: English - US ▼

Suppress @text: ☒ Suppress display of @text images

Suppress XML: ☒ Suppress display of XML images

Show TIF As PDF ☒ Show TIF images as PDF

Show All Images As PDF ☒ Show all images as PDF

Show PDF As images ☐ Show PDF images as images

Use PDF.js ☒ Show PDF files in pdf.js viewer

Change password

Confirm password:

7.15 Address Books

Address books are useful when faxing or emailing documents to third parties, such as customers, vendors, or employees. UnForm supports any number of address books, though it is also common to just have one. Address book entries are designed to support the UnForm deliver command as well as the browser interface. An address book entry is identified by an Entity ID, such as a customer ID or vendor number, and a document type. These two fields make it possible for UnForm to suggest email addresses when viewing a document, since the document's entity ID can be matched with records in the address books that are available. In addition, rule file coding that supports address book use can be designed to match entity ID and document type information to locate the proper email or fax information to deliver a document to.

Address books can be used by any user, but can only be edited by users granted permission by an administrator. In addition, when logged in as an administrator, a special toolbar is visible to allow creation and removal of address books, and also importing and exporting of address book records in CSV format.

The screenshot shows the 'Address Books' application window. At the top, there are 'Admin Options' including 'New Book', 'Delete Book', 'CSV Export', and 'CSV Import'. Below this is a 'Books' sidebar with a 'Customers' book selected. The main area displays a table of entries for the 'Customers' book. The table has columns for 'Entity ID', 'Doc Type', 'Entity Name', 'Contact Name', and 'Send To'. One entry is visible with Entity ID 'C000100', Entity Name 'Everest Industr A/P', and Send To 'ap@everestindi'. To the right of the table is an 'Edit Entry' form with fields for 'Entity ID', 'Document Type', 'Entity Name', 'Contact Name', 'Email/Fax', and a 'Combine' checkbox. The 'Entity ID' field is pre-filled with 'C000100', 'Entity Name' with 'Everest Industries', 'Contact Name' with 'A/P', and 'Email/Fax' with 'ap@everestindustries.com'. The 'Combine' checkbox is unchecked, with the label 'Combine documents with for single delivery'.

Entity ID	Doc Type	Entity Name	Contact Name	Send To
C000100		Everest Industr A/P		ap@everestindi

Edit Entry

Update Delete

Entity ID:
C000100

Document Type:

Entity Name:
Everest Industries

Contact Name:
A/P

Email/Fax:
ap@everestindustries.com

Combine:
☐ Combine documents with for single delivery

7.16 Administrator Options

The administration menu is available only when logged into the browser interface as an administrative user. The following options are available:

- [Libraries](#) - Manage archive libraries
- [Users](#) - Manage internal users
- [External Users](#) - Manage external users, who are users that can only view their own documents, based on entity ID
- [Groups](#) - Manage groups, which are groups of users who have similar permissions
- [Build Demo Libraries](#) - Create, or re-create, demo libraries
- [Server Manager](#) - opens the Server Manager, used to configure and monitor the UnForm server.

7.16.1 Libraries

Library Maintenance is used to add, edit, and manage libraries. Libraries are paths on disk on the UnForm server, used to store document archives. Libraries contain several files and sub-directories, though it is important to note that document image files are not stored directly in the file system.

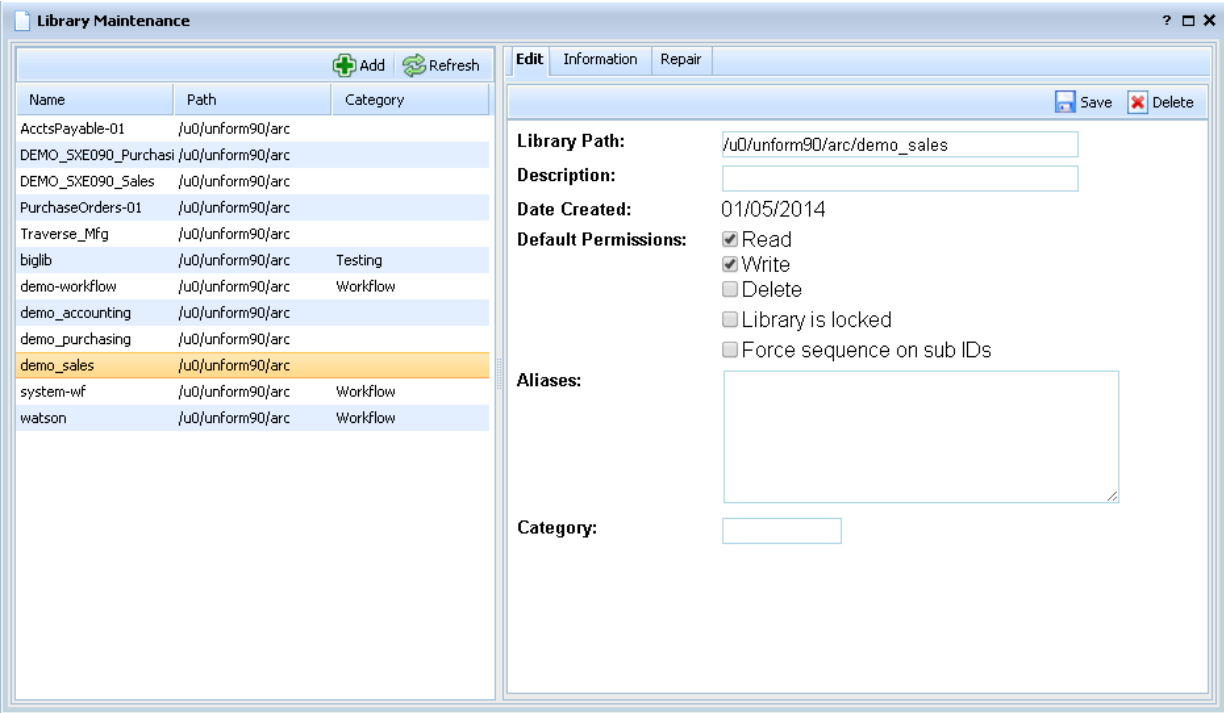
Adding a library will create an empty library path on disk. Deleting a library from this window will remove the library definition, but will not remove it from disk. To completely remove a library and its files from the system, you must remove its definition and then remove it manually from disk. Also, because libraries are automatically created when documents are added to them, often you will find a library appears without using the Add function of this window.

Default permissions are applied when a user or group has not been assigned specific permissions to the library. User and group permission assignments are defined in the user and group maintenance windows. Note that default permissions for an auto-created library are taken from the 'defperms=' setting in the uf101d.ini file.

Aliases is a list of alternate names for this library. When libraries are migrated between systems or renamed, external references to the old name will get errors indicating the library does not exist. To avoid this, add aliases to the active library. When an external reference, such as URL, uses an invalid library name, the system will scan aliases and use the correct library name that specifies that alias. Aliases should be full paths indicating the old name(s), one per line.

The Category field is used to provide structure to the browser interface when a library is to be selected. Libraries are listed in category groups, so you can establish groups like Sales or Accounting, or years, or whatever is appropriate for the libraries defined on a system.

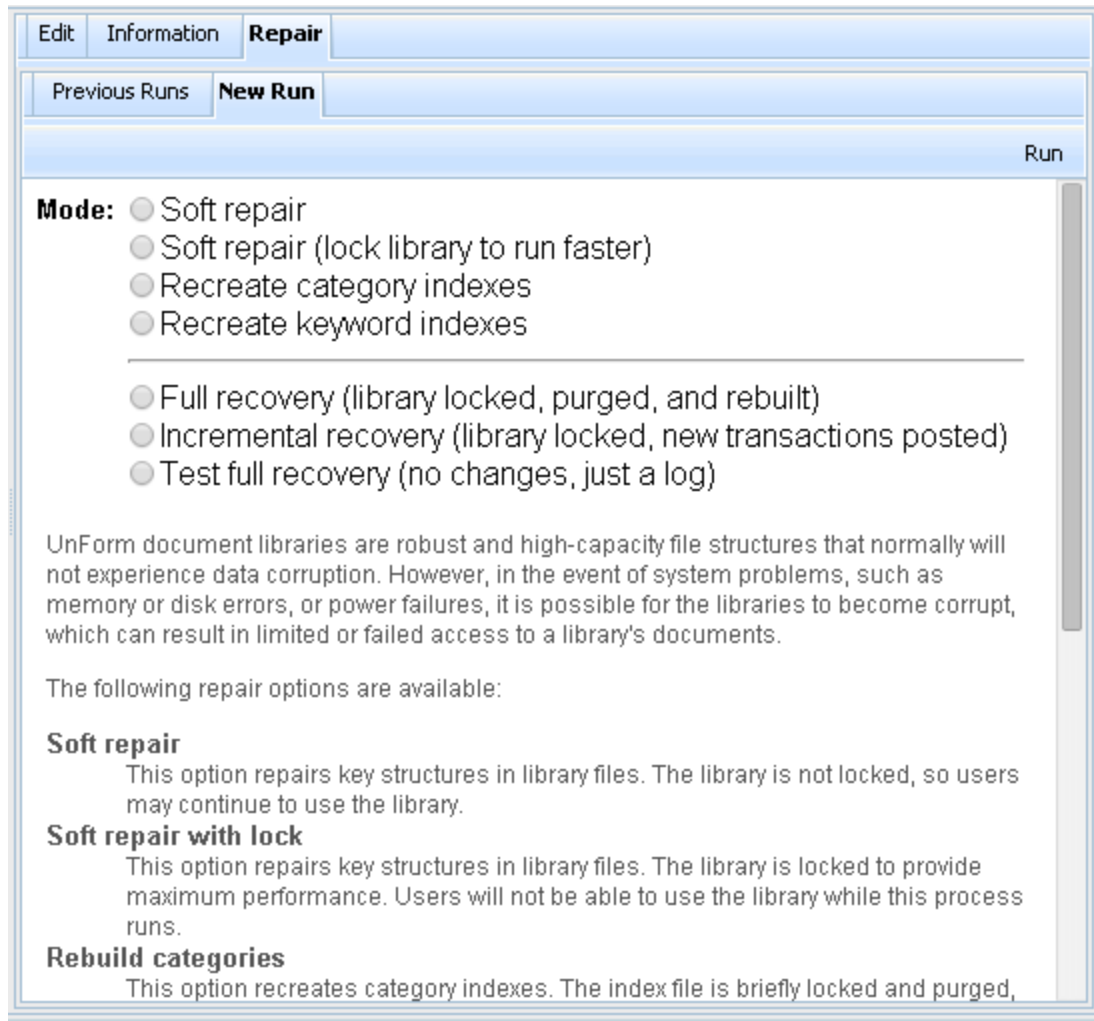
The Information tab provides some statistics about the library, such as number of documents, number of keywords, and key file sizes. The [Repair](#) tab is used to correct or recover from file corruption.



7.16.1.1 Library Repair

Libraries can occasionally encounter corruption, due to unexpected problems such as power or hardware failures, or disk space exhaustion. If you notice that a library displays corruption, typically noticed as unexpected error messages when running an archiving print job or in the browser interface, you can use the Repair tab to run one of several options against the library.

Repair options always run in background, and you can view the log of a previous run in the sub-tab "Previous Runs". To start a new repair job, use the New Run tab and select the desired repair mode.



Soft repair

This option repairs key structures in library files. The library is not locked, so users may continue to use the library.

Soft repair with lock

This option repairs key structures in library files. The library is locked to provide maximum performance. Users will not be able to use the library while this process runs.

Rebuild categories

This option recreates category indexes. The index file is briefly locked and purged, then reconstructed. After the initial purge, users may again use the library, but category indexes will not be fully available until the process is complete.

Rebuild keywords

This option recreates keyword indexes. The index file is briefly locked and purged, then reconstructed. After the initial purge, users may again use the library, but keyword indexes will not be fully available until the process is complete.

Full recovery

This option rebuilds the library from transactions in the library's recovery folder. The library is locked, fully purged, and reconstructed. Users will not be able to use the library while this process runs.

Incremental recovery

This option updates the library from transactions in the library's recovery folder. The library is not purged, and only new transactions are added. This option is normally used for replication purposes of a read-only library. The library is not locked during this process.

Test recovery

This option tests the recovery files in the library's recovery folder. It reads all recovery transaction records and produces a log, but does not perform any updates to the library. The library is not locked during this process.

Note that regardless of these library utilities, you should always maintain good backups of your libraries, to provide recovery in the case of catastrophic disk or system failure.

7.16.2 Users

The Internal Users window provides facilities to add and edit user details, including library access, group membership, and certain access permissions.

Internal users differ from External users in that Internal users are those that work within the organization that uses the UnForm server environment, whereas External Users are third party users that login and should only see documents that pertain to them. Internal users are granted certain permissions to whole libraries. External users are granted read only permissions only to their own documents, as determined by an Entity ID.

The left panel displays a list of active users, and a [+](#) to add a new user, or use the export and import functions to work with users in an external tool. The [Access](#) tab is used to specify the current user's library-specific permission settings.

External ID fields, name and password, not to be confused with External user definition, are used for custom requirements, such as for synchronization with an external service.

User ID	Name
Administrator	
Allen Miglore	Allen Miglore
Clark Jeppessen	Clark Jeppesen
Dan Schmitt	Dan Schmitt
Daryl Williams	Daryl Williams
Guest	
John Wilson	John Wilson
Miguel Escobar	Miguel Escobar
Patricia Osborn	Patricia Osborn
Ronald Anderson	Ronald Anderson
admin	Administrator
demo	demo
dpw	Daryl Williams
krbtgt	
mje	Michael Brumer

Edit

User ID:

John Wilson

User Name:

John Wilson

Email:

Company Name:

Telephone:

Date Entry Order:

☒ mm/dd/yyyy
☐ dd/mm/yyyy
☐ yyyy-mm-dd

Last Activity:

Permissions:

☐ Administrative user
☐ Workflow user

Key Permissions Settings

- Administrative users are granted full access to archive libraries through the browser interface, and to the design tool. They can also perform admin-only functions available on the Admin browser menu, like user and group maintenance. Note admin users are not automatically granted access to the Image Manager or DocFlow. Also note that if no admin user is defined, a default admin/admin user is created automatically, so be sure to maintain at least one administrative user.
- Allow Design Tool access enables a non-admin user to use the [Design Tool](#).
- Allow address book maintenance enables non-admin users to edit [address books](#).
- Allow browser Image Manager access enables this user to access the browser Image Manager. If this user is also an admin user, then document assignment and job history access enabled within the Image Manager interface.
- Allow job design in Image Manager enables access to the job configuration and custom script features of the browser Image Manager, or the job designer in the legacy image manager.
- A Password complexity level is defined in the [server configuration](#), and a Generate button is provided to generate a random password that meets the configured complexity level. User records can't be saved if the password doesn't meet the complexity level.

7.16.3 External Users

External user maintenance is similar to Internal Users, but external users do not have certain options enabled (they cannot be administrators, or design tool users, for example), and do have an Entity ID field that is required. Documents also have an Entity ID field, and external users can only view documents with a matching Entity ID value.

External users may be members of groups, and granted explicit [access](#) to certain libraries, but are only granted Read access to those libraries. Any library access must be explicit for the user or groups the user is a member of. Default library access, applicable to internal users, does not apply to external users.

The screenshot shows a web application window titled "External Users". On the left is a table with columns "User ID", "Name", and "Entity ID". The table contains one row: "everest", "Everest Industries", "C000100". Above the table are buttons: "Add", "Export CSV", "Import CSV", and "Refresh". On the right is an "Edit" form for the selected user. The form has tabs for "Edit" and "Access". The "Edit" tab is active, showing fields for "User ID", "User Name", "Password", "Confirm", "Email", "Company Name", "Telephone", "Date Entry Order", "Last Activity", and "Entity ID". The "User ID" field is "everest", "User Name" is "Everest Industries", "Password" and "Confirm" are masked with dots, "Email" is "ap@everest.bz", "Company Name" is empty, "Telephone" is empty, "Date Entry Order" has radio buttons for "mm/dd/yyyy" (selected), "dd/mm/yyyy", and "yyyy-mm-dd", "Last Activity" is empty, and "Entity ID" is "C000100". There is a "Show Me" button next to the password fields and a "Save" button at the bottom right of the form.

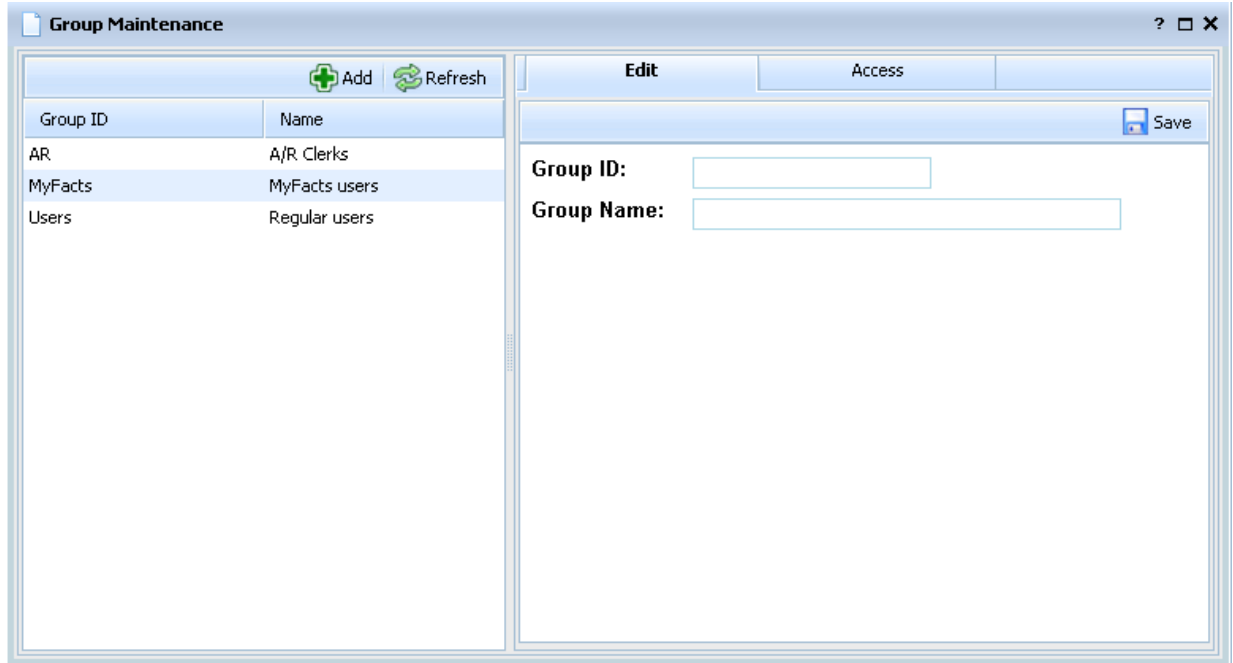
User ID	Name	Entity ID
everest	Everest Industries	C000100

Edit Form Fields:

- User ID: everest
- User Name: Everest Industries
- Password: (Show Me button)
- Confirm:
- ☐ User is inactive
- Email: ap@everest.bz
- Company Name: (Note: Company name and user telephone are used as defaults for fax cover sheets)
- Telephone: (empty)
- Date Entry Order:
 - ☒ mm/dd/yyyy
 - ☐ dd/mm/yyyy
 - ☐ yyyy-mm-dd
- Last Activity: (empty)
- Entity ID: C000100 (Note: User may only view documents with this entity ID.)

7.16.4 Groups

Groups are designed to simplify library permission management, by allowing a group of users the same access rights. As groups are defined, users can be given membership in those groups, and the group's [access table](#) is used as a default for those users.



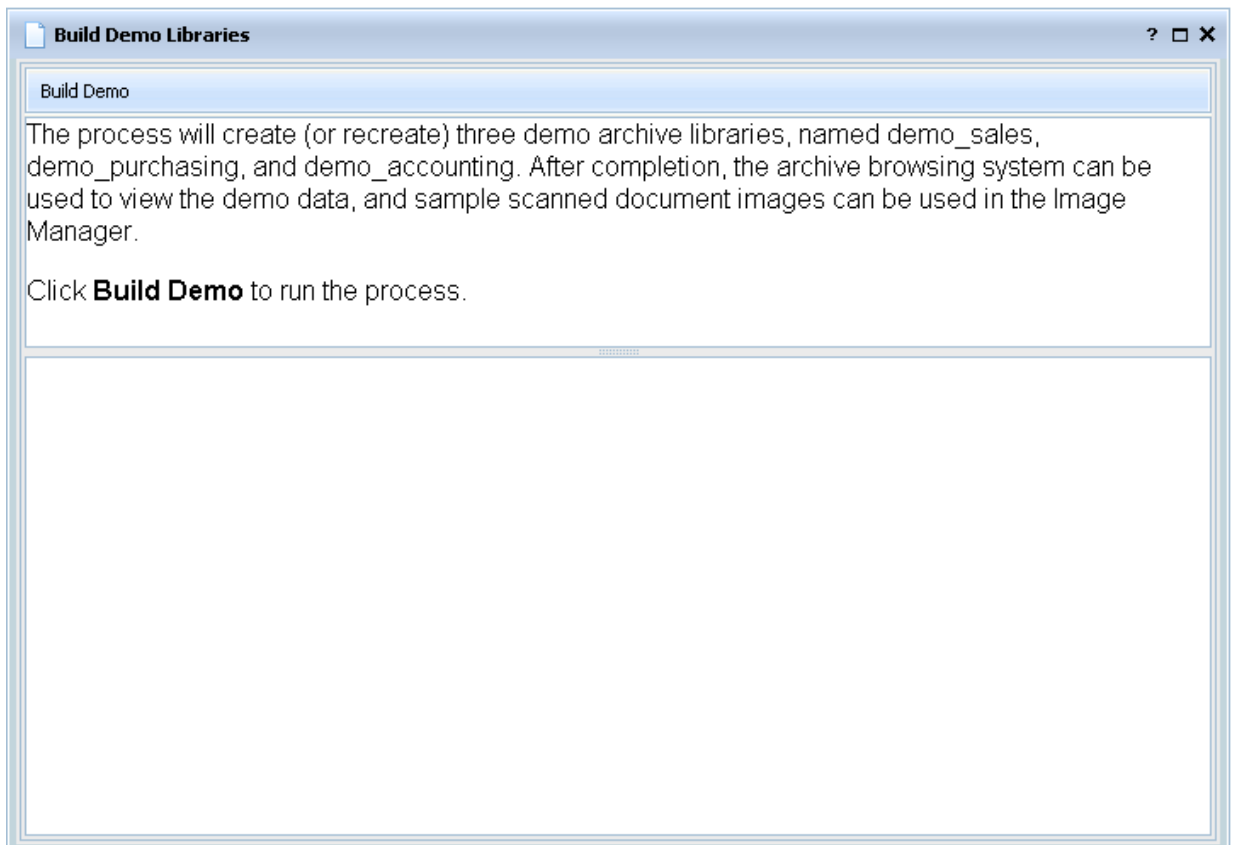
The **Group Maintenance** window is divided into two main sections. The left section contains a table with the following data:

Group ID	Name
AR	A/R Clerks
MyFacts	MyFacts users
Users	Regular users

Below the table are buttons for **Add** (with a green plus icon) and **Refresh** (with a green circular arrow icon). The right section has tabs for **Edit** and **Access**. The **Edit** tab is active, showing a **Save** button (with a floppy disk icon) and two input fields: **Group ID:** and **Group Name:**.

7.16.5 Build Demo Libraries

Use this window to create, or recreate, demo libraries. These are the demo_accounting, demo_purchasing, and demo_sales libraries.



The **Build Demo Libraries** window contains a section titled **Build Demo**. Below this title, the following text is displayed:

The process will create (or recreate) three demo archive libraries, named demo_sales, demo_purchasing, and demo_accounting. After completion, the archive browsing system can be used to view the demo data, and sample scanned document images can be used in the Image Manager.

Click **Build Demo** to run the process.

Below the text is a large, empty rectangular area, likely intended for a progress bar or additional instructions.

7.16.6 Access Tables

Users can be granted explicit permission settings to any library, or can be given default access based on the library definition. To guarantee no access, uncheck all boxes, including Default. To grant any explicit access, uncheck Default to enable Read, Write, and Delete check boxes.

Groups do not offer a 'default' option as group membership prevents default library access from being used; just enable the security desired for any member of that group who has not been granted explicit library permissions. If a user is a member of a group and does not have an explicit grant, then the group level permissions override library defaults.

Note that external users are not allowed write or delete permissions regardless of settings.

Edit

Access

Save

Library	Default	Read	Write	Delete
/u0/uniform90/arc/biglib	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
/u0/uniform90/arc/demo-workflow	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
/u0/uniform90/arc/demo_accounting	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
/u0/uniform90/arc/demo_purchasing	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
/u0/uniform90/arc/demo_sales	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

7.16.7 Active Directory/LDAP Synchronization

UnForm can synchronize some or all users and groups with an LDAP or Active Directory server. This process creates groups, users, and group membership from the LDAP server, while also allowing private UnForm users to be maintained as well. Any user created from the synchronization process is authorized against the same server, so no internal password is maintained. Any time the synchronization is performed, the LDAP-based user and group lists are updated, while specific UnForm properties, such as design tool or workflow access options, are maintained.

Active Directory uses the LDAP protocol, so this process works with that type of server as well.

Profiles

In addition to a default profile, stored in `ldap.ini`, you can add profiles by using the Add Profile button, or simply copying `ldap.ini.sds` to `ldap.profilename.ini`. This enables sites with multiple LDAP/AD servers to synchronize users and groups from different servers. Select the profile desired before running the sync process.

Sync

Instructions

Synchronize

Before running this process, be sure to backup the user and group databases. These files were last backed up on: .

- [Backup the user/group databases](#)

Profile:

Server:

[ssl:]tls:]server[:port]

Examples:

ssl:myserver:699 (uses ldaps, non-standard port)

192.168.1.130:389 (open, standard but specified port)

tls:10.10.1.100 (standard port, uses STARTTLS)

Domain:

i.e. company.local, or somewhere.com

Server Type: ☒ Active Directory

☐ LDAP (Posix)

Admin Login:

Password:

The server value supports three modes of operation, which must match how the LDAP server operates::

- Plain LDAP, without STARTTLS support (default port is 389)
- LDAPS, which connects to the server using SSL (prefix with ssl:, default port is 686)
- LDAP with STARTTLS, which converts to TLS/SSL mode after connection (prefix with tls:, default port 389)

The LDAP configuration expects a domain to form the base distinguished name values (sdsi.local, for example, becomes dc=sdsi,dc=local in LDAP searches).

The LDAP configuration found in ldap.ini support different database structures. Select one that works for the structure used. Additional structures can be configured easily. The ldap.ini file is distributed as "ldap.ini.sds", and copied automatically if not found when UnForm is installed or first started. If you customize ldap.ini, you can find the publisher-supplied version of definitions in ldap.ini.sds. The file is self-documented with comments.

The login and password are required to access the server and import user and group information. The LDAP server will validate these values.

If there are no errors, users, groups, and group membership information will be imported into the UnForm structures, and subsequent attempts to login to UnForm by an LDAP-based user will be authenticated against the LDAP server.

Since this process is potentially destructive, you can backup the user and group databases

7.16.8 Inbound Sources

An inbound source is a location where documents can arrive into the UnForm system. There are two types, directory and email. When files or attachments are received, UnForm monitor programs will retrieve them and either place them in an inbound library for Image Manager processing, or run them as print jobs via the rpq submission directory. There can be multiple monitors running, one for path sources, and one for each mail source.

The UnForm server will detect changes to sources every minute and will restart monitors if necessary.

Inbound Document Sources

List

+ Add

Inbound ID	Description	Source
doc1	Detect and run ANY job type	/tmp/inbound/doc1
doc2	Import Only	/tmp/inbound/doc2
mail	Detect and run Vendor AP jobs	inbound@synergetic-data.com
mje	mje Import Only	/tmp/inbound/mje
pick	Run picklist job	/tmp/inbound/pick
print		/tmp/inbound/print

Edit

Access

Save

Delete

Inbound ID:

doc2

Description:

Import Only

☐ Source is inactive

Type:

☒ Monitored Path

☐ Email Address

Source:

/tmp/inbound/doc2

Enter a directory to monitor

Filter:

*.pdf

*.png

*.tif

*.jpg

Enter file type wildcards (i.e. *.pdf), one per line, to restrict files loaded from this source

No OCR:

☒ No OCR processing needed

If documents from this source do not require OCR processing, initial parse performance can be improved with this option. This does not affect PDF files with embedded text already present.

Action:

☒ Import only

☐ Detect and run job

☐ Assign and run job

☐ Run as print

Type determines what type of source is being configured. The input fields change based on the type. For a monitored path, enter the directory to be monitored in the Source field. For email sources, fill in the email access information, including IMAP server information, mailbox, and login. Through mailbox aliasing, or mail list configuration, a single mailbox can deliver documents to alternate inbound sources. To do so, the alias or list address should contain +sourceid or .sourceid in the address (case-insensitive).

Filter is a list of wildcards indicating what types of files will be extracted from the source.

No OCR is a setting that determines if UnForm will utilize an internal Tesseract OCR engine when no text layer is found in the incoming images. OCR is necessary for some types of Image Manager jobs, but generating it is time consuming, so if files coming into this source do not typically require it, the processing can be skipped. An operator can run OCR processing later if needed. Note that incoming files

with existing text, such as ERP-generated PDF files, OCR is enabled regardless, as it is not time consuming.

OCR Service is visible in sites that have subscribed to the [UnForm OCR Service](#), an AI-based OCR processing service. If checked, when images require OCR processing in this inbound source, that service will be used rather than a local OCR engine.

If the service is enabled, you can also choose to "Upload color images when possible". If checked, and a color image is imported into the inbound source, the color image is uploaded to the OCR service. This can improve results if the images have color backgrounds. However, since color images are much larger than black and white images, this increases the network resources used to upload the image pages.

It is possible to determine page rotation from the OCR data returned by the service. A checkbox enables automatic rotation of pages to present their text left-to-right and top-to-bottom. Note enabling this option can increase service cost, as image pages that require rotation are processed a second time through the service. Note that the submission of rotated images uses 1-bit images regardless of the color option selection above.

Action determines what to do with the files as they arrive. They can simply be stored in an inbound library, awaiting Image Manager operator action. A job can be detected or assigned, and run. Or the files can be sent to the UnForm print system via the rpq directory. If this option is selected, an additional input field is provided to enter command line options, such as rule file and output designations.

When a print action is specified, the print job automatically receives some parameters. These parameters are accessed with a `prm("@name")` function in expressions and code blocks. The parameters are:

- For path sources, `@source` (source id) and `@filename` (basename of file)
- For email sources, `@source` (source id), `@from` (from email), `@to` (to email), `@subject` (email subject line), `@emlfile` (rfc822 email content file, usable by the [mailmessage](#) object's `parsefile` method)

Email Sources with Modern (OAuth2) Authentication

UnForm 10.1 supports XAUTH2 authentication with an IMAP server, as well as traditional LOGIN authentication. This requires configuring a dedicated [user record](#) with its Notes field filled in with text lines provided by the publisher through a registration procedure at <https://unform.com/support/oauthcode.html>. The user record can be disabled to prevent its use other than as a secure storage location for the authentication data.

After registering an application with your mail provider, such as the App Registration pages at Microsoft 365's Azure Active Directory, you can visit the above web page to configure, authorize, and receive the tokens required for UnForm to perform this type of authentication. Instructions are found at that page.

Once a user has been configured with the proper lines in its Notes field, you can specify which user contains the token information with a specially formatted password: `oauth:userid`. For example, if you set up a user "z_imap", use the password "oauth:z_imap" in the email configuration for the source. That format of password triggers the XAUTH2 authentication method.

The notes field should look something like this (lines here are truncated):

```
cid=2991095c-2b2a-4...
cs=Z2F8Q~5dZSM4v...
ep=https://login.microsoftonline.com/ab44...
tk=LCJhbGciOiJSUzI1NiIsIng1dCI6ImptT...
rf=0.ARIAlplEqxeM...
```

For Microsoft 365 users, here are more detailed [instructions](#).

7.16.9 Web Extension Deployment

The SDSI [Web Extension](#) provides added functionality for many UnForm users. However, each user requires configuration in order for the extension to communicate with an UnForm server. The deployment tool works with the UnForm user list, and can email users an instruction email that includes a copy/paste string that will automatically configure the extension for that user.

Fill in the form, select a user, or check all desired users, and click one of the toolbar buttons to send an email. The Email One button is used to send an email to one selected user, and you are given the opportunity to change the email address the message will be sent to. This enables an administrator to send an email to him/herself, or send it to an address not on file for the user. The Email All button will send an email to each user that is checked and has an email address on file.

To fill in the form:

- The server URL needs to be the base URL to the server that the extension will connect to. It is normally the same URL the administrator uses, but this can be overridden to ensure an external or network-wide address is used.
- Expiration is the number of hours the copy/paste content of the email is valid.
- Site list is a list of external URL's or wildcards that determine when the web extension displays PDF files in its custom viewer.

Web Extension Deployment

Email One Email All

Server URL:

The URL the web extension will connect to, including the path portion but no parameters. For example: https://unform.example.com:27392/arc. **Important:** If the web extension integrates with web sites served via https (secure sites), the UnForm server must also be served via https.

Expiration: hour(s)

Site List:

Users

<input type="checkbox"/>	User ID	Name	Email
<input type="checkbox"/>	demo_cole	Cole	demo@synergetic-data.com
<input type="checkbox"/>	demo_diaz	Diaz	demo@synergetic-data.com
<input type="checkbox"/>	demo_hank	Hank	demo@synergetic-data.com
<input type="checkbox"/>	demo_heyu	Heyu	demo@synergetic-data.com
<input type="checkbox"/>	demo_lee	Lee	demo@synergetic-data.com
<input type="checkbox"/>	demo_manu	Manu	demo@synergetic-data.com

8 DESIGN TOOL

The UnForm Designer is a browser-based product that is served by the UnForm web server. It provides a flexible environment for editing UnForm [rule files](#) and producing previews of jobs. The designer fully understands the structure of an UnForm rule file, and the many UnForm commands, and provides syntax highlighting, inline help for commands and functions, and a preview pane that can display either PDF documents or HTML5 documents.

When you use the designer, you are maintaining rule files on the UnForm server, and you are using the server to produce the previews of the jobs. For that reason, there is full compatibility between the designer and the UnForm server. What you see in the design environment is what you will see when live jobs are run, except for device differences. Test printing is supported as well, so you can verify printed output works correctly as well.

For those users who are already familiar with editing rule files manually, there are a few simple concepts to be aware of when using the designer.

- When the designer opens a rule file, it is opening a file on the server, not a local file on the client (unless, of course, the server is running locally).
- When the designer saves a file, it is saved on the server as a .rud file, rather than a .rul file, in order to avoid incomplete or untested changes to live rule files.
- When you are ready to make a rule file live, you *publish* it, at which time the designer saves it as a .rul file on the server. The designer's File menu has both Save and Publish options

The UnForm Design Tool is a browser-based product, accessible via the UnForm web server. You can access it via the web server's portal page, or directly with a URL. The URL structure is simple. If the web server is on IP address 192.168.1.10 and listening on the standard 27402 port, you would use this link:

<http://192.168.1.10:27402/arc?dsn=1>

The design tool is presented in a single [main window](#), used for editing and testing rule files. There are numerous [sample rule files](#) available for review that demonstrate various features and techniques.

8.1 About 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 commands and (usually) a list of parameters separated by commas. Commands instruct UnForm how to process the job. Here are some of the most commonly used commands:

detect	Used to detect which print stream is being handled by UnForm. For example, an invoice rule set might detect the word Invoice at a certain position.
cols	Scales output to fit a set number of columns.
rows	Scales output to fit a set number of rows.
text	Adds text to the output.
box	Adds a box or line to the output.
image	Adds an image to the output
barcode	Adds a barcode to the output
micr	Adds a MICR font line to the output
font	Changes the font of a region of input stream text.

UnForm includes many sample rule files, including simple.rul, advanced.rul, arcdemo.rul, and others, all found in the samples directory.

8.2 Sample Rule Files

There are numerous sample rule files in the samples directory provided with UnForm. All contain rule sets that demonstrate functionality and provide coding examples. For example, simple.rul provides four versions of an invoice, with increasing complexity with each rule set. The arcdemo.rul file shows several archiving techniques and is used to build the sample demo_* archives.

A number of rule files have been designed which focus on a particular command or technique. These rule files start with SampleCmd and SampleTool, respectively.

In the [Open Rule File](#) window, select the samples directory to see the full list.

8.3 Windows

8.3.1 Main Window

The design tool is a browser-based tool that provides tools to edit rule files on the UnForm server. The design tool is easily accessed from the UnForm [web server portal](#), or directly via the standard access URL with a "?dsn=1" suffix (i.e. <http://192.168.1.10:27402/arc?dsn=1>).

The screenshot displays the UnForm Design Tool interface. On the left, the 'simple4' rule set is being edited. The rule set includes commands for setting the sample name, detecting data, defining constants for columns, rows, and copies, and setting display options like DPI and graphics. On the right, the 'Preview' tab shows a generated invoice. The invoice header includes company name, address, and contact information. It details the order number, date, and ship date. A table lists items with descriptions, quantities, and prices. The bottom section shows a summary of sales, discounts, taxes, and freight, totaling \$830.00.

The elements of the Design Tool's main window include:

- A menu bar
- A rule set editor panel
- A preview panel, with preview, grid, sample, and [watch](#) tabs
- A syntax help panel, that displays syntax details for the current line in the editor

A menu bar provides access to:

- File menu for opening and saving rule files, and setting options
- Set menu for selecting and creating rule sets within the rule file
- Edit menu with features for searching, locating lines, inserting data, block comment/uncomment, code fold/unfold, and auto-formatting

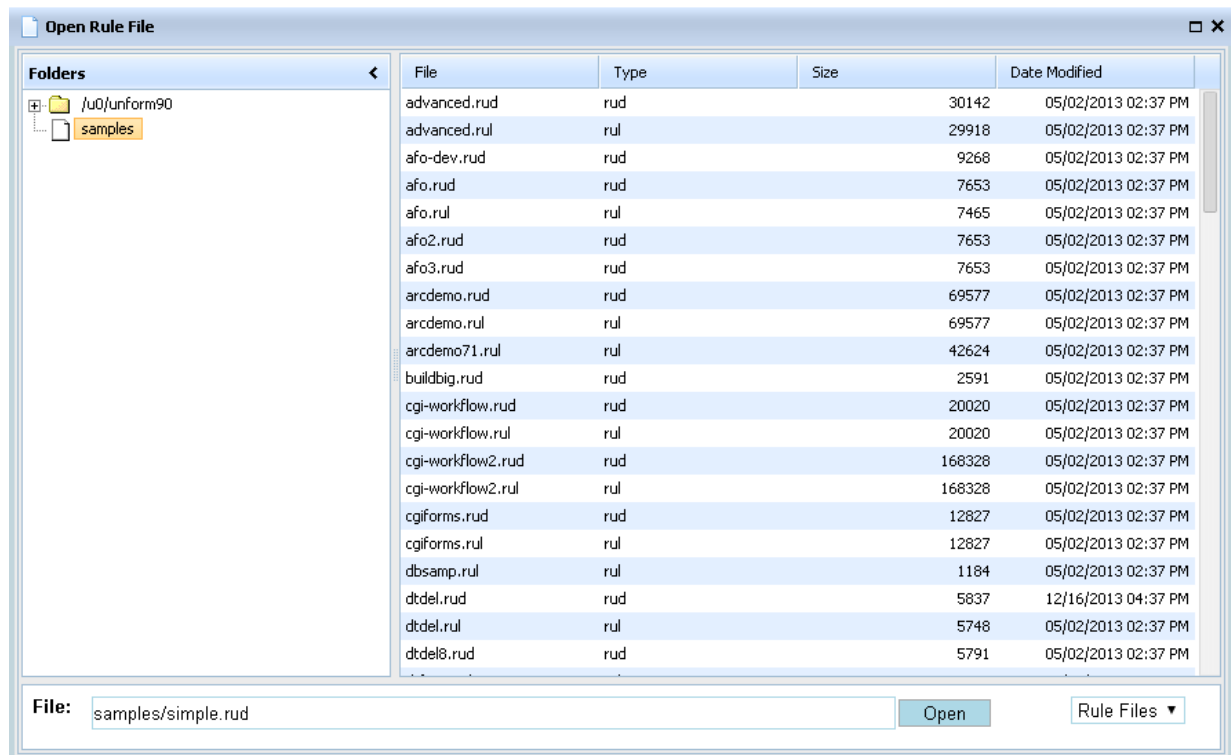
A second menu bar applies to the tab selected over the preview panel:

- Two run buttons launch a preview or grid preview, respectively, are available on all tabs
- Preview tab options include [Test Printing](#) and a set of [Options](#) pertaining to preview generation
- Sample tab options include options related to the [Sample](#) tab, such as creating a new sample, opening an existing sample file, selecting a dsn_sample file, and setting the text font size

Code folding is supported in the left-side gutter, as well as block highlighting (such as loops). Code folding is supported for comment groups and for indented sections of the rule set.

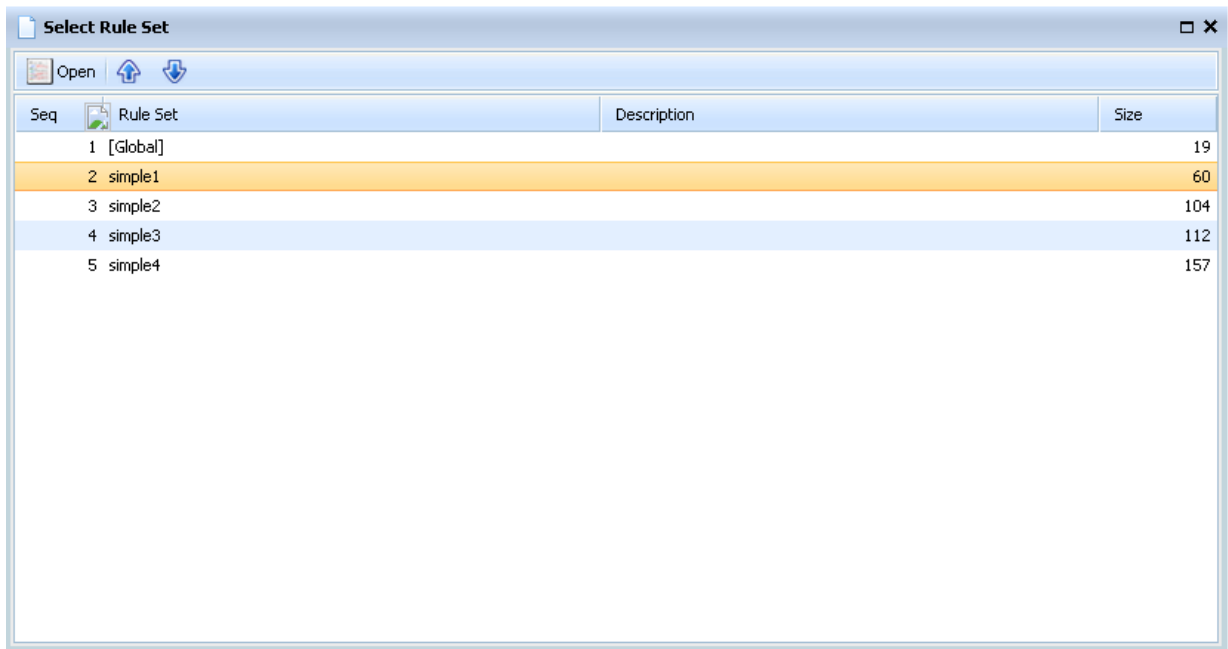
8.3.2 Open/Save File Window

When you open or save a rule file or sample file, the Design Tool presents a dialog window to help you navigate to the proper location and select or enter a file name. The file name to be used by the operation is shown in the File text box. At the lower-right of the window is a selector for file types to display, which will default to an appropriate setting based on the operation. You can sort the table of files by any column to help locate a file out of a large list. To proceed with the operation with a selected file, double-click the file name you want, or press the Open button when the file has been selected.



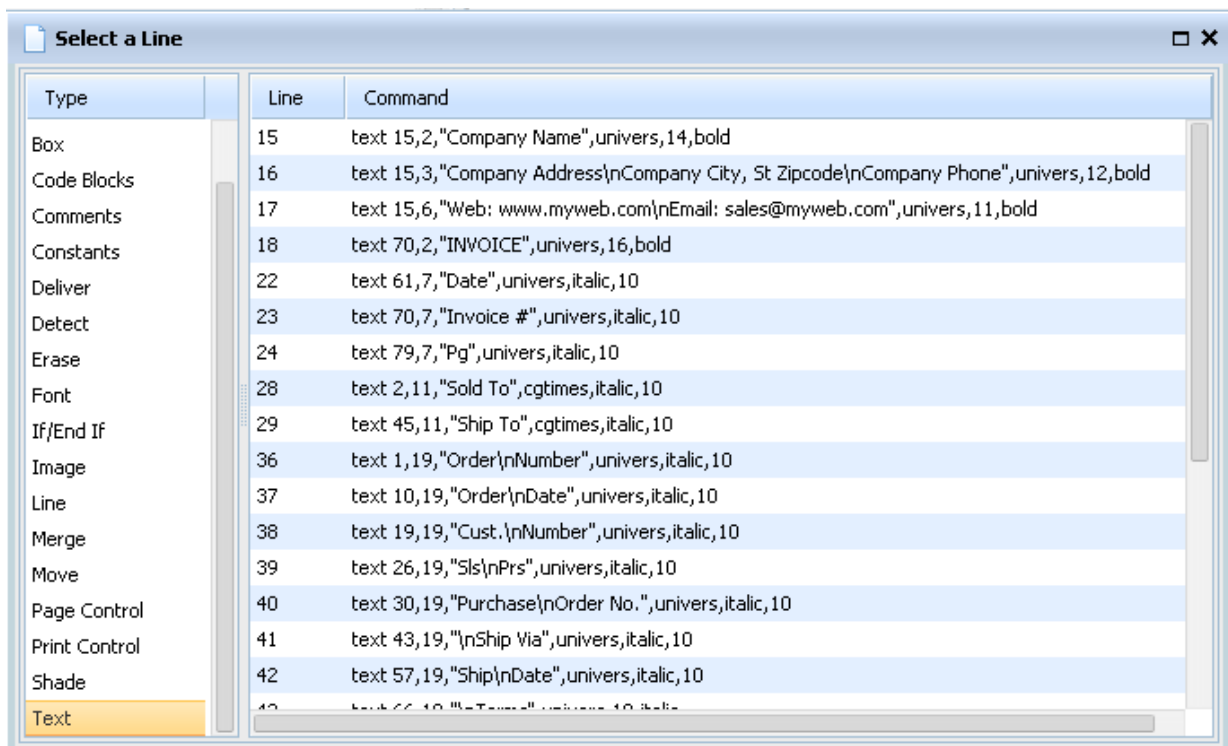
8.3.3 Select Rule Set

The Select Rule Set window is used to quickly open a rule set from a list of rule sets in the rule file. Just double click the desired set, or select it and click Open. You can also use this window to reorder rule sets within the rule file. Sometimes this is necessary to control the detection order when auto-detection is being used in an UnForm job. If two jobs might detect the same print stream, the one with more specific detection rules should be earlier in the rule file.



8.3.4 Select Line

The Select a Line window provides quick access to different types of commands in the current rule set. It is similar to search, but structured around commands or types of commands. Choose a command or category in the left panel, and click the line you want to jump to. That line will become the current line in the editor.



8.3.5 Search and Replace

The Search/Replace window provides quick location of specific content in the rule set or rule file. Enter a Search value, select one of the options, and press Search. A list of matching lines will appear in the lower panel. Clicking one of these lines will make that line and rule set active. You can also replace (or replace all) the Search value with the Replace value, by clicking the appropriate button.

Search/Replace

Search: Order

Replace:

Options:

Match case

Whole words

Search whole file

Search

Replace

Replace All

Ruleset	Position	Command
simple1	12,36	text 1,19," Order \nNumber",univers,italic,10
simple1	13,37	text 10,19," Order \nDate",univers,italic,10
simple1	23,40	text 30,19,"Purchase\n Order No.",univers,italic,10

8.3.6 Test Printing

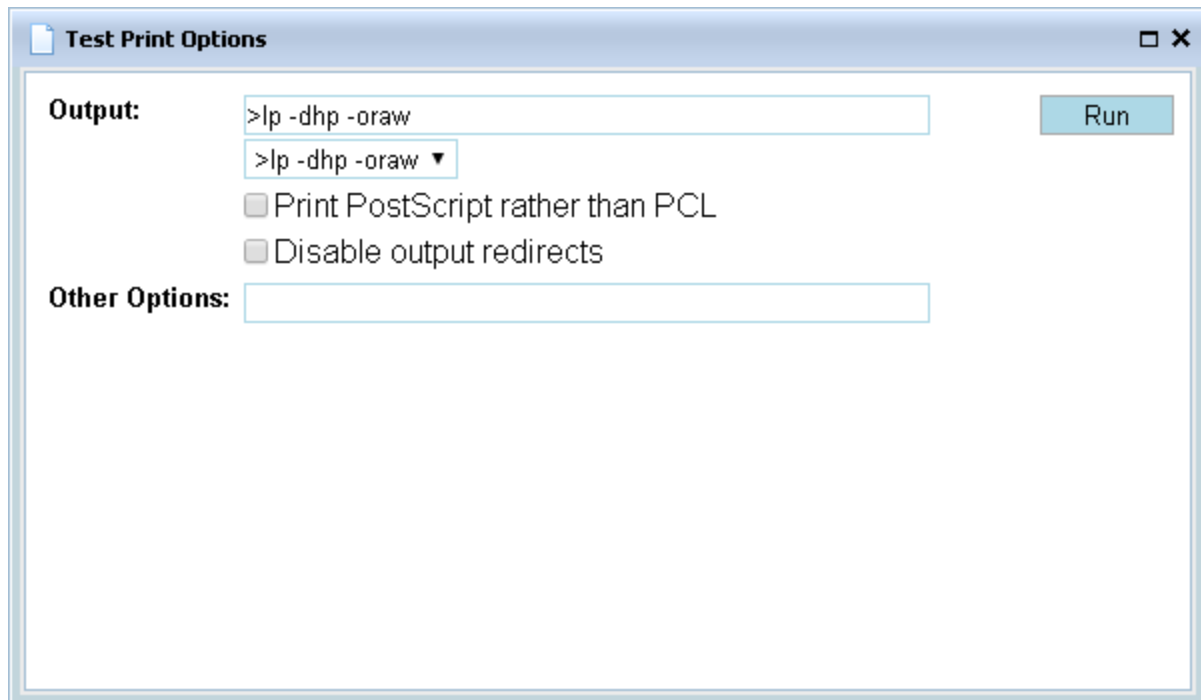
Test printing is important when you want to test a rule set using printed output rather than the PDF or HTML5 output used by the preview. Test prints go to a server printer, or can be returned to the browser (though they typically are not viewable by the browser and are just downloaded). Recent printers are remembered by the test print window, with a drop down selector.

Unix servers typically have printers referenced by piping to spooler commands, such as in the example below. On Windows, the printer can be a UNC (shared printer name). On both Unix and Windows, it can be a network address, entered as: "tcp:ipaddress" (i.e. tcp:192.168.1.45).

To test printing using Postscript, check the Print Postscript option. If the rule set redirects output, with an output command or setting the output\$ variable, you can check the Disable output redirects option to force all output to the main test print device (a watermark is printed on pages that would have gone to the alternate device).

Finally, if any other uf101c command line options are required for testing purposes, you can enter them in the Other Options field. Note that to test Zebra output to a file or printer, add -p zeban (*n* is 6, 8, or 12 dots per mm) to the command line options to override the default pcl or postscript value.

Local Viewing is attempted if Output is not specified (or Browser is selected in the drop down). In this case, PCL output is converted to PDF using a server-configured GhostPCL, and Postscript output is converted to PDF using a server-configured Ghostscript.



8.3.7 Preview

The preview button runs the currently edited rule file with the current sample to produce a preview, in either PDF or HTML5 format. The format is selected from the [Options](#) menu.

When in PDF mode, the preview is shown as an inline PDF document, using whatever internal PDF engine the browser is configured for. This is often Adobe Acrobat Reader, but some browsers default to their own internal engine, and there are third party tools available that can replace Adobe Acrobat as well. Whatever engine is used, the PDF rendering is under the control of the plugin and whatever features it offers.

When in HTML5 mode, UnForm generates HTML5 output, along with a designer-specific toolbar and some additional features. In this mode, the rule set and elements on the preview are synchronized. As you hover over different elements, they are highlighted, and if clicked, the current position in the rule set editor is updated. Likewise, moving the editor cursor to a new line will cause the preview to highlight the associated visual element. In the example below, the text command showing "Order Number" had been selected, and that value was highlighted in the preview.

Other toolbar options include a zoom feature, including fit width and height buttons, buttons to move an object up or down in the visual layering, and a button to reset the selections.

PCL and PostScript Previews

The checkboxes for these formats are available on the Options menu. Using these options will cause UnForm to use server-side tools (GhostPCL or GhostScript) to convert the raw pcl or ps output to PDF format for displaying in the preview pane. These programs must be configured at the UnForm server. If not configured, the preview option is disabled.

Zebra Previews


If you add "-p zebran [-paper *widthxheight*]" to the command line options defined in the File menu, AND have enabled use of the Labelary.com web service in the uf101d.ini file ([drivers] section, labelary=1), then the server will convert the ZPL output to PDF using the labelary.com web service, and present that in the preview window. If a rule set includes a paper command, it will override the optional -paper command line

option. If neither option is provided, the default is 'paper 4x6' (width and height are supplied in inches). The *n* value is for dots-per-millimeter (dpmm), as described in the -p command line option.

Note you can also choose the ZPL Preview option from the Options menu. This produces PDF like describe above, based on a 12dpmm setting and the rule set's paper setting.

Preview Grid Sample Watch

Width ▾



Company Name
 Company Address
 Company City, St Zipcode
 Company Phone
 Web: www.myweb.com
 Email: sales@myweb.com

INVOICE

05/30/97	0005138	1
Date	Invoice #	Qty

Sold To	000100 Everest Industries 123 Main St. Suite 111 San Bernardino, CA 93121	Ship To	000001 Everest Industries 403 Old Towne Road Rockville, MD 48833
----------------	---	----------------	---

Order Number	Order Date	Cust. Number	Stk. Pos.	Purchase Order No.	Ship Via	Ship Date	Terms
0001053	01/22/96	000100	JDP	EM-456	UPS GCD	01/22/96	1
Judy D. Peterson							Net (Due On Rec)

Qty Ord	Qty Ship	Qty Bkord	Item# Description	Unit Price	Extended Price
2	1	1	B100 24 Speed Trail Racer	EA 750.0000	750.00
2	2	0	1402 Attachable Tire Repair Kit	EA 12.0000	24.00
Returned Items Subject To 15% Restocking Charge. Return Authorization Required. Please Phone Our Customer Service Dept. For Details.					

8.3.8 Grid Printing

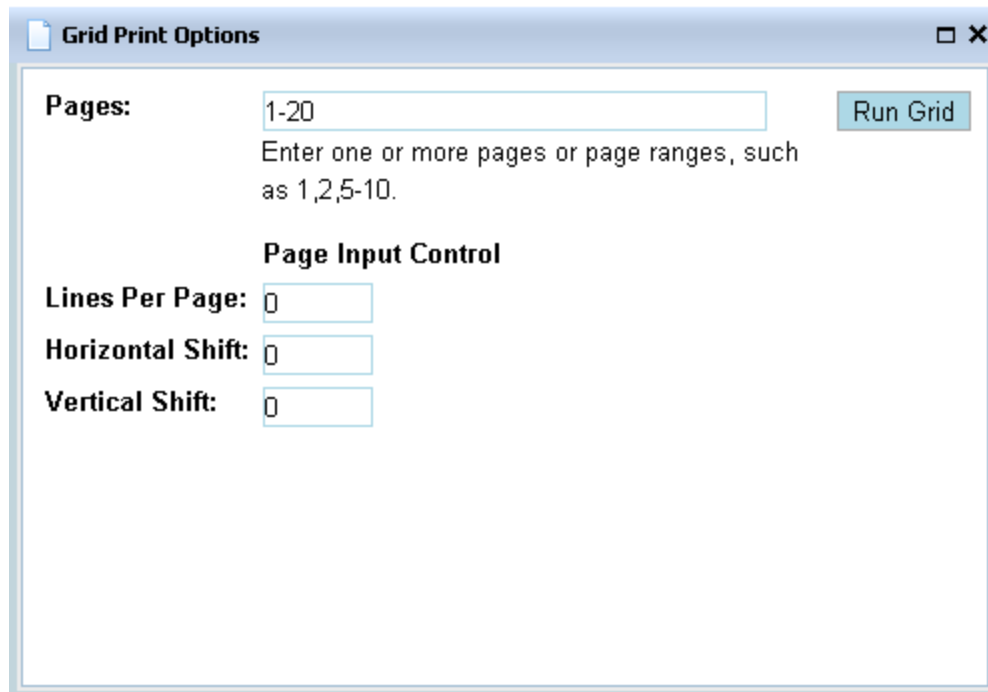
A grid preview uses the current sample as input, and prints a grid on top of it. No additional formatting takes place, so the grid positions accurately reflect the actual input print stream.

Choose which pages to print, based on the input stream, by entering page ranges or lists.

Choose the number of lines per page in cases where there are no page breaks in the input (a formfeed, for text data). Some applications, or some application print jobs, do not contain formfeeds and instead print an even number of lines per page. In those cases, UnForm needs to know how many lines per page to read.

Finally, if the rule set uses shift or vshift, which moves page text as it is read from the print stream, you can enter values in those two fields to offset the text.

Press Run Grid and the grid output will be displayed in the Grid panel, in either HTML5 or PDF format.



The image shows a dialog box titled "Grid Print Options". It contains a "Pages:" label followed by a text input field containing "1-20". To the right of this field is a "Run Grid" button. Below the input field is a hint text: "Enter one or more pages or page ranges, such as 1,2,5-10." Below this is a section titled "Page Input Control" which contains three labels and input fields: "Lines Per Page:" with a field containing "0", "Horizontal Shift:" with a field containing "0", and "Vertical Shift:" with a field containing "0".

8.3.9 Preview Options

When running a preview, the design tool saves a copy of the rule file and sample, then runs the job. The result of the job is presented in the Preview panel, in either PDF or HTML5 format.

Options for the preview include:

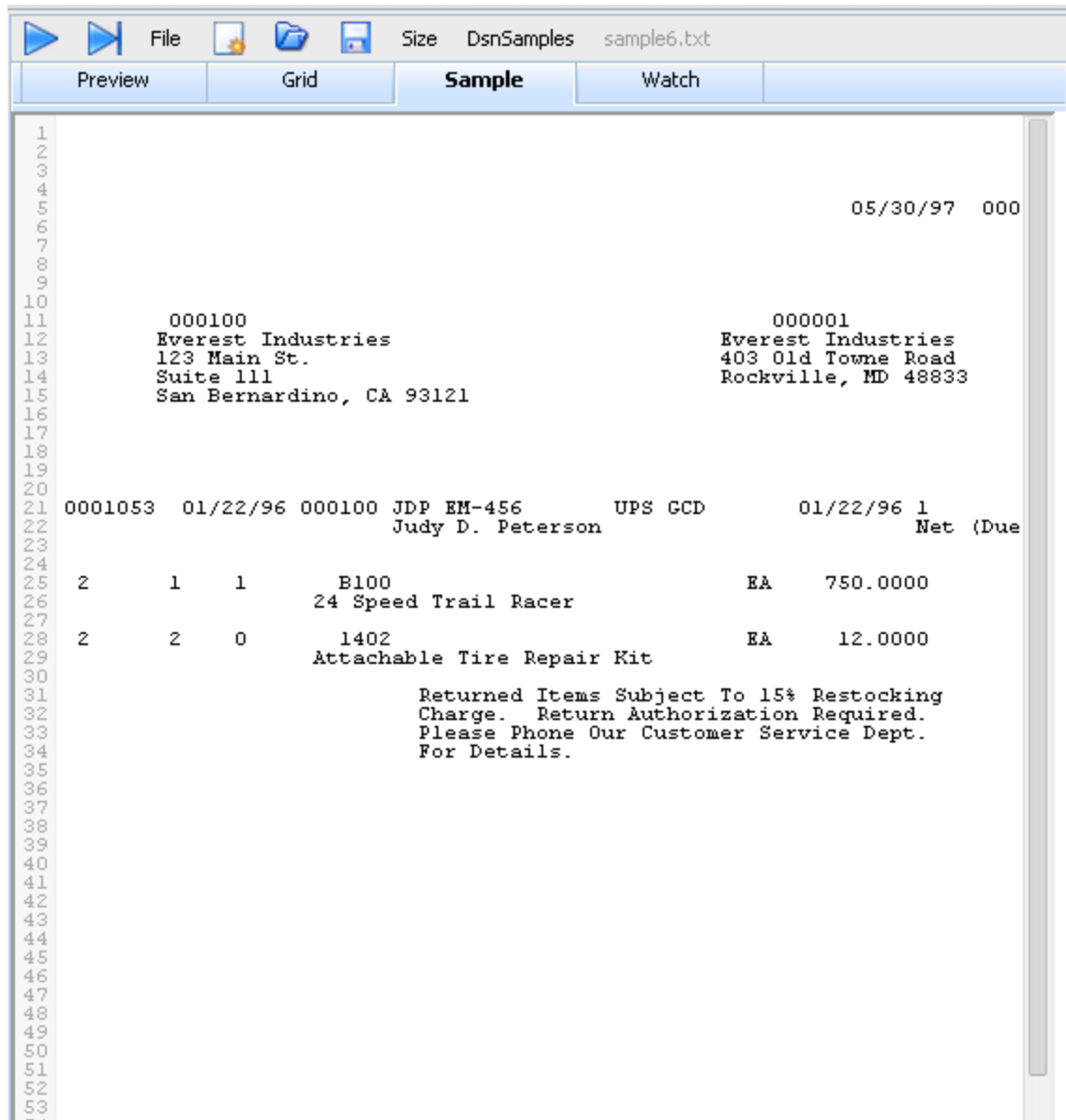
- Hide Preview, used when a Acrobat plugin obscures the Options menu itself
- Show Last Error, displays the last error message generated by a preview
- Force Rule Set, forces the current rule set to be invoked, regardless of detection logic
- Add Grid, adds a grid to the output (differs from a Grid preview, which presents just raw input text, not formatted output)
- Add Text Base, adds original text data to the output, which is especially helpful with [AFO](#) jobs
- PDF Preview, to generate previews in PDF format rather than HTML5

8.3.10 Samples

A sample provides input for a preview to be generated from, as if it had been printed by an application. This can be a plain text file produced by an application, or a PostScript or PDF file if the job is an [AFO](#) job. Note that samples can be easily captured by UnForm by adding the -debug command line option, which causes the server to leave files after the job is complete. The *.in files found in the temp directory are input files to UnForm jobs, exactly as UnForm receives them.

The Sample menu bar adds some new options to the standard Preview menu:

- File menu, with new, open, and save options (opened files are added to the dsn_samples commands in the rule set)
- Buttons for new, open and save
- Size dropdown for selecting a font size for text samples
- DsnSamples dropdown to select one of the dsn_sample files found in the rule set



8.3.11 Watch

The Watch tab provides a table of values and expressions from the last preview run. Enter a list of variables or expressions in the Vars, Expressions field. The next time the preview is run, the table will be populated with the actual values from that preview run, after each code block for the job and on each page and copy.

Preview	Grid	Sample	Watch	
Vars, Expressions: <input type="text" value="pagenum, get(8,12,20)"/>				
Variable	Block	Page	Copy	Value
pagenum	prejob	0	0	0
get(8,12,20)	prejob	0	0	Everest Industries
pagenum	prepage	1	0	1
get(8,12,20)	prepage	1	0	Everest Industries
pagenum	precopy	1	1	1
get(8,12,20)	precopy	1	1	Everest Industries
pagenum	postcopy	1	1	1
get(8,12,20)	postcopy	1	1	Everest Industries
pagenum	postpage	1	0	1
get(8,12,20)	postpage	1	0	Everest Industries
pagenum	postjob	0	0	1
get(8,12,20)	postjob	0	0	Everest Industries

9 IMAGE MANAGER

The UnForm 10 Image Manager is a new component designed to replace the former Windows-only Image Manager component from previous releases. The new Image Manager is browser-based for its user interface. It runs all jobs at the UnForm server, providing significant performance benefits and the ability to run jobs automatically without user intervention. There are other powerful enhancements, such as:

- Better multi-page image support, with multi-page zones and programmatic page manipulation
- Custom fields and parameters built into jobs
- Extensive and customizable filtering and validation functionality
- Lookups from both ODBC and other external services, such as REST API interfaces
- Integrated line item detail via grid management in both code and user input, with programmatic grid formatting and manipulation
- Common scripting language with other UnForm components
- Full access to all library management and other built-in UnForm objects

The former Image Manager, now called the UnForm Legacy Image Manager, is still available with UnForm 10, though the publisher plans no further enhancements to it and encourages all new projects to be designed in the new Image Manager.

9.1 Overview

The UnForm 10 Image Manager is a combination of server-side document handling capabilities, along with a browser interface to enable users to manage documents and the jobs that are run with those documents. Documents enter the Image Manager via configured inbound sources. Once a document has been identified and optional data fields specified, it can be uploaded to an archive library for later access and usage. In addition, with the scripting capability provided by UnForm, data from these documents can be used for direct integration with other applications, such as ERP products.

Users are assigned rights to run the Image Manager browser interface, and to design jobs. When users log into the Image Manager, they can work with a set of documents previously assigned to them, or they can self-assign documents from an available pool. Administrative users can additionally perform configuration functions, and perform manual assignment functions as necessary.

Inbound Sources are monitored for new documents. Jobs can be run automatically as the documents arrive, or documents can be simply queued for manual assignment to a user, who will perform actions related to identification and data entry. The following sources are supported:

- One or more email mailboxes can be monitored for document attachments, such as vendor invoices or customer purchase orders. Note that it is possible to create email aliases or mail groups that enable delivery of email to other configured sources as well as the email source itself.
- One or more system folders or directories can be monitored for files, such as those uploaded by a scanning device or application. Any method that can transfer files to a folder can be used, such as local drag and drop, FTP/SFTP, rsync, and more. It would even be possible to use an unform client with -i and -o options, though this does utilize a print job slot while performing the transfer.
- Image Manager users can upload documents from their personal system.

Inbound sources are maintained in Admin menu of the regular browser interface.

Browser Interface provides for user management of the inbound documents. Jobs can be run on selected documents, data validation performed, manual input entered. Image pages can be managed in different ways such as re-ordering scanned images, or splitting and joining pages. Advanced users can be granted the ability to create and edit job definitions and custom scripts. Administrative users can manage user and inbound source configuration, and also perform manual document assignment to other users.

The browser interface for Image Manager works with current versions of Chrome, Firefox, Edge, and Safari browsers. It does not support Internet Explorer.

Jobs are definitions that include custom parameters, OCR and barcode zones and detection, custom data fields, standard identification fields, and custom scripting. Jobs are run at the server, so they can execute automatically when documents arrive, or manually from the browser interface. The purpose of jobs is to capture and validate data from the documents so that they can be identified and properly indexed, and also so the data can be used for integration with other products. Examples include processing customer orders and vendor invoices for automated transaction creation in an ERP system.

9.2 Document Sources

Document sources are locations where UnForm can obtain inbound documents; that is, documents that are not generated via printing or direct uploading. Each source has an associated inbound library where documents and their properties are maintained. These libraries are not normally accessible in the regular browser interface, but an administrator can view them under the ~inbound library category.

Sources are maintained with [Inbound Sources](#) on the Admin menu of the archive browser interface

Each document in a source library is either assigned to a user or available for assignment by any user. Once documents pass all validation tests, which can be particularly extensive if a job is applied to a document, the document can be transferred to a standard UnForm archive library.

There are three types of document sources:

- Directories (Folders) on the UnForm server system. A directory can be anywhere that the UnForm server can read and remove files from. As documents arrive in that directory, UnForm picks them up and moves them to an inbound library, performs initial parsing for image and text extraction, and then optionally applies jobs to those documents. A common use for directory sources is for scan-to-disk operations.

A wildcard filter can be applied to limit the documents that are picked up by that source definition.

A zip file can be placed in the directory and files will be extracted from it for processing, rather than attempting to process the zip file itself as a document.

File system sources support sub-directories and process files from those as well.

- Dedicated email mailboxes, monitored by IMAP, with attachments extracted for processing, and the email and key attributes are stored as well. It is important to note that this must be a dedicated mailbox, not one shared with any user or other program, as UnForm will remove all email for either processing or discarding.

A white list enables spam control, where only mail from specific addresses or domains will be accepted. In addition, only certain attachment types can be specified.

An email source can receive mail with alternate To addresses, either via aliasing or mail list configuration. When an email arrives at the source inbox, the actual To address is tested to see if it contains `+sourceid` or `.sourceid` for an alternate inbound source. If so, the documents are delivered to that source rather than the defined email source library. For example, email addressed to `inbound+expvnd@somewhere.com` will deliver attachments to the source "expvnd" if it exists.

- Image Manager users can upload documents or zip files into a source library, and those documents will be assigned to them automatically.

Note that unlike standard document libraries, the system will purge library recovery data for dates prior to any existing documents in the inbound source libraries. This is due to the intentional transient nature of inbound documents.

Document Types

When files arrive in a source, they are parsed in various ways automatically.

- Zip files are extracted, and the content files processed individually.
- Image files are scanned for barcodes, and if OCR processing is not disabled, an attempt will be made to use a local OCR engine (Tesseract) to create a PDF file with text, which is then processed as if the PDF file were uploaded directly. In addition, the image pages are converted to single page PNG files for viewing.
- PDF files are scanned for text. If no text is found and OCR processing is not disabled, the PDF file is converted to an image and processed using the local OCR engine (Tesseract). If text is found in the PDF file, that text is used for all OCR zones in Image Manager jobs. In addition, the PDF pages are converted to images for barcode scanning and viewing.
- Other file types, other than text attachments, are simply loaded into the source library for browser viewing.

- If an email contains no attachments (other than text attachments), its body text is used as a document. This is particularly useful for sources that deliver to printing rather than the Image Manager, so that legacy email sources can be used for printing. Such emails also can produce an Image Manager job, though it will be plain text and cannot be managed in jobs with ocr or barcode zones.

Note that for document processing where the quality of OCR results is important, a commercial OCR tool can generally produce better text quality than free software. Any tool that can produce a PDF file with text can be used. Also, some scanner devices can perform OCR processing and scan to PDF files.

The very best "OCR" results are obtained without using OCR at all, but instead by processing PDF files directly from an application that produces PDF files natively. When automating the processing of inbound documents, it is worthwhile asking suppliers for such PDF files.

An AI-based [OCR service](#) is available that typically provides significantly better OCR accuracy than a local Tesseract engine.

9.3 Image Manager Users

Image Manager users are those users who have been granted Image Manager permission by an administrator in the [Internal Users](#) maintenance function. In addition, such users may also be granted permission to maintain job definitions. Note that Administrator users are not automatically granted Image Manager access permission, but those that are can assign inbound documents manually to any user, and have implicit access to all inbound source libraries.

9.4 Browser Interface

After logging in, the main browser window is shown. There are three panels and primary toolbar menu. The left panel shows a list of inbound sources. The center panel shows a list of documents, either those currently assigned to the user (also called pending documents), or those available for self-assignment. The columns in this panel are sortable. The right panel shows the last-selected document, with four tabs: custom fields, identification fields, images, and meta data. The images tab has its own toolbar for page and zoom control.

designer user can also create jobs that set specific values automatically.

- **Delete selected** removes the selected images from the inbound library. They are no longer available after this operation.
- **Merge selected** combines the images and properties of the selected documents. Images are appended, and categories, keywords, and links are merged.
- **Merge matching** combines the images and properties of selected documents with matching identification, that is matching library, doctype, docid, and subid values. Only documents with all four values can be merged. This process can result in several multi-page documents, depending on the identification values of the selected documents.
- **Reprocess/OCR** selected resets any OCR and barcode data and re-processes the selected images, as if they had just been received. This feature is useful if you have updated software such as PDF reading tools.
- **Fit columns** to data resizes the columns in the center panel to show all column data.
- **Edit Job Definitions** displays the job configuration window, where a job designer can create and edit job definitions.
- **Job History** displays the job history window, where logs of previously run jobs can be viewed.
- **Script Libraries** displays the custom code editor, where a job designer can create and edit custom filter, validation, lookup, and set values routines. These routines contain Basic code similar to code blocks in UnForm print jobs, to enable programmatic control over field assignments, data validation, external integrations, and more.
- **Assign Documents**, available to administrators, displays all assigned and unassigned documents in every inbound source library, and enables assignment (or unassignment) to any Image Manager user.
- **Force reprocess with OCR** re-parses selected documents using OCR, even if the original files were PDF with text data. Some text data cannot be successfully read from PDF files and better results may be obtained by forcing an OCR process. This option is only available if Tesseract is configured in UnForm.
- Logout exits the Image Manager and returns to the login screen.

9.4.1 Document Selection

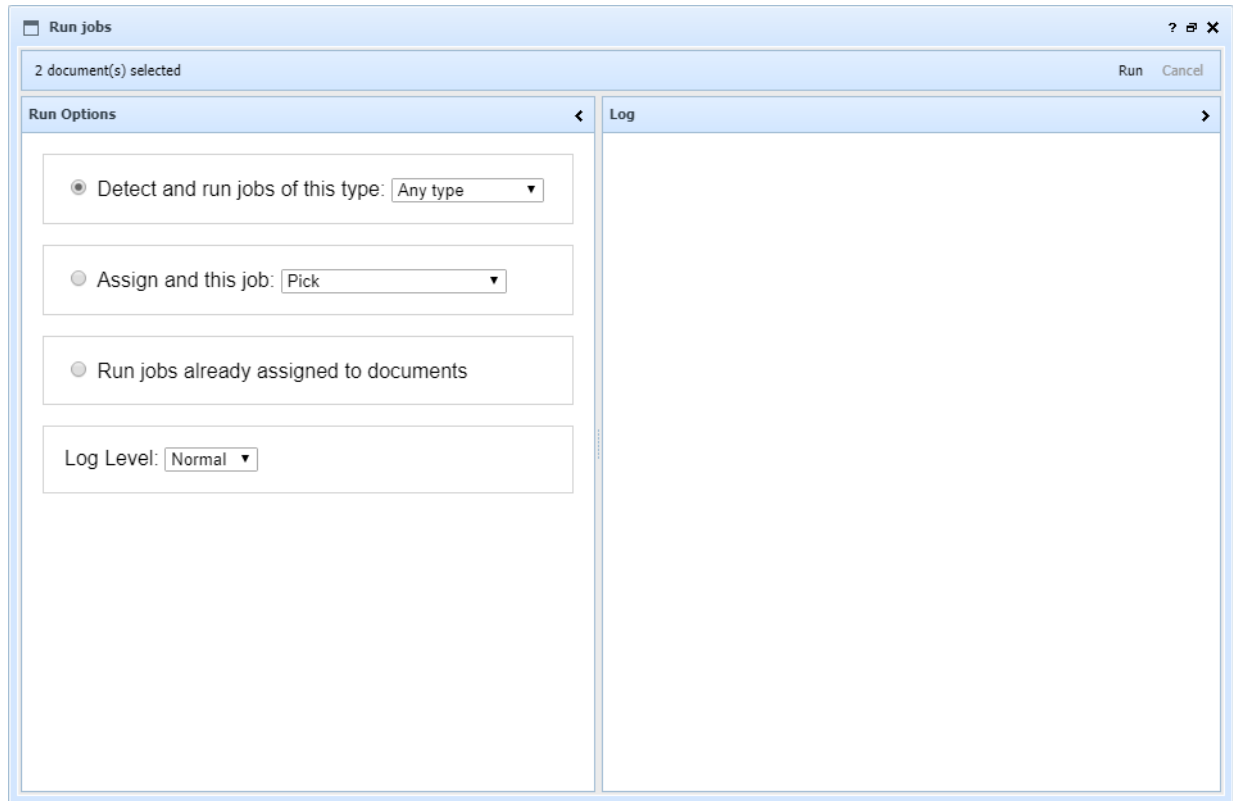
Documents in the pending or available lists can be selected by mouse or keyboard actions, using the shift key or control key as you click to select ranges or multiple documents. Clicking a row without the shift or control key selects just the clicked row. Checking the boxes in the left-most column retains the previously selected documents, and checking the column header checkbox selects or unselects all documents.

You can also use the toolbar selection buttons to select documents by category.

Once one or more documents have been selected, several toolbar buttons are available that apply to the documents selected. For example, if you click the Run jobs button, jobs will be run against each selected document in turn.

9.4.2 Running Jobs

The Run Jobs window enables job execution in one of three modes. Jobs run on the server, applying to each selected document in turn, and a log of the actions is displayed as the job runs. Select one of the Run Options, and click the Run button on the toolbar to start job execution.



Run Options

- **Detect and run jobs of *this type*** will perform zone extraction and detection logic on each document. Jobs that contain detection logic, and that are of the type selected, will be tested until the first one successfully passes detection. That job will be applied. If no jobs successfully pass detection, then processing continues with the next document.
- **Assign and run *this job*** will assign the selected job to each document, and execute the job.
- **Run jobs already assigned** will run jobs that have been previously assigned to the documents. Documents that have no job assigned are skipped. This option is useful to re-run jobs on documents.

The log level can be set to normal, detailed, or debug. The debug level is designed to assist job designers during development and testing of jobs.

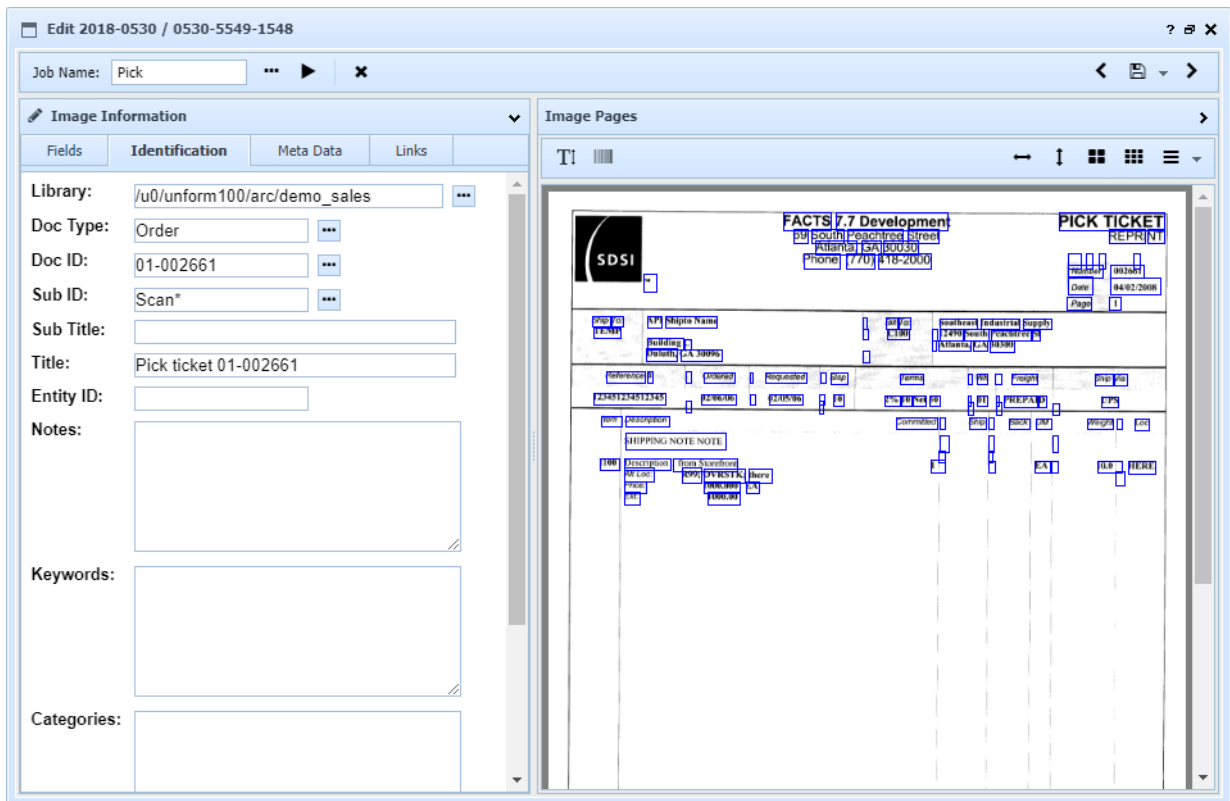
9.4.3 Manually Set Values

This window provides an easy way to apply specific identification values to all selected documents at once. Enter 'Set To Value' data, or check the Reset option to clear a given value, and press the Update *n* Documents button to apply the changes. A job designer user can also create jobs that perform these steps, and those jobs are available from the drop down menu on the left button on the toolbar.

Field	Reset	Set To Value
Library:	<input type="checkbox"/>	/u0/uniform100/arc/demo_sales
Doc Type:	<input type="checkbox"/>	Quotation
Doc ID:	<input type="checkbox"/>	
Sub ID:	<input type="checkbox"/>	Acceptance
Sub Title:	<input type="checkbox"/>	
Title:	<input type="checkbox"/>	
Entity ID:	<input type="checkbox"/>	
Notes:	<input type="checkbox"/>	
Keywords:	<input type="checkbox"/>	
Categories:	<input type="checkbox"/>	

9.4.4 Editing and Validation

Document field and identification values, and the associated images, are edited using the Edit Document window. This window provides two panels, the left for data and the right for image pages. The data panel has several tabs, for custom fields, identification fields, meta data, links to other documents or web pages, and if a job is run for this job from the toolbar, a job log. Any field or identification value that has not passed validation tests is highlighted in red. Move the mouse over the field to see an associated error message.



Toolbar Options

You can select a job to process against this image, using the ellipsis button to lookup jobs if desired, and the play button to run the job.

You can delete the image with the X button. This removes the image permanently from the inbound library.

If you've edited data, you can use the Save buttons to save and optionally move through the pending list, previous, current, and next, or choose the 'Save and Close' drop down option to save the document and close the edit window. The 'Save and Calc' option is similar to Save, but fields and identification values are recalculated if the job assigns them to expressions.

Image Pages

The image panel has a toolbar that offers several options. On the left are buttons for quick OCR and barcode zone drawing. Multipage images also offer a pagination button, so you can re-order the pages of a document via drag and drop actions. There are also several zoom options, and options to split multipage images in two or to single pages, and options to rotate and re-process the pages of the image. Double click an image to zoom in or out. Click an OCR word or draw a zone to enable copy/paste or drag/drop operations on text.

A caveat: rotated images cannot retain their OCR data, if any, so if you rotate images that originally had OCR data, it will be re-generated. If that original text data came from a PDF file with textual data, the resulting OCR will likely not be as accurate.

Grid Entry

Grid type data fields display a grid edit button, and also a special panel. When the grid data is displayed, the user can manually edit cells by double-clicking. There are also many row and column features available for data manipulation, as well as access to grid-oriented filters. In addition, if lookups have been

defined, they are available once a row has been selected. A lookup selection will replace the current cell with the lookup's defined value, and then update any cells whose column names match those of the lookup column names.

9.4.5 Transferring to Libraries

When a document is fully identified, which means if there is a job applied it has passed all validation tests, and otherwise includes at least a library and document type, the document can be transferred to the target library. Select the documents to transfer and press the Transfer Ready Documents button. After you confirm the action, the documents selected are transferred, and a log is presented. Note that only documents whose Ready column has a checkbox are transferred. The Ready checkbox indicates the document has passed all validation tests for a job, or has minimal identification values.

Before each document is transferred, a job's preupload code is run as a last verification step. After the document is transferred, a job's postupload code is executed, which can allow for related actions to be performed.

9.5 Job Definitions

Job definitions enable extensive automation of tasks associated with document identification and data capture. A job definition includes:

- [Parameters](#), which are easily maintained text value constants for a job
- [Zones](#), which capture OCR and barcode data from document images
- [Detection](#) criteria for jobs that should be auto-assigned based on zone content
- [Custom fields](#), which can be used to capture data from zones, the user, external lookups, or custom code
- [Identification fields](#), which are the data elements associated with the standard archive libraries that inbound documents get transferred to
- [Custom scripting](#), which provides complete flexibility in management of both image data and external data

Job definitions can be categorized to help manage large numbers of jobs, particularly with regard to automatic detection. Other job information can be specified in the [Name/Info](#) tab.

9.5.1 Overview

Jobs are created and edited using the Job Configuration window. There are three panels in this window. The left panel shows existing jobs, organized by category. Clicking a job will open it in the center panel. The center panel contains the job definition, organized in tabs. The right panel shows an example document, which is the last document selected in the pending list of the main Image Manager window. The example images show OCR and barcode data found in the sample, and provide a drawing surface when zones are being defined.

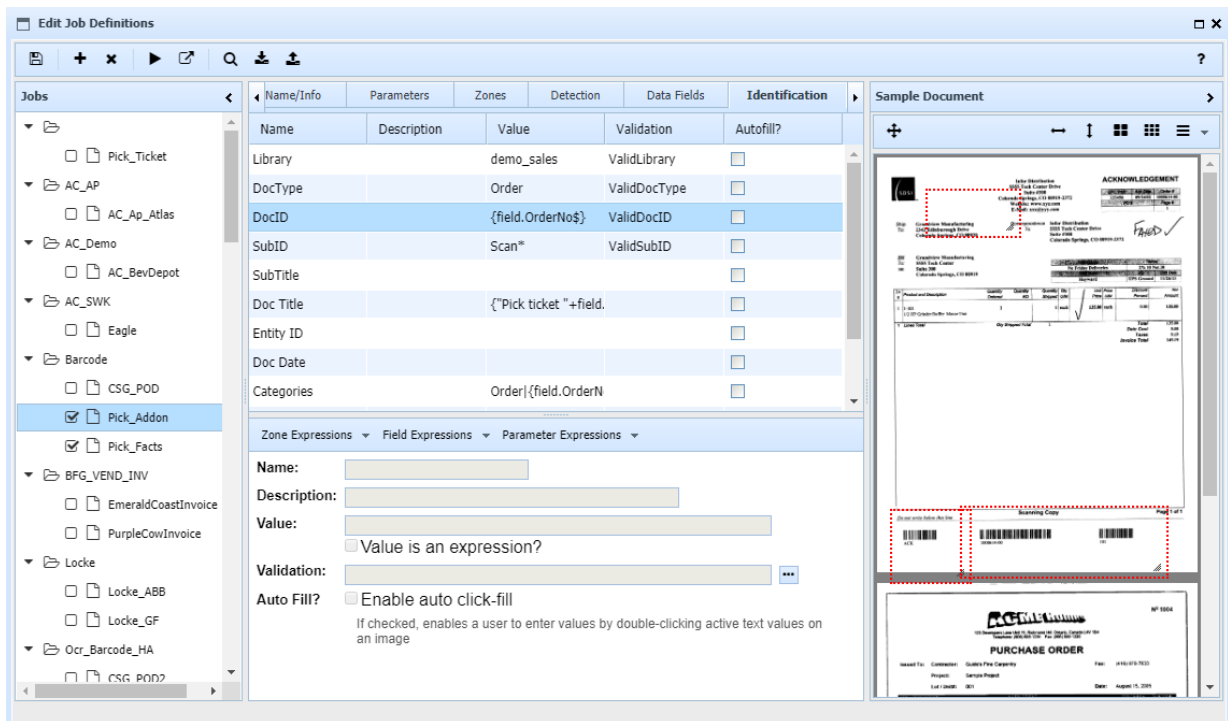
The toolbar provides several buttons, to save the current definition, add a new definition, or delete the current definition. If a sample document is loaded additional buttons are enabled to test the current job

and view document results. A 'save as' function is performed by entering a new job name and pressing the Save button. As long as there is no existing job with that name, a new definition is created.

Job Export and Import

The search, download, and upload buttons perform functions across job definitions, and provide the means to identify and deploy sets of jobs to other UnForm installations.

- The search button will prompt for text or a regular expression (~pattern entry) and check any job that contains that text. This is useful to locate jobs with specific filters, script variables, or other criteria.
- The download button exports checked jobs, as well as script library routines those jobs require, to a special file with a .ufmjobs extension. This file can be transported to another installation.
- The import button will prompt for a file, which is then imported, adding or updating jobs and script libraries. The server log is written for this import activity. In addition, a dated backup zip file is created in the "im/backups" directory, containing the jobs master file and script library files from before the import.



9.5.2 Name/Info

The Name/Info tab defines basic information about the job, and also allows importing and exporting of job definitions to enable copying of jobs between systems.

- **Job Name** is a short identifying name of the job. If the name of an existing job is modified, a Save operation creates a copy of the job.
- **Description** is a text description of the job for reference purposes
- **Type** is a category for this job, used to organize jobs in the left panel tree, and also to allow a restricted list of auto-detect jobs when running jobs against one or more images.
- **Single Page?** if checked will force multipage image imports, such as PDF and TIFF files, to be split into single page images when the job is run.
- **Allow Overwrite?** if checked allows replacement of document subid values when a document is transferred to an archive library. If not checked, a sequencer is always added to prevent overwriting.
- **New Libraries?** if checked allows a job to create new libraries when transferring documents to archive libraries.
- **Inactive?** if checked indicates this job will not run or be available for operator selection.

When viewing an existing job, an Export button is provided to generate a file that can be imported on another system running the UnForm Image Manager. When creating a new job, an Import button is provided to upload a previously created job definition export file.

9.5.3 Parameters

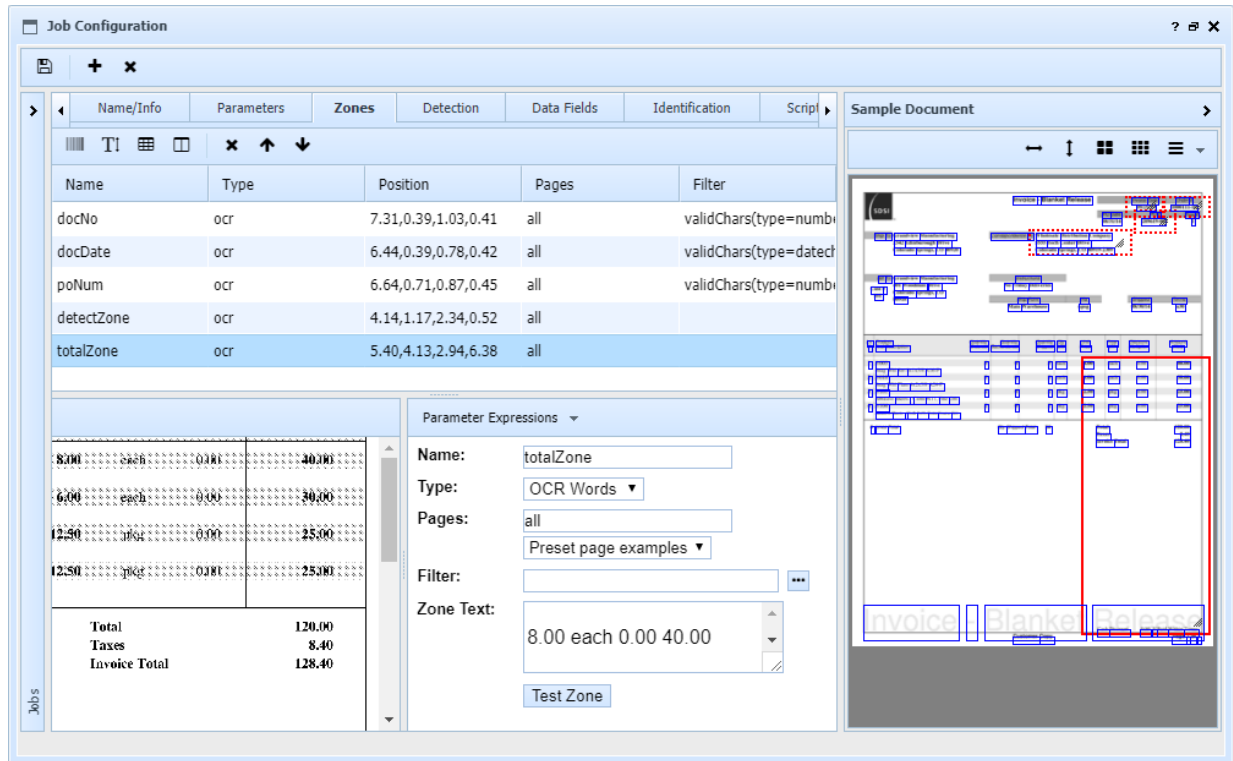
The Parameters tab provides an easy table to maintain job constant values, available in other portions of job design and in job scripts. Add, delete, and move parameters with the toolbar. Edit rows by clicking cells, or tabbing through them once in edit mode.

Parameters are automatically assigned values and inserted into script code routines, and also into filter, lookup, and validation objects exposed by those routines. The parameters take the form of string variables, named *param.name\$*.

Name	Description	Value
vendor		WDC001
GLaccount		790000
GLsubaccount		000-00-00
branch		SOFT
terms		30D
dtFormat		mm-dd-yy
stackedTotals		Yes
totalLabel		Invoice Total
merchTotalLabel		Total
taxLabel		Taxes
freightLabel		Freight
discountLabel		Order Discount Other Discount
otherLabel		

9.5.4 Zones

Zones are regions of one or more image pages, from which OCR or barcode data can be extracted. They are drawn using the mouse to position and size the zone's region on a sample document. Zones can span pages. Zone data can be filtered with a filter that has been defined in the filter script library.



Zone Definition Fields

- **Name** is a short alpha-numeric name for the zone. This name becomes a variable in job scripts, zone.name\$.
- **Type** is the type of zone: OCR words, barcode, OCR grid, or OCR column. A grid is a host for the columns that follow it. Note that for both barcode and OCR zones read from scanned documents, a resolution of at least 300 dpi is recommended. Non-scanned PDF files, such as those generated from an ERP application, will generally produce the best OCR results.

If an column zone name starts with an underscore, such as `_RangeCalc`, then it is not shown in the grid editing interface. It is considered a hidden column.

- **Pages** defines what pages this zone should extract data from. It can use specific page numbers, or named pages first, last, and all. Ranges can be specified with a hyphen, and expressions can be entered with curly braces. There is a drop down of common examples. Examples include "all", "1", "last", "1-{last-1}".
- **Filter** is a filter name, with optional parameters in parentheses. The ellipsis button can be used to lookup a filter. If the filter requires parameters, the lookup paste will supply the names so you can fill in appropriate values or job parameter expressions. Filters are used to modify the raw data captured from the zone, before it is used in later operations like custom or identification field assignments.

Filters can be chained by entering multiple filters, one per line. The result (output) of each filter is used as the source (input) of the next one.

When filters are applied to grid zones and their columns, column filters are applied first, then grid filters are applied, then the column values are set to the filtered results from the grid.

- **Zone Text** is a read-only field that shows the raw zone for the selected page. Press the Test Zone button to see the full zone value for all specified pages and after the filter has been applied.

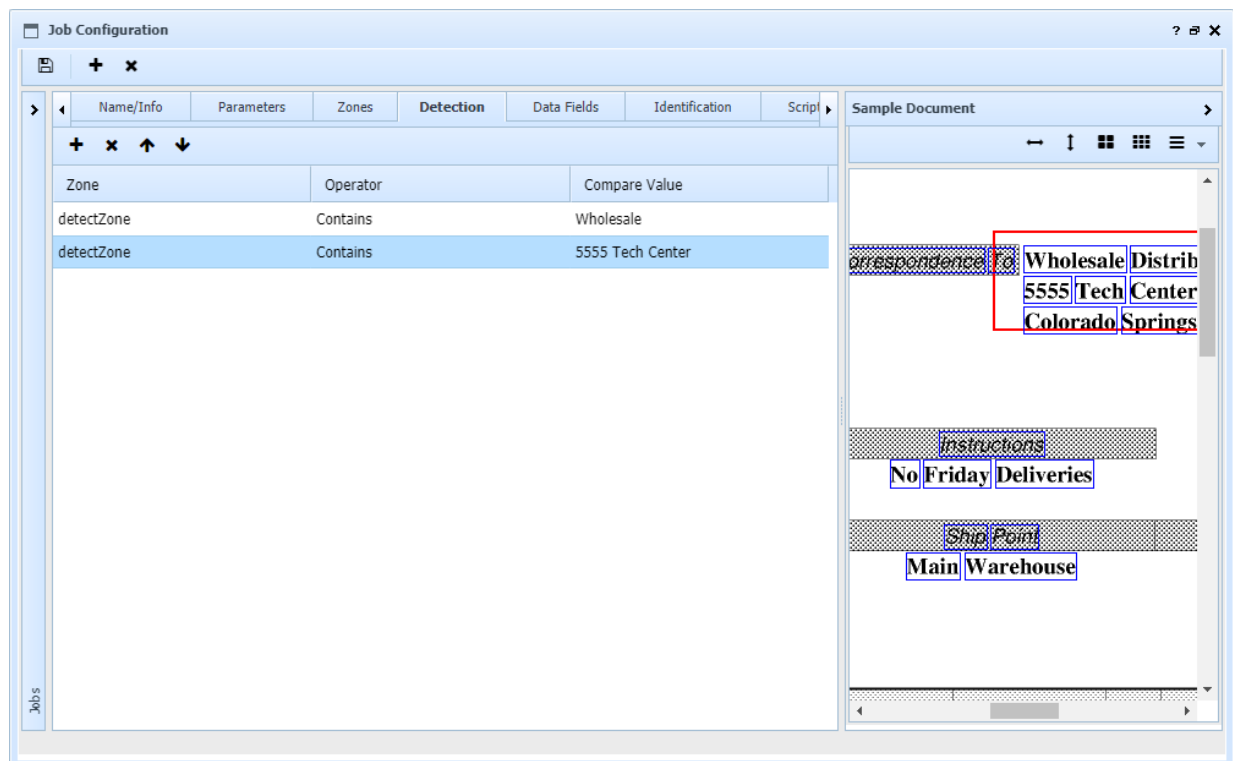
Note when drawing a zone, if you need precise positioning, you can zoom in on the sample image for greater drawing precision. You can also use the keyboard to adjust the zone in .01 inch increments as follows:

- **Ctrl-arrow** moves the zone up, down, left, or right
- **Shift-arrow** expands the zone in the direction of the arrow
- **Alt-arrow** reduces the zone from the edge opposite the arrow (i.e. left arrow reduces from the right edge)

9.5.5 Detection

Detection is the process a job goes through to determine if a given image should be processed by the job.

When jobs are run against a document or batch, there is a mode called 'Detect and run'. When running in this mode, the zones, parameters, and metadata are resolved, and then compared with known values to determine if the job should be applied to this document. In the example below, this job would be applied if an address region on the document contains both "Wholesale" and "5555 Tech Center", effectively limiting this job to one entity.



Detection Definitions

Use the toolbar to add, remove, or move detection lines.

For each detection line:

- Select from the dropdown list any zone, job parameter, or metadata value.
- Select the compare operator from the list provided. See the [compare object](#) for a list of valid operators.
- Enter the comparison value. In some cases this will be case sensitive, or may be a comma-separated list.

9.5.6 Data Fields

Custom data fields are used to capture data about the documents being processed by the job, in addition to the built-in identification fields used by all documents in an UnForm archive library. Data fields can be used to capture data useful in various ERP automation settings, such as accounts payable invoice processing. Data fields are presented to the user while editing a document, enabling user input as well as automated data calculations. A key feature of data fields is filters and validations, both of which are programmable functions that can be used to automate data formatting or manipulation, and to prevent invalid data from being stored and passed into other portions of the document management system, or to an ERP system.

The screenshot shows the 'Job Configuration' window with the 'Data Fields' tab selected. The table below lists the configured data fields:

Name	Description	Type	Value	Filter	Validation	Autofill?
poNum		text	{zone.poNum\$}			<input type="checkbox"/>
GLaccount		text	{param.GLaccount\$}			<input type="checkbox"/>
GLsubaccount		text	{param.GLsubaccount\$}			<input type="checkbox"/>
merchTotal		number	{zone.totalZone\$}	verticalTotal(label={pai		<input type="checkbox"/>
taxTotal		number	{zone.totalZone\$}	verticalTotal(label={pai		<input type="checkbox"/>
freightTotal		number	{zone.totalZone\$}	verticalTotal(label={pai		<input type="checkbox"/>
discountTotal		number	{zone.totalZone\$}	verticalTotal_combo(lal		<input type="checkbox"/>

Below the table, the configuration panel for the 'merchTotal' field is shown:

- Name:** merchTotal
- Description:**
- Type:** Number
- Options:**
- Value:** zone.totalZone\$
☒ Value is an expression?
- Filter:** verticalTotal(label={param.merchTotalLabel\$})
- Validation:**

Data Field Definitions

- **Name** is a short alpha-numeric name for the field. This name becomes a variable in job scripts, field.name\$.
- **Type** is the type of field: text, longtext, note, number, date, checkbox, radio button, list, lookup, label, grid, or hidden.
- **Options** apply to certain field types. For example, a radio button or list field requires a list of possible values. A lookup requires a lookup name. Screen help appears when required.

- **Value** can be a fixed text value, or an expression. This is what gets assigned to the field by default. If it is an expression, be sure to check the Value is an expression box. If it is empty, the field retains its value if previously set by another expression or has been edited before the job runs.

For a date type field, this must be empty or in YYYY-MM-DD format to be correctly interpreted by browsers.

- **Filter** is a filter name, with optional parameters in parentheses. The ellipsis button can be used to lookup a filter. If the filter requires parameters, the lookup paste will supply the names so you can fill in appropriate values or job parameter expressions. Filters are used to modify the data, if necessary, to correctly format it.

Filters can be chained by entering multiple filters, one per line. The result (output) of each filter is used as the source (input) of the next one.

- **Validation** is a validation name, with optional parameters in parentheses. The ellipsis button can be used to lookup a validation. If the validation requires parameters, the lookup paste will supply the names so you can fill in appropriate values or job parameter expressions. Validations are run whenever the document data is saved, and can report error message text back to the user. If any field fails validation, the document cannot be transferred to an archive library.
- **Autofill** allows this field to be a tab-stop during editing, enabling quick click-tab form fill operation to work.

9.5.7 Identification

Identification fields are built-in fields for UnForm document archive libraries. More detail about their use is found in the [Overview](#) chapter. Each document in the Image manager represents one document and image in a target archive library. When documents are transferred to libraries, these identification fields are used to index those documents and to provide properties to those documents. Each row in the Identification table is for a specific field. The first four, library, doctype, docid, and subid, identify the document and the image upon transfer. Other fields are used for additional information or indexing. Note that the Doc Date field's value should resolve to a yyyy-mm-dd format.

Name/Info	Parameters	Zones	Detection	Data Fields	Identification	Scripting	Autofill?
Library				/u0/unform100/arc/demo_accoun			<input type="checkbox"/>
DocType				Vendor Invoice			<input type="checkbox"/>
DocID				{field.vendorID\$+"-"+field.invoNum\$}			<input type="checkbox"/>
SubID				Scan*			<input type="checkbox"/>
SubTitle							<input type="checkbox"/>
Doc Title							<input type="checkbox"/>
Entity ID							<input type="checkbox"/>
Categories				poNumber{{field.poNum\$}}			<input type="checkbox"/>
Knowledge							<input type="checkbox"/>

Zone Expressions ▾ Field Expressions ▾ Parameter Expressions ▾

Name: DocID

Description:

Value: field.vendorID\$+"-"+field.invoNum\$
☒ Value is an expression?

Validation: ...

Auto Fill? ☐ Enable auto click-fill
 If checked, enables a user to enter values by double-clicking active text values on an image

- **Name** is supplied automatically based on the row selected. The names become variables for scripts, ident.name\$.
- **Value** is a fixed text value or an expression. If it is an expression, be sure to check the Value is an expression box. This becomes the default value for the field during job execution. The user can edit this value as needed in the Image Manager. If it is empty, any previous value is retained when the job is run.
- **Validation** is a validation name, with optional parameters in parentheses. The ellipsis button can be used to lookup a validation. If the validation requires parameters, the lookup paste will supply the names so you can fill in appropriate values or job parameter expressions. Validations are run whenever the document data is saved, and can report error message text back to the user. If any field fails validation, the document cannot be transferred to an archive library.
- **Autofill** allows this field to be a tab-stop during editing, enabling quick click-tab form fill operation to work.

9.5.8 Custom Scripting

Custom scripting is a feature that provides a great deal of flexible control over how a job operates. Much of the script code is generated by the server, based on the job's configuration. For example, in the screenshot below, you see all the param.name\$ variables are created automatically. Each script section, however, offers a custom code section where you can enter script code to perform further processing that the job design doesn't provide internally.

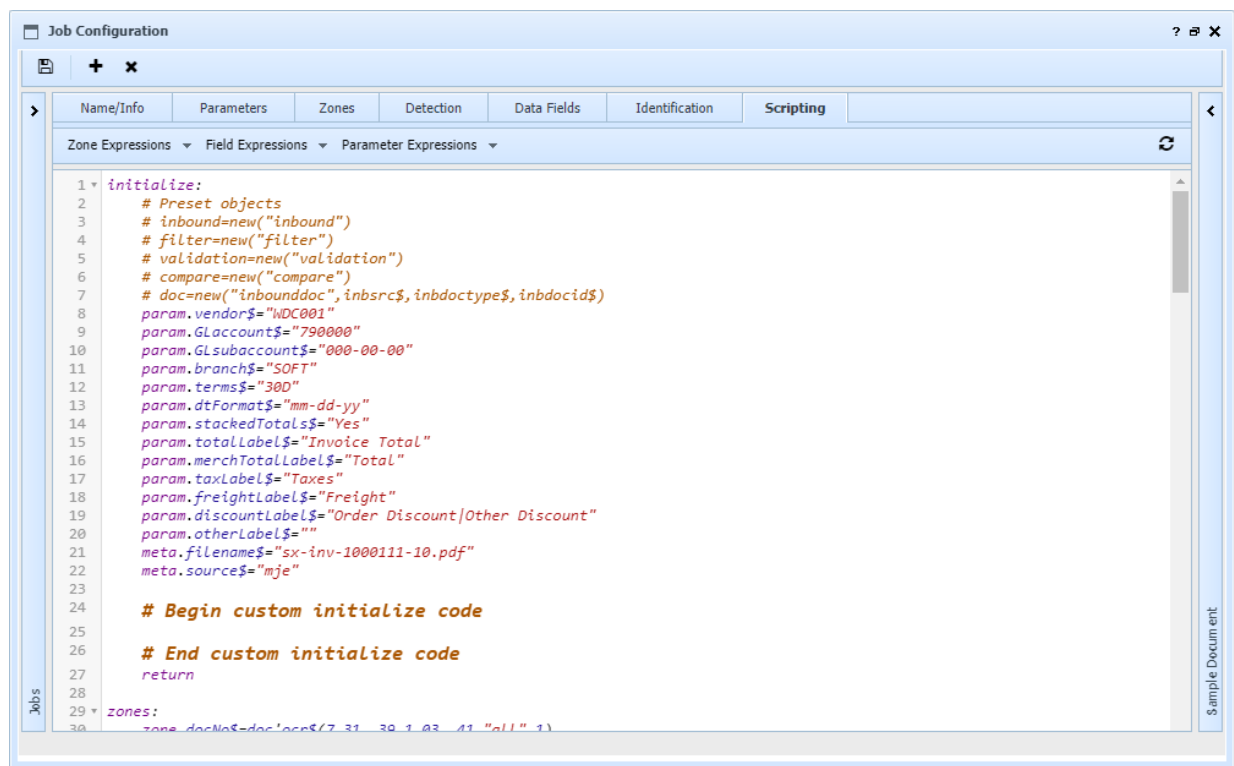
In the custom code section, you can enter any valid UnForm scripting code, which is based on the PxPlus Basic language. The code is identical to that used to [program code blocks](#), except that block if statements are not restricted from using curly braces (see Block If, below).

Scripting makes heavy use of objects, as several are exposed automatically while the script runs, used by the auto-generated code and available for custom code. The following objects are available in all sections of script code:

- [inbound](#) to manage the inbound libraries and contents
- `doc` to manage a single document, which is an instance of the [inbounddoc](#) object
- [filter](#) to run filter code (filters is an alias for filter)
- [lookups](#) to run lookup code
- [validation](#) to run validation code (validations is an alias for validations)
- [compare](#) for detection code to compare values

Also used if the job utilizes grid zones or fields:

- [grid](#) to manage grid zone data



There are several sections, each executed at specific times during processing, and each offering a custom section to add code to:

- **initialize** is run as each document is loaded into memory. This code should be used to create additional objects beyond those automatically provided as the Preset objects listed in the comment area.
- **zones** is run to capture zone data into variables, ahead of detection logic or other assignment logic.
- **detect** is run whenever auto-detection is specified for the job process. Its purpose is to set a variable "pass" to 1 (true) or 0 (false), to determine if this job should be applied to the document. If you wish to set a job name for later images in a job batch process, after a job has been detected, you can set `selectjob$=jobname$`.

- **fields** is run to assign values to custom data fields. Validations are also run here.
- **identification** is run to assign values to identification fields. Validations are also run here.
- **finalize** is run after the document has been processed. This code should clean up any objects or settings created in the initialize code.
- **preupload** is run before a document is transferred to an archive library. It is possible to cancel the transfer by setting the variable `errmsg$` to a message.
- **postupload** is run after a document has been transferred. A message can be provided to the user by setting `errmsg$` to a value, though unlike `preupload`, this does not cancel the transfer.

Reserved Variables

The following variables can be used for special purposes in scripting:

- **selectjob\$** to set the job name to a known value. This can be used in custom detection logic to fix a job name once detected for a batch, such as during header/attach logic where you want the first job detected to be the job used throughout the batch.
- **jobname\$** contains the name of the job that has been detected, or preassigned.
- **doclist\$** contains a line-feed delimited list of documents to be processed, where each line contains tab-separated source ID, doctype, and docid, or just doctype and docid. This list can be manipulated by script code if necessary. For example, if a document is split, the new documents can be added to `doclist$` for processing during the current batch. Be sure to separate lines with linefeed (`$0a$` or `chr(10)`) characters. To avoid endless loops, be careful to avoid setting this variable to a static value.
- **errmsg\$** can be set to a message in `preupload` code, to prevent the upload from proceeding.
- **targetlib** is a library object of the library the image has been transferred to, available in the `postupload` code.
- **subid\$** contains the subid of the newly transferred image file, available in `postupload` code.
- **xmlsubid\$** contains the subid of the newly created and transferred xml data file, available in `postupload` code.

Block If

This is valid in custom scripting:

```
if a=b then {
  # code if a=b
} else {
  # code if a<>b
}
```

This is valid in both custom scripting and code blocks:

```
if a=b then
```

```
# code if a=b
else
# code if a<>b
end if
```

9.6 Script Libraries

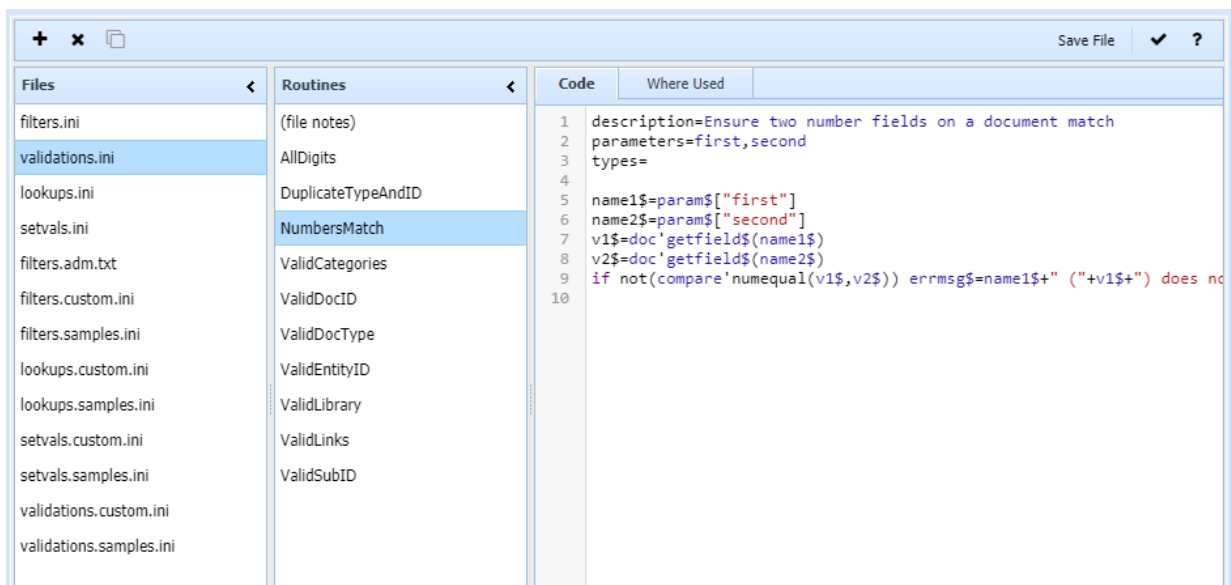
Script libraries are standard routines of code that perform functions available to Image Manager jobs. Code is run via the filters, lookups, and validations objects. There are different code library sets, and for each one there is a set provided by the publisher and one that is available to customize by the user. Any file with a "custom.ini" suffix is maintainable by users. If there are duplicate section names in both files, the custom.ini version takes precedence, though that may mask functionality added by the publisher at updates.

The left panel displays the files available to view or edit. The center panel displays the routines found in that file, along with a (file notes) section that provides documentation. The right panel contains the settings and code for that routine. Settings include specific name=value lines appropriate for the type of script. These lines, and comment lines starting with #, are ignored when the script code runs. All the other code executes and performs a specific task.

The Where Used tab displays a grid of where the selected routine is referenced in Image Manager and DocFlow jobs.

The Script Libraries tool provides support for alternate library files that can contain pre-defined routines for specific product integrations. These follow a different naming convention, where the suffix is *something*.ini rather than .ini or custom.ini. Routines in these files can be copied to *.custom.ini files, and visa versa. In some cases, it can even be beneficial to copy a whole file over the custom.ini version at a system level, to provide a starting set of routines for a specific ERP integration.

When selecting routines from a file other than the standard .ini or .custom.ini, the toolbar provides a copy button to enable copying one or more selected routines to the custom.ini version of this file.



9.6.1 Filters

Filters are executed via the [filters object](#), and are found in filters.ini and filters.custom.ini. They are implicitly run by job designs, or can be run by any custom code, including other filters. Filter routines are designed to convert one form of text to another, such as converting raw numbers into formatted strings, or removing invalid characters. The routine is designed to work with a variable in\$, and create a variable out\$. When the filter returns a value, it is returning out\$.

Filters can also have side effects that can be beneficial, such as updating properties in a inbounddoc object. When a filter runs, it typically has an [inbounddoc](#) object available to it in a variable "doc". When a filter is run from a DocFlow job, the doc object is a [docflowdoc](#) object. When a doc object is available and has a jobname\$ or flowid\$ property assigned (which is automatic when run in a job execution context), then that job's parameters are used to pre-create "param.name\$" variables for each parameter.

Filters provide three settings:

- **description** is a text value that is displayed in job design.
- **parameters** is a comma-separated list of parameter names. When the filter is run, it should be supplied these parameters either as an associative array or in the name structure, as described in the filters object documentation. When run within an Image Manager design, these parameters are supplied by the designer.
- **types** is a list of field or zone types this filter logically applies to. The Image Manager will use this value to display appropriate options when looking up filters during job design. Filters that apply to grid data must have "grid" in this list. General text filters can contain any type of bcd, col, and ocr in order to apply to those types, plus text, longtext, note, number, lookup, list, and link data fields. So, for example, if a filter specifies a type of "col", it will be offered by the job designer for text or longtext fields as well. Other data types must be named in the types list, including date, checkbox, and radio, as they have very specific formatting requirements.

9.6.2 Validations

Validations are executed via the [validations object](#), and are found in validations.ini and validations.custom.ini. They are implicitly run by job designs, or can be run by any custom code, including other validations. Validation routines are designed to test and confirm the validity of a value, such as ensuring a customer ID exists in a customer table, or that column total matches a form total. The routine tests the value in any way needed, and if there is a problem, it sets errmsg\$="some message". The presence of a message in that variable indicates the validation failed.

When validations are executed by jobs, this pass/fail result is managed for all the fields and identification values that have a validation configured. Documents with invalid values cannot be transferred to an archive library, and the Image Manager interface highlights and displays the problems found on the document.

Validations can also have side effects that can be beneficial, such as updating properties in a [inbounddoc](#) object presented as the doc variable. For example, a validation that tests the existence of a customer record can also update an associated customer name field on the document.

When a doc object is available and has a jobname\$ property assigned (which is automatic when run in a job execution context), then that job's parameters are used to pre-create "param.name\$" variables for each parameter.

Validations provide three settings:

- **description** is a text value that is displayed in job design.
- **parameters** is a comma-separated list of parameter names. When the validation is run, it should be supplied these parameters either as an associative array or in the name structure, as described in the validations object documentation. When run within an Image Manager design, these parameters are supplied by the designer.
- **types** is a list of field or zone types this filter logically applies to. The Image Manager will use this value to display appropriate options when looking up validations during job design.

9.6.3 Lookups

Lookups are executed via the [lookups object](#) and are found in lookups.ini and lookups.custom.ini. They are implicitly run by job designs, or can be run by any custom code. Lookup routines are designed to create a tab-separated-values string 'rows\$'. Often they will rely on parameters to limit the amount of data they return.

Lookups are also integrated into job designs in custom fields defined as Lookup types. These fields have an ellipsis button to execute the lookup and display results, allowing a user to select a specific record to obtain a value for the lookup field.

Lookups can also have side effects that can be beneficial, such as updating properties in a [inbounddoc](#) object, typically provided in a 'doc' variable. When a doc object is available and has a jobname\$ or flowid\$ property assigned (which is automatic when run in a job execution context), then that job's parameters are used to pre-create "param.name\$" variables for each parameter.

Lookups provide four settings:

- **description** is a text value that is displayed in job design.
- **parameters** is a comma-separated list of parameter names. When the filter is run, it should be supplied these parameters either as an associative array or in the name structure, as described in the filters object documentation. When run within an Image Manager design, these parameters are supplied by the designer.
- **cols** is a comma-separated list of column names used by the Image Manager when presenting the result of the lookup. This list should match the content of each tab-delimited row of the lookup result.
- **value** is the name of the column that will be pasted by the Image Manager into the lookup field. When the user selects a row in the lookup list, this column from that row is the value that is used.

9.6.4 Set Values

These are special routines that do not actually execute code. Instead, the routine consists of name=value pairs, with a list of specific names that correlate to the identification fields of a document, or a 'reset' option of the field to force it's value to null.

These routines are accessed in the tool bar of the [Manually Set Values](#) window. A description=text line is used in this window to describe the function of the routine. Other lines can be one or more of:

- `library=librarypath`
- `doctype=doctype`
- `docid=docid`
- `subid=subid`
- `subtitle=subtitle`
- `title=doc-title`
- `entityid=entityid`
- `notes=notes` (use `\n` for line breaks)
- `categories=categories` (pipe-delimited segments, `\n` line breaks)
- `keywords=keywords` (`\n` separator)
- `links=links` (`\n` separator, pipe-delimiter for lib|doctype|docid structures)
- `jobname=jobname`

To reset a value, use one of these names:

- `resetlibrary=1`
- `resetdoctype=1`
- `resetdocid=1`
- `resetsubid=1`
- `resetsubtitle=1`
- `resettitle=1`
- `resetentityid=1`
- `resetnotes=1`
- `resetkeywords=1`
- `resetcategories=1`
- `resetlinks=1`
- `resetjobname=1`

Only names that should be changed need to be

9.6.5 Global Code

The file `globalcode.custom.ini` can be edited inside fixed section names. Code found in these sections is inserted into every image manager and docflow job after any related custom code section. Use this to apply site- or application-specific features or cleanup code that should be applied universally to all image manager or docflow job definitions.

Image Manager Sections

- `im-initialize`
- `im-zones`
- `im-detect`
- `im-fields`
- `im-indentification`
- `im-finalize`
- `im-preupload`
- `im-postupload`

DocFlow Sections

- `df-initialize`
- `df-advance`
- `df-finalize`

9.7 Standard Filters

Standard filters are provided with every UnForm installation, available for use in Image Manager job definitions. The tables below describe the use of each of these standard filters. Using the script editing tool, it is possible to copy these filters to the filters.custom.ini file, and use the filter's code as the basis for a new custom filter.

All filters are designed to convert an input string to an output string. The output string is used in place of the original string when the filter is applied. The code of the filter must use two variables: in\$ and out\$, that contain the input and output data. The in\$ variable is pre-populated when the filter code starts to run, and it is the code's job to create the out\$ variable with the desired result. The out\$ filter always starts as null ("") when the filter code is run.

Parameters

Many filters use parameters, which are case-sensitive named values, passed inside parentheses as name=value pairs. Parameter values can be literal values or expressions enclosed in curly braces. Often an expression will refer to a job parameter, like {param.taxheader\$}, or another zone or data field. If a parameter value contains a comma or parenthesis, it should be quoted.

Chained Filters

In job design, filters can be chained together to achieve an end result, such as removing headers, removing footers, and joining rows. When chained, the output from one filter is used as the input for the next filter in the zone or field definition.

Page Headers

Many filters are applied to zone data, either ocr or barcode, which is text lifted off one or more pages in a specific position. Generally this data will have multiple lines, and page headers, so often filters are designed to get rid on this extra information. Page headers look like a line with this structure: [page]. Many filters remove these page header rows as part of their logic, using one of these functions:

- iszonepagehdr(x\$) returns true (1) if a text value, normally a single line, is a page header (the whole value is "[n]")
- removezonepagehdrs(x\$) removes all page headers from the string passed to it. There is also a filter named RemovePageHeaderRows that executes this function in a filter context.

Text Filters

These filters are designed to work with single- or multi-line text, where the text is not structured into columns. Examples are document identifiers, name and address blocks, subtotals and totals, or dates. They differ from grid data, which is composed of rows of discrete column data. These filters can also work on grid column zones, but to retain column relations across rows, avoid filters that remove headers or lines in grid column zones.

Name	Removes Headers	Description
anyText	Yes	Removes page headers and converts line breaks to spaces, so all words are on one line.

Name	Removes Headers	Description
concat	No	<p>Concatenates two values, with an optional delimiter. The values and delimiter are passed as parameters.</p> <p>Parameters: <code>val1=value</code> <code>val2=value</code> <code>dlim=delimiter</code></p> <p>Examples: <code>concat(val1=Customer,val2={field.CustID\$},dlim=": ")</code></p>
dateColumn	No	<p>Formats dates in a multi-line string, returning a string with the same number of lines, with each line either a formatted date or null. This is useful in a grid column with date values.</p> <p>Parameters: <code>DateOrder=mdy dmy ymd</code> - what order are the date segments, for date parsing purposes. <code>Format=format</code> - the format the dates should be in the output, using YYYY, YY, MM, or DD placeholders. <code>Alpha=y n</code> - if set to y or yes, month names are first converted to month numbers before the parse.</p> <p>Examples: <code>dateColumn(DateOrder=mdy,Format=YYYY-MM-DD,Alpha=n)</code></p>
extractPattern	Yes	<p>Returns data from a string that matches a regular expression pattern, or a simplified pattern. If the regex contains a parenthesized group, the group is returned. UnForm supports Perl compatible regular expressions (PCRE)</p> <p>Parameters: <code>regex=regular-expression</code> (may contain one parenthesized group) <code>pattern=pattern</code></p> <p>The <i>pattern</i> is designed to be simpler to use, but offers less flexibility than a regular expression. Patterns honor the following special matching characters:</p> <ul style="list-style-type: none"> • A matches any uppercase letter • a matches any lowercase letter • X matches any letter • # matches a digit • * matches any number of characters • ? matches any single character • [list] matches the list of characters (i.e. [+ -] matches either plus or minus) • \ escapes the next character so it is interpreted as a literal (\A matches only "A") • Other characters match themselves

Name	Removes Headers	Description
		<p>Examples:</p> <p><code>extractPattern(regex=[A-Z]{1,3}-\d\d\d\d)</code> returns three upper-case letters, a hyphen, and 4 digits, if they are present in the input.</p> <p><code>extractPattern(regex="Customer: (\d+)")</code> returns a customer number, if it is one or more digits following a text string "Customer: ".</p> <p><code>extractPattern(pattern=AA-####)</code> matches two uppercase letters, a dash, and four digits.</p> <p><code>extractPattern(pattern=\AX-*#)</code> matches an A followed by any letter, a dash, any number of characters, another dash, and a digit.</p>
firstLine	Yes	Returns the first line in a zone that is not empty or a page header.
firstWord	Yes	Returns the first word from the first line in a zone that is not empty or page header line.
forceUndetectedDecimal	No	<p>Scans lines looking for values that do not have a decimal. Numeric values that do not have a decimal point are assumed to be a misread and the value is adjusted by scaling it for decimals. For example, a value of "199 00" would become "199.00".</p> <p>Parameters:</p> <p><code>decimals=<i>n</i></code> - the number of decimal places expected in column values.</p>
lastLine	Yes	Returns the last line that is not empty or a page header
lastWord	Yes	Returns the last word from the last line that is not empty or a page header
MapValue	No	<p>Modifies values in a multi-line string based on a map of original and new values. For example, you can normalize unit of measure values by mapping vendor values to your own values.</p> <p>Parameters:</p> <p><code>FromList=<i>list</i></code> - a delimited list of values to convert from</p> <p><code>ToList=<i>list</i></code> - a delimited list of corresponding values to convert to</p> <p><code>Delim=<i>delimiter</i></code> - the list value separator, which defaults to a comma</p> <p>Examples:</p> <p><code>MapValue(FromList="Each,Case,Dozen",ToList="EA,CS,DZ")</code></p> <p><code>MapValue(FromList={param.VendorUOM\$},ToList={param.OurUOM\$})</code></p>
numColDecimals	No	<p>Reformats lines of numbers to ensure they have a certain decimal precision. Non-number lines become empty lines. Optionally provide a format for the numbers, to enable more control.</p> <p>Parameters:</p> <p><code>decimals=<i>number</i></code> - the number of decimal places numbers must have</p> <p><code>format=<i>mask</i></code> - a formatting mask, using #,0,-,. characters</p> <p>Examples:</p>

Name	Removes Headers	Description
		<p>numColDecimals(decimals=3) - a row with "1,123.24" would become "1123.240"</p> <p>numColDecimals(decimals=,format="###,###.000") - note the quotes to protect the format comma</p> <p>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.</p>
OCRLettersToNumbers	No	Replaces common character misreads in what should be numeric data.
parseDelimitedDate	Yes	<p>Converts a delimited date value to a new format.</p> <p>Parameters:</p> <p>DateOrder=mdy dmy ymd - what order are the date segments, for date parsing purposes.</p> <p>Format=<i>format</i> - the format the dates should be in the output, using YYYY, YY, MM, or DD placeholders.</p> <p>Alpha=y n - if set to y or yes, month names are first converted to month numbers before the parse.</p> <p>Examples:</p> <p>parseDelimitedDate(DateOrder=mdy,Format=YYYY-MM-DD,Alpha=n)</p>
RemoveChars	Yes	<p>Removes a set of characters from each line.</p> <p>Parameters:</p> <p>CharList=<i>chars</i> or <i>\$hexstr\$</i> - a sequence of characters or a hexadecimal string, each character of which is removed from each line</p> <p>Examples:</p> <p>RemoveChars(CharList="-,.#") - removes hyphens, commas, periods, and hash signs, using a quoted string to protect the comma from being parsed.</p> <p>RemoveChars(CharList=\$0922\$) - removes tab characters and quote characters.</p>
RemovePageHdrRows	Yes	Removes page header rows ([1], [2], etc.) from the zone.
RemoveSpaces	Yes	Removes spaces from each line in the zone. This is useful in cases where a value should not have spaces, but font calculations by the OCR engine have inserted spaces.
trimLines	Yes	Removes page headers and empty lines.
UnwrapHyphenWords	No	Looks for lines that end with a hyphen (-) and are followed by a line with data. Where found, the second line is appended to the first line, and the second line becomes an empty line. Note the hyphen is not removed, as often it is part of valid data. To remove it, use RemoveChars.
validChars	Yes	This filter looks for a specific line, then filters the value in that line so it only contains characters for a type of value. A similar but more flexible filter is ExtractPattern.

Name	Removes Headers	Description
		<p>Parameters:</p> <p><code>type=class</code> - defines the set of valid characters</p> <p><code>firstlast=first last lastallpages</code> - first line, last line of first page, last line of all pages</p> <p>The class type can be one of these:</p> <ul style="list-style-type: none"> • letters - upper and lowercase letters • alpha - letters and digits • alphanum=letters, digits, spaces, punctuation • alphanum2=uppercase letters, digits, punctuation • numbers=digits, commas, decimal points, minus sign • numbersparen=numbers and parentheses • digits=only digits • datechars=digits, forward slash, hyphen • datecharsalpha=digits, letters, comma, forward slash, space, and hyphen <p>After character filtering, if <code>type=datecharsalpha</code>, month names are converted to digit equivalents.</p>
VerticalTotalZoneMatch	Yes	<p>Scans lines for a label or pattern match, and returns a value found to the right of that label or pattern. This can be used to extract a specific value, such as a total, freight, or tax amount, from a zone. It can look for a single value, or it can sum multiple values.</p> <p>Parameters:</p> <p><code>label=string</code> or <code>string1 string2...</code> a label or ~regex pattern, or a pipe-separated list of them</p> <p><code>textmode=y</code> to return a textual value instead of a numeric one</p> <p>The label value can contain multiple items separated with a pipe (). Each of these can be either a text string or a ~regex pattern. A text string can contain spaces to look for more than one word preceding the target value.</p> <p>The <code>textmode=y</code> is only supported for single label values.</p> <p>Examples:</p> <p><code>VerticalTotalZoneMatch(label=~\w+ Tax:)</code> returns a numeric value to the right of a line that contains word characters, a space, and the word "Tax:"</p> <p><code>VerticalTotalZoneMatch(label=CA Tax LA Tax)</code> returns a sum of values to the right of "CA Tax" and "LA Tax".</p> <p><code>VerticalTotalZoneMatch(label=Carrier:,textmode=y)</code> returns any text to the right of Carrier:.</p>
wordAfter	Yes	Returns a word after a specified word or words. The input text is first flattened to not have any line breaks, and to only have single spacing.

Name	Removes Headers	Description
		<p>This is useful to extract a value that follows a header, either to the left or above a target value.</p> <p>Parameters: Searchfor=<i>word(s)</i> - a case-insensitive word or words that precedes the target value</p> <p>Examples: wordAfter(Searchfor=Purchase Order) - returns a word after or below "Purchase Order".</p>

Grid Filters

These filters are designed to manipulate grid data, which is internally stored as a tab-separated-values list with a first row of headers based on the column zone names that the grid contains. Grid filters are designed to retain column relations across rows. They often use the grid object, which provides a large number of methods to manipulate such data.

Name	Description
colTotalNamedCol	<p>Returns the sum of numbers in a given column, defined by name or column number.</p> <p>Parameters: colName=<i>name or number</i>, specifies which grid column to use.</p> <p>Examples: colTotalNamedCol(colName=Extension) colTotalNamedCol(colName={param.ExtCol\$})</p>
FieldFromGridRow	<p>Scans a grid column, bottom to top, for a value or a regular expression pattern. If found, get a value from the same row in another column, and place that value in a data field defined in the job. That row is removed from the grid. Note how this differs from a typical filter in that a data field value is updated, rather than simply being returned in out\$.</p> <p>Parameters: SearchFor=<i>value or ~regex</i> - what to look for in SearchCol SearchCol=<i>name or number</i> - what column to scan for SearchFor FieldName=<i>name</i> - the data field name in the job definition where a value should be placed FieldCol=<i>name or number</i> - the column that contains the value to place in the data field</p> <p>Examples: FieldFromGridRow(SearchFor=~Freight:,SearchCol=ItemDesc,FieldName=Shipping,FieldCol=Price)</p>

Name	Description
FindGridRow	<p>Scans a grid column for a value or pattern, and returns that row's value from another column. This is useful for extracting data values from a grid, such as freight or tax value, in a data field filter. It is similar to FieldFromGridRow, but is typically used in a data field rather than a grid zone.</p> <p>Parameters: SearchFor=<i>value</i> or <i>~regex</i> - what to look for in SearchCol SearchCol=<i>name</i> or <i>number</i> - what column to scan for SearchFor ReturnCol=<i>name</i> or <i>number</i> - the column containing the row data to return</p> <p>Examples: FindGridRow(SearchFor=CA Tax,SearchCol=Item,ReturnCol=Extension)</p>
initCol	<p>Initializes a grid column to all empty rows. This can be used to clear a column after another filter has used data in it and it is no longer needed.</p> <p>Parameters: Col=<i>name</i> - the name of the column to initialize</p>
JoinRows	<p>Joins rows together based on a value or pattern in a specific column or column list, by appending row values together in all columns. The logic groups rows together based on a value or pattern in a specific column. An empty row also starts a new group. This works for well structured line detail that has a top row for each line, and stacked data in some columns. When appending, a space separates the line values.</p> <p>Parameters: SearchCol=<i>name</i> or <i>name1,name2...</i> - columns to search SearchFor=<i>value</i> or <i>~regex</i> - a value or pattern to scan SearchCol for RowDelim=<i>text</i> to place between row values, defaults to a space</p> <p>Examples: JoinRows(SearchCol=LineNo,SearchFor=~\d,RowDelim=) - scans the LineNo column for any digit, when found treat that row as the top line, and subsequent lines are appended to this one in each column, until the next LineNo match or an empty line. A pipe () is inserted between each joined row.</p>
KeepPatternRows	<p>Filters the rows in a grid, retaining only those that match a value or pattern in a specified column. This is useful to filter out rows with just item details, often used after first joining rows.</p> <p>Parameters: SearchCol=<i>name</i> - the column to scan SearchFor=<i>value</i> or <i>~regex</i> - the value or pattern to match in SearchCol - non matching rows are removed</p> <p>Examples: KeepPatternRows(SearchCol=Item,SearchFor=~[A-Z0-9]{4,16}) - keep rows with Item column values from 4 to 16 uppercase letters or digits</p>
MapGridCol	<p>Modifies values in a grid column based on a map of original and new values. For example, you can normalize unit of measure values by mapping vendor values to</p>

Name	Description
	<p>your own values.</p> <p>Parameters: ColName=<i>name</i> - name of column to scan FromList=<i>list</i> - a delimited list of values to convert from ToList=<i>list</i> - a delimited list of corresponding values to convert to Delim=<i>delimiter</i> - the list value separator, which defaults to a comma</p> <p>Examples: MapGridCol(ColName=UOM,FromList="Each,Case,Dozen",ToList="EA,CS,DZ")</p>
RemoveEmptyRows	This removes rows that have no content in any column.
RemoveGridFooter	<p>Scans for the last occurrence of a value or regular expression one or more columns, and removes that row and those below it.</p> <p>Parameters: SearchCols=<i>col1 col2 ...</i> - an optional pipe-delimited list of column names or numbers, if empty all columns are searched. SearchValue=<i>value</i> or <i>~regex</i> - a value or regular expression to search for</p> <p>Examples: RemoveGridFooter(SearchCols=Item Description,SearchValue=Subtotal)</p>
RemoveGridHeader	<p>Scans for the first occurrence of a value or regular expression one or more columns, and removes that row and those above it. This filter can work at the document level, or it can scan page by page.</p> <p>Parameters: SearchCols=<i>col1 col2 ...</i> - an optional pipe-delimited list of column names or numbers, if empty all columns are searched. SearchValue=<i>value</i> or <i>~regex</i> - a value or regular expression to search for ByPage=<i>y</i> - set to y or yes to scan each page for headers to remove</p> <p>Examples: RemoveGridHeader(SearchCols=Item,SearchValue=Part No,ByPage=y)</p>
RemoveGridRows	This filter is deprecated, and only included for supporting existing jobs developed during the v10 beta cycle and still in use.
RemovePageHdrRows	Removes page header rows ([1], [2], etc.) from the grid.
RemovePatternRows	<p>Removes rows where a column matches a value or regular expression pattern. This is the opposite of the KeepPatternRows filter.</p> <p>Parameters: SearchCol=<i>name</i> or <i>number</i> - the column to search SearchFor=<i>value</i> or <i>~regex</i> - the value or pattern to search for in the specified column</p> <p>Examples: RemovePatternRows(SearchCol=LineNo,SearchFor=~[A-Z][A-Z]) - remove lines with uppercase letters in the LineNo column.</p> <p>RemovePatternRows(SearchCol=Item,SearchFor=Tax) - removes rows with Item values of "Tax".</p>

Name	Description
SplitColumn	<p>Splits values in a column into two separate columns. One column contains the original data, and retains the first portion of the split, and a second column receives the second portion of the split. This can be used in cases where one column contains two distinct pieces of information, such as a quantity and unit of measure.</p> <p>Parameters:</p> <p>FirstCol=<i>colname</i> - the column that contains the data to split, and retains the first portion of the split data</p> <p>SecondCol=<i>colname</i> - the column to place the second portion of the split data</p> <p>Splitter=<i>number</i> or <i>character</i> - defines how the split is performed</p> <p>Splitter Options:</p> <ul style="list-style-type: none"> • Positive number, the first portion is this many characters from the start of the data, the second portion is the remaining characters • Negative number, the second portion is this many characters from the end of the data, the first portion is remaining characters • <i>~^regex</i> - a regular expression anchored to the front of the string - the first portion is the matched characters, the second portion is the remaining characters • <i>~regex\$</i> - a regular expression anchored to the end of the string - the second portion is the matched characters, the first portion is the remaining characters • <i>~regex</i> - an unanchored regular expression - the second portion is the matched characters, the first portion is the original value with the matched portion removed. If there is a parenthesized group, that group's value is used as the second portion and the entire match is removed to form the first portion • <i>delimiter</i> - a string delimiter splits the value on that delimiter, and the first portion is the data before the delimiter, and the second portion is the data after the delimiter <p>Examples:</p> <p>SplitColumn(FirstCol=Qty,SecondCol=UOM,Splitter=-2) would split "123EA" into 123 and EA, by using the last 2 characters as the second portion.</p> <p>SplitColumn(FirstCol=Qty,SecondCol=UOM,Splitter=~^d+) would split "123EA" into 123 and EA, by matching one or more digits at the start of the string to be the first portion.</p> <p>SplitColumn(FirstCol=Qty,SecondCol=UOM,Splitter=~[A-Z]+\$) would split "123EA" into 123 and EA, by matching one or more uppercase letters at the end of the string to be the second portion.</p> <p>SplitColumn(FirstCol=Qty,SecondCol=UOM,Splitter=/) would split "123/EA" into 123 and EA, using "/" as a delimiter.</p>
StackedLineCleanup	This filter is deprecated, and only included for supporting existing jobs developed during the v10 beta cycle and still in use.

9.8 XML Document

Below is an annotated example of the XML document produced when a document is transferred to a library. This document has a subid based on the sequenced job ID (i.e. AcmeDocs*). When a transfer is run, the postupload code has a variable `xmlsubid$` that contains the subid value of the xml file, so you can retrieve the actual XML with a library object `'getimage()'` method.

The root node is `/inbounddoc`, with a `srcid` attribute of the source of the document.

```
<inbounddoc srcid="doc2">
```

The `/inbound/identification` node contains subnodes related to document identification and indexing, and also the job name of the image manager job applied to this image.

```
<identification>
<user>admin</user>
<library>demo_accounting</library>
<doctype>Vendor Invoice</doctype>
<docid>BRF010-25072969-00</docid>
<subid>Scan</subid>
<subtitle></subtitle>
<title></title>
<notes></notes>
<categories>poNumber|</categories>
<keywords></keywords>
<links></links>
<date></date>
<entityid></entityid>
<jobname>ap_wGrid_BRF</jobname>
</identification>
```

The `/inbounddoc/fields` node contains subnodes for each field defined in the job. A `type` attribute defines the type of field. Grid types include are further defined with `/rows`, `/rows/row`, and `/rows/row/col` tags.

```
<fields>
<vendorID type="text">BRF010</vendorID>
<vendorName type="text">BRF</vendorName>
<invoNum type="text">25072969-00</invoNum>
<docDate type="text">2018-03-27</docDate>
<poNum type="text"></poNum>
<GLaccount type="text"></GLaccount>
<GLsubaccount type="text">000-00-00</GLsubaccount>
<merchTotal type="number">2,724.78</merchTotal>
<taxTotal type="number">190.73</taxTotal>
<freightTotal type="number"></freightTotal>
<otherTotal type="number"></otherTotal>
<discountTotal type="number"></discountTotal>
<calcTotal type="number">2915.51</calcTotal>
<invoTotal type="text">2,915.51</invoTotal>
<flowDept type="text">Other</flowDept>
<terms type="text"></terms>
<receiptID type="text"></receiptID>
<Lines type="grid"><rows>
  <row>
    <col name="Qty">10.10</col>
    <col name="Item">312P300-040 4" PVC DWV 90 ELBOW</col>
    <col name="Price">5.220</col>
    <col name="Disc">.00</col>
    <col name="Extend">52.20</col>
    <col name="Uom">EA</col>
  </row>
```

```

<row>
  <col name="Qty">10.10</col>
  <col name="Item">312P101-040 4" PVC DWV FEMALE</col>
  <col name="Price">3.010</col>
  <col name="Disc">.00</col>
  <col name="Extend">30.10</col>
  <col name="Uom">EA</col>
</row>
...
</rows></Lines>
<LinesTotal type="number">2724.78</LinesTotal>
</fields>

```

The /inbound/metadata node contains subnodes related to the source, such as an original filename or to, from, and subject values from an email. For user-uploaded images, this is empty.

```

<metadata>
<filename>Scan-20180901-001.pdf</filename>
</metadata>

```

The /inbound/parameters node contains elements for each parameter defined for the job that this image used.

```

<parameters>
<VendorID>000100</VendorID>
</parameters>

</inbounddoc>

```

9.9 Bulk Job Update

If you have many jobs based on some standard, you can perform a bulk update to copy and synchronize several job attributes from a source job to multiple target jobs. The updates include job attributes, like job type, parameter names and values, new zones, field validations, identification values and validations, and script code.

To perform a bulk update, select a job to base the update on in the job editor, then click the toolbar button with the tooltip 'Update other jobs based on this one'. This opens the bulk update window with tabs to select the target jobs and the other items to update. Each element displayed includes a check box to select whether to update that value. Updates are limited when a target job already has a matching item. For example, matching data field rows will only update the validation and autofill settings, not the data calculation settings. However, when an item to update does not appear in the target job, the whole item is copied. The specifics of each tab's impact are displayed at the top of the window as tabs are selected.

Once the selections are made, the summary tab will show the specific items to be updated, along with some checkboxes to select additional synchronization options for the different functions. For example, unused parameters can be removed, and parameters can be reordered to match the source job. The summary page offers two buttons, for "Run" and "Run in Test Mode". Test mode will result in a report of what changes would have been made. Run will create a backup of the job definition file (as `im/backups/ufjobdef.yymmdd.hhmmss.dat`), perform the changes, and present a report. A copy of the report, in html format, is stored along with the definition file backup.

Update jobs based on AC_Ap_Atlas

Select parameters to update or ensure exist

Jobs	Attributes	Parameters	Zones	Fields	Identification	Script Code	Summary
Name	Description	Value	Update?				
Library		AC Finance AP	<input checked="" type="checkbox"/>				
DocType		Vendor Invoice	<input checked="" type="checkbox"/>				
vendor		AAVENDOR	<input type="checkbox"/>				
vendorName		Atlas	<input type="checkbox"/>				
GLaccount		50000	<input type="checkbox"/>				
GLsubaccount		000-000	<input type="checkbox"/>				
dtFormat		mm-dd-yy	<input type="checkbox"/>				
company	Acumatica company ID	PRODUCTS	<input type="checkbox"/>				
branch		PRODWOLE	<input type="checkbox"/>				
terms			<input type="checkbox"/>				
stackedTotals		Yes	<input type="checkbox"/>				
totalLabel		TOTAL DUE	<input type="checkbox"/>				
merchTotalLabel		SUBTOTAL	<input type="checkbox"/>				
taxLabel		PST 6.50% GST 3.20%	<input type="checkbox"/>				
freightLabel		SHIPPING & HANDLING	<input type="checkbox"/>				
discountLabel			<input type="checkbox"/>				
otherLabel		PAID	<input type="checkbox"/>				

9.10 External OCR Processing

Through proper configuration, UnForm can be configured to use a third party OCR processing tool rather than the default open source Tesseract tool. Commercial tools can provide higher OCR accuracy than Tesseract. The goal of this integration is to convert images and PDF files that do not contain text to PDF files that contain text. A PDF file with text enables automated text extraction during the inbound document parsing process, enabling Image Manager jobs to process those documents in an automated fashion, often without human intervention.

This integration is controlled by a configuration section in uf101d.ini. This section is not provided by default, but can be added to the file. Here is an example:

```
[ocrdrop]
path=c:/sdsi/ocrdrop
resultscontain=fromuf.
```

The `path=path` setting defines where incoming documents should be placed when they do not contain text that UnForm can use. This path should be a location that the OCR tool is configured to monitor and process. The OCR tool and UnForm do not necessarily have to be on the same system, but must be able to share a file system location, both products with full permission to control the contents of that path.

Whenever UnForm is parsing an incoming document, if it finds there is no text, it will create a copy of the incoming file in the specified path, and then remove the now redundant file from its inbound processing. The expectation is that the OCR processing tool will pick up the file, generate a PDF file with text, and drop that PDF in another path that UnForm is monitoring as an inbound source path. This will then become a new inbound document, this time with a text layer that can be extracted for processing.

To prevent UnForm from re-submitting a file that the external tool is unable produce text for in its generated PDF, a name match is performed when determining whether to submit the file to the external tool. If an incoming file name contains the string found in `resultscontain=string`, it will *not* be submitted to the configured *path*, but instead processed normally by UnForm. All UnForm-generated files created in *path* will contain a prefix "fromuf.", so if the OCR processing tool is configured to include the name of the file it receives from UnForm, the string "fromuf." is an appropriate value. However, any string can be specified, as long as the files generated by the OCR tool contain that string.

In a nutshell, if a third-party tool can monitor a directory for documents to process, and can place resulting PDF files with text in another directory UnForm is monitoring, you can configure UnForm to send it documents, and receive the results through an inbound source path.

10 DOCFLOW AND ANNOTATION

The DocFlow module provides programmable step-based workflow for document images found in archive libraries. A docflow for an image is started by a user, print rule set, or other programmable means, such as from an Image Manager postupload script. When a docflow is started, additional attachments and custom data fields can also be supplied to provide docflow users with additional information they need to monitor the status of the image as it moves through the steps of the flow. Flows can be used for many purposes, such as approval processes, document collaboration, or signature capture and other proof of delivery applications.

DocFlow is a browser-based product, accessible via the UnForm web server. You can access it via the web server's portal page, or directly with a URL. The URL structure is simple. If the web server is on IP address 192.168.1.10:28292, you would use this link:

<http://192.168.1.10:28292/arc?df=1>

The DocFlow module is accessible to any user who has been granted Docflow permissions in the [Internal Users](#) list.

10.1 Overview

The DocFlow module is composed of a browser interface, role and definition configuration tools, and a programmable object. Any number of flows can be defined, with varying degrees of complexity. A common use is to manage accounts payable approvals, where one flow might be used for expense invoices, another more complex one for inventory purchases. When an archive library image enters a flow, it is stored in a system library. Users perform actions on those images and move it from step to step, editing associated data, adding new attachments, notes, or drawings.

When the document reaches the final step, one or more new image files are added to the original source document (the parent document of the starting image): an xml document with the captured data and editing history of the document while it was in the flow, any attachments added by users, and an [annotated version](#) of the original image, if any annotations were performed.

Browser Interface

The browser interface supports any modern browser, such as Chrome, Firefox, Edge, and Safari (though not Internet Explorer). It is designed to be both *responsive*, meaning it works well on both small and large screens, and *offline capable*, meaning that once loaded, it can then run without a network connection. The offline capability requires a secure (https) website with a public certificate, as browsers do not support this feature on insecure connections. To load the application for offline use, use the Settings window.

Roles

Flow security and actions are controlled by roles. A role is simply a named-entity with one or more docflow users as members. Any user can be part of any number of roles.

Flow Definitions

Flow definitions specify a series of steps that a document moves through, along with custom field specifications to enable data capture. Each step has a responsible role assigned to it. When a document is at a step, any user who is a member of the assigned role can edit the document. Additionally, any user who is a member of any role specified by the flow has view access to the document. DocFlow users who are not members of any role used by the flow definition cannot see documents in that flow.

Flow definitions can enable annotations, which are user-drawn notations on the main image. This edited version of the image is uploaded as a separate PDF file into the source library archive, enabling signature capture or document markup.

Custom data fields include various text input types, checkboxes, radio buttons, selection lists, url links, and a grid feature.

Scripting is available at three points of processing: at initialization, when an image first arrives in a flow, at step advances, when the user clicks the Forward button in the interface, and at the finish point, after the parent document has been updated and the image is being removed from the flow. Scripting enables flows to be customize in various ways, such as to generate notifications, or update related documents, or control the step route that a document goes through. For example, a flow might have five steps defined, but only certain steps are required under some conditions. For example, an invoice under a threshold amount might skip a second approval level, but if over that threshold notify a second level approver and move to that step.

10.2 Browser Interface

The browser interface provides a main toolbar and two panels. The toolbar provides flow selection, a manual sync button and auto-sync toggle, sorting, and a drop down menu with additional options. The left panel displays documents in the flow, the right displays information about the currently selected document. The left panel can be displayed as a grid or a list, depending on settings and the screen width.

The screenshot displays the DocFlow and Annotation interface. On the left, a sidebar titled "Active Documents in AP_Factory" shows a list of documents, with "2018-07-24 (1)" selected. The main panel displays the details for document "0724-4402-5739".

Document Details:

- ID: 0724-4402-5739
- Started: 2018-07-24 12:13
- Step: AcctgClerkReviewPrep
- Updated: 2018-07-26 08:30:13
- Role: Accounting
- Due:
- totalAmount: 679.57
- Status: ~Approved
- flowDept: Factory

Document 0724-4402-5739 in AP_Factory

The document is an "Invoice - Direct Order" from SDSI. It is marked as a "DUPLICATE". The invoice details include:

- Ship To: Grandview Manufacturing, 2342 Edithborough Drive, Colorado Springs, CO 80920
- Consignee To: Wholesale Distribution Company, 5555 Tech Center Drive, Colorado Springs, CO 80919-2209
- Ship From: Grandview Manufacturing, 401 Woodman Drive, Colorado Springs, CO 80918
- Ship Date: 06/12/18
- Ship Time: 2a30

The invoice includes a table of items:

Ln	Product	Qty	Unit	Price	Amount
1	14001 Tap Extension, Size 04 Red	20	each	8.00	160.00
2	14002 Tap Extension, Size 04 Style B, 8"	8	each	8.00	64.00
3	14003 Tap Extension, Size 04 Style B, 8"	25	each	8.00	200.00

The total amount is 679.57.

Flow Selection

When the current user participates in more than one flow definition, the leftmost toolbar icon presents a flow selection list. The list indicates the last known update time, and also document counts. If due dates are specified in the flow documents, counts of color-coded due date categories are provided, green for future, blue for due, red for past due, based on the "Due Date Warn" value of the flow definition.

Synchronization

As the browser interface is designed for offline operation, there is a syncing process with the server any time a document step or data is changed, and also periodically to ensure that the current documents in a flow are up to date with server-based changes. You can set the auto-syncing interval in the Settings page (or turn it off), plus there are two buttons on the toolbar, a manual sync button, and a auto-sync toggle, which can turn auto-syncing on and off.

Sorting

When viewing documents in a grid format, clicking a grid header will sort the documents by that column. In addition, there is a toolbar sort drop down to enable the same sort functionality. Most sort orders also produce sub-headers. For example, the Due Date sort produces daily sub-headers. The My Docs option on the sorting menu turns on/off the suppression of rows not related to flow steps the current user participates in.

Batch Action

When you have two or more documents selected with shift-click and control-click actions, the batch back and forward buttons are enabled. Click these to move all selected documents backward or forward, as if you had chosen that option on each document's Action tab.

Administrator Options

Two options are available when an administrator is viewing a document. They are:

- Delete Document, which removes the document from the flow. It does not affect the original source document in the original library.
- Transfer Document, which moves the document to a different flow. A flow selection window appears, and after approval, the document is transferred. All attachments, data fields, notes, and history are also copied. A new history item is added indicating the transfer. Note that while data fields are copied, only those defined with the same name in the new flow are accessible.

Numeric Totals

The system will automatically calculate totals for number-type fields in each flow. These totals appear in the flow selection window, for the entire batch of documents in the flow, and also in the group headers available in different sort sequences, for the documents in each group.

Exporting

On the settings menu, choose Export List to generate a CSV file of the work flow table, or Export All Flows to generate a CSV file of all records in all flows available to the current user. It will be treated as a download by the browser, and can be opened in your default CSV viewing tool, such as Microsoft Excel, or any text editor. A [Settings](#) option enables the exporting of Notes in the CSV data.

Document Panel

The document panel has several tabbed subpanels. The first one displays the primary image, the original one that started the flow. This is often converted from PDF format to image format to facilitate easier viewing on mobile devices, and also to enable annotations if the flow is designed to enable them. For annotation-enabled flows, the user can draw directly on a copy of the image by clicking the editing button on the toolbar. The toolbar on this panel also provides access to document and step information, as well as zoom options. Note you can double click the image to zoom in and back out again.

Additional tabs are for [attachments](#), data entry and step [actions](#), [notes](#), and an edit [history log](#).

The top level drop down menu provides access to settings, help, logout, and for flow designers or administrators, [roles](#), [flow definitions](#), and a document deletion capability.

10.2.1 Attachments

The attachments panel enables viewing of any or all attachments to the current document. Attachments are additional files that were provided at the time the document entered the flow (named "attachment*"), and also those uploaded by users working with the document (named "upload*"). Any marked images stored in memory from an archive browser session can also be attached. The image can be opened in a standalone browser tab for fuller viewing, and the main and all attachment documents can be viewed side by side.

The screenshot displays the UnForm 10.1 interface. The top bar contains navigation icons and a dropdown menu. The main area is divided into two panels. The left panel, titled 'Active Documents in AP_Factory', shows a list of documents with a filter for '2018-07-24 (1)'. The selected document is '0724-4402-5739', with details: ID: 0724-4402-5739, Started: 2018-07-24 12:13, Step: AcctgClerkReviewPrep, Updated: 2018-07-27 11:14:50, Role: Accounting, Due: (blank), totalAmount: 679.57, Status: ~Approved, and flowDept: Factory. The right panel, titled 'Document 0724-4402-5739 in AP_Factory', contains tabs for Document, Attachments, Action, Notes, and Log. The 'Action' tab is active, showing a 'Back' button, 'Save' button, and 'Forward' button. Below these are form fields: 'totalAmount' (679.57), 'Status' (radio buttons for ~New, ~Pending, ~Hold, ~Disputed, ~Approved (selected), ~Denied), 'ReadyToGo' (checkbox for 'Check if it's ready to go'), 'flowDept' (Factory), and 'Due' (mm/dd/yyyy).

Active Documents in AP_Factory	
2018-07-24 (1)	
ID: 0724-4402-5739	Started: 2018-07-24 12:13
Step: AcctgClerkReviewPrep	Updated: 2018-07-27 11:14:50
Role: Accounting	Due:
totalAmount: 679.57	
Status: ~Approved	
flowDept: Factory	

Document 0724-4402-5739 in AP_Factory	
Document	Attachments
Action	Notes
Log	
< Back	
Save	
> Forward	
totalAmount	679.57
Status	<input type="radio"/> ~New
	<input type="radio"/> ~Pending
	<input type="radio"/> ~Hold
	<input type="radio"/> ~Disputed
	<input checked="" type="radio"/> ~Approved
	<input type="radio"/> ~Denied
ReadyToGo	<input type="checkbox"/> Check if it's ready to go
flowDept	Factory
Due	mm/dd/yyyy

10.2.3 Notes

The Notes tab enables any user, regardless of the role associated with the current step, to add notes. A history of notes is displayed as well.

Active Documents in AP_Factory

<

2018-07-24 (1)

ID: 0724-4402-5739

Started: 2018-07-24 12:13

Step: AcctgClerkReviewPrep

Updated: 2018-07-27 11:24:28

Role: Accounting

Due:

totalAmount: 679.57

Status: ~Approved

flowDept: Factory

Document 0724-4402-5739 in AP_Factory

>

Document

Attachments

Action

Notes

Log

Add Note

Date: 2018-07-27 11:24:28, By admin

Please have John look at the first invoice.

10.2.4 Log

The history tab displays all updates that have been performed on the document while in the flow. Updates by the current user are displayed in bold.

Active Documents in AP_Factory		Document 0724-4402-5739 in AP_Factory																																				
<div>2018-07-24 (1)</div> <div> ID: 0724-4402-5739 Started: 2018-07-24 12:13 Step: AcctgClerkReviewPrep Updated: 2018-07-27 11:24:28 Role: Accounting Due: totalAmount: 679.57 Status: ~Approved flowDept: Factory </div>		<table border="1"> <thead> <tr> <th>Document</th> <th>Attachments</th> <th>Action</th> <th>Notes</th> <th>Log</th> </tr> </thead> <tbody> <tr> <td>When</td> <td>Who</td> <td colspan="3">What</td> </tr> <tr> <td>2018-07-26 08:29:54</td> <td>wf_heyu</td> <td colspan="3">Updated field(s) Status, ReadyToGo</td> </tr> <tr> <td>2018-07-26 08:30:13</td> <td>wf_heyu</td> <td colspan="3">Updated field(s) Status</td> </tr> <tr> <td>2018-07-26 08:30:13</td> <td>wf_heyu</td> <td colspan="3">Moved to step AcctgClerkReviewPrep</td> </tr> <tr> <td>2018-07-27 11:14:50</td> <td>admin</td> <td colspan="3">Uploaded file smp_op_invoice_02.p</td> </tr> <tr> <td>2018-07-27 11:24:28</td> <td>admin</td> <td colspan="3">Updated notes</td> </tr> </tbody> </table>		Document	Attachments	Action	Notes	Log	When	Who	What			2018-07-26 08:29:54	wf_heyu	Updated field(s) Status, ReadyToGo			2018-07-26 08:30:13	wf_heyu	Updated field(s) Status			2018-07-26 08:30:13	wf_heyu	Moved to step AcctgClerkReviewPrep			2018-07-27 11:14:50	admin	Uploaded file smp_op_invoice_02.p			2018-07-27 11:24:28	admin	Updated notes		
Document	Attachments	Action	Notes	Log																																		
When	Who	What																																				
2018-07-26 08:29:54	wf_heyu	Updated field(s) Status, ReadyToGo																																				
2018-07-26 08:30:13	wf_heyu	Updated field(s) Status																																				
2018-07-26 08:30:13	wf_heyu	Moved to step AcctgClerkReviewPrep																																				
2018-07-27 11:14:50	admin	Uploaded file smp_op_invoice_02.p																																				
2018-07-27 11:24:28	admin	Updated notes																																				

10.2.5 Annotations

If a flow has annotations enabled, an annotation edit button is available in the main document display toolbar. This opens the document in a maximized image viewer with a drawing mode. The toolbar enables drawing or panning mode, mark removal and information, restore, zoom, and an ability to view marks made by specific users and times. When drawing mode is turned on, a prompt for user name is presented, and then drawing can proceed. When a drawing session is done, press the Save button on the right to record the annotation marks.


There can be any number of drawing sessions, allowing markup annotation by several users over time.

Edit Image TwoStep:2018-0819:0819-3911-0418

020	20.000	Dip and Nickel Plating	04/02/04	5.0000	100.00
030		Dip and nickel plating			
		Each part completely stripped of all oil			
				Total	800.00

Please Acknowledge Order And Confirm Prices. Our Order Number Must Appear On All Invoices, Packages, Etc.

Please Notify Us Immediately If You Are Unable To Completely Ship This Order By The Date Required.

Purchasing Agent

 Purchasing Agent

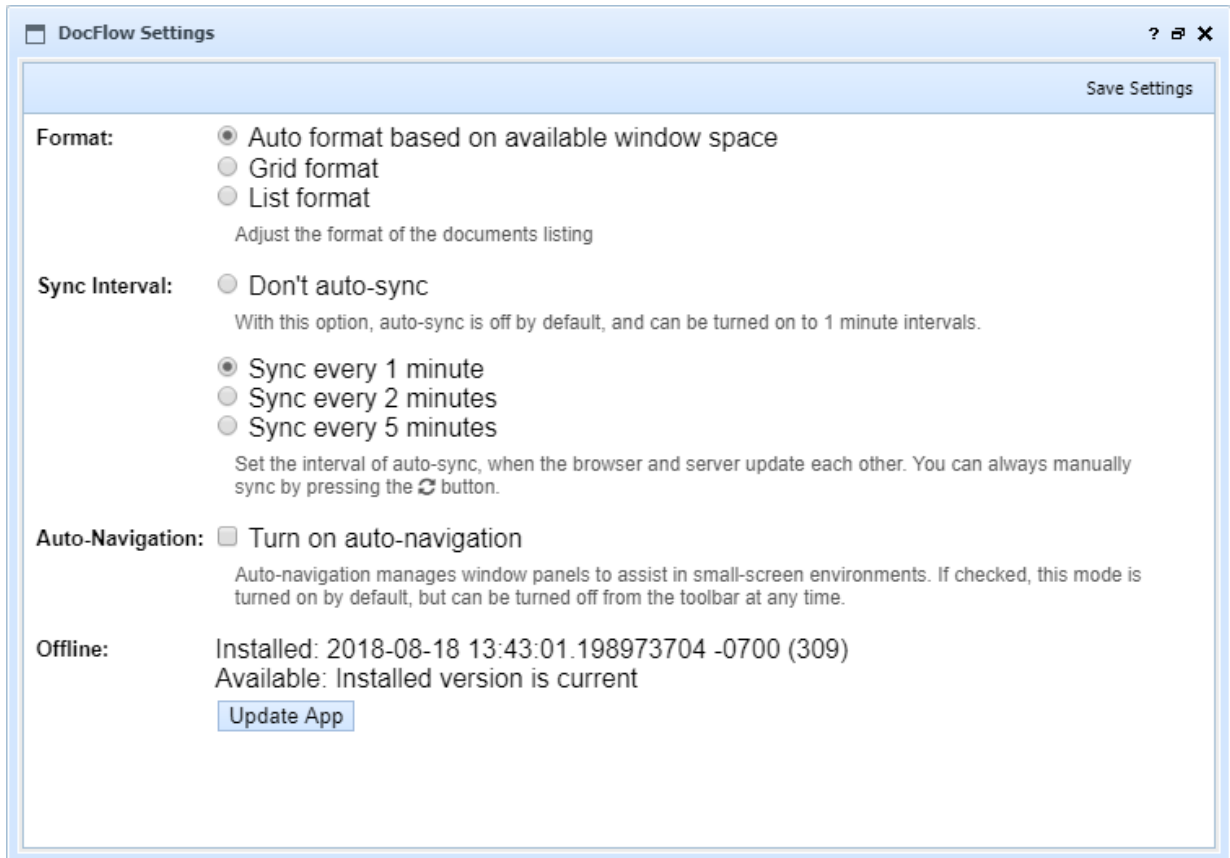
Vendor Copy

Operation Tips

- To quickly zoom into a region, double click/tap where you want to zoom in to. Another double click/tap will zoom out to full width.
- To see specific session marks, click the user drop-down and select the user/time of the annotation session. To view all marks again, click the user button itself.
- To edit specific marks, click the X toolbar button. Each visible mark will have "x" and "i" buttons, for removal and information.
- To restore the annotations to the starting marks, click the restore button.

10.2.6 Settings

The Settings window, accessed from the main toolbar, provides several features related to browser operation.



Format determines how the documents are displayed in the main window. Grid format provides a table with columns and rows. List format displays document data in a more vertical format, suitable for small screens.

Sync Interval determines how often the browser attempts to auto-sync with the server to ensure it's data is up to date. Manual syncing, and turning auto-sync on and off, is available on the main window toolbar.

Auto-Navigation is a feature that improves small screen operation by auto-expanding and closing panels.

Export Notes, if checked, will add a Notes column to the CSV files created by the Export List and Export All Flows options on the main menu.

Offline operation is available when the browser is connected to a secure (https, public certificate) site. The browser can then download files needed for offline operation and store them in browser cache. The Installed field shows the date of the currently installed version. The Available field shows if there is a more recent version available. Click the Update App button to install the available version. Do this only when you have a good connection to the server, so that the entire file set is cached. The download status is displayed as cache is being updated.

10.3 Flow Definitions

Flow definitions describe the steps that a document moves through when it is in a flow, the custom data fields that are captured from users, and any scripting that applies when a document arrives in the flow, is advanced through the steps, and when it is finished and moves out of the flow, updating the original parent document with data and additional attachments. Add, save, or delete flow definitions with the toolbar buttons. Select an existing flow in the left panel, and edit its definition in the right.

The screenshot shows the 'Edit Flows' window with the 'Overview' tab selected. The left sidebar lists a hierarchy of flows, with 'zDemo_SigCap' selected. The main area contains the following fields:

- Flow ID:** zDemo_SigCap
- Description:** Signature capture sample
- Type:** zDemo
- Default Due:** 1
 - ☐ Skip Saturday ☒ Skip Sunday
 - Set a default due date to this many days from the date a document is added to flow. Optionally skip Saturday or Sunday in this calculation.
- Due Date Warn:** 0
 - Due dates within this number of days are highlighted when flow lists are sorted by due date. Past due dates are always highlighted. Set to 0 to only highlight past due values.
- Enable Annotations?** ☐
 - Enable annotations (drawing) on primary image.
- Annotate SubId:** signed
 - If annotations are enabled, use this subid when creating an annotated version of the primary document. It is always auto-sequenced. If not provided, the default subid 'annotated*' is used.
- Enable Stamps?** ☐
 - Enable configured stamps to be placed on primary image.
- Enable Signature Pad** ☒ Position: 5.5,8,3,.75
 - Enable signature pad on primary image. Enter the position of the signature on the signed document in inches as left, top, width, height, such as '5.5,10,3,.75'.
- Auto Links:** ☐

Flows are identified by an alpha-numeric ID, up to 20 characters. Other fields on the main definition tab are:

- Description is a free text field.
- Type is used to categorize flows in the flow selection tree.
- Due Date Warn is a threshold of days considered non-urgent when sorting documents in the flow by due date. Due dates beyond this number of days are considered non-urgent, those due from today to this many days in the future are considered urgent. Those past due are given a third level of color coded priority.
- Enable Annotations, if checked, enables an annotated version of the main document, on which users can draw notes and signatures. This can be used to capture signatures and other information in a proof of delivery application.
- Enable Stamps, if checked, will enable users to add [stamp images](#) to the primary image of a docflow document. Stamps are configured as images in the `df/stamps` directory. If checked, a list of available stamps is presented for selection. Choose any number of stamps and those will be available for placement on documents in the flow.
- Enable Signature Pad, if checked, will enable the signature pad interface when editing a document image, similar to the annotation interface. A signature is captured as a JPEG file and stored as an attachment for automatic placement on the [annotated image](#). Use the Position value to specify where the signature should be placed on the annotated image.

Position is a comma-separated list of *left, top, width, height, page*. Specify in inches. The page number

can be set artificially high to print on the last page. Defaults are .25, 10, 3, 1, and 1.

- Annotate Subid is the image subid used to store the annotated version of the main image when the flow is complete and the flow data is written back to the library parent document of the main flow image. This feature is active when annotations, stamps, or the signature pad are enabled. See [Annotated SubID](#) for more details.
- Auto Links, if checked, will cause the system to look at links in the source document, and add any new image-level links (lib|doctype|docid|subid) as attachments whenever the flow document is accessed. Links added to the source document after the flow document is created are attached automatically.
- Auto Subids, if checked, will cause the system to look at non-text subids in the source document, and will add any new such subids as attachments whenever the flow document is accessed. SubIDs added to the source document after the flow document is created are automatically attached.
- Max Image Size sets the maximum width or height of attached JPEG images. When a DocFlow user attaches an image using a phone or tablet camera, the image can be very high resolution and require large amounts of bandwidth and storage space to manage the image. Setting this value will scale JPEG image down as needed when they are attached.
- Existing flow definitions can be exported to a file, downloaded at the browser. When adding a new definition, you can import previously exported file to create a new flow definition on any system.

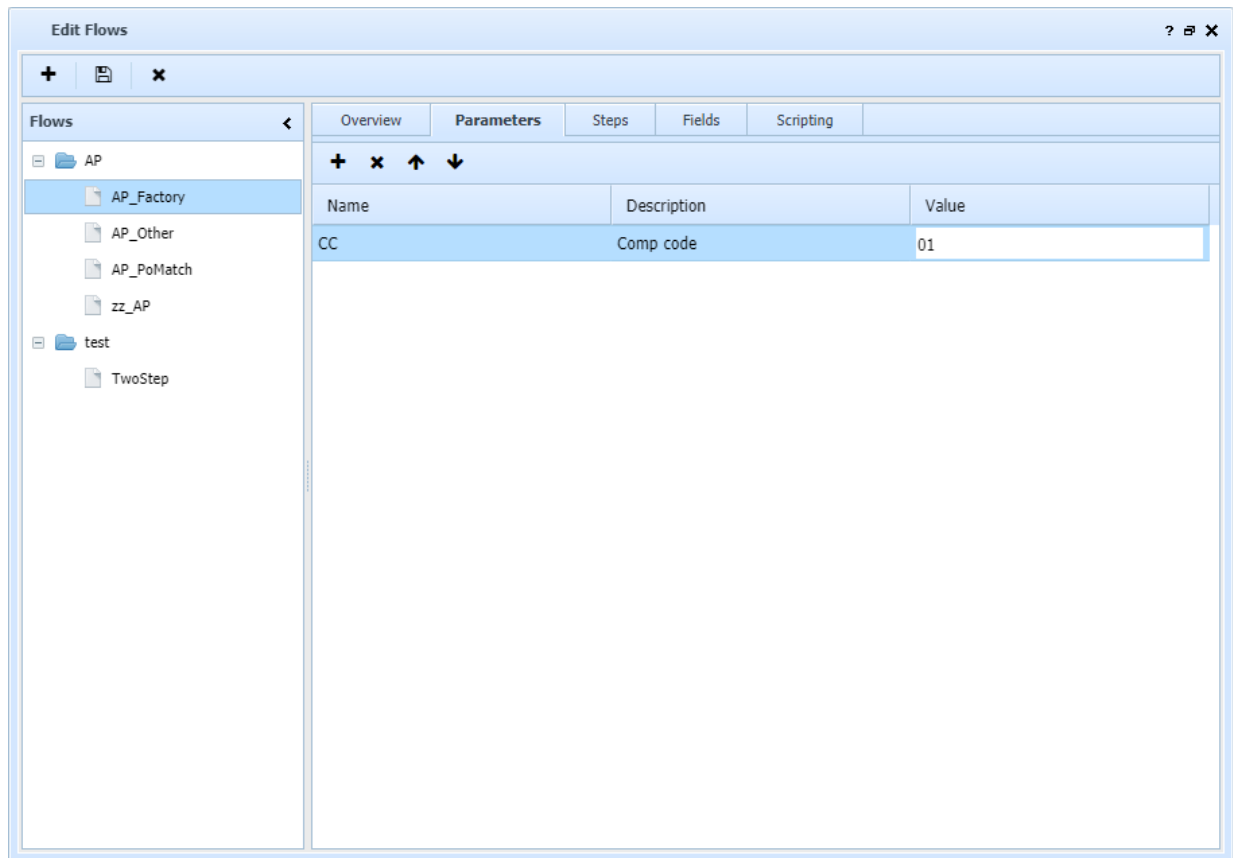
Each flow has an associated docflow library where documents and their properties are maintained. These libraries are not normally accessible in the regular browser interface, but an administrator can view them under the ~docflow library category.

Note that unlike standard document libraries, the system will purge library recovery data for dates prior to any existing documents in the docflow libraries. This is due to the intentional transient nature of docflow documents.

10.3.1 Parameters

Flow parameters are simple named text values that can be used elsewhere in the flow definition, particularly in scripting, to allow standardized routines to use editable rather than hard-coded values.

Parameters are automatically assigned values and inserted into script code routines, and also into validation objects exposed by those routines. The parameters take the form of string variables, named param.*name*\$.



10.3.2 Steps

Steps define what possible states a document can be in when it is in the flow. Each step is assigned a responsible role, and each can have notes or help that is visible when the document is at the step. Steps can also perform automatic notification of users in the responsible role, whenever a document arrives at that step. The server will perform this notification using the [notify](#) object, passing it the following tag values (while all tags are passed to the notify object, only the url\$ tag is required):

- url\$ is a hyperlink to the flow document
- fromstep\$ is the step before the current step was applied
- tostep\$ or step\$ is the current step whose role is to be notified
- duedate\$ is the due date of the document
- started\$ is the start date of the document
- updated\$ is the last updated date of the document
- title\$ is the document title
- library\$ is the library of the source document that created created the flow document
- doctype\$ is the doctype of the source document
- docid\$ is the docid of the source document
- subid\$ is the subid of the source document

Normally users can move a document forward and backward through the steps as designed in this table, but scripting can control what step a document moves to in based on any logic required.

Edit Flows

Flows

- AP
 - AP_Factory
 - AP_Other
 - AP_PoMatch
 - zz_AP
- test
 - TwoStep

Overview Parameters **Steps** Fields Scripting

Name	Description	Responsible Role
FactoryExpApprove		Factory
AcctgClerkReviewPrep		Accounting
ControllerApprove		Controller
DeniedRejected		Accounting
Complete		Accounting

Step Options

Step Name:

Description:

Responsible Role:

Help/Notes:

10.3.3 Fields

Flow fields describe data to be captured from users when a document is in the flow. Fields are displayed on the Action tab when viewing a flow document. In most cases, fields involve user entry, but there are also message fields and link (url) fields. Link fields can be helpful in opening related documents in a browser-based ERP application.

Edit Flows

Flows

- AP
 - AP_Factory
 - AP_Other
 - AP_PoMatch
 - zz_AP
- test
 - TwoStep

Overview Parameters Steps **Fields** Scripting

Name	Description	Type	Value	Validation	GridCol?
totalAmount	totalAmount	number			<input checked="" type="checkbox"/>
Status	Status	radio=~New;~Pendi			<input checked="" type="checkbox"/>
ReadyToGo	ReadyToGo	checkbox=Check if it			<input type="checkbox"/>
flowDept		text			<input checked="" type="checkbox"/>

Field Options

Field Expressions Parameter Expressions

Name:

Description:

Type:

Options:

Enter two or more items, separated by semicolons. Each item becomes a radio button caption. The user checks one button, which becomes the value of the field.

Example: Due on receipt;Net 15; Net 30

Value:

☐ Value is an expression?

The following options are available:

- Name is an alphanumeric name for the field. It is used to create variables, so it must start with a letter and only contain letters, digits, and underscores.
- Description is a free text field
- Type defines what type of data this field stores or displays. The types include:
 - Short text or long text, for single-line text entry
 - Multi-line text
 - Number for number entry
 - Date for date input (should be set to yyyy-mm-dd format)
 - Email for email input
 - Checkbox to enable on/off selection
 - Radio buttons or selection lists to allow one-of-many selection
 - Lookups to enable operator lookup based on scripted lookup definitions
 - Hyperlink to open a URL in a new tab or window, the URL value generally defined by script code
 - Grid to enable grid data entry, the initial contents generally defined by script code.
 - Label to display a subheader in the entry table
 - Message to display a textual message in the entry table
 - Hidden to store a value that is not visible to the user
- Options are required by some field types, such as radio buttons as shown above.

- Value is used to set an initial value, either hard-coded or by expression. The menu drop downs for field or parameter values can be used to generate expressions.
- Validation is a validation routine defined in the Image Manager tool, used to perform validation on the data provided for this field. A document cannot be finalized if any validation tests fail. Fields that fail validation are highlighted in the Action tab of the browser interface.
- Grid Column, if checked, adds this data field to the flow's document listing as a grid column or extra list field, so it can be used to sort the list.

10.3.4 Scripting

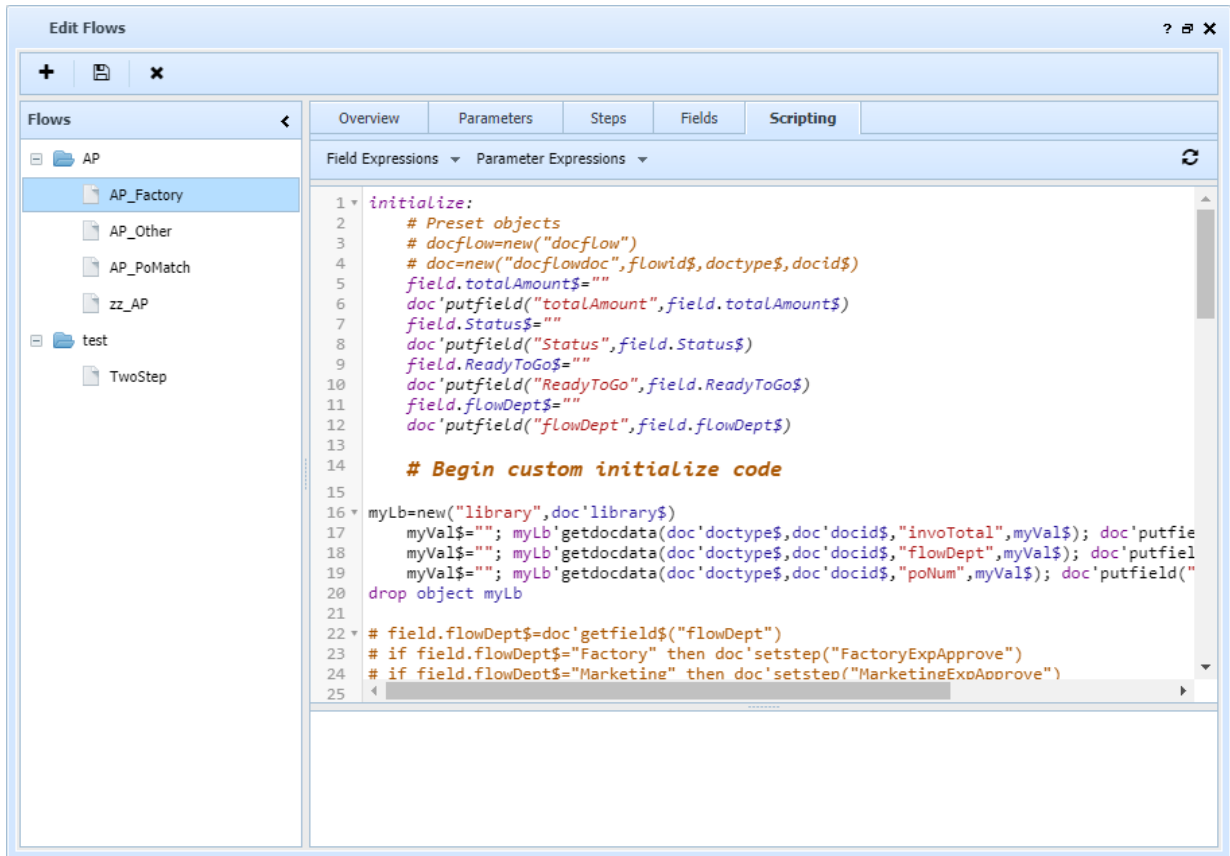
Scripting enables customized actions to be performed as documents enter, move through, and leave the flow. There are three sections: initialize, advance, and finalize. Each of these is pre-populated with parameter and field definitions, and each provides a custom section where additional code can be entered.

- Initialize code can be used to set initial field values, and conditionally set the initial step for the document. It can also be used to send notifications to role users.
- Advance code is executed whenever the user presses the Next button in the browser interface, normally to change the current step of the document. This code can set `errmsg$` to prevent the advance from changing the step. If the code sets a new step value, that step is honored. Otherwise the step is automatically set to the next step. This code can also send notifications to roles.
- Finalize code is executed when the document's current step is the last step, and the user presses the Finish button. After running the Advance code without `errmsg$` set, and without a different step being set by that code, the flow history, field data, and added attachments are updated in the source library document, and this code is run.

In the custom code sections, you can enter any valid UnForm scripting code, which is based on the PxPlus Basic language. The code is identical to that used to [program code blocks](#), except that block if statements are not restricted from using curly braces (see Block If, below).

Scripting makes heavy use of objects, as several are exposed automatically while the script runs, used by the auto-generated code and available for custom code, the most important of which is "doc", a [docflowdoc](#) object representing the current document in the flow, but also [docflow](#), [filter](#), [lookups](#), [validation](#), and [notify](#) object variables ("filters" is an alias for "filter", "validations" is an alias for "validation").

Also, in the finalize routine, a subdoc property template is provided, as returned by the [library](#) object's `getsubdoc()` method, for the newly created XML document generated automatically when the document is updated back into the source archive library. You can use this template to access the XML file, or to obtain document- or sub-id values for further processing with a library object.



Block If

This is valid in custom scripting:

```

if a=b then {
  # code if a=b
} else {
  # code if a<>b
}

```

This is valid in both custom scripting and code blocks:

```

if a=b then
  # code if a=b
else
  # code if a<>b
end if

```

10.4 Roles and Users

Roles are used to define groups of users who can edit a document when it reaches specific steps in a flow. Each step of a flow is assigned to a single role. Each role can have one or more users who are members and participate in editing the document and its data. Any user selected for the role can also

have a Notify setting. The [notify object's](#) emailrole() method honors these settings, only sending notifications to those members with Notify checked.

Existing Roles		Edit Role																																																																			
Add Role Delete <table border="1"> <thead> <tr> <th>Role ID</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>Accounting</td> <td>wf_cole, wf_diaz Supervisory</td> </tr> <tr> <td>ActgClerk1</td> <td>wf_manu</td> </tr> <tr> <td>App1</td> <td>Approval level 1</td> </tr> <tr> <td>App2</td> <td>Approval level 2</td> </tr> <tr> <td>Controller</td> <td>wf_orca</td> </tr> <tr> <td>Factory</td> <td></td> </tr> <tr> <td>Marketing</td> <td></td> </tr> <tr> <td>OfficeManager</td> <td>wf_lee, Approves for OTHER expense</td> </tr> </tbody> </table>		Role ID	Description	Accounting	wf_cole, wf_diaz Supervisory	ActgClerk1	wf_manu	App1	Approval level 1	App2	Approval level 2	Controller	wf_orca	Factory		Marketing		OfficeManager	wf_lee, Approves for OTHER expense	Save Role ID: <input type="text" value="Accounting"/> Description: <input type="text" value="wf_cole, wf_diaz Supervisory"/> Members: <table border="1"> <thead> <tr> <th>Member</th> <th>Notify</th> <th>Name</th> </tr> </thead> <tbody> <tr><td><input checked="" type="checkbox"/></td><td><input checked="" type="checkbox"/></td><td>admin - Administrator</td></tr> <tr><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td>bcj - Clark</td></tr> <tr><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td>collins - Collins (WF)</td></tr> <tr><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td>diaz - diaz</td></tr> <tr><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td>lee - lee</td></tr> <tr><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td>manooover - manooover</td></tr> <tr><td><input checked="" type="checkbox"/></td><td><input type="checkbox"/></td><td>mje - Michael Brumer</td></tr> <tr><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td>pao - Patricia re tickets WF</td></tr> <tr><td><input checked="" type="checkbox"/></td><td><input type="checkbox"/></td><td>wf_cole - Cole</td></tr> <tr><td><input checked="" type="checkbox"/></td><td><input checked="" type="checkbox"/></td><td>wf_diaz - Diaz</td></tr> <tr><td><input checked="" type="checkbox"/></td><td><input type="checkbox"/></td><td>wf_hank - Hank</td></tr> <tr><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td>wf_heyu - Heyu</td></tr> <tr><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td>wf_lee - Lee</td></tr> <tr><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td>wf_manu - Manu</td></tr> <tr><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td>wf_orca - Orca</td></tr> </tbody> </table>		Member	Notify	Name	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	admin - Administrator	<input type="checkbox"/>	<input type="checkbox"/>	bcj - Clark	<input type="checkbox"/>	<input type="checkbox"/>	collins - Collins (WF)	<input type="checkbox"/>	<input type="checkbox"/>	diaz - diaz	<input type="checkbox"/>	<input type="checkbox"/>	lee - lee	<input type="checkbox"/>	<input type="checkbox"/>	manooover - manooover	<input checked="" type="checkbox"/>	<input type="checkbox"/>	mje - Michael Brumer	<input type="checkbox"/>	<input type="checkbox"/>	pao - Patricia re tickets WF	<input checked="" type="checkbox"/>	<input type="checkbox"/>	wf_cole - Cole	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	wf_diaz - Diaz	<input checked="" type="checkbox"/>	<input type="checkbox"/>	wf_hank - Hank	<input type="checkbox"/>	<input type="checkbox"/>	wf_heyu - Heyu	<input type="checkbox"/>	<input type="checkbox"/>	wf_lee - Lee	<input type="checkbox"/>	<input type="checkbox"/>	wf_manu - Manu	<input type="checkbox"/>	<input type="checkbox"/>	wf_orca - Orca
Role ID	Description																																																																				
Accounting	wf_cole, wf_diaz Supervisory																																																																				
ActgClerk1	wf_manu																																																																				
App1	Approval level 1																																																																				
App2	Approval level 2																																																																				
Controller	wf_orca																																																																				
Factory																																																																					
Marketing																																																																					
OfficeManager	wf_lee, Approves for OTHER expense																																																																				
Member	Notify	Name																																																																			
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	admin - Administrator																																																																			
<input type="checkbox"/>	<input type="checkbox"/>	bcj - Clark																																																																			
<input type="checkbox"/>	<input type="checkbox"/>	collins - Collins (WF)																																																																			
<input type="checkbox"/>	<input type="checkbox"/>	diaz - diaz																																																																			
<input type="checkbox"/>	<input type="checkbox"/>	lee - lee																																																																			
<input type="checkbox"/>	<input type="checkbox"/>	manooover - manooover																																																																			
<input checked="" type="checkbox"/>	<input type="checkbox"/>	mje - Michael Brumer																																																																			
<input type="checkbox"/>	<input type="checkbox"/>	pao - Patricia re tickets WF																																																																			
<input checked="" type="checkbox"/>	<input type="checkbox"/>	wf_cole - Cole																																																																			
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	wf_diaz - Diaz																																																																			
<input checked="" type="checkbox"/>	<input type="checkbox"/>	wf_hank - Hank																																																																			
<input type="checkbox"/>	<input type="checkbox"/>	wf_heyu - Heyu																																																																			
<input type="checkbox"/>	<input type="checkbox"/>	wf_lee - Lee																																																																			
<input type="checkbox"/>	<input type="checkbox"/>	wf_manu - Manu																																																																			
<input type="checkbox"/>	<input type="checkbox"/>	wf_orca - Orca																																																																			

10.5 XML Document

Below is an annotated example of the XML document produced when a document is finalized and leaves a flow. This document has a subid based on the sequenced flow ID (i.e. APApproval*). When a flow is finalized, the "finalize" script code has a template variable subdoc\$ that has property information for this XML document, so you can retrieve the actual XML with a library object 'getimage()' method.

The root node is /docflow, with a flowid attribute of the flow ID name.

```
<docflow flowid="TwoStep">
```

The /docflow/primary node contains the identification of the source image file that originally started the flow.

```
<primary>
<library>/u0/unform100/arc/demo_sales</library>
<doctype>Order</doctype>
<docid>0001139</docid>
<subid>@unform</subid>
</primary>
```

The /docflow/started and /docflow/completed nodes contain date/time information in yyyy-mm-dd hh:mm format (24-hour clock).

```
<started>2018-07-25 07:32</started>
<completed>2018-07-25 00:18</completed>
```


The `/docflow/fields` node contains subnodes for each field defined in the flow. A type attribute defines the type of field. Grid types include are further defined with `/rows`, `/rows/row`, and `/rows/row/col` tags.

```
<fields>
<InitialReview type="longtext"/>
<ReadyToGo type="checkbox">1</ReadyToGo>
<CustomerID type="lookup"/>
<gridtest type="grid">
<rows>
<row>
<col name="one">1</col>
<col name="two">Row 1</col>
</row>
<row>
<col name="one">2</col>
<col name="two">Row 2</col>
</row>
</rows>
</gridtest>
<toGoogle type="link">https://google.com?asdf</toGoogle>
<message type="message">Some message text for the user.</message>
</fields>
```

The `/docflow/notes` node contains one or more note tags, each with date and who attributes, and with the value of the note entered by the user.

```
<notes>
<note date="20180728-073208" who="admin">note 1 </note>
<note date="20180728-073213" who="admin">note 2 </note>
</notes>
```

The `/docflow/updatehist` node contains a series of `/docflow/updatehist/update` nodes, each with `/docflow/updatehist/update/when`, `../who`, and `../what` tags describing what edit took place.

```
<updatehist>
<update>
<when>2018-07-25 07:33:04</when>
<who>admin</who>
<what>Annotation added or deleted</what>
</update>
<update>
<when>2018-07-25 07:33:13</when>
<who>admin</who>
<what>Updated field(s) ReadyToGo</what>
</update>
<update>
<when>2018-07-25 07:33:13</when>
<who>admin</who>
<what>Moved to step FinalReview</what>
</update>
<update>
<when>2018-07-25 07:33:15</when>
<who>admin</who>
<what>Updated field(s) ReadyToGo</what>
</update>
<update>
<when>2018-07-25 07:33:16</when>
<who>admin</who>
<what>Flow completed</what>
</update>
</updatehist>
```

The `/docflow/attachments` node contains two types of nodes. One is a series of `/docflow/attachments/file` nodes with a `subid` attribute, indicating what subid in the original parent document contains the image of the attachment. The value of each file node is the file title. The second is a series of `/docflow/attachments/link` nodes, with values of the library, doctype, docid, and subid of the linked image, separated by pipes.

```
<attachments>
<file subid="upload-00001">Invoice 0005199.pdf</file>
<link>/u0/unform100/arc/demo_sales|Order|0001134|TwoStep</link>
</attachments>
```

The `/docflow/annotation` node contains two types of subnodes. One `/docflow/annotation/subid` has the annotation image subid as its value. You can use this to retrieve the annotated image using the library object. In addition, there are multiple `/docflow/annotation/mark` nodes, each with a `seq` attribute, and `datetime`, `username`, and `location` subnodes. The `location` subnode has attributes for `latitude` and `longitude` if the browser provided this information (this feature requires user permission and a secure connection to the UnForm server using a public SSL certificate). Each mark node represents one drawn element on the annotation. The annotated version of the image (generally a PDF file) has these marks drawn and footnoted, so you can correlate the mark `seq` attributes with each mark on the PDF. The PDF files also have popup legends for these marks. Annotation marks are captured in sets, so the `datetime` value can be used as a reference for a set of marks drawn by one person in one annotation session. The `location` attributes will contain `latitude` and `longitude` if the drawing device (phone, tablet, etc.) allows it.

```
<annotation>
<subid>twostep-signed</subid>
<mark seq="1">
<datetime>2018-07-25 07:32:56</datetime>
<username>Administrator</username>
<location longitude="0" latitude="0"/>
</mark>
<mark seq="2">
<datetime>2018-07-25 07:32:58</datetime>
<username>Administrator</username>
<location longitude="0" latitude="0"/>
</mark>
<mark seq="3">
<datetime>2018-07-25 07:33:01</datetime>
<username>Administrator</username>
<location longitude="0" latitude="0"/>
</mark>
<mark seq="4">
<datetime>1969-12-31 17:00:00</datetime>
<username/>
<location longitude="0" latitude="0"/>
</mark>
</annotation>
</docflow>
```

10.6 Annotated SubID

When a flow with annotations or signature pad enabled is finished, part of the update process creates a PDF document with the annotated image. When this PDF file is generated, it includes footnotes identifying the annotation marks on the image, and a PDF popup with footnote definitions. This document is generated by a standard rule set "annotate" in the `df/docflow.rul` rule file.

However, if a different format for this document is required, it is possible to generate it using a custom rule set named `[annotate-flowid]` in a custom rule file called `docflow.custom.rul`. You can copy the default rule

set to benefit from the standard functionality, but add your own, such as color coded marks, or legends in a blank region of the page, or even adding a cover page with such information.

If the signature pad option is enabled, the signature image is placed on the document using the position and size specified in the flow's signature pad Position entry. This entry is the left, top, width, height, and page values specified as a comma-separated list with each value interpreted as inches. The page number can be set artificially high to place the signature on the last page. The image will be one of the uploaded attachments whose title value is "df-signature.jpg".

10.7 Stamps

Stamps are images that can be placed on a docflow document by users who are editing the document through the document's lifetime in the flow. A library of stamps is created by adding JPG or PNG image files to the df/stamps directory. These image should be sized as they are to be used, typically large enough to be obvious on a page image, but small enough to be placed without obscuring other information.

Any flow definition can enable stamps, and once enabled, the flow editor can select which stamps should be available for that flow.

On the main image of documents in that flow, users can select a stamp and move it into an appropriate position and page. Any number of stamps can be added to the image, and existing stamps can be removed by clicking the stamp's upper-right corner 'X'. Additions and deletions of stamps are logged in the flow's log.

Two sample images, approved.png and denied.jpg, are provided in the df directory, and can be copied to df/stamps to enable testing. The user should supply any other stamp images as needed.

11 SERVER MANAGER

The server manager is a browser-based tool accessed by adding ?sm=1 to the standard UnForm web server URL. For example:

<http://192.168.1.10:27402/arc?sm=1>

An administrator login is required. Once logged in, many server administrative tools are provided, including a job history table, an active connections table, a log viewer and analyzer, a configuration tool, scheduled jobs editor, and a server restart option.

The server manager window presents a toolbar menu many options to specific management tasks. These task windows are presented in full width frames below the menu. The task options presented include:

- [Jobs](#) - displays a job history table and offers filtering options
- [Connections](#) - displays active connections
- [Logs](#) - displays server logs and offers filtering and analysis options
- [Configuration](#) - maintains the server configuration file (uf101d.ini) through several forms
- [Scheduler](#) - maintains scheduled jobs, which are UnForm jobs run at specific intervals by the server
- [Restart Server](#) - provides a convenient tool to restart the server if the configuration changes

11.1 Jobs

This tab displays job history. The UnForm server maintains several days of job history (by default, seven days), showing job page counts, rule sets, status messages, and more. The toolbar provides filtering options and a refresh button. You can filter on status, ruleset, client IP address, or date, or clear the filter to return to a full listing.

In monitoring mode, the last 100 jobs are refreshed automatically, and displayed in top-down order, with the most recent jobs displayed first. When monitoring mode is turned off, you can scroll through the entire available job history, with the oldest jobs displayed first. In filtered mode, all records returned by the filter are displayed in browser memory.

Job history is normally maintained for 7 days, but this can be configured with the `age=days` setting in `uf101d.ini`.

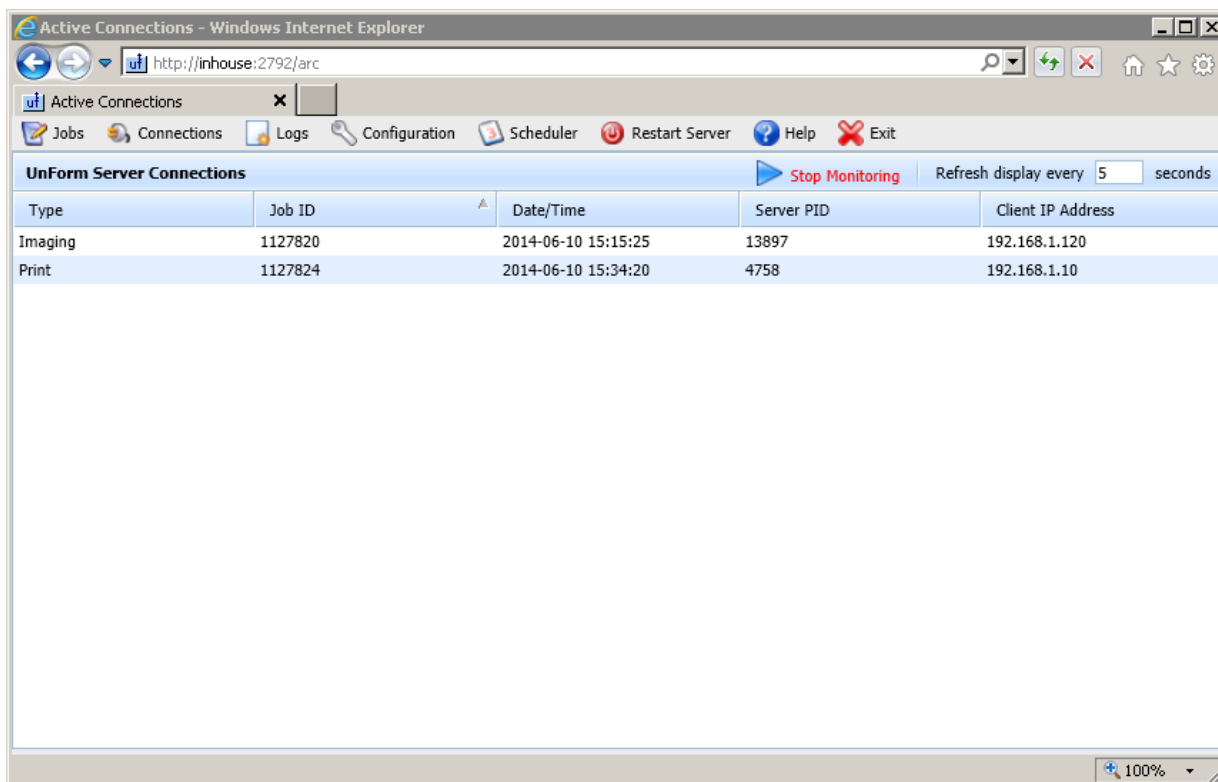
The status column can hold several values:

- 0 - initialized by client connection
- 1 - started
- 9 - complete
- E - error occurred, see the Message column for details

Id	Date	Time	Client Name	Client IP Address	Client User	Input Size	Input Read	Page	Status	Ruleset	Message
1127824	2014-06-10	15:34:20	inhouse	192.168.1.10	allenm	20505	0	0	1		
1127823	2014-06-10	15:16:21	inhouse	192.168.1.10	allenm	20505	0	0	E		
1127822	2014-06-10	15:16:17	inhouse	192.168.1.10	allenm	20505	0	0	E		
1127821	2014-06-10	15:16:13	inhouse	192.168.1.10	allenm	20505	0	1	9	test	
1127819	2014-06-10	14:45:22	lachamba.l	192.168.1.180	daryl	1463	1463	1	9	simple1	pdf
1127818	2014-06-09	16:47:49	inhouse	192.168.1.10	apache	291274	7606	1	9	FrmSOR_OrdAck	laser
1127817	2014-06-09	16:44:00	inhouse	192.168.1.10	apache	283695	4213	1	9	FrmSOR_OrdAck	pdf
1127816	2014-06-09	16:43:56	inhouse	192.168.1.10	apache	291274	7606	1	9	FrmSOR_OrdAck	pdf
1127815	2014-06-09	16:43:53	inhouse	192.168.1.10	apache	436791	763	3	9	N/A	pdf
1127814	2014-06-09	16:43:51	inhouse	192.168.1.10	apache	337677	3915	1	9	N/A	pdf
1127813	2014-06-09	16:43:17	inhouse	192.168.1.10	apache	283695	4213	1	9	FrmSOR_OrdAck	pdf
1127812	2014-06-09	16:43:15	inhouse	192.168.1.10	apache	291274	7606	1	9	FrmSOR_OrdAck	pdf
1127811	2014-06-09	16:43:13	inhouse	192.168.1.10	apache	436791	763	3	9	N/A	pdf
1127810	2014-06-09	16:43:11	inhouse	192.168.1.10	apache	337677	3915	1	9	N/A	pdf
1127809	2014-06-09	16:38:27	inhouse	192.168.1.10	apache	337677	3915	1	9	N/A	pdf
1127808	2014-06-09	16:37:47	inhouse	192.168.1.10	apache	337677	3915	1	9	N/A	pdf
1127807	2014-06-09	16:37:14	inhouse	192.168.1.10	apache	337677	3915	1	9	N/A	pdf
1127806	2014-06-09	16:36:35	inhouse	192.168.1.10	apache	337677	3915	1	9	N/A	pdf

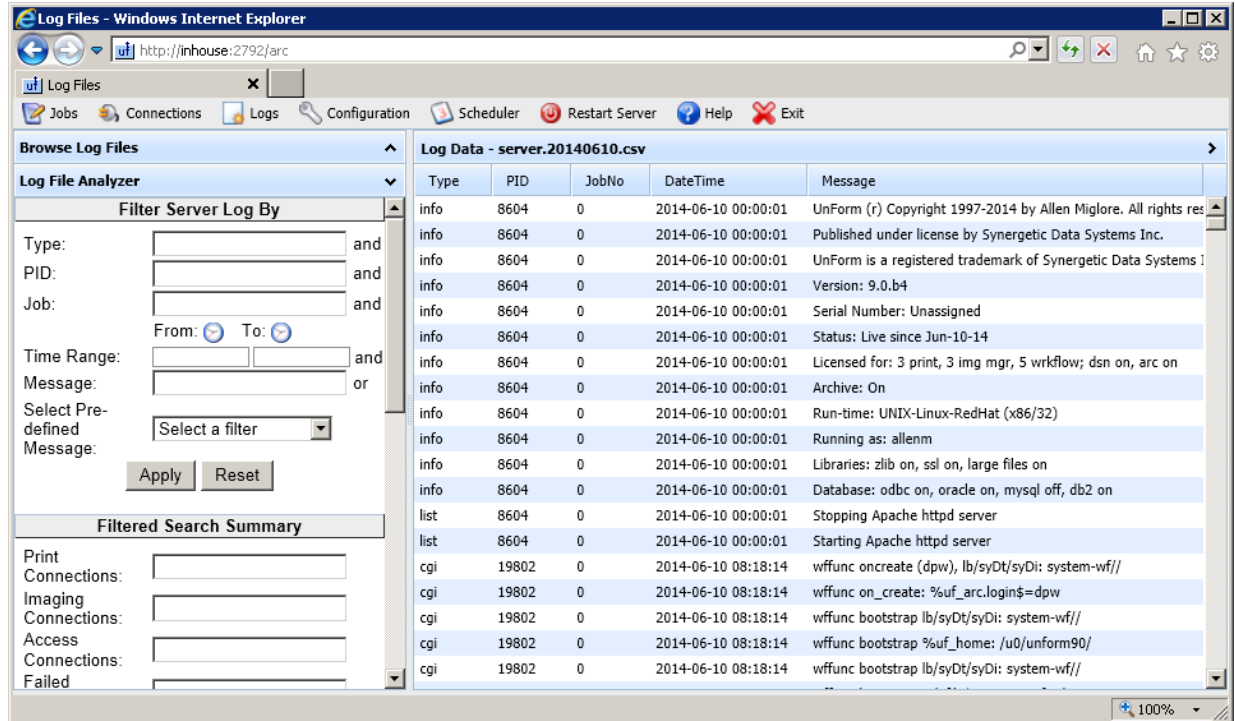
11.2 Connections

The connections tab provides a snapshot view of active connections to the server. A toolbar is provided to control the auto-refresh rate.



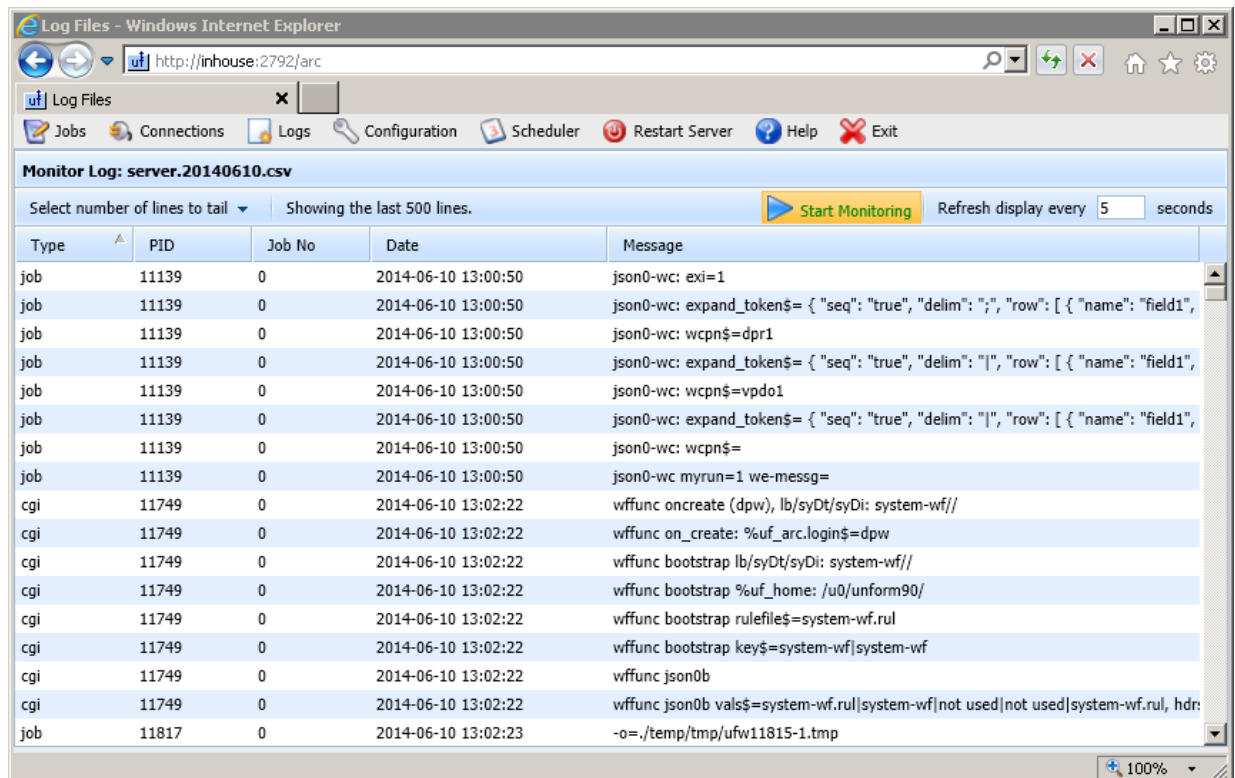
11.3 Log Browser

The logs tab provides extensive log viewing and analysis functionality. There are three panels: a file selector, a log viewer, and an analyzer. Once a file is selected, its lines are displayed in the viewer panel, and the analyzer panel opens. The analyzer tab offers several pre-defined filter options, plus user-specified filtering by various log fields. As filters are applied, the log view changes to show the filtered data, and analysis data is shown in tables in the analyzer panel.



11.4 Log Monitor

The monitor option provides an active view of the end of the current server log. The log file is polled for changes automatically, and the latest lines are displayed. A toolbar provides selectors for the number of lines to view, and polling frequency.



11.5 Configuration

The Configuration menu option provides a tabbed interface to many standard configuration elements, stored on the server primarily in `uf101d.ini`, with some email-specific elements in the `prog/mailcall.ini` file and `deliver.ini`. These are described in the [Configuration](#) chapter of the manual. Note you can also manually edit either of these files on the server.

Some items require that the UnForm server be restarted to take effect. These items have an asterisk (*) prefix.

When changes are complete, click Save. The server will make a backup of `uf101d.ini` (as `uf101d.ini.bak`), and save the new information.

Main

- Primary Server - the port the server listens on for client connections
- HTTPD - the port the private Apache HTTP server listens on, for browser and REST connections
- SSL options for both the Primary and HTTPD servers.
- Certificate, chain, and private key paths used when SSL is configured for either server. If not configured, a self-signed certificate is used. Note the uniform server/service user must have read access to these files.
- Directories - a list of directories searched for rule files and supporting files when print jobs run
- Purging - days the specified types of files and records are left on disk before automatically purged
- Rebuild - if checked, when the server is started it will attempt clean up of control files to eliminate many types of corruption
- CR Behavior - default handling of CR characters in text stream print job input
- Mail Server - default SMTP or SMTPS mail server
- Default From Address - default From address when emails are sent
- Default Login: SMTP authentication login
- Default Password: SMTP authentication password

Security

- Allowed IP's - IP addresses or wildcards that are allowed to connect to the server's main port
- Rule File Encryption - if checked, the design tool will save/publish rule files in encrypted format. The UnForm server can use such rule files like any other, and the design tool can view/edit them, but a standard text editor cannot

Logs

- Log Directory - the location on the UnForm server for log files
- Purge Age - the number of days server and http log files are retained
- Server Detail Level - the logging detail level for the server
- Job Detail Level - the logging detail level for print jobs
- Deliver Age - the number of days deliver and Image Manager upload logs are retained

Archiving

- Library Name - default library name used when a library function is used with an empty library name
- Access - when a process creates a new library, these are the default security options applied
- Auto Sequence - when a process creates a new library, this auto-sequence setting is applied. Auto-sequencing forces all duplicate images (sub ID's) to add a unique sequence value, preventing overwriting of any documents in that library.
- Session Lifespan - the number of hours browser sessions last. If 0, sessions terminate when the browser is closed by the user.
- Self-Manage - allow users to change their own passwords, and also to get an email of a forgotten password.
- Suppress @text - If checked, new browser sessions will by default not display @text images.

- Default Mail From - The default From address for browser interface emails, in cases where the logged in user does not have an email address configured.
- External URL - a URL prefix used when a link is built by the webapi object or the rac object. This value should point to web server address/port that forwards into the UnForm HTTP server "/arc" path. This is typically configured by an administrator as a reverse proxy address in an external web server, or as a port forward from the public address of a router.

External Tools

- Support Server - enter the hostname or IP address, and port, of the default Windows Support Server.
- Ghostscript - the full path to the Ghostscript executable, typically "gs" on Linux/Unix, and gswin32.exe or gswin64.exe on Windows. Use the latest version of GhostScript available. Note that [AFO](#) requires Ghostscript 9.06 or higher. If this is left blank, and a Windows Support Server is configured, UnForm will attempt to use a GhostScript configured in the Support Server, allowing a Windows version to be installed and used.
- Image Magick - the full path the the Magick "convert" or "convert.exe" program on the UnForm server. If this is left blank, and a Windows Support Server is configured, UnForm will attempt to use the Windows Support Server for image scaling and conversion.
- GhostPCL - the full path to the GhostPCL executable, if available. GhostPCL is used by the design tool for previewing PCL test prints on screen.
- Extension Printers - a list of name=*settings* lines. The list is read by the [UnForm Web Extension](#) and the [SDSI Web Extension](#), a tool that provides an enhanced PDF viewer in a compatible web browser. The names are presented in the extension, which can then submit that PDF file as an [UnForm AFO job](#).

TCP Ports

Enter the port configuration for network print capture. More details on this feature are in the [TCP/IP Monitor](#) chapter.

Email Sending with Modern (OAuth2) Authentication

UnForm supports XAUTH2 authentication with a SMTP server, as well as traditional LOGIN authentication. This requires configuring a dedicated [user record](#) with its Notes field filled in with text lines provided by the publisher through a registration procedure at <https://unform.com/support/oauthcode.html>. The user record can be disabled to prevent its use other than as a secure storage location for the authentication data.

After registering an application with your mail provider, such as the App Registration pages at Microsoft 365's Azure Active Directory, you can visit the above web page to configure, authorize, and receive the tokens required for UnForm to perform this type of authentication. Instructions are found at that page.

Once a user has been configured with the proper lines in its Notes field, you can specify which user contains the token information with a specially formatted password: `oauth:userid`. For example, if you set up a user "z_smtp", use the password "oauth:z_smtp" in the email configuration for the source. That format of password triggers the XAUTH2 authentication method.

The notes field should look something like this (lines here are truncated):

```
cid=2991095c-2b2a-4...
cs=Z2F8Q~5dZSM4v...
ep=https://login.microsoftonline.com/ab44...
tk=LCJhbGciOiJSUzI1NiIsIng1dCI6ImptT...
rf=0.ARIAlplEqxeM...
```


11.6 Scheduler

Scheduled tasks is a feature of the UnForm server that runs a rule set on a set schedule. Tasks are created and edited using the Scheduler tab.

Existing tasks are displayed in the left panel, along with a toolbar button to add a new task. Tasks are edited, and saved or deleted, using the right panel.

ID/Description

Tasks are identified by a name, and have a associated description. ID values can be up to 20 characters, and contain alpha-numeric characters only.

Command Line

Enter the options of a uf101c command line, without -i, or -o options, and without the uf101c command itself, as those items are supplied automatically. The job will not receive any input, but will run the prejob and postjob code blocks. The server will execute this command line at the scheduled intervals.

Disabled

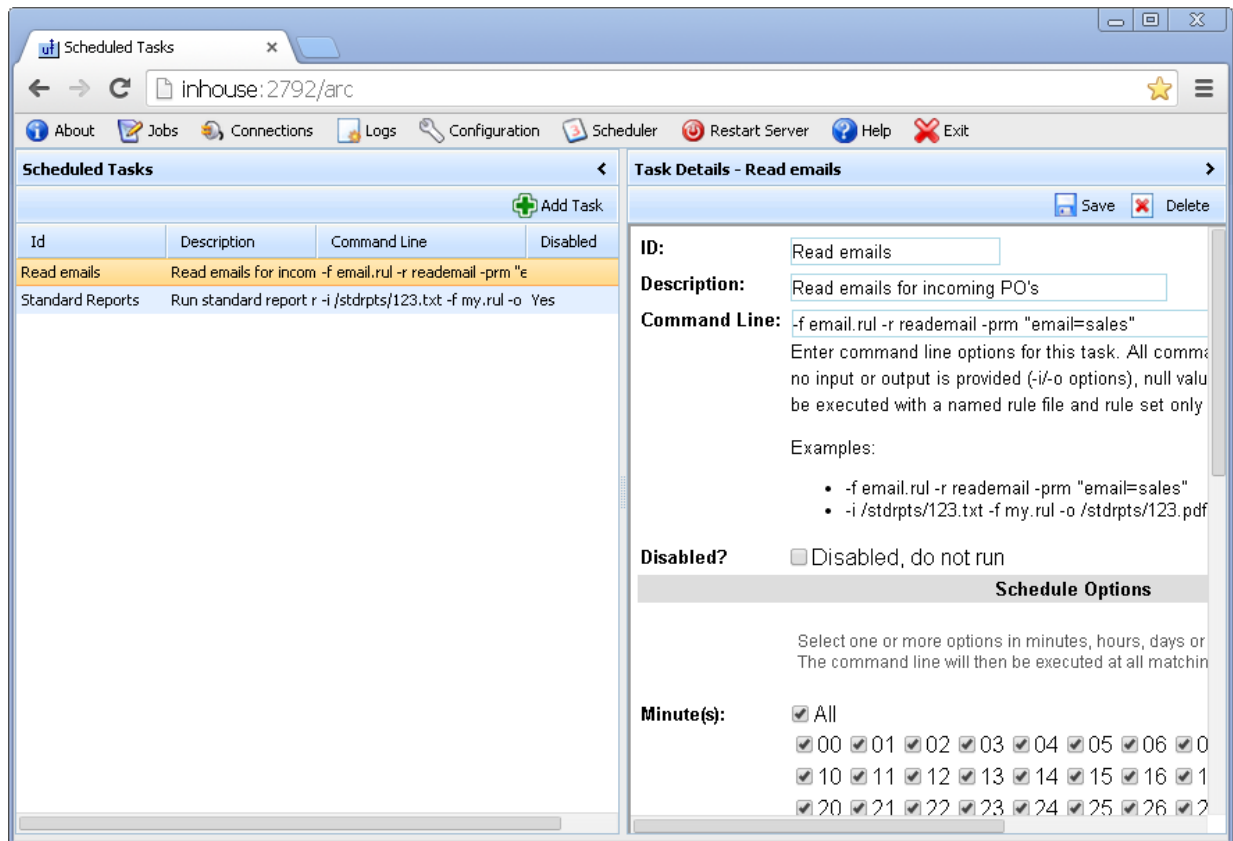
You can turn off execution of this job, without deleting it, by checking this.

Schedule

Scheduled times are allocated by minute, hour, day of month, day of week, and month. Each type of entry has an All option to select all the available intervals, such as all hours and all days. Select the appropriate intervals. Some examples:

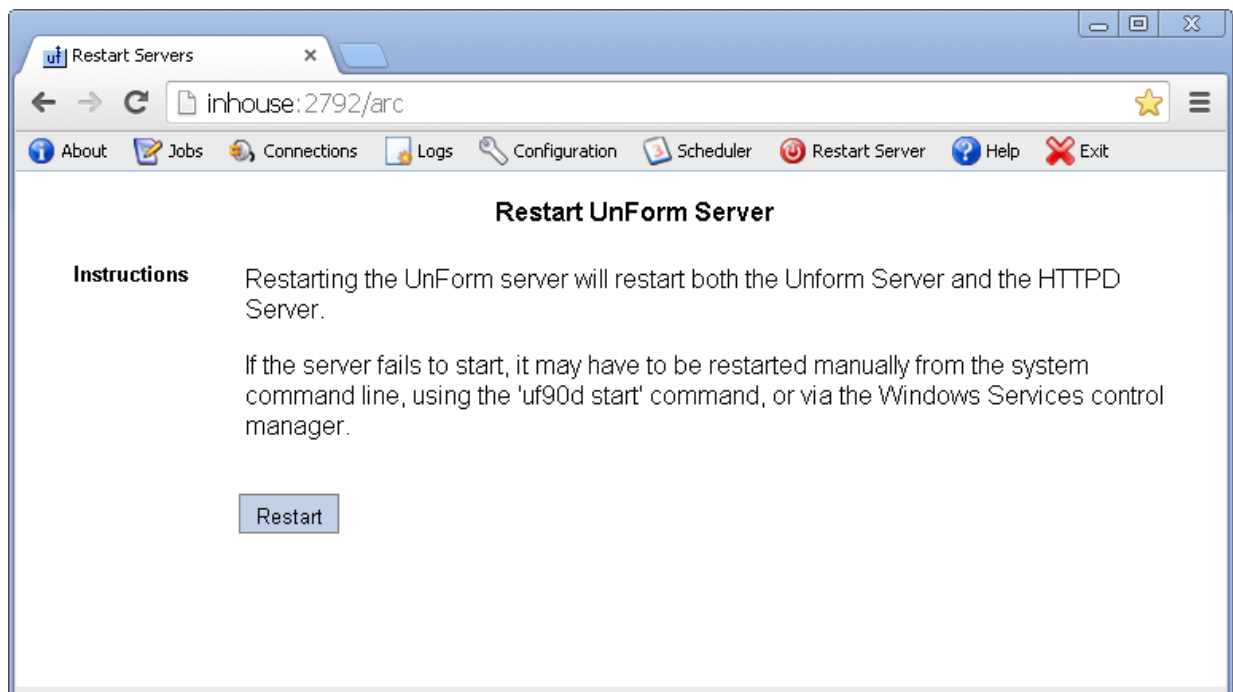
- 15th of each month, check all months, day 15, and an hour and minute for the task to run..
- Every Monday at 5:30PM. Check all months, Monday, and hour 17, minute 30.
- Every 5 minutes. Check all months, all days, all hours, and every 5 minutes.

Tasks can run as often as once a minute, or as infrequently as once per year.



11.7 Restart Server

Use this feature to restart the UnForm server, including the Apache HTTPD server. Note that if you change the HTTP port, you will need to reconnect using a new URL.



12 OTHER FEATURES

12.1 APPLICATION FORMATTED OUTPUT (AFO)

UnForm recognizes PostScript and PDF input streams, and will attempt to process them as pre-formatted print jobs. To do this, GhostScript 9.06 or higher must be configured. UnForm uses Ghostscript to both extract text and convert the job to PDF pages. An optional tool, MuPDF, can be installed and configured, in which case text extraction will use that program.

Caveat: Note that not all such print jobs contain text. Some contain images of text, and some contain a mixture of text and images of text. Only true text data can be used by UnForm as rule set data. In addition, sometimes text elements contain large regions of clear space around the text itself, posing some challenges for parsing text by location. The availability and usefulness of text is determined by the printing application and GhostScript, not UnForm.

The Design Tool can submit PostScript or PDF sample data to the UnForm server, and highlight each text element found on each page (by checking the Add Text Base option on the Preview menu).

The technique used by UnForm when it receives a PostScript or PDF print stream is to generate an overlay of each page in the output format of the job. UnForm graphical commands can then be used to add elements to this overlay, to scale it, and to erase regions from it. Other than selective erasure, it is not possible to modify the overlay. In many cases, there will be very limited, if any, cosmetic enhancements needed, allowing the implementer to focus exclusively on document management features such as electronic delivery and archiving.

The most common method of integrating AFO printing with UnForm is to use a Windows printer configured with a PostScript print driver and a TCP/IP port directed an UnForm [TCP/IP monitor](#).

The `-noafo` command line option can be used to suppress AFO processing for Postscript/PDF input, and may also be useful as an argument passed to a subjob in a `jobexec()` function, as subjobs of AFO jobs are by default treated as AFO jobs themselves.

The `-afo2` command line option can be used to utilize a different text parsing algorithm, which attempts to parse words from print stream data, which often uses phrases rather than simple words. This can impact detection logic and also the `gtext*()` functions, so a rule set designed for one mode may not work in the other mode. The new mode is used by the Image Manager when calculating word positions for zonal OCR text extraction.

Normally, when Ghostscript-created words are parsed, UnForm will attempt to sort them in a top-down, left-to-right order, but this algorithm can produce odd results under some circumstances with mixed character heights. An alternative is to set the `uf101d.ini [defaults] afogsorder` flag to 1, which suppresses this second sort, and accepts the order in which Ghostscript reported the words. Ghostscript versions after 9.12 produce more reliable results than older versions.

As the PDF pages are treated similar to overlays, the orientation of the UnForm job must match that of the PostScript or PDF input. For example, if the input uses landscape orientation, the UnForm rule set should include a `landscape` command.

Text vs. PostScript/PDF Print Stream Management

When working with plain text input, UnForm has commands that manipulate or apply enhancements to a text print stream, such as `font`, `bold`, and `erase`. Also, code blocks can manipulate the `text$[]` array, resulting in modified print stream text. However, when working with PostScript print streams, there is no text array, and commands that depend on it are not available. One exception is **erase**, which is

translated to be a shade command with a shade value of 0, resulting in erasure of the specified region of the overlay. Also, the **notext** command and its new synonym, **nooverlay**, may be used to suppress printing of the overlay on any copy or all pages.

The following commands are not compatible with AFO:

- across
- bold
- down
- font
- hline
- hshift
- italic
- light
- move
- page
- shift
- underline
- vline
- vshift
- any Zebra- or label-only commands

In addition, many commands support anchor text or patterns, which cause a search of the text content of the page to locate positions to apply enhancements. Supported commands that offer this feature, such as **barcode**, **box**, and **text**, continue to support the anchor search technique. However, since the location of PostScript text regions do not always correspond to the visual location or size of the text, accuracy can vary.

The margin command is honored (starting in version 10), to adjust the positioning and scaling of the overlay in cases where passed-through print data is outside of the printable margin of paper.

If AFO text regions vary from visual location or size, then detection logic may require greater flexibility than with simple text input streams. The **detect** command has been enhanced to support partial columns and rows, but it may be necessary in some cases to detect elements from the whole page rather than regions.

Text Array Limitations in Code Blocks and Expressions

Many code block functions that work with a text print stream are also not available. However, the `get()` and `mget()` functions have been enhanced to return text data from the PostScript print stream, plus three new functions have been added, `gtextcount()`, `gtextitem()`, and `gtextfind()`, which provide access to the text elements parsed from the PostScript print data. A new variable, `nooverlay`, can be set to 1 in prepage or precopy code blocks to suppress the printing of the overlay. This can be used to manage multi-format jobs, such as those with terms and condition attachments.

The following code block functions are not compatible with AFO:

- `cut()`
- `delpage()`
- `getpage()`
- `inspage()`
- `mcut()`
- `mset()`
- `putpage()`
- `set()`

The arrays `text$[]`, `textjob$[]`, and `textpage$[]` are not available.

New Functions For Accessing Text

- `gproperty()` returns values from PostScript DSC comments in the print stream.
- `gtextcount()` returns the number of text elements in a page.
- `gtextitem()` returns text and optional region information for a given text element on a page.
- `gtextfind()` searches for patterns in text and returns arrays of text and region information found.

12.2 WINDOWS SUPPORT SERVER

The Windows Support Server is a no-charge companion product that can be installed on any Windows 2003 or higher computer on the network where the UnForm server runs. If the UnForm server is running on Windows, then the Support Server is already installed and can be enabled through server configuration. If the UnForm server is not running on Windows, you can install the Support Server separately and configure these lines in the [defaults] section of `uf101d.ini`:

```
sshost=hostname or IP address of Support Server  
ssport=listening port number
```

In addition, the `sshost()` code block command can be used to set the support server machine and port during job processing.

The following features are supported:

- **Image scaling and conversion**
The UnForm server can utilize a local copy of Image Magick or Image Alchemy to perform image scaling and conversion, but these products are not always readily available for some Unix operating systems. In addition, the image command supports two options, `gamma n` and `rotate n`, which the Support Server honors. This feature is used automatically by UnForm whenever an image conversion or scale is required, if Magick or Alchemy is not configured for use at the server.
- **GhostScript-based Image Output**
The UnForm server can utilize a local copy of Ghostscript to produce image output and to convert PDF files to other formats. However, the latest versions of Ghostscript are not readily available on all operating systems. By installing a Windows version of Ghostscript on the support server, the UnForm server can rely on it to perform the conversions. This feature is used automatically by UnForm whenever a PDF-to-image conversion required, if GhostScript is not configured for use at the server.
- **Database Access**
The support server can be configured to access data base sources via ODBC or more recent database access technologies. UnForm rule files can connect to these data sources and retrieve data for use in UnForm jobs.

Data sources are configured using the Support Server configuration window. UnForm jobs can then use the `dbconnect()` and `dbexecute()` code block commands.
- **Microsoft Fax**
The Microsoft fax server, a free product available for Windows 2000 and up (and pre-installed on Windows XP and up), can be easily set up on the support server or another Windows server on the network. The support server can then use the Microsoft fax client to send faxes on behalf of UnForm jobs.

UnForm jobs can use the msfax() code block command as soon as Microsoft Faxing is configured.

The following table describes the various code block functions that are supported when the Support Server is available.

sshhost(SERVER\$,PORT)	Sets the support server hostname and port. Default values are defined in the uf101d.ini file in the sshost and ssport settings. This command allows for dynamic changing to a different server.								
dbconnect(NAME\$, TIMEOUT, EMSG\$)	Connects to the database source identified by name\$. The support server configuration is used to define the names and associate them with data source connection strings. Typically done in a prejob code block.								
dbexecute(NAME\$, CMD\$, TIMEOUT, FDELIM\$, RDELIM\$, RESPONSE\$, EMSG\$)	Executes the SQL command cmd\$ and returns zero or more result rows in response\$. Columns are delimited by fdelim\$ (tab - chr(9) - by default). Rows are delimited by rdelim\$ (CR-LF - chr(13)+chr(10) - by default).								
msfax(FILENAME\$, FAXNUM\$, TAGS\$ [, EMSG\$])	<p>Faxes filename\$, normally an UnForm-generated PDF file, to the fax number specified in faxnum\$. Numerous supported tags can be specified in tags\$, in the format tag1=value;tag2=value,...</p> <p>The format of faxnum\$ can be a simple phone number, or multiple numbers separated by semicolons, or tags in the format:</p> <p>name1=fax1; name2=fax2, ...</p> <p>Quote the entire tag if it contains semicolons: "Smith; Cline; Robert=9,1-555-555-5555". This will involve use of quote characters in the expression, using chr(34) or \$22\$. For example:</p> <p>Faxnum\$=chr(34) + name\$ + "=" + faxno\$+chr(34)</p> <p>Note that fax numbers may need to be complete, using for example "9,1" as a prefix for an outside line, a pause, and a leading 1 before the area code, depending on fax server use of dialing rules.</p> <p>Tags supported are:</p> <table border="1"> <tr> <td>Cover</td><td>Standard coversheet name based on the fax server, such as cover=generic.</td></tr> <tr> <td>Localcover personalcover</td><td>Personal or fax client-side cover sheet.</td></tr> <tr> <td>Subject</td><td>Subject for cover sheet.</td></tr> <tr> <td>Note or Notes</td><td>Notes for cover sheet. Use \n for hard line breaks.</td></tr> </table>	Cover	Standard coversheet name based on the fax server, such as cover=generic.	Localcover personalcover	Personal or fax client-side cover sheet.	Subject	Subject for cover sheet.	Note or Notes	Notes for cover sheet. Use \n for hard line breaks.
Cover	Standard coversheet name based on the fax server, such as cover=generic.								
Localcover personalcover	Personal or fax client-side cover sheet.								
Subject	Subject for cover sheet.								
Note or Notes	Notes for cover sheet. Use \n for hard line breaks.								

	Time	A human-readable date and time to send the fax, if not immediately.
	Receipt Attachfax	An email address to send fax result reporting. Note that you must be using a Server version of Microsoft Fax with Microsoft Exchange and enable SMTP receipt delivery for this to work. If a receipt is specified, you can additionally use the attachfax option to have the receipt email include the fax image.
	Alert	This tag's presence requests that the fax client issue a message box regarding the fax disposition.
	Server	Set the Microsoft Fax Server computer name if the server is not running on the same system as the UnForm support server.
	ToName	If a single fax number is supplied, this tag is an alternate way to specify the recipient name.
	FromName	Alternative tag to set the sender name.
	FromCompany	Alternative tag to set the sender company name.
<p>Other sender tag names that may be used by a cover page:</p> <ul style="list-style-type: none"> name title company department title homephone officephone faxnumber email streetaddress city state zipcode country 		

Configuring Microsoft Fax

If faxing is not already configured, use Control Panel Add/Remove Programs to install the Microsoft Faxing service. First, click the Add/Remove Windows Components button, then enable the Fax Services checkbox. You may need your Windows installation media to install Fax Services.

There are a few common pitfalls to working with Microsoft Fax. Things to be aware of:

- Since the Support Server typically runs as a service, it is important to ensure that the user it runs under has permission to use Microsoft Fax. To test this, login as the same user as the service and try

manually faxing. Adjust security for the Fax device in printer configuration.

- The typical document type faxed from UnForm is a PDF file. PDF files must be converted to TIFF images before faxing, so Ghostscript must be installed and configured.
- Further, the system must be able to fax a TIFF image, which requires a TIFF viewer program be active and that it supports the Windows "printto" shell operation. Some systems do not have a TIFF viewer installed by default. On some versions of Windows servers, this is part of a package called the "Desktop Experience", which can be downloaded and installed from Microsoft. This article describes this product:

<http://technet.microsoft.com/en-us/library/cc772567.aspx>

- Cover pages are user-specific, so if you choose to configure cover pages for use the the Support Server faxes, be sure to login as the same user the service is configured to run under before editing covers.

12.3 DATABASE ACCESS

UnForm supports access to databases from rule set code, using one of two techniques. Note the preferred technique is Server-based Access.

Server-based Access

The second technique, added in UnForm 9.0, supports access to database sources directly within the UnForm server. When using this method, you connect to a data source identified with a string construction, optionally supplying a user and password login, as well as other optional arguments. The `sqlconnect()` function provides the functionality, and returns a connection channel number.

Note that secure passwords can be configured in the browser interface and referenced in the `sqlconnect()` function, using the syntax "store:/D" rather than a plain text password.

After connecting, send SQL commands to the database channel using the `sqlexecute()` function. Access the data returned by the command, if any, using the `sqlfetch()` function, which can return one, many, or all rows from the query, in a delimited string.

When done with the data source, you can close the channel with the `close(chan)` command.

The syntax of the three functions is:

```
chan=sqlconnect(datasource$[,user$ ,pswd$ [,otheroptions$ [,errmsg$]])
[success=]sqlexecute(chan,command$[,errmsg$[,result$[,fdelim$[,rdelim$]]]])
count=sqlfetch(chan,result$[,count [,errmsg$ [,fdelim$ [,rdelim$]]]])
```

There are four types of databases supported, though not all types are supported on all platforms. The "uf101c -v" command shows which database types are supported. The four types are ODBC, Oracle, DB2, and MySQL. Note that ODBC is supported on Unix/Linux, as well as Windows, if either the unixODBC or iODBC package is installed.

The syntax of the `datasource$` argument identifies the database type and data source:

- `odbc:dsn` connects to the ODBC data source name (also called the DSN), as configured in the Windows ODBC administrator or in the unixODBC/iODBC configuration.

- `oracle:sid` connects to the Oracle System ID, using local Oracle client libraries.
- `db2:database` connects to the DB2 database specified.
- `mysql:database[:hostname]` connects to the MySQL database named, optionally on the host specified.

Most databases require a login and password in order to access a database. The user and password must be supplied in those cases.

Additional options that can be supplied in the `otheroptions$` argument, as a semicolon-delimited list.

Options include:

- `access=read|write`
- `strip` (if present, causes trailing spaces to be trimmed from fields)
- `textmax=val` (sets the maximum amount of text returned from a text field, default=4096)
- `timeout=seconds`

Once a connection channel has been created, you can then send SQL commands to the channel using the `sqlexecute()` function. That function can optionally fill a results variable with all the rows returned by the query, or you can use the `sqlfetch()` function to return rows one or many at a time.

Below is a simple example showing how to use the three functions:

```
prejob{
chan=sqlconnect("odbc:sampdb","userid","password")
if chan>0 then:
    e=sqlexecute(chan,"select member_id, last_name, first_name from member")
    if e>0 then:
        while sqlfetch(chan,row$)
            row$=sub(row$,$09$,"|")
            allrows$+=row$+$0a$
        wend
    end if
    close(chan)
end if
}

text 10,2,{allrows$}
```

Windows Support Server Access

An older technique uses the Windows Support Server to access data sources available on the machine where the support server runs. The Windows Support Server configuration window enables database connections to be configured and given a name, and two code block functions: `dbconnect()` and `dbexecute()` are provided to communicate with the named connection to return the results of a query. The syntax of these two functions is:

```
[success=]dbconnect(name$[,timeout[,errmsg$]])
[success=]dbexecute(name$, command$, timeout, fdelim$, rdelim$, response$ [,errmsg$])
```

Both functions return 1 if successful, 0 if not. This technique was available in starting with UnForm 7.0.

12.4 ADDRESS BOOKS

UnForm supports multiple address books to assist in delivery of documents to email or fax addresses. Address books can be created and maintained directly in the archive browser interface, or programmatically via rule set code blocks.

Address books utilize a concept of an Entity ID, which is an identifier for a particular entity, such as a vendor or customer, or any other unique contact that might be required for an application. In addition to an entity ID, an address record is identified by an optional document type. This allows a single address book to be utilized, for example, for customer addresses for sales, delivery, or ad hoc contact. Each combination of entity ID and document type can have a delivery address, either email or fax.

Specifically, the fields maintained in an address book record are:

- Entity ID
- Document Type
- Entity Name
- Contact Name
- Send To (email address or fax number)
- Combine

The combine field can be leveraged by the **deliver** command, which has the capability to combine multiple documents in a batch that are targeted to the same delivery address.

The browser interface provides address book maintenance features to users who are granted address book maintenance rights. This maintenance feature includes an ability to import and export an address book in a CSV format for easy maintenance using third-party tools or text editors. Many applications and report writers can produce CSV files, allowing the upload of address book information.

In addition, when address books are populated with entity ID's that match document entity ID's, email address suggestions are offered when viewing documents in the browser interface.

In addition to user interfaces, address books can be programmed within rule sets. There are two functions for simple read and write of address book records, `getaddress()` and `putaddress()`, plus the "addrbook" object, that provides greater flexibility. Using these facilities, you can create rule sets for importing and management of address book entries, as well as utilizing address books for delivery addresses for emailing, faxing, or the new **deliver** command, which offers batch handling and automated email and fax delivery capabilities.

12.5 DYNAMIC RULE FILE TRANSLATIONS

At runtime, a translation file can be associated with a rule file in order to perform dynamic substitution of text and barcode values, and also anchor text to which many enhancements can be related. The purpose of this feature, added in version 8.0.25, is to ease the effort of translating rule sets for different languages. By editing a messages file that is associated with hard code text fragments in a rule file, UnForm can perform word and phrase substitutions dynamically.

The structure of the translation file is an "ini" file, with sections that match the names of rule sets in the current rule file. At runtime, UnForm will load assignment lines from the section that matches the current rule set name, plus any assignment lines found in the file before the first section header. For example, if a rule file contains the rule set "Invoice", then lines in the [Invoice] section of the translation file are used. Any assignment lines at the top of the file are appended to these lines, allowing for global settings to be used as well as ruleset-specific settings. An example of a translation file is the

samples/advanced.spanish.ini file, which contains translation samples associated with some rule sets in the advanced.rul file.

A translation file can be specified in the uf101c command line with a `-trans "filename"` option. In addition, the active file can be changed at runtime in any code block, using the `settrans("filename")` code block command.

The format of an assignment line is simply a *name=value* pair. The *name* value can include a context prefix, where the context can be one of the words text, barcode, or anchor, followed by a tilde (~). For example, the line `text~Invoice=Factura` would replace "Invoice" in a text command with "Factura". A line without the context prefix (just `Invoice=Factura`) may apply to both text and barcode commands.

Values are case sensitive, and only whole values are replaced. For example, if a text command specifies "Invoice No:", then the assignment line might be: `Invoice No:=Factura no:`.

When assignment lines are evaluated, they are first evaluated in a context of text, barcode, or anchor. If a match is found, that assignment is used. If not, text and barcode values are next evaluated without a context, and if a match is found, that assigned value is used. Without a match, the original text is used.

Anchor context substitutions apply to content that is searched on the page and *only* work in context mode (the "anchor~" prefix is required). While text and barcode substitutions apply to content added to the job, anchor substitutions apply to content in the print stream of the job. Anchor substitutions do not change text values, but instead change the text values searched for. Many commands support anchors in their syntax, when the first command is a quoted string. For example, a font command might look like this:

Font "Continued@50,60,80,66",0,0,9,1,cgtimes,12,bold

This command will search each page, in columns 50-80 and rows 60-66, for the word "Continued". When it finds the word, it applies a font at 0 columns and 0 rows offset from the position, for 9 columns and 1 row. When a translation file is active, UnForm will look for a line `anchor~Continued=value`, and search instead for *value*. Additional numerical adjustments are performed on anchor contexts:

- The difference in lengths of the original text and new text is calculated
- The difference is added to the ending column if `@col,row,endcol,endrow` syntax is used
- If the offset column is 0, the difference is added to the number of columns, to accommodate enhancements designed to apply to the text that is located.
- If the offset column is not 0, the difference is added to the offset column itself, to accommodate a new relative position based on the new text.

Anchors can include the following prefixes: ~, !=, and !~. These prefixes modify how the anchor text and the search are performed, and are not considered part of the assignment name. A ~ prefix indicates the text is a regular expression, and the != and !~ options indicate "not", so match all positions where the search does not find the sought string or regular expression. Since regular expressions can result in variable lengths, the above described numerical adjustments are not performed with regular expression anchors. Another limitation of anchor-based translations is that they are not supported in Application Formatted Output (AFQ) jobs, though text and barcode translations are supported.

Since both "~" and "=" characters have special meaning, if they should be part of the actual name they must be escaped with a backslash. For example, if the *name* value should contain an equal sign, which normally separates it from the *value*, that equal sign must be escaped: `"Cost="`, for example, should appear as `Cost\= =value`.

It should be noted that the dynamic translation feature is a simple substitution function, and it does not account for cases where a substitution dramatically alters the space used by a particular text fragment, or

when a different font or character set should be used. In some cases, it might be necessary to use fit or wrap options, expressions for positioning, or code block `exec()` functions to accommodate some aspects of translation.

Code blocks can also utilize the features of text translation through three functions:

- `settrans("filename")` sets the translation file dynamically as the job runs. This overrides what might be set via a `-trans` command line option.
- `gettrans()` returns the active translation file.
- `translate(name$ [,context$], forcecontext)` returns the value associated with the specified name, based on the translation file and current rule set. The context value can be "text", "barcode", or "anchor", and if `forcecontext` is true (non-zero), only context-based names are searched.

There is a rule file, `samples/trans.rul`, with the rule set "trans", available in the samples directory. This rule set is designed to scan any rule file sent through it as input, and build as output a skeleton translation file based on hardcoded text found in the rule sets. In order to run this rule set, use this command line:

```
uf101c -i "rulefile" -f trans.rul -r trans -o "inifile"
```

12.6 HTML5 OUTPUT

HTML5 output, generated by use of the `-p html5` option, is designed to create page-oriented output, similar when viewed to PDF output. In addition to general document features, such as text, images, and box drawing, the HTML5 driver provides hooks to allow a developer to insert custom HTML and Javascript code into specific sections of the output.

HTML5 output is designed to be stand-alone, not requiring access to any particular web server. For example, if your rule set specifies an image file such `logo.jpg`, that file is converted into a data URL structure that can be embedded in the output. In more typical HTML, such an image would be referenced and retrieved from a web server at view time. With UnForm, you can specify an external image this way, but need to reference it with an `"http://"` or `"https://"` prefix so UnForm knows not to look for a local image and embed it.

The output generated by the `html5` driver is generally compatible with modern web browsers on most platforms. For Internet Explorer specifically, it is supported by IE9 and above.

Command Differences

The `image`, `attach`, and `overlay` commands normally reference a local file, and that file will be embedded into the output as a data URL. However, these commands also recognize a standard URL structure starting with `http://` or `https://`, and will use the URL supplied instead of a file-based data URL.

The `image` and `text` commands support two new options to support hyperlinks. These are created using an HTML `text or image` structure.

- `link "url" | {urlexpr}` to specify a target URL as a literal or expression, which becomes the `href=` value of the `<a>` tag.
- `linkopt "options" | {optionexpr}` to specify additional `<a>` tag options as a literal or expression.

For example, the following will turn My Company into a hyperlink to mycompany.com, opening it in a new, blank window.

```
text 10,1.5,"My Company",16,bold,Helvetica, link "http://mycompany.com", linkopt "target=_b
```

Note that URL values must be url-encoded. This is particularly important if the URL includes a query string, where values can contain spaces and other non-alphanumeric characters. Use the `urlencode()` function to assist in creating url-encoded strings.

The `image` and `attach` commands will insert their content differently depending on the type of file being inserted. If the file appears to be an image (its MIME type matches `image/*`, like a jpeg or png file would), then it is referenced with an `` tag. Otherwise, it is referenced with an `<iframe>` tag. Note that some types of `iframe` content, in combination with some plugins, such as Adobe Acrobat Reader, render on top of all content, so PDF files might not be suitable attachments and should be converted to images.

The `outline` command is supported, and causes automatic insertion of a toolbar feature to navigate the outline structure.

TrueType fonts are supported, though the viewing browser must also support them via the `@font-face` CSS rule, and that excludes Internet Explorer 9 and earlier.

Inserting Custom HTML Output

Each HTML5 job has, at runtime, an `html5` object variable that can be used to insert HTML and Javascript into the output from code block code, such as `prepage` or `precopy`.

- `html5'putpage(html$)` inserts page content in `prepage` or `postpage` code blocks
- `html5'puthdr(head$)` add content to the `<head>` `</head>` section
- `html5'putsript(javascript$)` add lines of script code following body, inside the `<script>` and `</script>` tags
- `html5'puttoolbar(html$)` add lines to toolbar `<div>` section. The output can include a toolbar that resides at the top of each page. If any code injects toolbar content, the toolbar will be visible.
- `html5'putfooter(footer$)` add lines to footer following the `</body>` and `</html>` tags

The "html5" subdirectory under the UnForm server path includes several files that are automatically inserted into the output. All file names starting with "uf" are included with UnForm and are overwritten by updates, but this directory is also a convenient place to add your own code fragments that can be used by the `html5` object methods, using the `getfile()` function.

Here is an example from the `simple.rul` file in the `samples` directory. This is a `postjob` code block that inserts two toolbar features from files found in the `html5` directory. Note the condition on the `html5` object being present.

```
postjob{
  if html5 then
    html5'puttoolbar(getfile("ufsmg_zoom.html"))
    html5'puttoolbar(getfile("ufsmg_paging.html"))
  end if
}
```

12.7 MAILCALL

UnForm includes a copy of the MailCall utility that enables emailing of attachments from within UnForm. This is most often used to send PDF files, but it can be used to send any file at all, subject to limitations imposed by the mail server (which may limit email size, filter attachment types, or impose quotas).

The MailCall utility is used internally by the [deliver](#) command, the [email](#) command, which emails a complete PDF-formatted job, and the `email()` code block function, which can send email(s) in mid-job, possibly with attachments resulting from sub-jobs managed by the `jobxxx` series of code block functions. These features are capable of handling most email requirements. However, within a code block, you can use the MailCall program directly, for any degree of control required.

For details about using MailCall, you can refer to the [MailCall manual](#). All features except those the provide a user interface are available to UnForm jobs. Where there are references to dialects of the Basic language, note that UnForm 10.1 uses PxPlus 14 or higher.

The UnForm instance of MailCall supports UnForm's encrypted password store. Define the email password under a name in the browser admin interface, and then specify the name with the syntax `"store:name"`.

12.8 LOAD BALANCING/FAIL OVER

UnForm supports load balancing and automatic fail-over by enabling a client connection to attempt to access one of several servers. The job can then run on an available server.

A list of servers can be provided in the `-server` option of the `uf101c` command line, or in a `server=` line in `uf101c.ini`. The servers are delimited by semicolons, so the whole option should be quoted to avoid misinterpretation by shells.

```
-server "192.168.1.10;192.168.1.60"
```

When multiple servers are specified, the `uf101c.ini` `minavail=n` setting is used. The first server with either 0 jobs in use, or at least *n* job slots available, is chosen. If *n* is 0, and all servers have at least one job slot in use, the one with the least proportion of jobs in use vs. licensed is chosen.

Archiving

If archiving is used, one server should be designated as the archiving server. All archive updates are then sent to that server, or queued if it is not available, so that only one set of libraries is created. The archive server is defined in the `[archive]` section, `server=server:port` setting. This value should not be set on the archive server, but should be set on all the other servers used to run jobs.

File Synchronization

When there are several servers that might produce jobs, managing rule files and their associated dependencies across the different servers can be tedious. To help with this, you can configure automatic file synchronization across the machines.

There should be a primary server where rule files and other files are maintained. On this server, maintain the `[syncfiles]` of `uf101d.ini`. This can be done manually or in the Server Manager browser interface. This section contains a list of file names, or file wildcards, one per line. Note that wildcards can only be in the file portion of the name, not a directory portion, which must be specified explicitly.

On the child systems, in their `uf101d.ini` files, set `syncfrom=server:port` in the [defaults] section. Optionally update the `syncinterval=n` value as well. Every *n* minutes, the client systems will ask the primary server for a list of files and their content hashes. These are compared with local copies of those files, and any changed files are copied from the primary server. Pathnames of the files will be consistent, and will be auto-created on the child machines. Note that this means if you rely on relative names, then all machines should have UnForm installed in the same path.

Unique Server Settings

If different servers use different `authkey` or `proxy` options, you can specify them in the `-server` option. Each server can have up to five colon-delimited segments:

servernameOrIP:port:authkey:proxynnameOrIP:proxyport

Any segment value overrides what might be in `uf101c.ini` or other specific command line options.

Library Access

While archive command results are automatically on a configured archive server (see Archiving, above), if a job requires access to a library during its processing, that job must run on the server where the libraries are. For example, if a library object is required, or the `getimage()` function is used, the job must execute on the server where the libraries reside. To support this in a load balance environment, you can add the following code to rule sets that require it:

```
prejob{
  infile=new("infile","uf101d.ini")
  arcserver$=infile'getitem$("archive","server")
  drop object infile
  if arcserver$>"" then
    runjobon(arcserver$)
    skip=1
  end if
}
```

See the **runjobon** function in the [Internal Functions](#) table for more details.

12.9 FAIL RECOVERY

In any client-server environment there can be problems submitting jobs to the server. The problems can be a result of a network problem, or a license problem, or just a mistake in the client command line, such as a bad rule file or output destination. In the client's `uf101c.ini` file, a `failhist=days` value can be specified so the client will retain job input, command lines, and error messages when a job fails to run at the server.

When this parameter is set to a number greater than 0, then the client will store failure history so that the failed jobs can be resubmitted, optionally with overridden command line options to correct problems related to command line mistakes.

Failed Job Storage

The queue for failed jobs is found in one of two places:

- Linux clients store failure history in "failhist" under the client's home path (where `uf101c.pl` is located)
- Windows clients store failure history in `%ProgramData%\SDS\uf101\failhist`

Under this directory, history is stored by date in directories formatted as `yyyymmdd`. Each failed job is stored with three files. A `.in` file contains the job's print stream, a `.arg` file contains the job's command line arguments, and a `.err` file contains the error message pertaining to that job's failure.

Failed job history is maintained for the number of days specified in the failhist parameter of uf101c.ini. Directories older than that date in the failhist directory are removed as jobs are run.

Recovery

To re-run a job, or several jobs, you use the command line client, uf101c on Linux or uf101cc.exe on Windows, with the -rerun option. Arguments before the -rerun option are used as overrides to any found in the failed job's arguments file, so you can use this to fix problems with the original command line. For example, if jobs failed because of a bad rule file name, you can specify a new rule file with -f *correctname.rul* in the command line before the -rerun option.

After the -rerun should be one or more file name references from the failhist directory structure. These names can be plain file names or full path names. They can include any of the .in, .arg, or .err extensions. The process of re-running jobs normalizes the names. There are three special circumstances:

- If no names follow -rerun, the standard input is read for a list of files to process. This makes it easy to use script routines to provide the list of names. For example, on Linux the grep command can be used to get a list of files containing a certain message, and the list passed to the uf101c command line:

```
grep --files-with-matches 'Bad rule file' failhist/*/*.err | uf101c -f myrule.rul -rerun
```

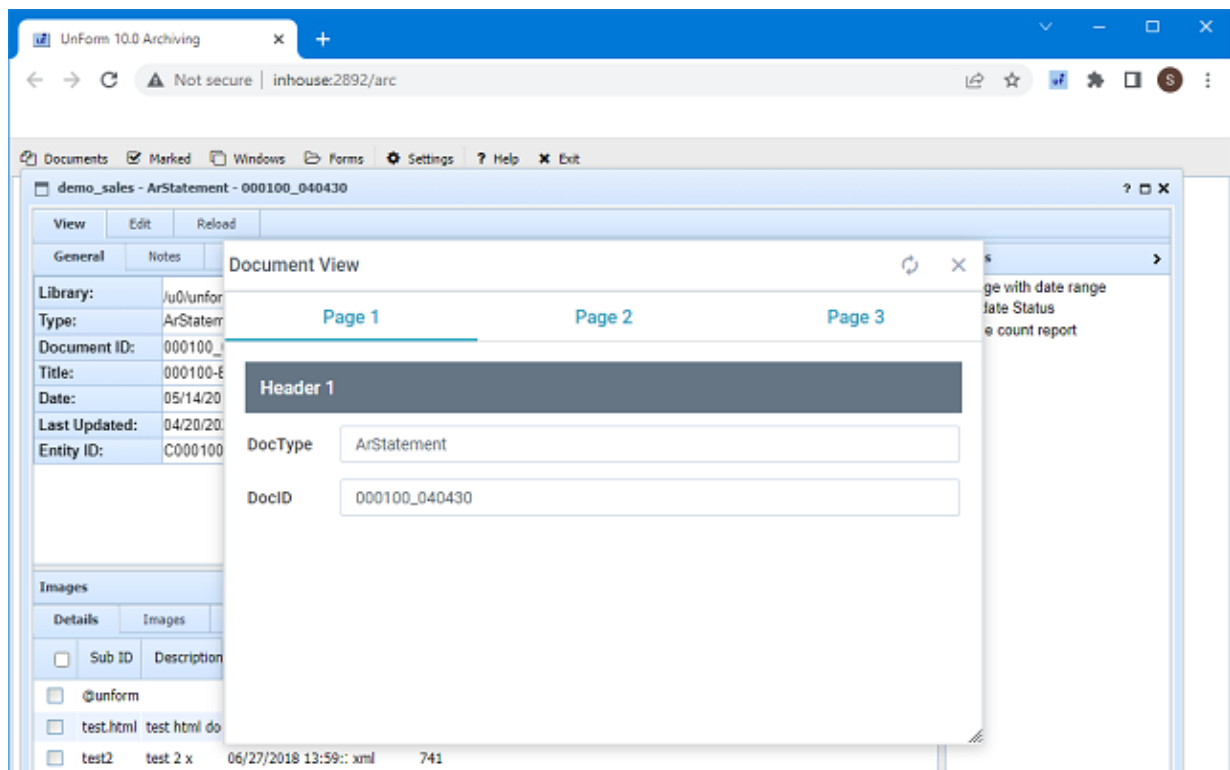
- Use **-rerun all** to process all files in all failhist directories are re-submitted. This could be used, for example, to recover from a down server.
- Use **-rerun yyyyymmdd** to process all failhist files for the specified date.

12.10 UNFORM WEB EXTENSION

The UnForm Web Extension is a browser extension that is compatible with Chrome and Edge browsers. It is designed to integrate browser-based ERP applications with an UnForm server, adding buttons and other integration touch points to the ERP application screens to enable seamless integration between the ERP and UnForm's document management modules. In addition, documents can be printed from the browser through UnForm for both archiving and print enhancement purposes.

The extension is configured via special rule files that contain detection, screen-content and user-entered data fields, and launch button configuration. The launch buttons pass captured data to the rule file, which returns responses that enable the browser to open documents, indexed lists, thumbnail views, and more from the UnForm server.

In addition, virtual extension printers can be configured in the UnForm [Server Manager](#) and these printers are available when the user prints from their browser. Such print streams can be enhanced and archived by UnForm as if they had been submitted through normal print channels.



The UnForm Web Extension can be installed from this link:

<https://chrome.google.com/webstore/detail/unform-web-extension/gcemegkcpgcalfjijnbhjgkhhbhdgdp>

12.10.1 Configuration

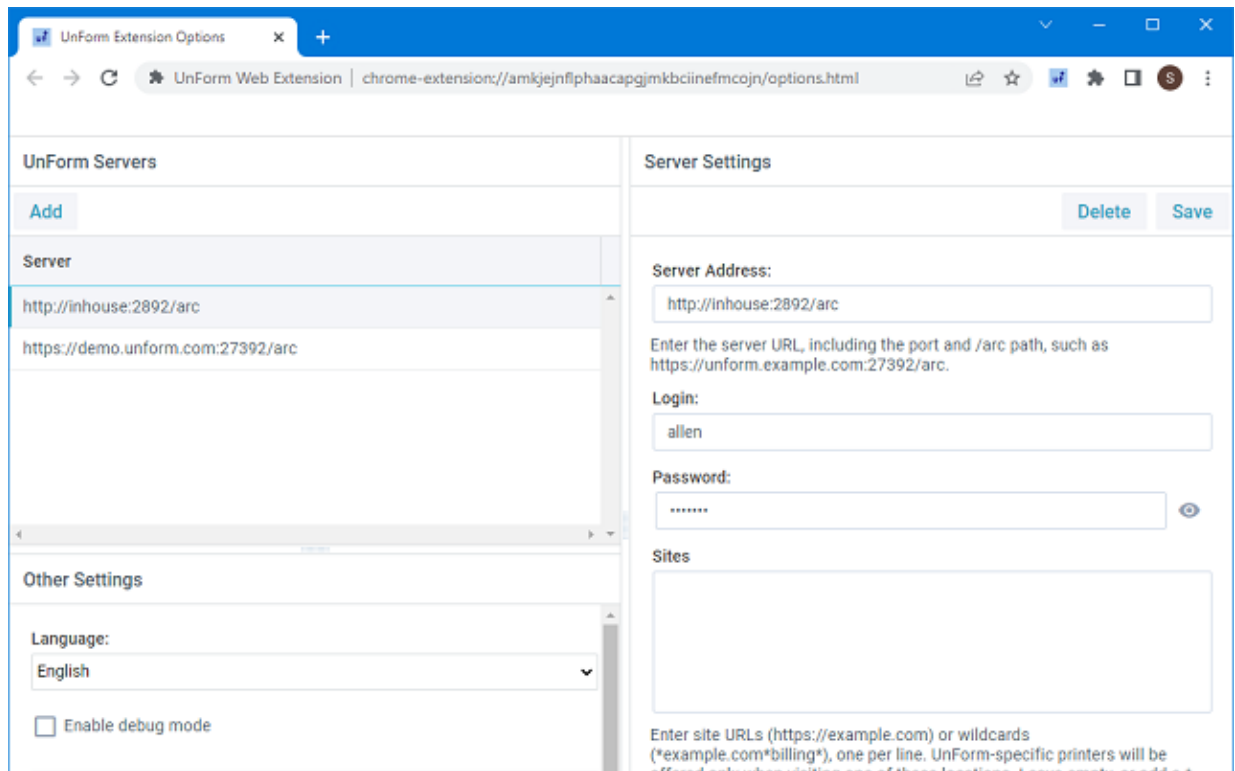
After installation, the extension must be configured.

The web extension is configured from within the browser's Extensions/Add-on's management page. In Chrome, go to the `chrome://extensions` page or toolbar icon. Right-click the UnForm extension icon and choose Options. The extensions configuration screen will be presented, where you can enter the UnForm server address and login information. While rarely used, it is possible to configure more than one UnForm server.

An important consideration is that if the browser is viewing websites over https, the UnForm server must also be accessed over https.

Server Configuration

An UnForm server requires a server address, including the port and `/arc` path, as well as login and password information. Extension rule files are loaded from the configured server(s) and used to monitor browser navigation that matches detection logic in the rule file. In addition, if the browser is viewing sites that match lines in the Sites field, configured UnForm web printers are added to the available printers when the user chooses to print a web page. This is particularly useful when the browser is viewing a PDF document because UnForm will receive an exact copy of the PDF being viewed.



Note when developing extension rule files, any time the rule file changes it should be reloaded into browser memory. Click the Reload Site Rules in the Other Settings panel to do this.

12.10.2 Rule Files

Web extension rule files define screen integration elements for specific web pages. Any number of rule files can be configured, each listed in `uf101d.ini` in a `[webrulefiles]` section. This section is composed of lines in the format: *rulefile=URL-pattern*. For example, if you want a rule file named "classic-erp.rul" to be used whenever visiting the site "https://classic-erp.com/sdsi", use a line like this:

```
classic-erp.rul=https://classic-erp.com/sdsi
```

Note that the URL pattern is treated as a regular expression, enabling complex pattern matching in cases where there might be specific URL parameters or fragments to should apply.

Once a matching site is viewed in the browser, the configured rule file is applied to pages and frames the user navigates to. Frames are very common in browser-based applications, and each frame is treated as a unique page for integration purposes.

Detection

The standard UnForm detect statement is used to determine which rule set, if any, to apply to any given page or frame. The detection is evaluated in the following way:

detect *x,y,"match value"*

0,0 title + " " + url

0,1 title

0,2 url

0,3 host+path

0,4 query string (without the ?)

0,5 selector existence (selectors are jQuery/CSS selectors, used to check for ERP-specific input elements)

The *match value* for options 0 to 4 should be a full case-insensitive text match, but can begin with a prefix:

!= matches text not equal to the match value

~ treats the match value as a regular expression, and matches text that matches the expression

!~ treats the match value as a regular expression, and matches text that does not match the expression

The *match value* for option 5 is a jQuery/CSS selector. If a match is found, the detection pattern passes.

All detects must pass for the rule set to be applied to the page.

For any page or frame that is detected, a webaction button and form are created based on the below commands. The button can be clicked to show the form.

Action Buttons

An action button is presented as a small UnForm icon or specified html somewhere on the browser screen, within the frame it is associated with. The default location of this button is the upper right corner, but the position and size can be adjusted with CSS style tags, and it can also be anchored to a selected element, to allow buttons to appear alongside specific page content. An action button is specified with the **webaction** command. There can be more than one button in a rule set, and all are presented on the page or frame. The button last clicked is the one submitted with the web form data.

```
webaction "name" [,style "button CSS styles"] [,beforebegin|afterbegin|beforeend|afterend "selector"] [,tip "tooltip" ] [,autorun] [, html "<tag... @ufaction>...</tag>"]
```

If no html option is present, an UnForm icon image is used. If html is provided, it must be a valid tag or tag set, such as "<button @ufaction>Click Here</button>". The string "@ufaction" should be present and is replaced by the extension at runtime with attributes required for identification and event handling. The code can otherwise be any html fragment, including nested tags.

Form Specification

A combination of fields and items are presented in a web form, either as a single page or with a tab-style interface. The elements of the form are defined with three commands:

Fields are text items that extract information from the current web page or frame. A jQuery/CSS selector is used to identify which field or screen element to read. If the element has a "value" attribute, such as an input element, that is used. If not, the innerHTML property is used, for example to get a table cell or span element's contents.

The specified attributes are added to the input tag, so should be a space-delimited list of HTML <input> attribute names and values.

```
webfield "name","selector"[,attributes]
```

If a selector is not found, then the field is not added to the form. Note if the selector needed contains a period (.), it should be escaped (\.) to avoid conflict with selector class lookups.

Items are addition input fields the user can interact with. An input widget is created of the type specified (input, textarea, radio, checkbox, select, button), with optional attributes to apply. Certain types support a list of options (radio, checkbox, select), which is a simple semi-colon delimited string. A button option submits all the fields to the same rule file and rule set, which can access the form data via the `cgi.data$` structure, or in individual fields as `cgi.field_name$`. The name of the button pressed is sent as `cgi.button$`.

The specified attributes are added to the associated tag, so should be a space-delimited list of HTML `<input>` attribute names and values.

```
webitem "name", input|textarea|radio|checkbox|select|button|file ["list"] [, attributes]
```

The file type of webitem allows the user to upload a local file to the UnForm server, for handling by the rule set associated with the current browser screen.

Panels add tabs to the form interface. Each webpanel command creates a new tab for the fields and items that follow it in the rule set.

```
webpanel "title"
```

Form Submission

When a web item of type "button" is pressed, or if the web action button contains an 'autorun' option, the form is submitted to the UnForm server, to process the active rule set. The rule set is explicitly specified via -f and -r arguments to an UnForm command line on the server. Code blocks, such as prejob, will execute with specific variables defined with form content, and the result can be either a PDF file generated by normal rule file processing, or the content of a `cgiresponse$` variable, which can contain any HTML text, messages, or script code.

More details are in the [code block response](#) page.

Notes

UnForm code block expressions are not supported in any of the commands. However, global and rule set constants (global, local, const commands) are supported, enabling parameterization of command elements, like style or attribute tags.

Due to parsing methods, quotes in the above commands are part of the syntax and must be double quotes ("). Portions passed to HTML can use single or double quotes, such as in attribute lists.

The extension loads all configured rule files from the UnForm sites when loaded, to limit the overhead imposed by the load process. If you are maintaining a rule file and need it reloaded, you can use the "Reload Site Rules" button on the extension options window.

CSS Selectors

CSS/jQuery selectors are used in detection, webaction, and webfield commands. A selector is used to locate a particular element in a web page's document object. Standard CSS selectors are well documented in many sites on the Internet, such as:

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors

https://www.w3schools.com/cssref/css_selectors.asp

https://en.wikipedia.org/wiki/Cascading_Style_Sheets#Selector

jQuery is the actual engine used by the Web Extension when identifying elements with selectors. jQuery supports standard CSS syntax along with a few extensions. You can find its selector documentation here:

<http://api.jquery.com/category/selectors/>

Special Selector Prefixes

The default context for a selector is the current window.document object. This is often the document of a frame of a larger web page. If you need to access another document object within the larger web page, such as the top level or a sibling frame, you can use one of these prefixes to the selector:

- parent:*selector*
- top:*selector*
- opener:*selector*
- \$(*value*):*selector*

The first three formats use the document object associated with the named window object. If the \$(*value*) syntax is used, a jQuery selector function is executed and the first matching returned jQuery object is used as the *selector* context.

12.10.3 Code Block Response

Once one of the button-stype webitem buttons is pressed, the equivalent of a web form is submitted to the UnForm server, to run the same rule set that contains the detect and button commands (-f and -r are used, along with any args options and a hard-coded "-p pdf"). The following values are available in the cgi\$ template:

cgi.action\$	The name of the webaction button that was pressed to load or autorun the web form.
cgi.button\$	The name of the button webitem that was pressed.
cgi.panel\$	Returns the name of the active panel when the button was pressed, or "*" if no panels are defined.
cgi.field_name\$	One field for each webfield and webitem in the form. Checkboxes can have multiple items in the value, each separated by chr(1) (\$01\$). Any spaces in the name of the associated webfield or webitem is replaced with underscores (a web field named "Doc ID" becomes cgi.field_doc_id\$).
cgi.dataurl_name\$	For file-type webitems, the file content is provided as a data url value, which can be decoded with the dataurldecode() function. Other dataurl* functions are also provided, documented in the internal functions table.

Use the prejob code block to interpret these items, and generate a cgiresponse\$ string value. This value is returned to the web extension for processing, based on the content:

- http: or https: prefix displays a URL in a new window, generally used to open a document or indexed view page
- error: *message text* (use \n or
 for line breaks) displays a message box titled "error"
- message: *message text* displays a message box titled "message"
- thumbnails + documents - a multi-line response (use \$0a\$ or chr(10) between lines) beginning with "thumbnails" presents clickable thumbnails of UnForm images, specified as additional lines with pipe-delimited syntax *library|doctype|docid|subid*
- multiview + documents - a multi-line response beginning with "multiview" presents a multiview frame with UnForm document or other pages, with UnForm documents identified with pipe-delimited syntax *library|doctype|docid|subid* (subid element optional), or as http/https url values
- other text renders in local browser window, and assumes HTML content

The [webapi object](#) can be used to generate http URL responses. Note these URL values can also be used in tags in a pure html response.

If no `cgireponse$` is available, the print job result is returned instead, allowing artificial print jobs to execute (by generating text pages in a prejob code block, and allowing normal rule set processing to handle the job). The print job will be in PDF format.

12.11 SDSI WEB EXTENSION

NOTICE: The SDSI Web Extension is based on Chrome technology known as Manifest V2. Chrome is phasing out MV2 extensions gradually, and it will no longer be supported at all in 2024. This stance is subject to change and is publicly documented here:

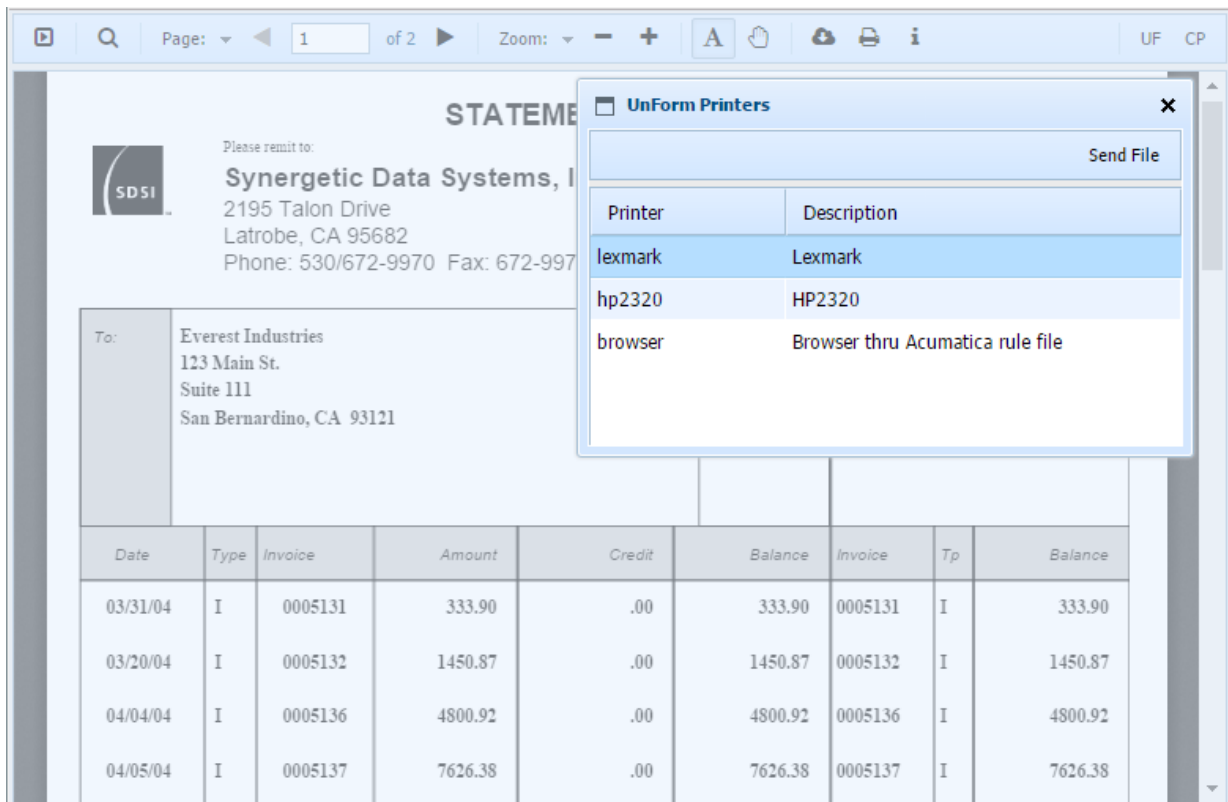
<https://developer.chrome.com/docs/extensions/mv3/mv2-sunset/>. SDSI has developed the [UnForm Web Extension](#) on Manifest V3. Among the features removed from MV3 is the ability to capture PDF files (and other data) as it is downloaded by the browser, so the UnForm Web Extension is not designed to display PDF files and offer a print through UnForm option from the custom viewer. However, it does offer the ability to capture standard browser printing, so there is still the capability of printing PDF files through UnForm, just via a different method. In addition, the UnForm Web Extension offers an enhanced feature set related to browser-based application integration. Users should plan to migrate to the UnForm Web Extension prior to the removal of the MV2-based SDSI Web Extension from the Chrome store.

The SDSI Web Extension provides an easy method to send PDF files to UnForm printers [configured](#) in the server's [webprinters] section or via the Server Manager on the Configuration, External Tools tab. The extension provides an enhanced PDF viewer in a compatible web browser. It is designed to streamline print workflows when using browser-based applications that produce reports in PDF format, by replacing the need to install local print capability to a remote printer.

The web extension also offers deep UnForm integration capability in browser-based applications. Using specially designed rule files, UnForm can detect specific pages on specific sites and add integration points that present the user with web forms or direct posting of data read from the browser application. For example, on an ERP web page displaying customer information, the web extension can use the customer ID to present a list of invoice documents, or while editing a purchase order, the vendor and purchase order information can be used to pull up archived documents such as inbound invoices and check records.

The web extension also provides integration with CirrusPrint, enabling a user to send a PDF file to any user or output device configured for the user's company. This enables immediate PDF distribution and cloud printing.

The web extension can be installed in the Chrome browser from the uniform.com website.



GET/POST Requests

The extension's custom PDF viewer works by intercepting web requests as soon as they start to arrive at the browser, by checking the response headers. If the response indicates it is a PDF from a [configured](#) site, the browser redirects the request to the custom viewer, causing the PDF to be retrieved again.

There are two ways that browsers can submit request, using GET or POST request types. POST types are normally used when submitting forms, and browsers generally warn you about re-submitting POST requests because, depending on how the target website was written, it may result in a double transaction. It is rare that a website will respond with a PDF to a transaction-oriented POST request, but since the possibility exists, the extension will not redirect PDF responses to POST requests except for sites specified in the server configurations. Instead of redirecting the request, the standard browser PDF viewer will be used.

12.11.1 Configuration

After installation, the extension must be configured.

The web extension is configured from within the browser's Extensions/Add-on's management page. In Chrome, go to the <chrome://extensions> page. The extension can be used strictly as an alternate PDF viewer, by checking "Use PDF viewer on all sites", but it also designed to provide enhanced functionality to both UnForm and CirrusPrint installations. You can add any number of UnForm or CirrusPrint servers to interact with. Click the Add button and fill out the Server Settings form, or select a server to edit an existing entry.

Each UnForm or CirrusPrint server can be configured to work with specific web sites or domains. When visiting a configured site, the PDF viewer toolbar presents a UF and/or CP integration button, enabling the user to submit the PDF file to the server.

Server Configuration

An UnForm server requires a server address, including the port and "/arc" path, as well as login and password information. You can optionally request that login information be sent to an email address associated with an UnForm login.

A CirrusPrint server requires a server address, but without a path. Login information as a browser user is required, typically including a company ID, such as john@sdsi. If you know the login, but not the password, you can request a setup link, which will be sent to the email address associated with the login. You can copy and paste that full link URL into the Setup Link field and press the Submit button. The login information will be retrieved and stored in the extension.

The Sites field is a list of URL's or domain names, one per line. The internal PDF viewer toolbar presents an UnForm or CirrusPrint integration button whenever visiting one of these sites. If the Sites field is empty, then the integration buttons are presented on all sites. Note that the extension will intercept POST requests for sites listed here.

A UF integration button presents a list of printers derived from the UnForm server's Web Printers configuration. Selecting a printer and pressing Send File will submit the PDF file as a print job to the selected printer.

A CP integration button presents a list of all company output devices and users. The PDF file can be sent to any of these destinations.

CirrusPrint/UnForm Servers		
Server	Type	Login Name
http://inhouse:2792/arc	uf	admin
https://demo.unform.com:27292/arc	uf	admin
https://demo.cirrusprint.com:8483	cp	allen@sdsi

Other Settings	
Language:	English ▼
Sites:	<input checked="" type="checkbox"/> Use PDF viewer on all sites If not checked, non-UnForm/CirrusPrint configured sites use default PDF viewer
Debug:	<input checked="" type="checkbox"/> Enable debug mode Adds messages to browser console
Reload:	<input type="button" value="Reload Site Rules"/> Reloads site rule files from UnForm servers

Server Settings	
Add	Delete
Save	
Enter UnForm or CirrusPrint server and login information. A UF or CP button can be added to the PDF viewer toolbar when viewing any, or selected, sites. The button will communicate with the server configured.	
Type:	<input checked="" type="radio"/> UnForm <input type="radio"/> CirrusPrint
Server Address:	http://inhouse:2792/arc
Login:	admin
Password: <input type="checkbox"/> Show password
Email Login:	<input type="text" value="User ID"/> <input type="text" value="Email address"/> <input type="button" value="Send Login"/>
You can request that the UnForm server send you login information by email. Enter either the User ID or an email address associated with your login, and press Send Login to receive information to the email address on file.	
Sites	<input type="text" value="http://svrsdsi/acumatica60"/> <input type="text" value="http://inhouse:2792"/>

12.11.2 Rule Files

Web extension rule files define screen integration elements for specific web pages. Any number of rule files can be configured, each listed in `uf101d.ini` in a `[webrulefiles]` section. This section is composed of lines in the format: *rulefile=URL-pattern*. For example, if you want a rule file named "classic-erp.rul" to be used whenever visiting the site "https://classic-erp.com/sdsi", use a line like this:

```
classic-erp.rul=https://classic-erp.com/sdsi
```

Note that the URL pattern is treated as a regular expression, enabling complex pattern matching in cases where there might be specific URL parameters or fragments to should apply.

Once a matching site is viewed in the browser, the configured rule file is applied to pages and frames the user navigates to. Frames are very common in browser-based applications, and each frame is treated as a unique page for integration purposes.

The default style of the popup web forms is a desktop style called "skyblue" in the normal UnForm browser interface. An alternate style, called "material" (designed based on Google's Material Skin principles for more device-independent usability) is also available. To specify the material style, append ";material" to any `[webrulefiles]` line, such as:

```
classic-erp.rul=https://classic-erp.com/sdsi;material
```

Detection

The standard UnForm detect statement is used to determine which rule set, if any, to apply to any given page or frame. The detection is evaluated in the following way:

```
detect x,y,"match value"
```

```
0,0 title + " " + url
```

```
0,1 title
```

```
0,2 url
```

```
0,3 host+path
```

```
0,4 query string (without the ?)
```

```
0,5 selector existence (selectors are jQuery/CSS selectors, used to check for ERP-specific input elements)
```

The *match value* for options 0 to 4 should be a full case-insensitive text match, but can begin with a prefix:

!= matches text not equal to the match value

~ treats the match value as a regular expression, and matches text that matches the expression

!~ treats the match value as a regular expression, and matches text that does not match the expression

The *match value* for option 5 is a jQuery/CSS selector. If a match is found, the detection pattern passes.

All detects must pass for the rule set to be applied to the page.

For any page or frame that is detected, a webaction button and form are created based on the below commands. The button can be clicked to show the form.

Action Buttons

An action button is presented as a small UnForm icon or specified html somewhere on the browser screen, within the frame it is associated with. The default location of this button is the upper right corner, but the position and size can be adjusted with CSS style tags, and it can also be anchored to a selected element, to allow buttons to appear alongside specific page content. An action button is specified with the **webaction** command. There can be more than one button in a rule set, and all are presented on the page or frame. The button last clicked is the one submitted with the web form data.

```
webaction "name" [,style "button CSS styles"] [,beforebegin|afterbegin|beforeend|afterend "selector"] [,tip "tooltip" ] [,autorun] [, html "<tag... @ufaction>...</tag>"]
```

If no html option is present, an UnForm icon image is used. If html is provided, it must be a valid tag or tag set, such as "<button @ufaction>Click Here</button>". The string "@ufaction" should be present and is replaced by the extension at runtime with attributes required for identification and event handling. The code can otherwise be any html fragment, including nested tags.

Form Specification

A combination of fields and items are presented in a web form, either as a single page or with a tab-style interface. The elements of the form are defined with three commands:

Fields are text items that extract information from the current web page or frame. A jQuery/CSS selector is used to identify which field or screen element to read. If the element has a "value" attribute, such as an input element, that is used. If not, the innerHTML property is used, for example to get a table cell or span element's contents. An alternative to a selector is an expression in curly braces, which is resolved as a string.

The specified attributes are added to the input tag, so should be a space-delimited list of HTML <input> attribute names and values.

```
webfield "name","selector{expression}"[,attributes]
```

If a selector is not found, then the field is not added to the form. If an expression is used, it is resolved via a function at runtime, and should resolve to a string. When using an expression, quotes and curly braces are required (i.e. "{top.location.href}").

Items are additional input fields the user can interact with. An input widget is created of the type specified (input, textarea, radio, checkbox, select, button), with optional attributes to apply. Certain types support a list of options (radio, checkbox, select), which is a simple semi-colon delimited string. A button option submits all the fields to the same rule file and rule set, which can access the form data via the cgi.data\$ structure, or in individual fields as cgi.field_name\$. The name of the button pressed is sent as cgi.button\$.

The specified attributes are added to the associated tag, so should be a space-delimited list of HTML <input> attribute names and values.

```
webitem "name", input|textarea|radio|checkbox|select|button ["list"] [, attributes]
```

Panels add tabs to the form interface. Each webpanel command creates a new tab for the fields and items that follow it in the rule set.

```
webpanel "title"
```

Form Submission

When a web item of type "button" is pressed, or if the web action button contains an 'autorun' option, the form is submitted to the UnForm server, to process the active rule set. The rule set is explicitly specified via -f and -r arguments to an UnForm command line on the server. Code blocks, such as prejob, will execute with specific variables defined with form content, and the result can be either a PDF file generated by normal rule file processing, or the content of a cgiresponse\$ variable, which can contain any HTML text, messages, or script code.

More details are in the [code block response](#) page.

Notes

Expressions are not supported in any of the commands. However, global and rule set constants (global, local, const commands) are supported, enabling parameterization of command elements, like style or attribute tags.

Due to parsing methods, quotes in the above commands are part of the syntax and must be double quotes ("). Portions passed to HTML can use single or double quotes, such as in attribute lists.

The extension loads all configured rule files from the UnForm sites when loaded, to limit the overhead imposed by the load process. If you are maintaining a rule file and need it reloaded, you can use the "Reload Site Rules" button on the extension options window.

CSS Selectors

CSS/jQuery selectors are used in detection, webaction, and webfield commands. A selector is used to locate a particular element in a web page's document object. Standard CSS selectors are well documented in many sites on the Internet, such as:

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors

https://www.w3schools.com/cssref/css_selectors.asp

https://en.wikipedia.org/wiki/Cascading_Style_Sheets#Selector

jQuery is the actual engine used by the Web Extension when identifying elements with selectors. jQuery supports standard CSS syntax along with a few extensions. You can find its selector documentation here:

<http://api.jquery.com/category/selectors/>

Special Selector Prefixes

The default context for a selector is the current window.document object. This is often the document of a frame of a larger web page. If you need to access another document object within the larger web page, such as the top level or a sibling frame, you can use one of these prefixes to the selector:

- parent:*selector*
- top:*selector*
- opener:*selector*
- {*expression*}:*selector*

The first three formats use the document object associated with the named window object. If the expression format is used, the expression should resolve to a Document object.

12.11.3 Code Block Response

Once one of the button-type webitem buttons is pressed, the equivalent of a web form is submitted to the UnForm server, to run the same rule set that contains the detect and button commands (-f and -r are used, along with any args options and a hard-coded "-p pdf"). The following values are available in the cgi\$ template:

cgi.action\$	The name of the webaction button that was pressed to load or autorun the web form.
cgi.button\$	The name of the button webitem that was pressed.
cgi.panel\$	Returns the name of the active panel when the button was pressed, or "" if no panels are defined.
cgi.field_name\$	One field for each webfield and webitem in the form. Checkboxes can have multiple items in the value, each separated by chr(1) (\$01\$). Any spaces in the name of the associated webfield or webitem is replaced with underscores (a web field named "Doc ID" becomes cgi.field_doc_id\$).

Use the prejob code block to interpret these items, and generate a cgiresponse\$ string value. This value is returned to the web extension for processing, based on the content:

- http: or https: prefix displays a URL in a new tab
- error: *message text* (use \n or
 for line breaks) displays a message box titled "error", capitalized the same was as the prefix.
- message: *message text* displays a message box titled "message".
- other text renders in local browser window, and assumes HTML content

The webapi object can be used to generate http responses. Note these http values can also be used in tags in a pure html response.

If no cgiresponse\$ is available, the print job result is returned instead, allowing artificial print jobs to execute (by generating text pages in a prejob code block, and allowing normal rule set processing to handle the job). The print job will be in PDF format.

12.12 DESKTOP DELIVERY AND FORMS

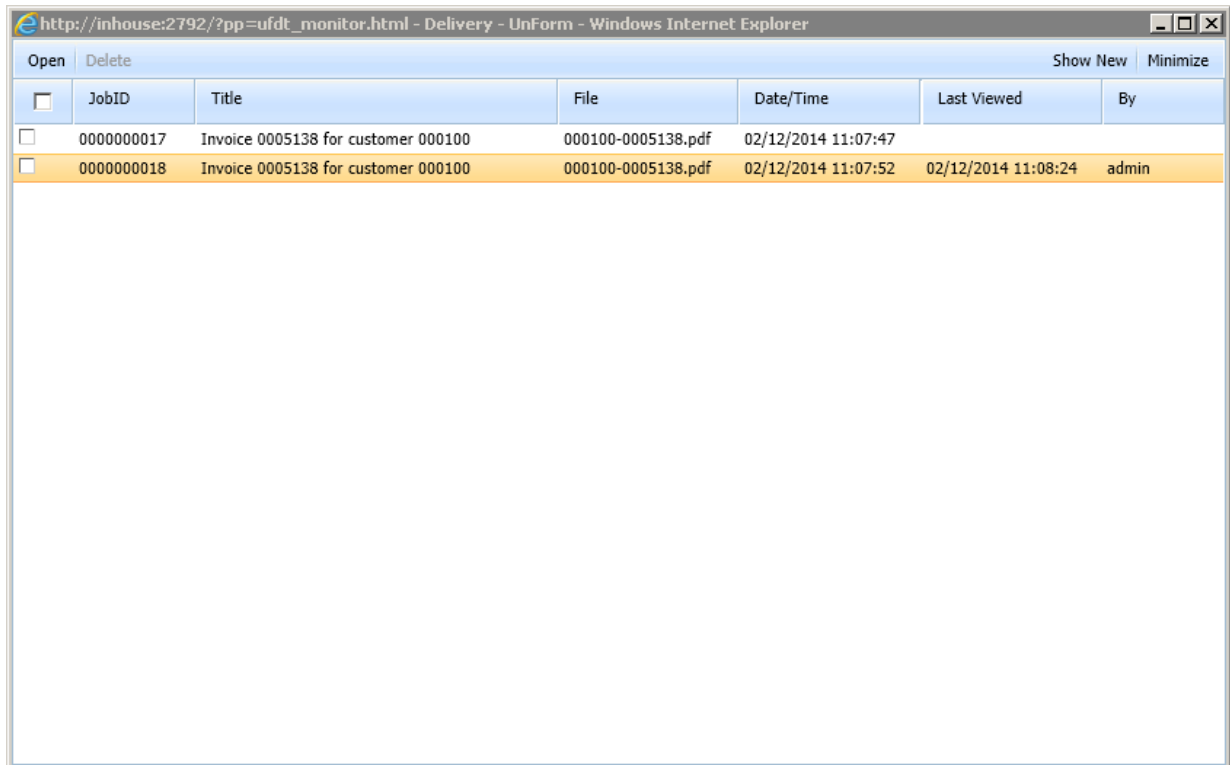
UnForm 10.1 provides desktop delivery features that enable users on the network where the UnForm server runs to receive documents or fill out forms at the time UnForm jobs run. The feature is accessed via a web browser, either via the UnForm server's web portal, or directly via a URL.

http://servername:port	Presents a portal page with links to the delivery monitor.
http://servername:port/arc?pp=ufdt_monitor.html	Directly open the delivery browser, which displays documents waiting for the user, and monitors for form requests. This page can be minimized to provide a small display that shows document counts as buttons to open existing and new document lists.

Optionally, you can supply a query string suffix "&ips=*uniqueid*" to the above URL structures. The *uniqueid* value can be something that will add uniqueness to the IP address that the browser monitors

with, supporting Terminal Server environments where the same IP address would be used for all users. Typically there is an environment variable, such as %SESSIONNAME%, that identifies a particular session on the server. By passing this information to both the URL and UnForm job submissions, deliveries and forms can be sent to the correct user session.

Documents are pushed to a desktop delivery session via the `dt-del()` and `dt-form()` code block functions. For an example of these functions, view the `dt-del.rul` and `dt-form.rul` sample rule files.



The screenshot shows a web browser window titled "http://inhouse:2792/?pp=ufdt_monitor.html - Delivery - UnForm - Windows Internet Explorer". The browser displays a table with the following columns: JobID, Title, File, Date/Time, Last Viewed, and By. There are two rows of data, both highlighted in yellow.

JobID	Title	File	Date/Time	Last Viewed	By
0000000017	Invoice 0005138 for customer 000100	000100-0005138.pdf	02/12/2014 11:07:47		
0000000018	Invoice 0005138 for customer 000100	000100-0005138.pdf	02/12/2014 11:07:52	02/12/2014 11:08:24	admin

As with any access to the UnForm server, when users connect to the server, they must login unless they have an active session.

12.12.1 Desktop Delivery

The delivery of documents is performed by the `dt-del()` code block function. When this function is executed, a copy of the file to deliver is stored securely on the server. The desktop browser and monitor poll the server for new documents to display, and when new documents are available, they are listed as available or are immediately displayed.

Documents are not stored indefinitely. The system purges documents based on the `dtage` and `dtviewage` parameters in the [defaults] section of `uf101d.ini`. These values specify the number of days unviewed documents, and viewed documents, are maintained, respectively.

The `dt-del` function has the following syntax:

```
dt-del(filename$,title$,userid$,ip$[,style[,errmsg$]])
```

The `filename$` argument can be either a file that resides on the UnForm server or a message in the format of `"msg:message text"`. It will typically be a PDF file that was generated using a `jobexec()` command, or it could be the job's output file, delivered in a postdevice code block. If it is a message, the text is merged with the message template `"http/files/msg.html"`, and it is always treated as a popup (`style=1`). As the popup is based on an HTML template, you can embed HTML code in the *message text*. You can also use `"\n"` as a synonym for `"
"` as a convenience.

The `title$` argument is a document title that will display in the delivery browser.

The `userid$` and/or `ip$` arguments are used to identify which user should receive the document. The preferred method is to specify a `userid$` value. If no user ID is specified, then an IP address can be used. Only a browser logged in as the specified user, and/or connected from that IP address will be notified of the delivery and will be able to access it. The IP address reported in `uf.clientip$` is typically the IP address of the computer that submitted the UnForm job, though in some circumstances it will be the address of a server computer rather than a user computer.

As IP addresses can change if DHCP is used or static IPs are re-assigned, or may not be available if the user is accessing the server via a router with NAT translation, care must be taken when using an IP address. They are suitable in local networks for popup styles of delivery, assuming that purge times (`dtage` and `dtviewage`) settings are short.

A suffix can be appended to the `ip$` value. This suffix must match the value used in an `ips=uniqueid` query string in the browser monitor launch. For example: `uf.clientip$+clientenv("SESSIONNAME")`.

The `style` argument can be 0 or 1. A value of 0 indicates the monitor and browser windows will display the presence of the delivery. A value of 1 indicates that the browser will immediately display the document as well. If the `style` argument is not supplied, 0 is assumed.

The `errmsg$` variable will return an error message if an error occurs while storing the document. It will return null (`""`) if no error occurs.

12.12.2 Desktop Forms

Desktop form support is configured the same way as desktop delivery. The same HTTP server and clients are used. Unlike desktop delivery, which will store documents for when the user logs in, it is critical that the user be connected when the form is to be presented.

The form is launched with the `dtform()` code block function. The user specified is notified that a form is ready to be presented, and he or she can accept or cancel the form. If the user neither accepts nor cancels the request within a specified amount of time, the request times out.

Data is sent to the form, and returned from the form, using a URL-encoded data string. There are several functions provided to manage this string.

The form itself is an HTML document, stored anywhere in the browser interface directories, such as under `web/en-us/forms`. The form must an `action=` attribute of the UnForm CGI script name, and have the following fields defined:

```
<input type="hidden" name="cancel" value="0">
<input type="hidden" name="rest" value="dtputform">
[jobno hidden]
```

In addition, any standard HTML form widgets can be used, including text boxes, text areas, radio buttons, checkboxes, and selection lists. A submit button must be provided, and an optional Cancel button can be used, which must set the value of the "cancel" field to 1 when submitting the form.

An example form, "emailform.html", is found in the web/en-us/forms directory, and a sample rule file samples/dtform.rul is provided using this form.

Note that the required fields and form location are different in version 9 from previous releases. so you must update existing forms to follow the new specification.

To provide default values for form fields, specify the named value(s) in the data string argument of the dtform command, and include `~name~` tags in the HTML document. For example an input field for a To address might look like this:

```
<input type="text" size="30" name="to" value="~to~">
```

If the data string supplied to the dtform function contains to=someone@somewhere.com, then that email address will be presented as the default value when the form is displayed.

The dtform function has the following syntax:

```
dtform(formname$,title$,userid$,ip$,datastr$,response[,timeout[,errmsg$]])
```

The formname\$ value is the name of the html form file, which can be a full path or a name found in the standard browser interface search paths.

The title\$ value is what is shown to the user when notified that the form needs to be displayed.

The userid\$ and/or ip\$ arguments are used to identify which user should receive the document. The preferred method is to specify a userid\$ value. If no user ID is specified, then an IP address can be used.

Only a browser logged in as the specified user, and/or connected from that IP address will be notified of the delivery and will be able to access it. The IP address reported in uf.clientip\$ is typically the IP address of the computer that submitted the UnForm job, though in some circumstances it will be the address of a server computer rather than a user computer.

A suffix can be appended to the ip\$ value. This suffix must match the value used in an ips=*uniqueid* query string in the browser monitor launch. For example: uf.clientip\$+clientenv("SESSIONNAME").

The datastr\$ is a URL-encoded string with form field values defined as name=value pairs as in normal web programming. It must be a string variable in order to receive form values back, which can then be decoded using the urlgetfld() function. If the string contains values when dtform is executed, those values are used in the form, wherever a `~name~` tag is found. To create the string with URL-encoded name=value pairs, use the urlsetfld() function.

The response variable returns one of these codes:

- 0 indicating the form was submitted
- 1 indicating the form was cancelled
- 2 indicating the form request timed out
- 3 indicating the user refused the form

Any non-0 responses are logged in the server log file. Rule sets that use the dttdel() function should query and react to non-0 responses as appropriate.

The timeout value is the number of seconds the user has to respond to the form request. Once the user accepts the form, they may take as long as needed to complete the form. However, the job will be halted waiting for the form submission, so users must understand that forms they accept should be submitted as soon as possible. If the user doesn't accept the form within the specified number of seconds, a timeout response will be provided. The default timeout value is 30 seconds.

If a timeout of -1 is specified, then the form request step is skipped, and the form is displayed automatically. If no monitor is running or the user does not respond to the form, the job will be hung, so do not use this option unless you know the monitor is active and the user is available. This step might be used as an immediate follow up to a previous form that the user did respond to.

If an unexpected error occurs, it will be returned in `errmsg$`, if provided in the function arguments.

URL-encoded strings are comprised of name-value pairs with special character encoding. Use the following functions to create or parse the URL-encoded data string:

<code>urlgetfld(datastr\$,name\$)</code>	Returns the value of the <code>name\$</code> field. The value is returned without URL encoding. <code>mailto\$=urlgetfld(datastring\$,"to")</code>
<code>urlsetfld(datastr\$,name\$,value\$)</code>	Returns a URL-encoded string with the field <code>name\$</code> set to <code>value\$</code> . The field is added if necessary. <code>datastring\$=urlsetfld(datastring\$,"to",someone@somewhere.com)</code>
<code>urldelflds(datastr\$,names\$)</code>	Returns the a URL-encoded string after removing the fields specified in <code>name\$</code> from the URL-encoded string <code>datastr\$</code> . Multiple fields can be separated by commas. <code>datastring\$=urldelflds(datastring\$,"to,from,subject,body")</code>
<code>urlgetnames\$(datastr\$)</code>	Returns a list of field names in the data string. <code>fldlist\$=urlgetnames\$(datastring\$)</code> <code>count=parsec(fldlist\$,"(',')')</code>

12.13 DESKTOP CLIENT

The UnForm Desktop Client (DTC) client is an optional Windows application that provides streamlined access to UnForm document management facilities from a user's Windows desktop. DTC communicates with UnForm via the internal HTTP server. It downloads a specialized rule file that controls its processing.

The primary purpose of DTC is as a monitor that watches for windows to appear and gain focus, present buttons related to those windows, and to submit data to the uniform server to retrieve documents or perform other actions related to the uniform server.

The configuration of DTC includes a user login and password, which is used to submit the data for rule set processing. Note this does not replicate to the user's browser, so separate session logins are required when a job launches a browser window.

The rule file contains rule sets, which are composed of three things: detection statements, button and panel commands, and a prejob code block that is executed when the user clicks a button.

When the rule file is loaded by DTC, it then begins monitoring for window focus changes on the desktop. When a window passes detection for a rule set, a small, user-sizable window is displayed with the defined buttons. When the user clicks a button, a job is run on the server, which executes the rule set and returns the value of `cgiresponse$` to DTC, which then displays the response in one of several ways.

Note that since detection can be based on user rather than window, it is also possible to construct an interface that always displays regardless of what window currently has focus on the user's system.

12.13.1 Deployment

DTC can be installed directly from the server's "dtcinst" folder, by running the `ufdtc_setup.msi` file. It can also be installed with a browser via the internal HTTP server portal, using the server's address and HTTP port, such as `http://192.168.1.10:27402`. Note this is one level higher than the normal `/arc` path to the server.

Once run, DTC will attempt to communicate with the server, but will need to be configured with a login and password, and an optional rule file. Right-click the "uf" DTC system tray icon to configure. A default rule file can be configured in the `[dtc]` section of `uf101d.ini`, setting `airule=rulefile`.

12.13.2 DTC Rule Sets

Desktop Client rule sets are similar to print job rule sets in many respects, but are limited in structure to a small number of commands. Other commands are ignored.

DTC rule sets require three things: detection logic, button definitions, and a prejob code block that creates responses to buttons that are pressed by the user and data that is submitted from DTC. Detection statements are used to specify which applications and/or window titles to monitor as focus changes between applications and windows on the user's workstation. As the active window or application changes, different rule sets become active and an associated DTC application window will appear. The contents of that window are controlled by other rule set commands: `dtcbutton`, `dtchelpfile`, and `dtcpanel`. These commands are used to construct and present a user interface associated with a window or application. The user can then copy, paste, or type data into the button text fields, and when a given button is pressed, the rule set is executed on the server, and the prejob code block is executed.

The syntax for the above mentioned rule set commands is as follows:

12.13.2.1 Detect

Detect 0, *item*, "[^][~]match text"

- 0,0 tests space separated exe name, window title, and UnForm login name
- 0,1 tests window title
- 0,2 tests exe name
- 0,3 tests UnForm user login
- 0,4 tests Windows domain\user

A prefix on the detection text can specify a case-insensitive match, regular expression match, or case-insensitive regular expression match:

- ^ case insensitive text match
- ~ case-sensitive regular expression
- ^~ case-insensitive regular expression

Each time window focus changes on the user's system, detection is processed for all rule sets. Any rule sets that pass detection cause presentation of that set's application popup window. It is possible for multiple windows to be displayed at the same time.

12.13.2.2 Title

Title "*window title*"

The title command can be used to specify the application popup window title. The title defaults to the rule set name.

12.13.2.3 DTCPanel

DTCPanel "*title*"

The dtcpanel command names a tab panel for subsequent button commands. When a form is submitted, panel titles become section headers for the cgi.data\$ field, which is an INI file structured string. If no panel commands are present, a single un-tabbed panel is presented, and the cgi.data\$ field contains a single section header, [*].

12.13.2.4 DTCHelpfile

DTCHelpFile "*filename*"

If present, the DTC window will offer a help toolbar button. The purpose is to allow integrator-generated help content associated with the rule set's DTC window. The filename should be an HTML file available on the UnForm server. The file is loaded into a browser control.

12.13.2.5 DTCButton

DTCButton "*title*" [,style clipboard|nbclipboard|text|nbtext|button|title|nbtitle] [,args "*job arguments*"] [,width *chars*] [,match "*regex*"] [,library "*name*"] [,doctype "*value*"] [,parsevalue ["*ruleset*"]]

The DTCbutton command is interpreted by DTC for presentation and action upon click. Note that expressions are not supported, only literal values.

Style

This optional argument defines the style of the button provided to the user. The default style is "clipboard".

Clipboard	Provides a text box and a small submit button, and monitors the clipboard for changes to place text in the text box.
Nbclipboard	Provides a text box, but no submit button. It monitors the clipboard for changes to place in the text box.
Text	Provides a text box and a small submit button. This is intended for simple user entry.
Nbtext	Provides a text box, but no submit button.
Button	Provides a submit button with the title as the button caption.

Title	Provides a text box and a small submit button, and monitors the window title for changes to place in the text box.
Nbtitle	Provides a text box, but no submit button. It monitors the window title for changes to place in the text box.

Args

Options passed to the to the UnForm job running the rule set. There are automatic rule file and rule set arguments (-f and -r, respectively) to ensure the rule set with the button configuration is the one executed when a submission takes place.

Width

This defines the width of the button, in nominal characters. Without a width, the title width is used. Use this to ensure consistent widths when multiple buttons are presented.

Match

The match option is honored by the clipboard and nbclipboard styles. The clipboard value must match the specified regular expression in order to be pasted into the text field.

Library, Doctype

These two options in tandem are honored by the clipboard and nbclipboard styles. The clipboard value must be a valid document ID within the library and doctype specified in order to be pasted into the text field.

ParseValue

If this option is present, the value from the clipboard or window title is sent to the server for parsing. The server will run the current rule set or the optionally specified one, and use the value returned from the server in cgiresponse\$ in place of the clipboard or title value. The rule set receives cgi.button\$, cgi.panel\$, and cgi.parsevalue\$ when the request is sent.

12.13.2.6 Code Block Response For Buttons

Once one of the buttons is pressed, the equivalent of a web form is submitted to the UnForm server, to run the same rule set that contains the detect and button commands (-f and -r are used, along with any args options and a hard-coded "-p pdf"). The following values are available in the cgi\$ template:

cgi.button\$	Returns button title, indicating which button was pressed.
cgi.selected\$	Returns text of a text field associated with the button.
cgi.panel\$	Returns the name of the active panel when the button was pressed, or "" if no panels are defined.
cgi.data\$	<p>Returns all the panels and text fields in an INI file format. Panel titles are used as section headers, and text and clipboard field text boxes are returned in name=value format. If no panels are provided, a single section header [*] is supplied.</p> <p>This format allows multiple fields of data to be submitted and interpreted by the rule set, by using the getinival() function, passing cgi.data\$ as the first argument. For example, name\$=getinival(cgi.data\$, "CustPanel", "Name") would set name\$ to the value of the Name field in the CustPanel panel.</p>
cgi.parsevalue\$	Returns the value of the clipboard or window title in cases where the parsevalue option is specified. Note that if this value is present, DTC is

	submitting a request for reformatting of this data. It is not a result of the user pressing a DTC submit button. No <code>cgi.selected\$</code> or <code>cgi.data\$</code> is sent.
--	---

Use the prejob code block to interpret these items, and generate a `cgiresponse$` string value. This value is returned to DTC for processing. DTC interprets the response in these ways:

- `http:` or `https:` prefix performs shell launch to display URL in the default browser
- `error:` message text (use `\n` for CRLF) displays an error style message box
- `message:` message text (use `\n` for CRLF) displays an information style message box
- other text renders in local browser control, and assumes HTML content

The `webapi` object can be used to generate http responses. Note these http values can also be used in `` tags in a pure html response.

If no `cgireponse$` is available, the print job result is returned instead, allowing artificial print jobs to execute (by generating text pages in a prejob code block, and allowing normal rule set processing to handle the job). The print job will be in PDF format.

12.13.2.7 Code Block Response For ParseValue Requests

If a clipboard or title button is configured with the `parsevalue` option, DTC will submit that value to the server for parsing.

The following values are available in the `cgi$` template:

<code>cgi.button\$</code>	Contains the button title, indicating which button contains the <code>parsevalue</code> option.
<code>cgi.panel\$</code>	Contains the name of the active panel, or <code>"**"</code> if no panels are defined.
<code>cgi.parsevalue\$</code>	Contains the value of the clipboard or window title in cases where the <code>parsevalue</code> option is specified.

Use the prejob code block to interpret these items, and generate a `cgiresponse$` string value. This value is returned to DTC for processing. DTC interprets the response in these ways:

- `error:` message text (use `\n` for CRLF) displays an error style message box
- other text is used to replace the original value from the clipboard or window title

13 RULE FILES

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

rule set to work with based on either a command line option (-r), or **detect** commands contained in the rule set.

The region in a rule file above the first rule set is known as the *global area*. This area can contain global constants (defined with the **const** or **global** commands) that apply to all rule sets, and global **merge** commands, which are internally loaded at the top of all rule sets in the file.

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

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

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

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

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

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

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

13.1 Content-based Rule Sets

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

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

13.2 ACROSS

Syntax

```
across n [,gap]
```

Description

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

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

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

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

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

Drivers: pcl, pdf, ps

[AFO](#) jobs not supported.

13.3 ANNOTATE, CANNOTATE

Syntax

1. `annotate col{numexpr}, row{numexpr}, cols{numexpr}, rows{numexpr}, "msg | url"{expr}, [text|link|stamp] [,name name{expr}] [,title title{expr}] [,width width] [,color colname] [,rgb rrggbb] [,opacity opacity] [,style style]`

2. `annotate "text|!=text|~regex|!~regex[@left,top,right.bottom]", col{numexpr}, row{numexpr}, cols{numexpr}, rows{numexpr}, "msg | url"{expr}, [text|link|stamp] [,name "name"{expr}] [,title title{expr}] [,width width] [,color colname] [,rgb rrggbb] [,opacity opacity] [,style style]`

Description

This PDF-only command adds an annotation element at the specified position and size. Three types are supported: text, link, and stamp. The default is text.

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

If syntax 2 is used, then the region is defined relative to any occurrence of the *text*, or of text that matches the regular expression *regex*. 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 *regex* 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 *regex*, it is necessary to specify "\@".

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

Text annotations display as an icon, which if clicked will open a text window displaying the message with the optional title in the title bar. The border can be controlled by the style, color, and width options. The icon can be set with the case-sensitive name option: Comment, Help, Insert, Key, NewParagraph, Note, or Paragraph. The default icon is Note. To ensure the case is maintained, the *name* should be enclosed in quotes.

Link annotations perform an action when clicked. The action is defined in the message. It can be a URL, such as `http://abc.com` or `mailto:sales@abc.com`, or it can be a Javascript action, entered as `javascript:script code`. Note the **javascript** command can be used to add functions at a document level which can be used in annotation actions.

Stamp annotations place a stamp rather than an icon on the form. When clicked, the message is displayed as in a text annotation. The stamp image shown is identified by the case-sensitive name value, which can be one of these:

- Approved
- Experimental
- NotApproved
- AsIs
- Expired
- NotForPublicRelease
- Confidential
- Final
- Sold
- Departmental
- ForComment
- TopSecret
- Draft
- ForPublicRelease

Style and Width values apply to the annotation border. The style can be S (solid), D (dashed), B (beveled), I (inset), or U (underline). Width is expressed in pixels at the current dpi.

Examples

annotate 70.5,64,10,2.25,"http://acme.com/docs/terms.htm",link,style U, title "Click to view our Terms and Conditions"

annotate "TOTAL:",0,1,15,1.5,"Please contact our credit department at 555-123-4567 for more information", title "Credit Terms?",name "Help"

Drivers: pdf

13.4 ARCHIVE

Syntax

```
archive "libpath" {expr} , "doctype" {expr} , "docid" {expr} [, subid subid {expr}] [, title title {expr}] [, notes notes {expr}] [, keywords kws {expr}] [, categories|cats cats {expr}] [, link|links links] [, entityid|entid entity ID {expr}] [, args args {expr}] [, dtm|date yyyyymmddhhmmss {expr}] [, subtitle subtitle {expr}] [, subdtm|subdate yyyyymmddhhmmss {expr}] [, server "name[:port]" {expr}] [, flowid "flowid" {expr}] [, docdata "docdatavals" {expr}]
```

Description

This command causes UnForm to add two versions of the current document to the library specified. The first document is a PDF version formatted as the current rule set specifies, the second is the input stream

text from which the PDF document was generated. The PDF version has a default sub ID of "@uniform", but this can be overridden by specifying a *subid*. The sub ID of the text version is "@text".

Note that as the formatted document is generated as a PDF, the rule set must be designed to successfully produce PDF output. In particular, any images or attachments need to be available in PDF format, or be designed to use automatic image conversion.

The library, doctype, and docid elements of the archive command are evaluated as each page of the job is printed. If they change, then a sub-job is executed using the pages to that point as input. In this manner, a batch job with multiple documents can be archived as multiple documents rather than as a single large document. For example, if an invoice run is processed, and the *docid* is derived from the invoice number, then each new invoice number during the job will produce a new document in the archive.

The library is a path name on the UnForm server. If it doesn't exist, it will be created and a library pointer record will be created. If it isn't a full path, the library is created using the full path of the "arc" directory under the UnForm server, such as "/usr/lib/uniform90/arc/library".

An archive document is identified in a library by the document type *doctype* and document ID *docid*. A document can also contain further information: title, notes, keywords, date and time, and categories. Further, each document can contain multiple versions identified by a *subid*, each of which contains a title and date and time. See the **Archiving and Document Management** chapter more information about each of the archive elements.

If any archive command elements are not supplied, the following defaults are used:

- The document type is set to the rule set name
- The document ID is set to a unique date/time stamp
- The title is set to the value of the **title** command, if any, or is derived from text input
- Keywords are derived from unique words in the content, up to a default limit on the number of words
- The date and time is set to the current date and time
- Command line arguments, such as `-arclib` or `-arcdotype`, supply remaining defaults

If the *subid* ends with an asterisk, such as "Formatted*", then UnForm will not overwrite duplicate sub ID values. Instead, a 5-digit sequence will be added to ensure up to 99,999 versions of a document can be added.

If the keywords value begins with "*,", the * is replaced with auto-derived keywords based on content, so you can have both auto and custom keywords using this structure. The keywords parameter can also be the word "all", or a number, indicating the maximum number of keywords to calculate (-1 means the same as 'all', and 0 means no keywords should be calculated).

Categories should be structured with vertical bars separating segments and semi-colons separating categories. For example: "CustPO|"+custname\$+"|"+custpo\$ + ";" + "Salesperson|"+slspid\$. There can be any number of categories, and each category can contain up to ten segments. A category index ends at a null segment. The browser interface treats a null segment as a signal to display document records. It is therefore improper to have an empty segment in the middle of a category sequence (i.e CustPO| 12345). If you have empty data and still would like to index this, replace the null with an "!", which will sort lower than any other viewable data and provide a visual indicator of missing data as well.

Links provide outbound linking to other documents, within or without the archive system. This value is a semi-colon delimited list of links, each of which can be in one of the following formats:

- A full URL, optionally matching a URL used to load a document or image from a library, or a URL to an outside page or document. This structure, if it begins with `http://` or `ftp://`, can be prefixed with a title in

the format of *title=URL*. If the title is specified, that becomes the visible link in the browser.

- A simplified pipe-delimited structure of *library|doctype|docid[|subid]*, which is displayed in the browser interface as a URL link to the document or image named by library, document type, document ID, and optionally image sub ID.

There can be any number of links in the list.

An entity ID can be set to tie this document to a particular code that can be used to filter access in the browser interface. A user login can be assigned to an entity ID, and that user can only view documents with a matching entity ID.

The args option can be used to specify UnForm command line arguments to pass to the sub-job used to generate the archive PDF file. For example, if you only want to archive copy 1 of a job, you could pass "-ce 1" (copies enabled 1).

The server option, if specified, overrides a default server specified in the [archive] section of uf101d.ini. This value is used to forward archive data to the remote server for library storage, useful in sites that maintain multiple UnForm servers for disaster recovery or load balancing. A single archiving server can be configured to ensure that library data does not get scattered among several machines.

The flowid option is used to create a docflow record when the PDF document is stored in the library. The argument must specify a previously created docflow definition.

The docdata option specifies custom name-value pairs in *name=value;name2=value2* syntax. It is used extensively by UnForm internally, for Image Manager and DocFlow data. It is intended for programmatic use, and the data is not user-editable.

If more than one archive command is present in a rule set, then archives are generated for all of them. For example, a second archive command might be added to produce a full job archive in addition to archives for individual documents. Note this differs from version 7.x, where only one archive command was honored.

Examples

```
archive "demo_accounting","ApAging"
```

This first example simply archives the A/P aging report to the demo_accounting library, under the document type "ApAging". The document ID will be automatically generated as a 10-digit sequential number, and the entire job is archived as a single document. The title and keywords are derived automatically from the content.

```
archive "demo_sales","ArStatement",{arcid$},title {arctitle$},cats {arccats$}, args "-ce 1"
```

This example archives statements to the demo_sales library. The document ID, title, and categories are expressions derived from code block variables. The sub-job that generates the PDF document will have a "-ce 1" command line argument, which enables copy 1 only, so the archived copy will only be of the rule set's first copy. The sample rule file arcdemo.rul contains the full Statement rule set where this example comes from.

Drivers: pcl, pdf, ps

13.5 ATTACH

Syntax

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

Description

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

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

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

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

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

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

PCL Attachments

An easy way to create an attachment file is to use a Windows workstation and install a PCL5 type printer (not a PCL6 or PCL/XL driver, which will produce the wrong type of format). Set the port for the printer to FILE:. Then create the attachment using any word processor and print to that printer. Windows will ask for a file name, and when printing is complete, the resulting file is suitable for use as an attachment. If your document contains fonts that are not present in the printer you will be using, be sure to modify the print driver to print True Type Fonts as graphics, if possible.

PostScript Attachments

PostScript attachments are rendered simply as full page images, meaning the file can either be an EPS file or a JPG file (JPG files are only supported by color printers). UnForm simply prints the image, scaled to the printable region of the page.

PDF Attachments

UnForm attaches PDF documents by merging the objects on page one with those of the current page of output. Objects are placed in the exact same position and size as found in the attached document.

UnForm 10.1 supports PDF files up to version 1.4 (introduced with Acrobat 5). Some files that specify a later revision are still compatible, but new a file structure element was added at PDF 1.5 that is not supported by UnForm. Specifically, the unsupported feature is called an *Object Stream*. Former versions of UnForm did not support Linearized (also known as Optimized or Fast Web View) PDF files, nor files with incremental updates. Version 10.1 now supports these formats.

To create an attachment, use a PDF printer or other method to save a document in PDF format, choosing, if possible, to generate a file compatible with Acrobat 5 or below, or version 1.4 or below. There are many free and commercial tools available to produce PDF files, from Adobe and other vendors. Of note, Microsoft Office supports a "Save As" PDF feature with an Add-in that can be downloaded from Microsoft's web site.

PDF files are often available from third parties or government entities, and many of these are compatible with UnForm. Sometimes these files are designed with unusual page sizes or internal offsets that cause their elements to be in unexpected positions. UnForm is unable to re-position objects, so such files might need to be re-created using a PDF printer or other tool that will realign the objects using normal page dimensions.

Note that the object merging technique can cause issues when a landscape attachment file is designed to perform landscape formatting by rotating a portrait page, as UnForm executes landscape via a landscape page size rather than rotation. The result is a fundamental incompatibility between the two documents. To work around this, consider using UnForm commands to produce the document, or use the image command with a 'page *n*' option, supported when Ghostscript is available and configured.

Drivers: pcl, pdf, ps

13.6 AUTHOR

Syntax

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

Description

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

Drivers: pdf only

13.7 BARCODE (PCL,PDF, PS)

Syntax

1. barcode *col*{*numexpr*}, *row*{*numexpr*}, "*value*"{*expr*}, *symbology*, *height*, *spc-pixels* [, *text*] [, rotate *degrees*] [, *start char*] [, *stop char*] [, *truncate*] [, *mode mode*] [, *cols cols*] [, *rows rows*][, *notrim*] [, *security|errorlevel n*] [, *symbolno n*] [, *totsymbol n*] [, *notbc*]
2. barcode "*text*"~*regexpr*!=*text*!~*regexpr*[@*left*,*top*,*right*,*bottom*]", *col*{*numexpr*}, *row*{*numexpr*}, "", *symbology*, *height*, *spc-pixels*, *getoffset cols*, *getcols cols* [, *eraseoffset cols*] [, *erasescols cols*] [, *text*] [, rotate *degrees*] [, *start char*] [, *stop char*] [, *truncate*] [, *mode mode*] [, *cols cols*] [, *rows rows*][, *notrim*] [, *security|errorlevel n*] [, *symbolno n*] [, *totsymbol n*] [, *notbc*]

Description

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

value is a text string, up to 28 characters, to barcode (when using the Support Server, there is no limit on characters). Often this is symbology-dependent. If check digits are required, they are generated internally in UnForm. Within barcode families, if a unique symbology is associated with a specific length, then UnForm will internally select the correct symbology. For example, if a 9-digit zip code is specified with symbology 900 (5-digit post net), then symbology 905 will be used automatically.

expr is an 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 Post net – 5 DIGIT
905	USPS Post net – 9 DIGIT
910	USPS Post net ABC – 11 DIGIT
950	USPS Planet
960	USPS Intelligent Mail (Aka: OneCode, the 4-State Customer Barcode, 4CB and USPS4CB)
1000	PDF417
1010	MicroPDF
1100	Maxicode
1200	Data Matrix
1300	Aztec
1310	Aztec with binary content flag
1400	QR Code
1410	Micro QR Code

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

spc-pixels is the number of pixels allocated to spacing between bars, from 1 to 50, the default being 2.

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

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

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

Normally, UnForm will trim trailing spaces from the value being barcoded. If the notrim option is present, trailing spaces will not be trimmed. This can be critical if the barcode data contains spaces that must be retained. For example, a Code 128 barcode uses encoding that can produce space characters for valid data, so the notrim option should be used to prevent valid data from being truncated.

For 1D symbologies, you can specify 'cols *n*' to fit the barcode into the specified number of columns (specified in integers), overriding the space-pixel setting. This provides more precise space allocation.

Symbology Notes

- Use symbolno *n* and totsymbol *n* for linked 2d barcodes, supported by QR code, Aztec, and Maxicode
- Use "security *n*" or "errorlevel *n*" to set the error correction level with Aztec (range 1-89), QR (range 0 to 3), and PDF417 (range 1 to 8)
- PDF417 also supports "mode 1" to enable binary compaction, and uses cols and rows for number of columns (1 to 30) and rows (3 to 90)
- MicroPDF417 uses "cols *n*" to specify a symbol size, described in the following table:

Index	Symbol size (rows x cols)	Index	Symbol size (rows x cols)
0	automatic calculation	18	3 x 15
1	1 x 11	19	3 x 20
2	1 x 14	20	3 x 26
3	1 x 17	21	3 x 32
4	1 x 20	22	3 x 38
5	1 x 24	23	3 x 44
6	1 x 28	24	4 x 4
7	2 x 8	25	4 x 6
8	2 x 11	26	4 x 8
9	2 x 14	27	4 x 10
10	2 x 17	28	4 x 12
11	2 x 20	29	4 x 15
12	2 x 23	30	4 x 20
13	2 x 26	31	4 x 26
14	3 x 6	32	4 x 32
15	3 x 8	33	4 x 38
16	3 x 10	34	4 x 44
17		3 x 12	

- MicroPDF417 uses "mode *n*" to define an emulation mode:

0	Default format
1	UCC/EAN/GS1-128 Emulation
2	Code 128 Emulation
3	Code 128/FNC2 Emulation
4	Linked UCC/EAN/GS1-128
5	05 Macro
6	06 Macro
7	CC-A Data Mod

- QR format uses "cols *n*" to specify a symbol size:

Index	Symbol size (rows x cols)	Index	Symbol size (rows x cols)
0	automatic calculation	21	101 x 101
1	21 x 21	22	105 x 105
2	25 x 25	23	109 x 109
3	29 x 29	24	113 x 113
4	33 x 33	25	117 x 117
5	37 x 37	26	121 x 121
6	41 x 41	27	125 x 125
7	45 x 45	28	129 x 129
8	49 x 49	29	133 x 133
9	53 x 53	30	137 x 137
10	57 x 57	31	141 x 141
11	61 x 61	32	145 x 145
12	65 x 65	33	149 x 149
13	69 x 69	34	153 x 153
14	73 x 73	35	157 x 157
15	77 x 77	36	161 x 161
16	81 x 81	37	165 x 165
17	85 x 85	38	169 x 169
18	89 x 89	39	173 x 173
19	93 x 93	40	177 x 177
20	97 x 97		

- MicroQR format uses "cols *n*" to specify a symbol size

Index	Symbol size (rows x cols)	Index	Symbol size (rows x cols)
0	automatic calculation	3	15 x 15
1	11 x 11	4	17 x 17
2	13 x 13		

- Data Matrix uses "cols *n*" to specify a symbol size:

Index	Symbol size (rows x cols)	Index	Symbol size (rows x cols)
0	automatic calculation	16	64 x 64
1	10 x 10	17	72 x 72
2	12 x 12	18	80 x 80
3	14 x 14	19	88 x 88
4	16 x 16	20	96 x 96
5	18 x 18	21	104 x 104
6	20 x 20	22	120 x 120
7	22 x 22	23	132 x 132
8	24 x 24	24	144 x 144
9	26 x 26	25	8 x 18
10	32 x 32	26	8 x 32
11	36 x 36	27	12 x 26

12	40 x 40	28	12 x 36
13	44 x 44	29	16 x 36
14	48 x 48	30	16 x 48
15		52 x 52	

- Data Matrix uses "mode *n*" to specify a format:

Index	Format
0	Default format
1	GS1/UCC/EAN
2	Industry
3	Macro 05
4	Macro 06

- Aztec uses "cols *n*" to specify a symbol size:

Index	Symbol size (rows x cols)	Index	Symbol size (rows x cols)
0	automatic calculation	17	83 x 83
1	15 x 15	18	87 x 87
2	19 x 19	19	91 x 91
3	23 x 23	20	95 x 95
4	27 x 27	21	101 x 101
5	31 x 31	22	105 x 105
6	37 x 37	23	109 x 109
7	41 x 41	24	113 x 113
8	45 x 45	25	117 x 117
9	49 x 49	26	121 x 121
10	53 x 53	27	125 x 125
11	57 x 57	28	131 x 131
12	61 x 61	29	135 x 135
13	67 x 67	30	139 x 139
14	71 x 71	31	143 x 143
15	75 x 75	32	147 x 147
16	79 x 79	33	151 x 151
And 3 special sizes, usually used only for reader programming			
34	19 x 19 (reader progr.)	36	27 x 27 (reader progr.)
35		23 x 23 (reader progr.)	

- Aztec uses "mode *m*" to specify a format:

Index	Format
0	default format
1	GS1/UCC/EAN
2	Industry

Version 10.1 notes

The internal barcode engine has been upgraded to use the TEC-IT barcode toolkit, which enables many new features and barcode symbologies, and eliminates the need for the Windows Support Server, which was required for certain features in previous releases. The legacy 1D barcode kit is still available, by specifying a "notbc" option.

Barcodes are now generated before other raster data, such as boxes or text, in order to support human-readable text output. This might change job appearance if boxes or text are drawn on top of the barcode image.

Version 5 Note

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

Drivers: pcl, pdf, ps

Examples:

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

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

13.8 BARCODE (ZEBRA)

Syntax

barcode *col*{*numexpr*}, *row*{*numexpr*}, ("*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*, *symbolno* *val*, *totsymbol* *val*, *chkhmn* *val*, *magfactor* *val*, *ecis* *val*, *errctrl* *val*, *menusymbol* *val*, *appendid* *val*, *model* *val*, *hqml* *val*, *nabk* *val*, *symboltype* *val*, *sepheight* *val*, *segwidth* *val*, *width39* *val*, *ratio39* *val*, *height39* *val*, *heightpdf* *val*, *widthpdf* *val*, *quality* *val*, *escchar* *val*

Description

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

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

symbology is one of:

Symbology	Name	Options Used
1	Code 11	rotate,checkdigit,height,text,above
2	Interleaved 2 of 5	rotate,height,text,above,checkdigit
3	Code 39	rotate,checkdigit,height,text,above
4	Code 49	rotate,height,text,mode
5	Planet	rotate,height,text,above
7	PDF417	rotate,height,security,cols,rows,truncate

8	EAN-8	rotate,height,text,above
9	UPC-E	rotate,height,text,above,checkdigit
A	Code 93	rotate,height,text,above,checkdigit
B	CODEABLOCK	rotate,height,security,cols,rows,mode
C	Code 128	rotate,height,text,above,checkdigit,mode
D	UPS Maxicode	mode,symbolno,totsymbol
E	EAN-13	rotate,height,text,above
F	Micro PDF417	rotate,height,mode
I	Industrial 2 of 5	rotate,height,text,above
J	Standard 2 of 5	rotate,height,text,above
K	ANSI Codabar	rotate,checkdigit,height,text,above,start,stop
L	LOGMARS	rotate,height,above
M	MSI	rotate,checkdigit,height,text,above,chkhmn
O	Aztec	rotate,magfactor,ecis,errctrl,menusymbol,symbolno,appendid
P	Plessey	rotate,checkdigit,height,text,above
Q	QR Code	rotate,model,magfactor,hqml,nabk
R	RSS (Reduced Space Symb)	rotate,symboltype,magfactor,sepheight,height,segwidth
S	UPC/EAN extensions	rotate,height,text,above
T	TLC39	rotate,width39,ratio39,height39,heightpdf,widthpdf
U	UPC-A	rotate,height,text,above,checkdigit
X	Data Matrix	rotate,height,quality,cols,rows,formatid,escchar
Z	Postnet	rotate,height,text,above

Many options are required only by certain symbologies. The options used are given in the table above. For details about use and required values for options, see the ZPL reference manual available from Zebra Technologies Corporation (<http://zebra.com>).

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

Segment 1:

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

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

Segment 2:

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

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

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

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

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

Rotate will rotate the barcode the given number of degrees.

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

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

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

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

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

stop char will set the stop character.

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

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

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

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

For other options, see reference materials offered by Zebra Technologies Corporation (<http://zebra.com>).

Drivers: zebra only

13.9 BIN

Syntax

bin *bin-code*

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-code is printer-specific, with 1 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.

The printer model's (-m command line option) PPD file (or generic pcl.ppd or ps.ppd files) can specify *OutputBin *bin-code* entries which are used if present.

Drivers: pcl, ps

13.10 BOJ, BOP, EOJ, EOP

Syntax

```
{boj | bop | eoj | eop}"text string" | {expr}
```

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 as a quoted text string or an expression in braces.

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

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

An expression can take advantage of the getppd() function to load control sequences for PCL or PostScript out of the printer's PPD file (as specified by the -m command line argument or as the generic pcl.ppd or ps.ppd).

Prior releases supported an unquoted format for hex strings. UnForm no longer supports this syntax. If your rule file contains hex strings, convert the commands like these examples:

```
boj 1b266c3247 change to      boj { ath("1b266c3247") }
boj 1b 26 6c 32 47          change to      boj { ath(stp("1b 26 6c 32 47",3," ")) }
```

Examples:

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

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

Drivers: pcl, ps

13.11 BOLD, ITALIC, LIGHT, UNDERLINE, CBOLD,...

Syntax

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

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

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

Description

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

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

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

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

Examples:

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

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

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

[AFO](#) jobs not supported.

13.12 BOX, CBOX

Syntax

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

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

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

Description

A box of the indicated dimensions will be drawn. All dimensions can be specified to 2 decimal places, in the range of -255 to +255. Whole number *col* and *row* represent center points; lines are drawn to the center point of the character position identified in order to facilitate connections between lines. This differs from the **shade** keyword, which shades full character cells. It may be easier to use the **box** keyword's shade parameter than to calculate shade positions that are offset from similar box parameters. To draw lines rather than boxes, simply set the *cols* or *rows* to 1 or 0 (1 is a special rule maintained for historical reasons), or use the **line** command. If both *cols* and *rows* are 1, then a vertical line is drawn 1 character high. To draw a box that is 1 column wide or 1 row deep, use 1.01 or .99. If used, *numexpr* is a Business Basic expression that generates a numeric value for the column, row, columns, or rows.

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

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

Line Thickness

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

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

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

Shading

The optional *shade* parameter may be used to specify a "percent gray" value from 1 to 100. Most laser printers can only print about 8 different shades of gray, so a value of 45, for example, may print the same pattern as 50. Note that if you specify a shade level of 0, this differs from not specifying any shade at all: a shade level of 0 will force a white interior, even if another box or shade command draws shading inside the bounds of the box. If an interior color is specified, shading is ignored. A shade value of -1 is equivalent to no shading at all.

Color

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

Grids

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

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

Examples:

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

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

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

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

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

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

Drivers: All (*gridcols* and *gridrows* options supported only in pcl, ps, pdf), zebra only support 0% or 100% shading.

13.13 BOXR, CBOXR

Syntax

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

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

Description

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

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

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

Line Thickness

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

Corner Rounding

To specify the degree of rounding for different sides, specify values for *tl*, *tr*, *bl*, and *br*, as desired. The specification for each corner is *col:row:scale*, where *col* is the number of columns from the corner to begin the rounding, *row* is the number of rows from the corner to begin rounding, and *scale* is the level of rounding, from -100 for fully convex, to 100 for fully concave, where 0 becomes a straight line from the column and row break points. If no rounding options are specified at all, then UnForm will apply default rounding to all four corners. If any rounding is specified, then any unspecified corners become square corners.

Shading

The optional *shade* parameter may be used to specify a "percent gray" value of from 1 to 100. Most laser printers can only print about 8 different shades of gray, so a value of 45, for example, may print the same pattern as 50. Note that if you specify a shade level of 0, this differs from not specifying any shade at all:

a shade level of 0 will force a white interior, even if another box or shade command draws shading inside the bounds of the box.

Color

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

Grids

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

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

Zebra Printers

Zebra output supports rounded corner boxes somewhat differently than laser/pdf output. All corners have the same scale of rounding, so the first corner option (tl, rt, bl, br) is used for all corners. The scale value must be a number from 1 to 8, indicating the scale of rounding the printer performs. For example, tl::6 would apply a rounding scale of 6. The col and row parameters of the corner specification are ignored.

Examples:

boxr 10,9.5,70,4.25,2.5,lcolor=blue will draw a box with default rounding on all corners, with a 2 pixel edge and 5% shading. The edge line will be drawn in blue if the output device supports color.

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

Drivers: pcl, pdf, ps (pcl cannot have -nohpgl specified), zebra (see notes)

13.14 CIRCLE

Syntax

1. circle *col*{*numexpr*}, *row*{*numexpr*}, *radius*{*numexpr*} [*thickness*] [*shade*]{*numexpr*} [*color*]{*color* *colname*} [*scolor* *colname*] [*color*]{*color* rgb *rrggbb*} [*scolor* rgb *rrggbb*]

2. circle "*text*!|=*text*!~*regex*!~*regex*[@left,top,right.bottom]", *col*{*numexpr*}, *row*{*numexpr*}, *radius*{*numexpr*} [*thickness*] [*shade*]{*numexpr*} [*color*]{*color* *colname*} [*scolor* *colname*] [*color*]{*color* rgb *rrggbb*} [*scolor* rgb *rrggbb*]

Description

A circle with the center at the column and row specified, with the radius specified, will be drawn. All dimensions can be specified to 2 decimal places, in the range of -255 to +255. If used, *numexpr* is a Business Basic expression that generates a numeric value for the column, row, and radius. The radius is specified as a number of columns. For a fixed measure radius, use an expression with the *inchtocols()* or *cmtocols()* function.

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

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

Line Thickness

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

Shading

The optional *shade* parameter may be used to specify a "percent gray" value of from 1 to 100. Most laser printers can only print about 8 different shades of gray, so a value of 45, for example, may print the same pattern as 50.

Color

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

Examples

The following will draw a circle centered in an 80 by 66 form, with a 2.5 inch radius, a blue 2-pixel wide border, and a 5 percent interior shade.

```
circle 40.5,33,{inctocols(2.5)},2,5,lcolor blue
```

Drivers: all

pcl cannot have -nohpgl specified

zebra diameter in dots, thickness in dots, line color – black or white only, no shading

13.15 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

13.16 COMPRESS, NOCOMPRESS

Syntax

compress
nocompress

Description

If zlib support is available on the UnForm server (most operating systems support it), then UnForm will use the "deflate" compression method by default. This produces very compact PDF files. If you do not wish to produce such compressed files (for example, you want to see the PDF commands contained in the file), then you can use the **nocompress** option (or the `–nocompress` command line option) to turn off this default compression mode.

If no zlib support is available, the **compress** command can be used to use the RLE compression algorithm. This is most effective when repeated characters like spaces are present in the output, such as wide reports with empty space between columns. Compression requires extra processing and will therefore affect performance.

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

You can determine if zlib support is enabled by viewing the version information produced by the `uf101c –v` command line.

Drivers: pdf

13.17 CONST, GLOBAL, LOCAL

Syntax

const|global|local *ID=value*

Description

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

const COLPOS=22.25

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

A given constant ID can be reused, and references to it in subsequent rule set lines will reflect the new value. Also, a constant defined before the first rule set in the rule file will apply to any rule sets in the file, unless the same ID is reused in any particular rule set. The global command may be used in place of `const` for one of these pre-rule set constants. Likewise, the local command can be used in place of `const` inside a rule set.

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

Constant names are limited to 255 characters, and constant values are limited to 65,536 characters. If you use a quoted value, the outer quotes are removed before the value is substituted into the rule file commands. You can therefore include quotes inside a quoted constant. Unquoted values are trimmed of leading and trailing spaces.

Long constant values can be built by including the constant name in multiple `const` commands, like this:

```
Const VAL="Initial Value"  
Const VAL="VAL plus this appended data"  
Const VAL="VAL and still more appended data"
```

Drivers: all

13.18 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 together for each page.

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.

```
pcopies 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

13.19 COVER

Syntax

```
cover "ruleset" [{expr} [, "rulefile" [{expr} [, "args" [{expr} ]]]
```

Description

Processes the named rule set (optionally in a different rule file) as a subjob, using the first page of text of the current job as the input stream. The resulting one page of output is used as an initial page of the main job. Both pcl and ps output will generate cover pages for each output file when the job is broken into multiple output designations. If arguments are specified, they are passed to the subjob, in addition to the -r/-f options named by the ruleset and rulefile options.

In addition to the cover command, the -cover command line argument, as well as the coverset\$, coverfile\$, and coverargs\$ code block variables, can be used to generate cover pages. Also, setting nocover=1 in a code block will disable cover page generation. This can be used to turn off the effect of a -cover command line option.

Example

This example will generate a cover page from the corpcover rule set in covers.rul, passing it a name and number parameter. The corpcover rule set could retrieve the name and number in a prejob code block, using prm("name") and prm("number").

```
cover "corpcover", "covers.rul", {"-prm "+quo+"name="+name$+"";number="+faxnum$+quo}
```

13.20 CPI

Syntax

```
cpi characters-per-inch
```

Description

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

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

Examples:

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

Drivers: pcl, pdf, ps, zebra

13.21 CROSSHAIR

Syntax

crosshair

Description

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

Drivers: pcl, pdf, ps

13.22 DELIVER

Syntax

```
deliver "send to"{{expr},"docid"{{expr}} [,combine yes|true|1 {{expr}} [,args "list"{{expr}} [,tag "value"{{expr}}, ...] [,async 0|1{{expr}} [,ufnotify "address"{{addr$}} [,ufnotifycc "address"{{addr$}} [,ufnotifyon "err | rty | ok" | {{list$}} ]]
```

Description

The deliver command generates subjobs whenever the document ID and output format changes, producing files as needed based on the list of "send to" values, and then sending them via email, fax, or custom configured methods. Custom configuration provides facilities to deliver the generated documents to disk, program or script, web servers, cloud services, and more. The output format is determined by the [deliver.ini](#) file, and if the recipient is an email, fax, or other destination. Multiple destinations can be provided in a comma-separated list and each will get their own copy of the document. If the combine option is on (yes, true, or 1 turn it on), then all documents for a given destination are combined into a single transaction. Note that not all fax systems offer support for multiple documents in a single transaction (i.e. msfax). The document ID is used as the basis for the file name to be sent, which can be useful when emailing to provide meaningful attachment file names.

The args option specifies command line options passed to the subjob. The deliver.ini configuration can also add more options. Other tag names are used to substitute values in the delivery gateway's configuration lines in deliver.ini. More details are provided in that file, and in the Deliver Configuration chapter.

If the `async` option is specified, it overrides an `async` setting in `deliver.ini` for this particular job. If turned on, the delivery is queued rather than delivered as part of the current job. Queued files are stored as encrypted objects in the `deliver` directory and processed independently of the job that generates them. When a queued file encounters an error, it is retried until successful, based on some [deliver.ini](#) configuration settings.

Any number of tags can be specified, using user-defined tag names. When the delivery is executed, the `deliver.ini` configuration is scanned and occurrences of `%tagname` are substituted with the provided value. Note that tag values cannot contain linefeed characters (`$0A$`) due to the method of parsing `deliver.ini` tag definitions. Use `"\n"` sequences instead in message bodies, for example, where a linefeed character may be necessary.

Multiple delivery commands can be used. This may be desired if different arguments, such as cover pages or `-prm` parameters, are desired to distinguish email from fax jobs for formatting. If this is the case, the document ID should also vary, since only a unique ID and output type cause a new subjob to be executed.

When the subjob is executing, both `uf.subjob` and `uf.deljob` are true (1).

CSV formatted logs are maintained in the `./deliver` directory (or other configured directory named in `logdir=` in `deliver.ini`). The logs are named `yyyymmdd.csv` and record date/time, to, file, success, response and error messages, and optionally the tags.

Note that code blocks can also use the `deliver()` function, managing the file to be delivered with `jobstore/jobexec` functions or other techniques.

Notification Options

You can specify a notification address, along with an optional notification cc, and a selection of when to send notification, by using the following reserved tags (added in 9.0.24):

- `ufnotify` sets a notification address, which is required to enable notification
- `ufnotifycc` sets an optional notification cc address, where an additional copy of the notification is sent
- `ufnotifyon` is a space-separated list of status values for which notification is sent. This defaults to "err". You can set it to "err rty" to notify on both errors and pending retries, or "err rty ok" to send notifications in all cases.

A notification is sent immediately after the delivery attempt. If there is an error sending the notification, the error is logged to the server's log file. No additional notifications are attempted. The notification relies on server and from address settings defined in the `prog/mailcall.ini` file, not the `deliver.ini` file.

SMTP Logging

If you specify a `logfile` option, a detailed log of the mail server communication is written to the specified file. This name should be unique to avoid errors if two jobs require the same log file name at the same time. The filename can include tags to assist in uniqueness. Note the directory where log files are to be placed must exist and be accessible to the UnForm server. The tags are:

- `@d` for the date in YYYYMMDD format
- `@t` for the time in HHMMSS (24-hour clock) format
- `@p` for the process ID of the UnForm task generating the file
- `@j` for the UnForm job number, a sequential counter
- `@k` for a millisecond value, generally appended to a `%t` tag

Examples

Email with an html message body from the variable `htmlbody$`, using the `async` option to spool the email submission:

```
deliver {sendto$},{ "E"+invno$}, subject "Please review and pay your invoice", note {htmlbody$}, bodymime "text/html", async 1
```

Fax with cover tags:

```
deliver {faxnum$},{ "Invoice "+invno$}, name {contact$}, subject {"Invoice "+invno$},
  note "Your invoice is attached.\n\nThank you for your business.", logfile "/tmp/smtplogs/@d-@t.@k-@p.log"
```

Deliver a file to a disk location:

```
deliver "store.pdfdoc",{ "Inv"+invno$}, store_path="/docs/invoices", subdir {dte(0:"YYYY-MM")}
```

13.23 DETECT

Syntax

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

Description

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

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

column and *row* can contain ranges in the format *from-through*, such as '20-25' for the columns (or rows) 20 through 25. Column ranges must include the entire number of columns to search, so to detect the word "Invoice", the second number must be at least the rightmost possible position of the "e".

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

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

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

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

DTC Rule Sets

When the UnForm Desktop Client processes detect statements, it honors certain options. For details, see the Desktop Client [rule set section](#).

Web Extension Rule Sets

When the SDSI Web Extension processes detect statements, it honors certain options. For details, see the Web Extension [rule files section](#).

Examples:

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

detect 10-22,4,"~../../.." would match a date format between columns 10 and 22, on row 4.

detect 65-76,6-8,"!~../../.." would match a date format NOT occurring between columns 65 and 76, on rows 6 through 8.

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

Drivers: all

13.24 DOWN

Syntax

down *n* [,*gap*]

Description

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

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

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

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

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

Drivers: pcl, pdf, ps

PostScript input not supported

13.25 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: pcl only

13.26 DSN_SAMPLE

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

13.27 DTCTBUTTON

Syntax

DTCButton "*title*" [,style clipboard|nbclipboard|text|nbtext|button|title|nbtitle] [,args "*job arguments*"] [,width *chars*] [,match "*regex*"] [,library "*name*"] [,doctype "*value*"] [,parsevalue ["*ruleset*"]]

Description

The DTCButton command is interpreted by the [UnForm Desktop Client](#) for presentation in an application integration window. For each DTCButton command, an input field is constructed, along with a submission button, and presented on the form associated with the rule set. The DTCPanel command can be used to categorize the buttons in panels. Input fields are presented in the same order as presented in the rule set. Note that expressions are not supported.

Style

This optional argument defines the style of the button provided to the user. The default style is "clipboard".

Clipboard	Provides a text box and a small submit button, and monitors the clipboard for changes to place text in the text box.
Nbclipboard	Provides a text box, but no submit button. It monitors the clipboard for changes to place in the text box.
Text	Provides a text box and a small submit button. This is intended for simple user entry.
Nbtext	Provides a text box, but no submit button.
Button	Provides a submit button with the title as the button caption.
Title	Provides a text box and a small submit button, and monitors the window title for changes to place in the text box.

Nbtitle	Provides a text box, but no submit button. It monitors the window title for changes to place in the text box.
---------	---

Args
Options passed to the to the UnForm job running the rule set. There are automatic rule file and rule set arguments (-f and -r, respectively) to ensure the rule set with the button configuration is the one executed when a submission takes place.

Width
This defines the width of the button, in nominal characters. Without a width, the title width is used. Use this to ensure consistent widths when multiple buttons are presented.

Match
The match option is honored by the clipboard and nbclipboard styles. The clipboard value must match the specified regular expression in order to be pasted into the text field.

Library, Doctype
These two options in tandem are honored by the clipboard and nbclipboard styles. The clipboard value must be a valid document ID within the library and doctype specified in order to be pasted into the text field.

ParseValue
If this option is present, the value from the clipboard or window title is sent to the server for parsing. The server will run the current rule set or the optionally specified one, and use the value returned from the server in cgiresponse\$ in place of the clipboard or title value. The rule set receives cgi.button\$, cgi.panel\$, and cgi.parsevalue\$ when the request is sent.

13.28 DTCHelpFile

Syntax

DTCHelpFile "filename"

Description

When the [UnForm Desktop Client](#) presents an application integration form (defined using DTCTButton and DTCTPanel commands), there is a toolbar help button available. If a DTCHelpFile command specifies a file, the help button is enabled, and when pressed, the HTML file specified is loaded in a browser window on the DTC user's workstation. The help file must be in HTML format, and must be available to the UnForm server using normal HTTP interface locations, in the ./web/en-us or ./web/language path, in the home UnForm directory.

13.29 DTCTPanel

Syntax

DTCTPanel "panelname"

Description

The DTCPanel command can specify a panel name for DTCButtons that follow. The buttons will be presented within a tab panel, using *panelname* as the title. This command is interpreted by the [UnForm Desktop Client](#).

13.30 DUMP

See the **image** command.

13.31 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 (1/720th inch) that indicate how far to shift the page printing from the left and top edges, respectively. Note that margins may need to be adjusted (with the **margin** keyword) if offsets are used.

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

The printer model's (-m command line option) PPD file (or generic pcl.ppd or ps.ppd files) can specify *Duplex *mode* entries which are used if present.

Note that when working with multiple copies to produce pages or unique formats in duplex mode (i.e. terms and conditions on the back page of primary forms), only the **pcopies** command will work, as it is critical that the copies be printed in sequence rather than at a job-level.

Examples:

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

Drivers: pcl, ps (the left offset and top offset options are ignored in PostScript, use margin instead)

13.32 EMAIL

Syntax

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

Description

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

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

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

In order to use this command, the prog/mailcall.ini file must be edited to configure a mail server (server=value) line, and optionally fixed login and password information. The file is self-documenting, or you can use the [server manager](#) to configure the standard entries. Note also the [deliver](#) command offers more flexibility than the email command. The email command is a legacy command.

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

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

cc	Followed by a literal that is, or an expression in curly braces that resolves to, a list of email addresses separated by commas. These addresses become the CC, or carbon copy, list for the email.
bcc	Followed by a literal that is, or an expression in curly braces that resolves to, a list of email addresses separated by commas. These addresses become the BCC, or blind carbon copy, list for the email. Blind carbon copy addresses are stripped from the email header before the message is sent.
attach	Followed by a literal that is, or an expression in curly braces that resolves to, a list of additional attachment files, separated by commas. Note that the PDF job itself is always emailed as an attachment, so only use this option for adding additional attachments to the message.
otherhead or oh	Followed by a literal that is, or an expression in curly braces that resolves to, one or more line-feed or "\n" delimited custom email headers.

login	Followed by a literal that is, or an expression in curly braces that resolves to, a login name. Some mail servers are configured to require a login and password for authentication. This value and the password value are then required.
password or pswd	Followed by a literal that is, or an expression in curly braces that resolves to, a login password. Some mail servers are configured to require a login and password for authentication. This value and the login value are then required.
logfile	Followed by a file name to which SMTP logging will be written. The file name may contain substitution tags as documented in the deliver command.

Example

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

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

Drivers: pdf only

13.33 ERASE, CERASE

Syntax

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

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

Description

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

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

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

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

When erase is used with PostScript input, it is converted internally to a **shade** command with a shade percent of 0, resulting in erasure of the region from the overlay. Rule set output commands, such as text or box, are layered on top of the erased region.

Examples:

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

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

Drivers: all

13.34 FIXEDFONT

Syntax

`fixedfont fontcode`

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

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

Drivers: pcl only

13.35 FONT, CFONT

Syntax

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

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

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

Description

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

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

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

Font Names and Numbers

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

Symbol Sets

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

Point and Pitch Sizes

size is a numerical value that specifies the point size of a proportionally spaced font or the pitch size of a fixed font. Values range from about 4 to 999.75. The default is based on the rows per page. Note that for proportional fonts, the larger the number, the larger the size printed. Fixed fonts are the opposite. Size can be specified as an expression, if it is prefixed with the keyword "size".

Attribute Styles

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

Shaded Text

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

Fixed and Proportional Text

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

Color

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

Justification

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

Text Case Conversion

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

Fit to Width

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

Weight and Style

Some laser printer fonts must be specified with given weight or style in order to be selected by the printer.

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

Note that if you use identical font commands for two adjacent or overlapping regions, UnForm will combine the regions. For proportionally spaced fonts, the result will be misaligned columns. To avoid this, you can add non-operational options, like "black" or "shade 100" to alternating commands, so UnForm will not treat them as identical. Alternatively, use the column option, specifying a unique column between 0 and 222 to prevent region combining.

Examples:

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

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

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

Drivers: all, but note the following:

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

Ps: maps pcl font names and numbers to a setting defined in the [psmap] section of ufparam.txt. Matching Type1 font files can be installed in the psfont directory.

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

The fit option is only supported in pcl, ps, and pdf drivers.

[AFO](#) jobs not supported.

13.36 GS

Syntax

gs [yes | on | no | off]

Description

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

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

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

Example:

```
gs on

if copy 2
  gs off
  output {"\vfx -n " + faxnumber$ + " -F pcl"}
end if
```

Drivers: pcl only

13.37 HLINE

Syntax

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

Description

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

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

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

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

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

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

Example:

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

Drivers: all

PostScript input not supported.

13.38 HSHIFT

See the **shift** command.

13.39 IF COPY ... END IF

Syntax

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

Description

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

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

Examples:

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

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

Drivers: all

13.40 IF DRIVER ... END IF

Syntax

```
if driver name  
...  
end if
```

Description

The command **if driver** will cause any commands to apply only when the rule set is evaluated under the driver *name*. The driver is specified with the command line option "-p", and defaults to "laser". Common drivers are laser, pdf, ps, and zebra. If Ghostscript drivers are configured, then other driver names are available based on the [drivers] configuration in the server's uf101d.ini file.

The command 'if driver zebra' applies to any Zebra driver specification (Zebra drivers include suffix-based information).

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

13.41 IF EXPRESSION ... END IF

Syntax

```
if expression  
...  
end if
```

Description

The *if expression* block test evaluates the *expression* as Basic syntax to determine if the enclosed UnForm rule set commands should be included in the current job.

Before 10.1, this expression could only reference values available at parse time, not runtime, so expressions could only evaluate prm() or gbl() functions, or uf.xxx variables that would contain command line settings. Starting in 10.1, however, the expressions are generally evaluated at runtime, allowing output commands like text, box, and image to be conditionally resolved during document production.

Some commands must be resolved at parse time, such as merge and const/global/local, however, so expressions that contain prm(), gbl(), or uf.xxx are still resolved at parse time.

end if indicates that conditional processing of the rule set is done, and normal parsing continues. The **end if** keyword may also be entered as **endif** or **fi**.

Examples

```
if uf.pdftitle$=""
  title "Default Title"
end if
```

```
if prm("email")>""
  # command line contained -prm email=xxx
  email {prm("email")},...
end if
```

Drivers: pcl, pdf, ps, zebra

13.42 IMAGE

Syntax

```
image col{numexpr}, row{numexpr} [, cols{numexpr}, rows{numexpr}, "file" | {expr} [,color], [,nocache]
[,option code] [,shade percent] [,gamma gamma] [,rotate rotate][,page pagenum{expr} [,link "url"{urlexpr}]
[,linkopt "urlopts"{urloptexpr}]
```

Description

The **image** command is used to print an image file specified by *file* or the *expr* which resolves to a file name 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 optional *cols* and *rows* parameters are used in most circumstances. Specifically, they are not used for native PCL images (.pcl or .rtl files), or for raw Zebra images (.zpl). In those cases, the columns and rows options are ignored. For Postscript and PDF images, and for images that are scaled and converted for use in PCL output, columns and rows are required. If not supplied in those circumstances, then each defaults to the cols and rows that measure one inch. It is generally advisable to include these parameters to ensure that all versions of output will produce the desired size whenever possible.

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

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

Images typically require some degree of processing or parsing during a print job. To avoid extra overhead for images that are used repeatedly, the final images are cached for reuse in later jobs. However, in some cases it may be preferable to avoid caching images. For example, signature images are often used only once, and caching them would waste space and cache management processing time. To turn off caching of an image, use the nocache option. Note that cached images that remain unused for a period of time (defined by imageage=days in uf101d.ini) are removed from the cache file (images.dat) automatically.

The color option indicates that a color image should be produced. The `-ci` and `-color` command line options set the color option on by default for all images. This option is ignored when printing native format images, such as `.pcl` or `.rtl` images, but it may determine what file name substitutions might occur in some output formats.

The shade option can be used to apply shading to the image to reduce its intensity and allow other data to show through the image. In PDF output, transparency must be enabled for this to work.

The gamma value can be used to modify the colors when an image is being converted to native format. Images that contain RGB color information often display differently on different devices and as a result can appear darker or lighter than expected when printed. A gamma value greater than 1 lightens an image, while a value less than 1 darkens it. This value is passed to the conversion program (Image Magick or the Windows Support Server). It is ignored when using image files that are already in native format for the output.

Rotation can occur when an image is being converted to native format. This value is passed to the conversion program (Image Magick or the Windows Support Server). It is ignored when using image files that are already in native format for the output.

The page option is supported for PDF images and PDF output. See the discussion below.

Image handling varies considerably based on the output format, and the type of image provided to the image command. Image files can be considered "native", meaning they can be inserted by UnForm directly into the output with little or no processing, or they can be other image formats that require conversion first. Conversion is automatic, assuming either Image Magick or the Windows Support Server are configured and available for UnForm's use.

The publisher also maintains an image conversion tool at <http://unform.com>. This tool allows users to upload images in non-native format and produce native images for use within UnForm jobs.

PCL Output

UnForm considers an image with a `.pcl`, `.pm`, or `.rtl` extension to be a native PCL image. These images are parsed to remove PCL instructions that could cause a page eject, and positioning code is inserted at runtime to place the image in the correct location on the page. Otherwise, the image is passed through to the output unchanged.

If an image is not in native PCL format, such as a `.jpg` or `.tif` file, then it must be converted at runtime to PCL format. During this conversion process, the image is also scaled to the size required. The conversion and scaling process is accomplished with Image Magick, if configured in the `uf101d.ini` file, or via the Windows Support Server, if configured in `uf101d.ini` or with a `sshost()` code block command. Internally, the image is first converted to a bitmap in color or black and white format, and then resolved into a PCL image internally.

Notes on Native PCL Images

Some PCL images contain width and height information. However, since passing through a width and height would apply a default crop size to any additional image without width and height information, UnForm strips out this width and height coding. Unfortunately, color PCL images must contain width and height information to prevent the printer from displaying a black band from the right edge of the image to the right margin. Therefore, when printing to a color printer, UnForm must pass the image width and height coding to the printer. To trigger this parsing behavior, you can do one of two things: add a "color" option to the image command, or add a `-gw`, `-ci`, or `-color` option to the UnForm command line.

One side effect of passing this coding through is that you can't also use images that do not have size information. Generally, this means you can't mix color and black and white PCL image files in the same job.

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. This feature only works with pcl images that include positioning data.

PDF Output

If UnForm is producing PDF output, and the image file name ends in .pcl, .pm, or .rtl, then the file name is modified to have a .pdf extension automatically. This allows a single fixed file name to accommodate both laser and PDF output without special logic.

Because PDF files can contain various image formats, including full-color (24-bit) jpeg files, and supports both color and black and white image data, UnForm goes through several logical steps to determine how best to insert the image.

- If the image file extension is .pcl, .pm, or .rtl, UnForm instead looks for a file of the same name, but with a .pdf extension. If the file is not found, a warning error is issued and no image is produced. Otherwise, native .pdf handling is performed.
- If the image is in .pdf format, and the 'page *n*' option is not used, UnForm parses the file for the first image on the first page, and inserts that image object in the output. Note that UnForm's parser supports PDF files revision 1.4 and below.

If the 'page *n*' option is used, then UnForm will use Ghostscript 8.10 or higher to produce a scaled image of the specified page. This page image will be in either color or black and white format, depending on whether color or black and white image output is being produced. If Ghostscript is not configured in the uf101d.ini file or via the Windows Support Server, this option results in a warning message and no page image is produced.

- If the image command or command line indicates color output, and the image file extension is jpg or jpeg, Unform checks its size and colors setting. If it is a 24-bit jpeg file and its size is no more than 50% larger than the specified size (based on cols and rows), then the file is inserted directly into the output.
- If no conversion facilities are available (no Image Magick configuration and no Windows Support Server), and the image is a 24-bit jpeg file, it is inserted into the output.
- UnForm attempts to convert and scale the image file into either a black and white pdf image, or a 24-bit color jpeg image, depending on the whether the output is color or not.

Due to internal buffer management, PDF file names can't exceed 75 characters.

PostScript Output

PostScript image output must be in either eps or jpeg format. Black and white laser printers do not support jpeg images. Color laser printers support both eps files and both gray scale (8-bit) and color (24-bit) jpeg images.

UnForm performs these logical steps to determine how to insert the image:

- If the image file extension is .pcl, .prn, or .rtl, then the file name is modified to have either a .eps extension (for black and white output) or a .jpg extension (for color output). If the file exists, the new file name is processed.
- If the file extension is jpeg and the output is color, the file is inserted in the output.
- If the file extension is eps, the file is inserted in the output.
- Other image formats are converted to jpeg or a black and white PostScript raster image, using either Image Magick or the Windows Support Server. If these facilities are not available, a warning error is issued and the image is not inserted.

HTML5 Output

The file name can start with "http://" or "https://" to refer to it as a URL rather than an embedded file. The link and linkopt options can be used to specify a URL hyperlink value, and HTML <a> tag options, respectively, to make the image a hyperlink.

Zebra Output

UnForm scans the first block of the file to determine if it is a native ZPL image file. ZPL images begin with the characters "~DG" followed by some comma-delimited data. If so, the image is inserted in the output.

If the file is not a native ZPL image, UnForm attempts to convert it to a bitmap using Image Magick or the Windows Support Server, and then to a ZPL image. During this process, the image is scaled to the size specified by the cols and rows parameters. If the conversion fails, then a warning error is issued and no image is inserted.

Zebra does not support color or shaded images.

Examples:

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

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

Drivers: all.

13.43 IMAGES

Syntax

images "*filelist*"[{*expr*} [,across *n*] [,down *n*] [,res|resolution *n*] [,color] [,tray value]{*expr*}]

Description

Appends image files from the filelist, which can be a literal or an expression. The list may contain any number of semi-colon delimited file names. Each is converted to an image in sequence and added to pages following the current page, optionally scaled and tiled based on the across and down options.

The images are produced at 300 dpi unless otherwise specified. The images will by default be produced in black and white unless color images are indicated with a `uf101c -color` (or `-ci`) command line option, or if the color option is specified in the images command.

If a file in the list is a PDF file, and the job's output is in PDF format, and there is no tile or resolution setting, then PDF pages are inserted in vector format rather than being converted to images first. This improves performance and will normally result in smaller PDF output.

Support for PDF files requires Ghostscript availability. Support for image files requires Image Magick availability.

The tray option is provided to draw paper from the tray specified for the pages produced by the image attachments. The tray value should match the syntax found in the **tray** command, or can be an expression that produces an equivalent text value.

An images command may be placed inside an 'if copy' block or be run with an `exec()` command in a code block in order to append images only on selected copies or pages.

Examples

```
images "termsconditions1.jpg;termsconditions2.jpg"
```

```
images {all_invoices$},across 2, down 2, resolution 150, tray 5
```

```
images "terms.pdf"
```

Drivers: pcl, pdf, postscript

13.44 ITALIC

See the **bold** keyword.

13.45 JAVASCRIPT

Syntax

```
javascript "text" [{expr}]
```

Description

This command adds document-level javascript to the pdf document. This code is executed as the document opens, so can be used to invoke actions when the document is opened or to define functions for use in annotation actions specified with a javascript: url in the **annotate** command.

Examples

```
javascript "function showMessage(msg) { alert msg; }"
```

```
prejob{
  crlf$=$0d0a$
```

```
js$="function showMessage(msg)+"\n"+"alert msg;"+"\n"}"
}
javascript { js$}
```

Drivers: pdf only

13.46 KEYWORDS

Syntax

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

Description

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

Drivers: pdf only

13.47 LANDSCAPE, RLANDSCAPE

Syntax

landscape or rlandscape

Description

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

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

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

Also see the **portrait** keyword.

Drivers: pcl, ps, pdf (rlandscape is pcl only)

13.48 LCOPIES

Syntax

lcopies *n* | {*expr*}

Description

Setting this value will cause UnForm to add a thermal printer copies command (^PQ in Zebra) to the print output, specifying the printer generate *n* or numeric expression *expr* duplicate labels. This is different than using an UnForm **copies** or **pcopies** command, as it instructs the printer to generate duplicate labels at the printer, rather than allowing for distinct formatting for different copies. The benefit of using this command is that the copies are produced by the hardware and not the print stream output, so overhead is reduced and performance is higher.

Alternatively, in a prepage or prejob code block, the variables `lcopies$` or `zcopies$` can be set to a numeric string (i.e. `lcopies$=str(10)`).

Drivers: zebra

13.49 LDARKNESS

Syntax

`ldarkness n | {expr}`

Description

Setting this value will cause UnForm to send a thermal printer darkness command (~SD in Zebra) to the printer. The darkness parameter can be a number *n* or a numeric expression *expr*. The value of *n* should be an integer between 0 and 30, based upon current ZPL documentation.

Alternatively, in a prepage or prejob code block, the variables `ldarkness$` or `zdarkness$` can be set to a numeric string (i.e. `ldarkness$=str(5)`).

Drivers: zebra

13.50 LIGHT

See the **bold** keyword.

13.51 LINE

Syntax

1. line *col1*{*numexpr*}, *row1*{*numexpr*}, *col2*{*numexpr*}, *row2*{*numexpr*} [*thickness*] [,color|color *colname*] [,rgb *rrggbb*]

2. line "*text*!|=*text*|~*regex*!~*regex*[*@left,top,right.bottom*]", *col1*{*numexpr*}, *row1*{*numexpr*}, *col2*{*numexpr*}, *row2*{*numexpr*} [*thickness*] [,color|color *colname*] [,rgb *rrggbb*]

Description

A line is drawn between the first column and row and the second column and row. All dimensions can be specified to 2 decimal places, in the range of -255 to +255. If used, *numexpr* is a Business Basic expression that generates a numeric value for the columns and rows.

If syntax 2 is used, then the line is drawn relative to any occurrence of the *text*, or of text that matches the regular expression *regex*. In these cases, there may be no lines drawn, or several. *col1* and *row1* are 0-based, in these formats, and can be negative if required, and *col2* and *row2* are the number or columns and rows to draw from the offset position. The search for *text* or *regex* can be limited to a region on the

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

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

The origin point of a line is the center of a cell. In other words, position 1,1 is the center of the first character cell, 0.5,0.5 is the upper left corner of the cell, and 1,1 is the lower right corner of the cell. This is consistent with the box command.

This positioning is different from version 7.0, requiring that column positions be shifted 0.5 columns left, and row positions be shifted 0.5 rows up.

Line Thickness

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

Color

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

Examples

The following will draw a light green line from the upper left corner to the lower right corner of a page:

```
line 0.5, 0.5, 80.5, 66.5, 1, rgb 008000
```

The following will underline the word "TOTAL", in conjunction with a font command:

```
font "TOTAL:",0,0,5,1,cgimes,12
line "TOTAL:", 0, .75, {textwidth("TOTAL:", "cgimes",12,0)},0
```

Drivers: pcl, pdf, ps (pcl cannot have -nohpgl enabled)

13.52 LOAD

Syntax

```
load "filename"
```

Description

The load command is similar, but not identical, to the merge command. Its purpose is to insert rule set text from a disk file. The entire contents of the file is loaded into the rule set in place of the load command.

This differs from merge in that the merge command loads a specific rule set by its name (identified in the file as [*name*]), whereas the load command loads the entire file.

13.53 LOCKCOLS

Description

When the label dimensions and cols setting are established, UnForm scans mono-spaced internal fonts for the closest match that will not exceed the cols specified, then recalculates the cols to agree with the font selected. This allows print stream text and all other enhancements to scale together. However, it also causes labels to shrink in printable area width, sometimes very noticeably, resulting in graphical commands not being placed where expected. This option was added to prevent this recalculates from occurring, at the expense of losing the print stream scale matching. With this option, graphical commands will print where expected on the label, but may not align with print stream output.

Drivers: zebra only

13.54 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

13.55 LSPEED

Syntax

lspeed n | {expr}

Description

Setting this value will cause UnForm to send a thermal printer speed command (^PR in Zebra) to the printer. The value of *n* or result of *expr* should be an integer between 2 and 6, or between 8 and 12, based upon printer documentation.

Alternatively, in a prepage or prejob code block, the variables `lspeed$` or `zspeed$` can be set to a numeric string (i.e. `lspeed$=str(2)`).

Drivers: zebra

13.56 MACRO

Syntax

macro *n*

Description

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

Drivers: pcl only

13.57 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: pcl only

13.58 MARGIN

Syntax

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

Description

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

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

Drivers: pcl, pdf, ps

13.59 MERGE

Syntax

```
merge "ruleset" [ , "rulefile" ]
```

Description

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

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

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

Drivers: pcl, pdf, ps, zebra

13.60 MICR

Syntax

```
micr col{numexpr}, row{numexpr}, "account"{expr}, "check"{expr} [,"font"| bold | light | narrow | {expr} ]
```

Description

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

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

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

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

There are several fonts pre-configured for the various output formats, named micr, micrb10, micrl15, and micrn1, each offering a slightly different appearance, the latter three being for bolder, lighter, and narrower output. These are provided because of slight tolerance variations in bank check reading equipment as well as paper and toner differences. If your bank does not accept the default font, you can try one of the other three.

Optionally provide one of the mnemonic keywords bold, light, or narrow.

Example:

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

Drivers: pcl, pdf, ps

13.61 MOVE, CMOVE

Syntax

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

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

Description

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

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

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

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

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

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

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

Examples:

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

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

Drivers: pcl, pdf, ps

PostScript input not supported.

13.62 NOTEXT, NOOVERLAY

Syntax

notext or nooverlay

Description

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

Example:

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

Drivers: all

13.63 OUTLINE

Syntax

outline [*/leve/*]

Description

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

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

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

Drivers: pdf only

13.64 OUTPUT

Syntax

output "*output-device*" | {*expression*}

Description

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

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

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

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

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

Example:

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

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

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

13.65 OVERLAY

Syntax

```
overlay "pdffile"{{fileexpr}}, page{{numexpr}}, pclleft{{numexpr}}, pcltop{{numexpr}}, resolution{{numexpr}}
```

Description

The overlay command is used to place a specified page of a PDF file as a background on the printed page. This command relies on Ghostscript being configured at the server or available via the Windows Support Server. The specified page is extracted to an image and placed as a full page background.

The pclleft and pcltop values are used as offsets to the upper-left corner of the image, to accommodate margins within the pdf file. Both numbers are entered as pixels, so related to the resolution. Resolution defaults to the rule set's dpi, or 300 if not set. All parameters can be expressions. Pcl offsets are ignored in ps and pdf output.

PostScript Note

When producing PostScript output, the overlay command (more specifically, Ghostscript) produces an EPS overlay that contains features that require PostScript 3. The -p ps and -p ps3 command line options have no effect on this.

Example:

```
overlay "invoice_image.pdf",1
```

```
overlay {backdrop$},1,75,75,300
```

Drivers: pcl, ps, pdf

13.66 PAGE

Syntax

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

Description

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

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

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

Examples:

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

page 42

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

Drivers: all

PostScript input not supported.

13.67 PAPER

Syntax

paper size

Description

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

Common sizes for laser and PDF output are defined in ufparam.txt in the [paper] section. Such sizes include:

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

In addition, you can specify the size in a format *widthxheight*. This feature defaults to inches and also supports specification in centimeters or millimeters, using a "cm" or "mm" suffix, such as **paper 20x30cm**.

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

Drivers: all

13.68 PORTRAIT, RPORTRAIT

Syntax

portrait or rportrait

Description

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

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

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

See also the **landscape** keyword.

Drivers: pcl, ps, pdf (rportrait is pcl only)

13.69 PRECOPY, PREDEVICE, PREJOB, PREPAGE, POSTCOPY,...

Syntax

```
precopy | postcopy | prejob | predevice | postjob | prepage | postpage | postdevice {
code block
}
```

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

Description

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

- * **prejob** executes after the rule set has been read, and after the first page is read, but before any printing takes place. Use this code to open files, define string templates, create user-defined functions, and initialize job variables.
- * **postjob** executes after the last page has been printed. Use this to close out your logic, such as adding totals to log reports. There is no need to close files, since UnForm will RELEASE Business Basic.
- * **predevice** executes just after a device has been opened. With the laser driver, the output device can be changed with the **output** command or by modifying the output\$ variable in a prepage or precopy code block. Whenever a new device is opened for any given copy, this code block is executed. The programmer can then store information from the page that causes the device to be opened, such as a customer code or fax information.
- * **postdevice** executes just after the output device has been closed. Use this code block to perform processing with prior output device, once UnForm has closed the device. For example, if the output

device changed when the customer number changed, then one or more pages for a given customer would be in the output file and could be sent as a group to a fax product.

- * **prepage** executes after each page is read, but before any printing takes place. Use this to gather data associated with any page, or to modify the content of the text if you need such modifications to apply to all copies.
- * **postpage** executes after the last copy of each page has printed.
- * **precopy** executes before each copy is printed. Use this to modify copy text content, to skip specific copies, or to modify a copy's output device.
- * **postcopy** executes after each copy is printed.

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

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

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

Example:

This example shows how to use various routines to make copy 2 of a form be a conditionally faxed invoice, using a CSV formatted file containing a customer ID and a fax number.

```
prejob {
exportfile$="/exports/faxnums.csv"
today$=dte(0:"YYYY-MM-DD")
faxlog$="/exports/logs/fax"+today$+.log"
}

prepage {
invoice$=get(65,5,7)
custid$=get(65,4,6)
custname$=trim(get(10,10,35))
faxnum$=getfilefield(exportfile$, custid$, 2)
}

precopy {
if copy=2 then:
if faxnum$>"" then:
output$="|fx -n "+faxnum$
log(invoice$+" "+custid$+" "+custname$, faxlog$)
end if
end if
}
```

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

13.70 PROTECT

Syntax

```
protect [print] [,annotate] [,extract] [,modify] [,password "password" | {expr}] [, owner "password" | {expr}]
[,128]
```

Description

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

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

print adds the ability to print the document.

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

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

modify adds the ability to modify document contents.

password "xxx"{expr} sets user password required for opening document

owner "xxx"{expr} sets owner password. If the owner password is used to open a document, Acrobat will allow modification to document permissions. Note an owner password won't automatically enable restricted options. Instead, it only allows changing of those permissions and saving of those new settings.

128 turns on 128-bit encryption, which is much stronger than the default 40-bit encryption.

Password and owner expressions are interpreted at the start of the job, immediately after the prejob code block is executed; therefore only page one data, or variables defined in the prejob code block, are available for use.

Drivers: PDF only

13.71 ROWS

Syntax

```
rows n
```

Description

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

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

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

Examples:

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

Drivers: all

13.72 SHADE, CSHADE

Syntax

1. `shade col{numexpr}, row{numexpr}, cols{numexpr}, rows{numexpr}, percent{numexpr} [,extend] [,color] [,rgb rrgbbb]`
2. `shade col{numexpr}, row{numexpr}, cols{numexpr}, rows{numexpr}, percent{numexpr}, skip, times [,extend] [,color] [,rgb rrgbbb]`
3. `shade "text|!=text|~regex|!~regex[@left,top,right.bottom]", col{numexpr}, row{numexpr}, cols{numexpr}, rows{numexpr}, percent{numexpr} [,extend] [,color] [,rgb rrgbbb]`

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

Description

The region indicated by `col`, `row`, `cols`, and `rows` will be shaded, using the `percent` as the percent-gray value. The region parameters can be specified as decimal values to 1/100 character. The region is based on the full character cell, starting at the upper left corner of the cell. This differs from the **box** keyword, which measures from the center point of a cell. The `percent` can be any value from 0 to 100, where 0 is white (useful for erasing regions), and 100 is black. The default shade value is 5% (which renders as 10% in PCL5 devices). PCL5 printers actually support only eight levels of gray, generally: 2%, 10%, 20%, 35%, 55%, 80%, 99%, and 100%. Values less than these are rounded up to the next supported value. If you wish to issue a shade command that will do nothing, use -1 as the percent.

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

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

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

If syntax 3 is used, then the shading is drawn relative to any occurrence of the `text`, or of text that matches the regular expression `regexpr`. In these cases, there may be no shaded regions, or several.

column and *row* are 0-based, in these formats, and can be negative if required. The search for *text* or *regexpr* can be limited to a region on the page by adding a suffix in the format '@left,top,right,bottom'. To use a literal "@" character in *text* or *regexpr*, it is necessary to specify "\@".

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

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

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

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

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

With PostScript input, use a shade percent of 0 to erase a region of the overlay and allow UnForm graphical enhancements to remain visible.

Examples:

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

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

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

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

Drivers: pcl, pdf, ps

For Zebra (0% or 100% only), use a box command.

13.73 SHIFT

Syntax

shift *n*

Description

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

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

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

Example:

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

Drivers: all

[AFO](#) jobs not supported.

13.74 SUBJECT

Syntax

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

Description

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

Drivers: PDF only

13.75 SYMSET

Syntax

symset "*symbolset*"

Description

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

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

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

Drivers: pcl only

13.76 TEXT

Syntax

1. text *col*{*numexpr*}, *row*{*numexpr*}, "*text*" | *@name* | *\$name* | {*expression*} [, *fontname*] [, *font fontcode* | {*expr*}] [, *symset symset*] [, *size* | *size {sizeexpr}*] [, *bold*] [, *italic*] [, *underline*] [, *light*] [, *shade percent* | {*numexpr*}] [, *rotate angle*] [, *fixed* | *proportional* | *prop*] [, *color*] [, *rgb rrggbb*] [, *justification*, *cols*|*cols ncols* | {*numexpr*} , *ccols endcol*] [, *wrap*] [, *fit*] [, *spacing spacing*] [, *weight w*|*weightname*] [, *style style*|*stylename*] [, *nohpgl*] [, *link "url"*{*urlexpr*}] [, *linkopt "urlopts"*{*urloptexpr*}]
2. text "*text*!|=*text*!~*regex*!~*regex*[@*left,top,right.bottom*]", *col*{*numexpr*}, *row*{*numexpr*}, { "*text*" | *@name* | *\$name* | {*expression*} } [, *fontname*] [, *font fontcode* | {*expr*}] [, *symset symset*] [, *size* | *size {sizeexpr}*] [, *bold*] [, *italic*] [, *underline*] [, *light*] [, *shade percent* | {*numexpr*}] [, *rotate angle*] [, *fixed* | *proportional* | *prop*] [, *color*] [, *rgb rrggbb*] [, *justification*] [, *cols*|*cols ncols* | {*numexpr*} , *ccols endcol*] [, *eraseoffset cols*, *erasescols cols*] [, *getoffset cols*, *getcols cols*] [, *wrap*] [, *fit*] [, *spacing spacing*] [, *weight w*|*weightname*] [, *style style*|*stylename*] [, *nohpgl*] [, *link "url"*{*urlexpr*}] [, *linkopt "urlopts"*{*urloptexpr*}]

Description

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

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

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

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

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

If *text* contains line-feed characters (CHR(10) or \$0A\$), or the mnemonic character string "\n", then UnForm will break the text into multiple lines and space them according to the *spacing* value. For example, if the point size is 12, and *spacing* is set to 1.5, then line spacing is set to 18 points. The default *spacing* is calculated from the number of rows per page, so multi-line text data will match the vertical placement of single line text data.

If syntax 2 is used, then UnForm will search for occurrences of *text* or the regular expression *regex*. In this case, *col* and *row* become 0-based offsets from each location where matches are found. In addition, the *erasescols cols* and *eraseoffset cols* can be used to remove match text. The search for *text* or *regex* 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 *regex*, it is necessary to specify "\@".

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

Font Names and Numbers

Fonts are identified by numbers, generally those assigned by HP for PCL5 output. Internally, all font names are converted to font numbers, as UnForm's font architecture is based on PCL5 methods. A name is mapped to a number in the [fonts] section of `ufparam.txt`. For PCL5 output, this number is used to specify the font used for this text element. For other output formats, the number is mapped to an appropriate font setting by using other sections of `ufparam.txt`:

- [psmap] maps numbers to Postscript built-in fonts, or to Type1 font file name in the "psfont" directory.
- [pdfmap] maps numbers to built-in PDF font names.
- [tffont] maps numbers to TrueType font files, located by full path or in the tffont directory. Note UnForm will look in the psmap and pdfmap sections first for a given number, then look in tffont.

There are three fonts that are universally available or readily mapped to a similar native font for all output formats. Those names are Courier (the default), CGtimes, or Univers. These fonts are standard on all PCL5 compatible printers and easily mapped to similar fonts in other environments, such as Times Roman and Helvetica in the Postscript and PDF worlds.

Alternately, a specific *fontcode* supported by your printer can be specified by its font number. For example, if your printer supports Arial, specify "font 16602". If you supply a fontcode this way, there is no mapping of name to number, but the format-specific mapping still takes place in order to select a similar font appropriate for the output format.

Note if you need to customize `ufparam.txt`, you should first copy it to `ufparam.txc` and edit that file, to avoid losing edits when UnForm is updated.

Symbol Sets

symset can be any symbol set supported by your printer. The default symbol set is "9J", using the Windows Latin 1 character set (similar to ISO-8859-1, but with some additional characters defined). You can also specify symbol sets by name from the "ufparam.txt" file. Only symbol set 9J is supported by the PDF and Postscript drivers.

To include non-printable characters, such as control codes or 8-bit characters from a specific symbol set, include the character's numeric (ASCII) value in angle brackets. For example, to include a copyright symbol from the Desktop (7J) symbol set, use something like this: "<165>2000 Synergetic Data Systems Inc.", or use an expression with the `CHR(num)` function or a hex literal, such as \$80\$ for character 128..

Point and Pitch Size

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

Fit and Wrap Options

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

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

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

Attribute Styles

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

Shading

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

Rotation

The "rotate" option will cause the text to be rotated around the baseline left edge at the angle specified. If the `-nohpgl` option is used while producing pcl laser output, then the angle must be 90, 180, or 270 degrees. Any angle can be specified for other formats.

Fixed and Proportional Spacing

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

Color

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

Justification

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

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

Weight and Style

Some laser printer fonts must be specified with given weight or style in order to be selected by the printer.

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

Get Text From Input Stream

If "getoffset" and "getcols" are specified in a syntax 2 command, then the value printed is taken from the data stream at the offset and length specified from each occurrence (any *text* value supplied is ignored).

Further, "erasetoffset" and "erasescols" can be used to remove any data stream text from the point of occurrence as well.

Barcode Note

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

Text in this syntax: "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.

Special Symbol Fonts

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

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

MICR Fonts

There are four standard MICR fonts available in all output formats. To utilize these, specify a font name of micr, micrbold (or micrb10), micrlight (or micrl15), or micrnarrow (or micrn1), along with 'fixed, 8' to ensure proper sizing. These fonts are default, bold, light, and narrow versions of the MICR fonts that are supplied with UnForm. Use these MICR text commands lines in cases where the micr command can't be used, such as with deposit slips, unusual bank account numbers, or non-standard check sizes.

Postscript Type1 Fonts

All Postscript printers include a base set of fonts, including Courier, Times-Roman, and Helvetica. The [psmap] section of ufparam.txt links font numbers to Postscript font names, and the most often used font numbers are mapped in the default ufparam.txt file. The [psmap] section looks like this:

```
[psmap]
# Maps pcl font numbers to postscript font names. Each number is
# set to up to four font filenames for normal, bold, italic, and
# bold-italic fonts in order. If a fontname.pfa or fontname.pfb
# file is present in the psfont directory, then it is downloaded when used.
# Such fonts must be Adobe Type1 fonts.
#
# psfont/<fontname>.afm files provide metrics.
```

```
4099=Courier,Courier-Bold,Courier-Oblique,Courier-BoldOblique
4101=Times-Roman,Times-Bold,Times-Italic,Times-BoldItalic
4102=Courier,Courier-Bold,Courier-Oblique,Courier-BoldOblique
4141=ZapfDingbats
```

In order to use custom Type1 fonts, follow these steps:

- Install the font's .pfa or .pfb files, as well as the metric file (.afm) in the psfont directory.

- Assign a unique number to the fonts, using the same normal, bold, italic, and bold-italic sequence found in the standard font mapping, if the font provides these versions.
- Optionally map a font name to the number in the [fonts] section of ufparam.txt.
- Text or font commands can now specify the font by number (font *n*) or name.

TrueType Fonts and Unicode

TrueType fonts can be configured by assigning font numbers to font file names in the [ttmap] section of ufparam.txt, in a manner similar to the [psmap] section described above. In the Windows environment, the system's Fonts directory is automatically searched as well as the ttfnt directory under the UnForm server, so there is no need to define full paths to system fonts on Windows.

When using a TrueType font, text data must be in Unicode format, where each character is represented with two bytes rather than one. Standard single-byte encodings therefore need to be converted to Unicode, by mapping the characters in the string to the proper double-byte characters in Unicode. Most character sets use the same characters in the 0-127 character range, but vary in the 128-255 range. Unicode can represent any of these characters with a specific 0-65535 value, allowing any character set to be represented in Unicode. The tables for most PCL symbol sets as well as several common character sets are included with UnForm in its unicode directory, and UnForm provides internal and code block functions to perform mapping.

Normally, UnForm will assume that the text data provided to the command is single-byte data, and it will internally map the text to a Unicode string based on the symbol set setting. The default symbol set (9J) matches the ISO-8859-1 character set, so that is the default translation.

If you wish to specify Unicode directly, you can do so in a variety of ways. You must add a "uc" or "unicode" option to the text command to indicate the text data is already in Unicode format.

- Use the touc(text\$,charset\$) function in an expression, which returns the Unicode version of text\$, given that text\$ is in the identified character set or symbol set. For example, {touc("Total Sales","9J")}.
- Specify raw double-byte sequences as hexadecimal in an expression, using \$hex\$ notation, such as {\$00300039\$}.
- Use the <xhex> syntax in literal text, such as "<x00300039>".

Turn Off HP/GL

In PCL output, some text

URL Links

The html5 driver supports the link and linkopt options, used to specify a URL hyperlink value, and HTML <a> tag options, respectively.

Examples:

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

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

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

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

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

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

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

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

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

text 10,1.5,"Sample Industrial",16,bold,helvetica,link "http://sample_indust.com", linkopt "target=_blank" will place a company title on the page and make it a hyperlink to a company website, to be opened in a new blank page, when producing HTML5 output.

Drivers: all. PDF driver fonts map to Courier, Helvetica, or Times-Roman based on the [pdfmap] section if ufparam.txt, and support only symbol set 9J (ISO8859-1 with extensions). PS maps based on the [psmap] section in ufparam.txt. 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 decimal justification, or wrap and fit options. **Light** and **underline** options are only supported by the pcl driver.

13.77 TITLE

Syntax

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

Description

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

The title command is also honored by the [UnForm Desktop Client](#), though expressions are not supported.

Drivers: pdf only

13.78 TRANSPARENCY

Syntax

transparency on|yes|off|no

Description

The transparency command turns on or off PDF transparency mode, which affects shading created by the image, text, box, and shade commands. When transparency mode is on, shading is implemented as a partial transparency, which allows objects to show through the shade region.

Transparency can also be controlled in the uf101d.ini file (pdftrans=*n*) or with the -pdftrans or -nopdftrans command line options.

Drivers: PDF only

13.79 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.

Trays can be varied by copy, to pull different paper for different copies.

The *paper-source* is printer dependent. Typically, tray 1 is an upper tray source, tray 2 is a manual feed source, and tray 4 and 5 are a lower and middle paper sources. These will likely not coincide with physical tray numbers labeled on the printer itself, unfortunately. To determine the proper tray values, see your printer's documentation for the paper source command.

The printer model's (-m command line option) PPD file (or generic pcl.ppd or ps.ppd files) can specify *InputSlot *paper-source* entries which are used if present.

When producing output to a *winprt* Windows printer, the tray command specifies a tray number that is assigned by the vendor, typically a number over 256, though there are pre-defined Windows values published by Microsoft that some vendors honor. A list of tray numbers for a Windows printer can be obtained with the system object, using the winprttrays\$(printer\$) method.

Example

tray 5

```
if copy 2
  tray 4
end if
```

Drivers: pcl, ps, *winprt* only

13.80 UNDERLINE

See the **bold** keyword.

13.81 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: pcl, ps, PDF

13.82 VLINE

Syntax

vine "*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 **vine** keywords in a rule set, if needed.

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

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

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

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

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

Example:

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

Drivers: all

PostScript input not supported.

13.83 VSHIFT

Syntax

vshift *n*

Description

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

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

Example:

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

Drivers: all

[AFO](#) jobs not supported.

13.84 WEBACTION

Syntax

webaction "*name*" [,style "*button CSS styles*"] [,beforebegin|afterbegin|beforeend|afterend "*selector*"] [,tip "*tooltip*"] [,autorun]

Description

Used by the [SDSI Web Extension](#) as part of an integration rule set. This command creates a small clickable HTML button with the UnForm icon. When clicked, the web form defined by other elements of the rule set is presented. The button by default is placed in the upper right corner of the page or frame. If the beforebegin, afterbegin, beforeend, or afterend option is specified, then an element will be selected using the provided CSS *selector*, and the button will be placed relative to that element, within the normal HTML page flow.

The *name* value is supplied in the cgi.action\$ value when the associated web form is submitted.

The style option can be used to add CSS style attributes to the button, to adjust the position, size, color, border, and so on. The style value is appended to the default style and can therefore override any setting of the default style.

If a tip is provided, it is added as a HTML title attribute, which normally displays as hover text and can be used for small help information.

If autorun is specified, the web form is not presented, but is automatically submitted.

Examples

This creates an action button moved down from the upper right corner, with a default opacity of 60% (partially transparent).

```
webaction "Default", style "margin-top:2.5em;opacity:.60", tip "UnForm web form action"
```

This creates an action button anchored to a table cell that is the adjacent sibling of a table header cell containing "Doc ID", itself a descendent of a table that is a descendent of a div tag with an id attribute of "panel1". The button is placed at the end of the cell's content, but inside the closing </td> tag.

```
webaction "DocID", beforeend "div#panel1 table th:contains(Doc ID) + td", autorun
```

13.85 WEBFIELD

Syntax

```
webfield "name", "selector"[, attributes]
```

Description

Used by the [SDSI Web Extension](#) as part of an integration rule set. This command creates an input field whose value is derived from the *selector*, which is a jQuery/CSS selector that identifies a particular element in the web page or frame. If multiple elements are selected, the first one is used. The value is either the element's "value" attribute, if present, or its innerText property.

The *name* value is used as a label for the field, and as `cgi.field_name$` when the form is submitted. Spaces are converted to underscores in the cgi field name (i.e. `cgi.field_doc_id$`).

To add additional attributes to the input field, specify them as HTML attribute tags after the selector option. Attributes should be space-separated, as they are added directly to the <input> tag.

Examples

This creates a web field whose value is that of an <input name="invoiceno"> field that is a descendent of a form whose ID is "custinv". The field is styled for width, and is readonly.

```
webfield "Doc ID", "form#custinv input[name=invoiceno]", style="width:10em;" readonly
```

13.86 WEBITEM

Syntax

webitem "*name*", input|textarea|radio|checkbox|select|button ["*list*"] [, *attributes*]

Description

Used by the [SDSI Web Extension](#) as part of an integration rule set. This command creates an input widget of the type specified, and with additional attributes specified. In the case of a radio, checkbox, or select widget, a semi-colon delimited list is provided with the available values.

The *name* value is used as a label for the field, and as `cgi.field_name$` when the form is submitted. Spaces are converted to underscores in the cgi field name (i.e. `cgi.field_doc_id$`).

To add additional attributes to input, textarea, button, or select widgets, specify them as HTML attribute tags. Attributes should be space-separated, as they are added directly to the widget tag.

Examples

This creates a simple input element, 120 pixels wide, with a placeholder (suggestion text).

```
webitem "Comment", input, style="width:120px;" placeholder="Enter a comment"
```

This creates three radio buttons, one of which can be selected. When submitted, `cgi.field_options$` will have the checked value.

```
webitem "Options", radio "Option 1;Option 2;Option 3"
```

This creates a password field that will obscure the input.

```
webitem "PIN Code", input, type="password" size="6"
```

This creates a submission button, which when clicked will cause the web form to be submitted with `cgi.button$="Get Invoice"`.

```
webitem "Get Invoice", button, style "padding-left:5px;padding-right:5px;"
```

13.87 WEBPANEL

Syntax

webpanel "*title*"

Description

Used by the [SDSI Web Extension](#) as part of an integration rule set. This command creates a tab panel to house the webfields and webitems that follow. Multiple webpanel commands can be used to organize the web form into distinct sections, one of which is visible at a time. When the form is submitted, `cgi.panel$` will contain the value of *title* for the active panel.

13.88 ZCOPIES, ZDARKNESS, ZSPEED

See the **lcopies**, **ldarkness**, and **lspeed** commands.

14 SAMPLE RULE FILES

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

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

```
uf101c -i sample1.txt -f simple.rul -r simple1 -o output-device
```

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

```
uf101c -i sample-file -f advanced.rul -o output-device
```

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

```
uf101c -i sample1.txt -f advanced.rul -o "|lp -dhp -oraw"
```

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

```
uf101c -i sample1.txt -f advanced.rul -p pdf -o invoice_sample.pdf.
```

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

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

14.1 simple.rul

Enter topic text here.

14.1.1 SIMPLE1 - invoice rule set

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

```
uf101c -i sample1.txt -f simple.rul -p pdf -o client:simple1.pdf
```

A title header prefixes all rule sets, which is just a unique name enclosed in brackets.
[simple1]

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

The detect statements below indicate that

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

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

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

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

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

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

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

20% shading between rows 6 and 8.

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

invoice # section

cbox 60,4,80.5,8,crows=6 8::20,ccols=69 78

text 61,7,"Date",univers,italic,10

text 70,7,"Invoice #",univers,italic,10

text 79,7,"Pg",univers,italic,10

bill to / ship to section

cbox .5,10,80.5,18.5,5,ccols=7::20 43.5 50::20

text 2,11,"Sold To",cgtimes,italic,10

text 45,11,"Ship To",cgtimes,italic,10

ribbon section

cbox .5,18.5,80.5,22.5,5,crows=20.5::20,ccols=9 18 25 65

special internal grid in ribbon box

cbox 29,18.5,65,21.5

cbox 42,18.5,56,21.5

text 1,19,"Order\nNumber",univers,italic,10

text 10,19,"Order\nDate",univers,italic,10

text 19,19,"Cust.\nNumber",univers,italic,10

text 26,19,"Sls\nPrs",univers,italic,10

text 30,19,"Purchase\nOrder No.",univers,italic,10

text 43,19,"Ship Via",univers,italic,10

text 57,19,"Ship\nDate",univers,italic,10

text 66,19,"Terms",univers,10,italic

detail section

cbox .5,22.5,80.5,56.5,5,crows=24.5::10,ccols=5 10 16 51 55 69

text 1,23,"Qty\nOrd",univers,italic,10

text 6,23,"Qty\nShip",univers,italic,10

text 11,23,"Qty\nBkord",univers,10,italic

text 17,23,"Item & Description",univers,italic,10

text 52,23,"U/M",univers,italic,10

text 56,23,"Unit\nPrice",univers,italic,10

text 70,23,"Extended\nPrice",univers,italic,10

footer section

cbox 57,57,80.5,65,crows=59 63,ccols=69::20

text 58,58,"Sales Amt",univers,11

text 58,61,"Sales Tax",univers,11

text 58,62,"Freight",univers,11

text 58,64.25,"TOTAL",univers,bold,14

14.1.2 SIMPLE2 – add font control and text

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


```
uf101c -i sample1.txt -f simple.rul -r simple2 -p pdf -o client:simple2.pdf
```

```
[simple2]
```

```
# to use this rule set, you need to FORCE the rule set with the -r option
# or remark (#) out the detect command in the rule sets above.
```

```
#
```

```
# This rule set takes the rule set above and improves it by adding
# fonting and justification.
```

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

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

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

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

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

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

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

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

```
# invoice # section
cbox 67,4,80.5,10,1,crows=6 8,ccols=74::20
text 68,5,"Date",univers,italic,10
text 68,7,"Invoice",univers,italic,10
text 68,9,"Page #",univers,italic,10
```

```
# cut data from old position and place in new
text 75,5,{cut(61,5,8,"")},cgtimes,bold,10
text 75,7,{cut(71,5,7,"")},cgtimes,bold,10
text 75,9,{cut(79,5,2,"")},cgtimes,bold,10

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

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

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

```
cfont 1,21,8,21,cgtimes,bold,10,center # order #
cfont 10,21,17,21,cgtimes,bold,10,center # order date
cfont 19,21,24,21,cgtimes,bold,10,center # cust #
cfont 26,21,28,21,cgtimes,bold,10,left # sls prs code
cfont 26,22,64,22,cgtimes,bold,10,left # sls prs name
cfont 30,21,41,21,cgtimes,bold,10,center # po #
cfont 43,21,55,21,cgtimes,bold,10,center # ship via
cfont 57,21,64,21,cgtimes,bold,10,center # ship date
cfont 66,21,80,22,cgtimes,10,center # terms

# detail section
cbox .5,22.5,80.5,56.5,5,crows=24.5::10,ccols=5 10 16 51 55 67
text 1,23,"Qty\nOrd",univers,italic,10,right,cols=4
text 6,23,"Qty\nShip",univers,italic,10,right,cols=4
text 11,23,"Qty\nBkord",univers,10,italic,right,cols=4
text 17,23,"Item & Description",univers,italic,10
text 52,23,"U/M",univers,italic,10,center,cols=3
text 56,23,"Unit\nPrice",univers,italic,10,right,cols=11
text 68,23,"Extended\nPrice",univers,italic,10,right,cols=12
```

This section performs two distinct fonting functions. First, the detail item columns are fonted. Note that you can't simply font the entire detail section in a proportional font, as the spacing between columns would be lost. Instead, each column is fonted individually.

However, the data stream for the invoice also contains memo lines in the middle of the detail item lines, and those memo lines should not be broken into individual columns.

Therefore, an additional font command is added after the column fonting, which will override any font characteristics defined for any given data position in a prior font command. This memo section fonting uses a technique that will scan the page for a pattern (in this example, 4 spaces in the region outlined by column 1, row 25 through column 4, row 56), and change font characteristics relative to those locations found. In this example, wherever the 4 spaces are found, fonting will occur 17 columns to the right, 0 rows down, for 60 columns and 1 row. These are the memo lines found in the midst of the item detail lines.

```

cfont 1,25,4,56,cgtimes,10,bold,right    # qty ord
cfont 6,25,9,56,cgtimes,10,bold,right    # qty shipped
cfont 11,25,15,56,cgtimes,10,bold,right   # qty b/o
cfont 17,25,50,56,cgtimes,10,left        # item # & desc
cfont 52,25,54,56,cgtimes,10,bold,center # u/m
cfont 56,25,66,56,cgtimes,10,bold,right   # unit price
cfont 68,25,79,56,cgtimes,10,bold,right   # extended

# memo section
font " @1,25,4,56",17,0,60,1,cgtimes,10,left

# footer section
cbox 57,57,80.5,65,crows=59 63,ccols=67::20
text 58,58,"Sales Amt",univers,11
cfont 58,60,66,60,univers,11,left
text 58,61,"Sales Tax",univers,11
text 58,62,"Freight",univers,11
text 58,64.25,"TOTAL",univers,bold,14
cfont 68,58,79,65,cgtimes,bold,right,14    # totals

```

14.1.3 SIMPLE3 – add copies, barcode, watermark

This rule set adds copy handling, a watermark, and a barcode. To produce this sample, use this command:

```
uf101c -i sample1.txt -f simple.rul -r simple3 -p pdf -o client:simple3.pdf
```

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

[simple3]

to use this rule set, you need to FORCE the rule set with the -r option

or remark (#) out the detect command in the rule sets above.

#

This rule set takes the rule set above and improves it by adding
copies, watermark, and a barcode.

```
dsn_sample "sample1.txt"
detect 61,5,"~.././.. ....." # invoice date and #
detect 9,11,"~....."           # customer code
detect 10,21,"~.././.. ....." # ord date and cust code
```

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

This rule set produces two copies of each page, with each copy sequentially produced as each page is read from the data stream. The pcopies command indicates this page-oriented copy production. There is also a copies command, which produces job-oriented copies for laser jobs. Note that PDF output always is produced as page-oriented copies, whether copies or pcopies is used.

When copies are produced, all rule set content that is not bracketed within 'if copy' blocks is produced on all copies. The majority of this rule set is outside of such blocks, so the majority will be applied to both copies. Near the bottom of the rule set is some code that will apply distinctly to each copy.

```
# copies
pcopies 2
```

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

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

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

```
# ribbon section
cbox .5,18.5,80.5,22.5,5,crows=20.5::20,ccols=9 18 25 65
# special internal grid in ribbon box
```

```
cbox 29,18.5,65,21.5
cbox 42,18.5,56,21.5
text 1,19,"Order\nNumber",univers,italic,10,center,cols=8
text 10,19,"Order\nDate",univers,italic,10,center,cols=8
text 19,19,"Cust.\nNumber",univers,italic,10,center,cols=6
text 26,19,"Sls\nPrs",univers,italic,10,center,cols=3
text 30,19,"Purchase\nOrder No.",univers,italic,10,center,cols=12
text 43,19,"Ship Via",univers,italic,10,center,cols=13
text 57,19,"Ship\nDate",univers,italic,10,center,cols=8
text 66,19,"Terms",univers,italic,10,center,cols=14
```

```
cfont 1,21,8,21,cgtimes,bold,10,center # order #
cfont 10,21,17,21,cgtimes,bold,10,center # order date
cfont 19,21,24,21,cgtimes,bold,10,center # cust #
cfont 26,21,28,21,cgtimes,bold,10,left # sls prs code
cfont 26,22,64,22,cgtimes,bold,10,left # sls prs name
cfont 30,21,41,21,cgtimes,bold,10,center # po #
cfont 43,21,55,21,cgtimes,bold,10,center # ship via
cfont 57,21,64,21,cgtimes,bold,10,center # ship date
cfont 66,21,80,22,cgtimes,10,center # terms
```

detail section

```
cbox .5,22.5,80.5,56.5,5,crows=24.5::10,ccols=5 10 16 51 55 67
text 1,23,"Qty\nOrd",univers,italic,10,right,cols=4
text 6,23,"Qty\nShip",univers,italic,10,right,cols=4
text 11,23,"Qty\nBkord",univers,10,italic,right,cols=4
text 17,23,"Item & Description",univers,italic,10
text 52,23,"U/M",univers,italic,10,center,cols=3
text 56,23,"Unit\nPrice",univers,italic,10,right,cols=11
text 68,23,"Extended\nPrice",univers,italic,10,right,cols=12
```

```
cfont 1,25,4,56,cgtimes,10,bold,right # qty ord
cfont 6,25,9,56,cgtimes,10,bold,right # qty shipped
cfont 11,25,15,56,cgtimes,10,bold,right # qty b/o
cfont 17,25,50,56,cgtimes,10,left # item # & desc
cfont 52,25,54,56,cgtimes,10,bold,center # u/m
cfont 56,25,66,56,cgtimes,10,bold,right # unit price
cfont 68,25,79,56,cgtimes,10,bold,right # extended
```

memo section

```
font " @1,25,4,56",17,0,60,1,cgtimes,10,left
```

This text line adds a large text watermark on line 56, centered horizontally. The text is printed in cgtimes, 120 point, with 10% shading applied.

.

watermark

```
text 1,56,"INVOICE",cgtimes,120,shade=10,center,cols=80,fit
```

footer section

```
cbox 57,57,80.5,65,crows=59 63,ccols=67::20
text 58,58,"Sales Amt",univers,11
cfont 58,60,66,60,univers,11,left
text 58,61,"Sales Tax",univers,11
```

```
text 58,62,"Freight",univers,11
text 58,64.25,"TOTAL",univers,bold,14
cfont 68,58,79,65,cgtimes,bold,right,14      # totals
```

The barcode command can be used to add barcodes in many symbologies. It is similar to other commands, in that you provide a column, row, and value. In addition, you specify a symbology (400 is Code 3 of 9), a point size or pixel height (14.0, being a decimal rather than integer value, is treated as point size), and a bar spacing value in pixels. Like most commands, you can use expressions in the value element of the command. In this example, the data stream data from column 9, row 11, for 6 characters is used on each page, using the get() function within an expression.

```
text 2,58,"Customer code as 3 of 9 barcode",univers,italic,10
barcode 2,58.67,{get(9,11,6)},400,14.0,4
```

The following lines produce different output for each of the two copies. Copy 1 is labeled with a text command to say it is the "Customer Copy", while copy 2 is labeled as "Accounting Copy". Any commands outside of 'if copy' blocks are applied to all copies.

```
# copy name section
if copy 1
    text 1,65.5,"Customer Copy",univers,12,bold,center,cols=80
end if
if copy 2
    text 1,65.5,"Accounting Copy",univers,12,bold,center,cols=80
end if
```

14.1.4 SIMPLE4 – constants, color, expressions

This rule set demonstrates the use of constants, graphical shading, colors, and expressions to produce explanatory notes in the document. To produce this sample, use this command:

```
uf101c -i sample1.txt -f simple.rul -r simple4 -p pdf -o client:simple4.pdf
```

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

```
[simple4]
# to use this rule set, you need to FORCE the rule set with the -r option
# or remark (#) out the detect command in the rule sets above.
#
# This rule set takes the rule set above and improves it by adding
# constants, graphical shading, increases the resolution,
# and adds explanatory text commands for cust # and ship to #.
#
# Also adds a copy for a packing slip with no prices.
```

```
dsn_sample "sample1.txt"
detect 61,5,"~../.. ....." # invoice date and #
detect 9,11,"~....."         # customer code
detect 10,21,"~../.. ....." # ord date and cust code
```

Constants are simple text names that are replaced by values later in the rule set. They can be used to simplify maintenance of the rule set, or to make it easier to read. In this example, a series of constants is defined using the const command, and you will find the names referenced throughout the balance of the rule set.

```
const MAXCOLS=80          # max cols to output
const MAXRCOLS=79        # MAXCOLS-1
const LEFTCOL=.5         # use 1 if empty
const RIGHTCOL=80.5      # use LEFTCOL for symmetry
const MAXROWS=66         # max rows to output
```

```
cols MAXCOLS
rows MAXROWS
```

```
# copies
const CUSTOMER_COPY=1
const FILE_COPY=2
const PACK_COPY=3
pcopies 3
```

The dpi setting applies to laser output only, and instructs the printer to produce output at 600 dpi, providing a typically crisper, more professional look. In addition, the gs on command turns on graphical shading mode, so that shade regions and shaded text are rendered as graphical data rather than using pcl's internal, typically coarse, shade macros.

```
dpi 600
gs on          # turn on graphical shading
```

```
# enhancement constants
const HSHADE=30
const ISHADE=20
const DSHADE=10
const HFONT=univers,11
const IFONT=univers,italic,10
const DFONT=cgtimes,10
const DBFONT=DFONT,bold
```

```
# header section
cbox LEFTCOL,.5,RIGHTCOL,{MAXROWS+.5},5
image 1,1,12,6,"sdsilogo.pcl"
text 1,2,"Company Name",HFONT,14,bold,center,cols=MAXRCOLS
text 1,3,"Company Address\nCompany City, St Zipcode\nCompany
Phone",HFONT,12,bold,center,cols=MAXRCOLS
text 1,6,"Web: www.myweb.com\nEmail: sales@myweb.com",HFONT,bold,center,cols=MAXRCOLS
text 1,2,"INVOICE",HFONT,16,bold,right,cols=MAXRCOLS
```

```
# invoice # section
cbox 67,4,RIGHTCOL,10,1,crows=6 8,ccols=74::ISHADE
text 68,5,"Date",IFONT
text 68,7,"Invoice",IFONT
text 68,9,"Page #",IFONT
# cut data from old position and place in new
text 75,5,{cut(61,5,8,"")},DBFONT
text 75,7,{cut(71,5,7,"")},DBFONT
```

```
text 75,9,{cut(79,5,2,"")},DBFONT
```

The cbox command shown here uses constants defined above, plus shows the use of color options, which are supported by PDF and color laser output. In this example, the interior is colored in cyan, and the lines are colored in blue. Alternately, RGB hex triplets (such as 800000 for dark red) can be specified using the rgb, scolor rgb, or lcolor rgb options.

```
# bill to / ship to section
cerase 1,11,MAXCOLS,11 # erase cust#,ship# used later in text commands
cbox LEFTCOL,11,RIGHTCOL,18.5,5,cyan,lcolor=blue,ccols=7::ISHADE 43.5 50::ISHADE
text 2,12,"Sold To",IFONT,center,cols=5
cfont 8,11,40,11,DBFONT,left
cfont 8,12,40,15,DBFONT # sold to address
```

This text command shows an example of how to use an expression to construct a message using a combination of hard-coded text and information from the data stream. In this example, the phrase "Your customer code is" is concatenated with the data at column 9, row 11, for 6 characters, on each page, and the result is printed at column 9, row 18, using the specifications provided by the constant IFONT, defined earlier in the rule set.

```
text 8,18,{"Your customer code is "+get(9,11,6)+"."},IFONT
```

```
text 45,12,"Ship To",IFONT,center,cols=5
cfont 51,11,80,11,DBFONT,left
text 51,12,{mcut(51,12,30,4,"","Y","Y")},DBFONT
text 51,18,{"Your ship to code is "+get(55,11,6)+"."},IFONT
```

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

```
cfont 1,21,8,21,DBFONT,center # order #
cfont 10,21,17,21,DBFONT,center # order date
cfont 19,21,24,21,DBFONT,center # cust #
cfont 26,21,28,21,DBFONT,left # sls prs code
cfont 26,22,64,22,DBFONT,left # sls prs name
cfont 30,21,41,21,DBFONT,center # po #
cfont 43,21,55,21,DBFONT,center # ship via
cfont 57,21,64,21,DBFONT,center # ship date
cfont 66,21,80,22,DBFONT,center # terms
```

```
# detail section
if copy PACK_COPY
```



```

        erase "~\.[0-9][0-9]@62,25,79,56",-6,0,11,1
    endif
cbox LEFTCOL,22.5,RIGHTCOL,56.5,5,crows=24.5::DSHADE,ccols=5 10 16 51 55 67
text 1,23,"Qty\nOrd",IFONT,right,cols=4
text 6,23,"Qty\nShip",IFONT,right,cols=4
text 11,23,"Qty\nBkord",IFONT,right,cols=4
text 17,23,"\nItem & Description",IFONT
text 52,23,"Unit\nM",IFONT,center,cols=3
text 56,23,"Unit\nPrice",IFONT,right,cols=11
text 68,23,"Extended\nPrice",IFONT,right,cols=12

cfont 1,25,4,56,DBFONT,right      # qty ord
cfont 6,25,9,56,DBFONT,right      # qty shipped
cfont 11,25,15,56,DBFONT,right    # qty b/o
cfont 17,25,50,56,DFONT,left      # item # & desc
cfont 52,25,54,56,DBFONT,center   # u/m
cfont 56,25,66,56,DBFONT,right    # unit price
cfont 68,25,79,56,DBFONT,right    # extended

# memo section
font "    @1,25,4,56",17,0,60,1,DFONT,left

# watermark
if copy CUSTOMER_COPY,FILE_COPY
    text 1,56,"INVOICE",DFONT,120,shade=DSHADE,center,cols=MAXCOLS,fit
endif
if copy PACK_COPY
    text 1,56,"PACK SLIP",DFONT,120,shade=DSHADE,center,cols=MAXCOLS,fit
endif

# footer section
cbox 57,57,RIGHTCOL,65,icolor=red,crows=59 63,ccols=67::HSHADE
text 58,58,"Sales Amt",HFONT
cfont 58,60,66,60,HFONT,left
text 58,61,"Sales Tax",HFONT
text 58,62,"Freight",HFONT
text 58,64.25,"TOTAL",HFONT,bold,14
cfont 68,58,79,65,DBFONT,right,14      # totals

text 2,58,"Customer code as 3 of 9 barcode",IFONT
barcode 2,58.67,{get(9,11,6)},400,14.0,4

```

Note the use of constants to make this section easier to read.

```

# copy name section
if copy CUSTOMER_COPY
    text 1,65.5,"Customer Copy",HFONT,12,bold,center,cols=MAXCOLS
endif
if copy FILE_COPY
    text 1,65.5,"Accounting Copy",HFONT,12,bold,center,cols=MAXCOLS
endif
if copy PACK_COPY
    text 1,65.5,"Packing Slip",HFONT,12,bold,center,cols=MAXCOLS
endif

```

end if

This text line demonstrates the use of multi-line text forced to fit within a certain number of columns. UnForm scans each of the two lines (delimited by the \n character sequence, or it could contain data with line-feed or carriage-return line-feed delimiters) to determine the width, beginning with the point size 12 specified in the command. The size is reduced until both lines will fit within the 20 columns specified with the cols option. Once the correct point size is determined, the lines are spaced normally for that height. For example, if the size required is 8.25 points, then the lines will be spaced 8.25 points apart. If spacing had been set to 1.5, then the lines would be spaced 12.33 points apart.

```
text 2,62,"This sample message text, which contains\nline breaks, is sized to fit in 20 columns.",cols
20,cgtimes,12,fit,spacing 1
```

14.2 advanced.rul

Enter topic text here.

14.2.1 INVOICE - from preprinted to UnForm

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.

```
uf101c -i sample1.txt -f advanced.rul -p pdf -o client:invoice.pdf
```

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

```
[Invoice]
```

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

The detect statements below indicate that

```
* a date (mm/dd/yy format) followed by 2 spaces, followed by 7 more characters will appear at column
61, row 5
* 6 characters will appear at column 9, row 11
* a date, a space, and 6 characters will appear at column 10, row 21
detect 61,5,"~../../.. ....." # invoice date and
#detect 9,11,"~....." # customer codedetect
10,21,"~../../.. ....." # ord date and cust code
```

The following lines define several constants that are used elsewhere in the rule set. Wherever the constant names appear in a command, the value is substituted. Constants are not variables and are not interpreted while the job is processed. They are simply literal placeholders used while UnForm reads rule set lines.

```
# set up document constantsconst MAXCOLS=80
# max cols to outputconst MAXRCOLS=79
```

```

# MAXCOLS-1const LEFTCOL=.5
if emptyconst RIGHTCOL=80.5
for symmetryconst MAXROWS=66
output
# use 1
# use LEFTCOL
# max rows to

```

The following lines define the page size and orientation. The dpi command sets the printer to 600 dots per inch. The rows and cols commands set the dimension for positioning and scaling. All positioning will be based on 80 columns and 66 rows appearing within the printed margins of the page. The gs on command triggers the use of graphical shading, which improves the look of shade regions over the native pcl shading of most laser printers, especially at higher dpi settings and shade percentages. In addition, UnForm will generate two copies of the job, with each page producing two copies as processed (collated).

```

portraitdpi 600gs on
graphical shadingcols MAXCOLS
max output columnsrows MAXROWS
max output rows# to print more copies, increase value and add
copy titles in prejob
pcopies 2
#
#
#
# max # of copies

```

If this rule set is used to produce a PDF document, then the title of "Sample Invoice" will be added to the PDF file. For laser output, the title command is ignored.

```

title "Invoice Sample"
properties
# view in PDF

```

The prejob code block is executed once at the beginning of the job, after the first page of data has been read and the rule set parsed. This example is simply setting a variable form_title\$ to a literal value INVOICE. This variable is used later in the rule set.

The prepage code block is executed once per page, just after UnForm has read the text for the page, but before any copies of that page have been printed. Within a prepage code block, you can insert any valid Business Basic code (though you need to be careful not to insert any UnForm commands.) This code initializes a variable shipzip\$ to null, then looks for a regular expression pattern of 5 digits on line 15. If the pattern is found, it sets shipzip\$ to the zip code. After the code block is closed, a barcode command is used to place a postnet barcode below the shipping address. The barcode command uses the syntax "{shipzip\$}", indicating the expression shipzip\$ should be used to generate the data to barcode.

Once the prepage code block creates shipzip\$, it then scans a range of rows looking for special memo format lines. It marks these lines with the characters "mL" in the first two columns. Later in the rule set, you'll see how these markers are used to treat memo lines differently than standard invoice lines.

The order of execution is controlled by UnForm. There is actually no need to place the barcode command below the prepage code block, as UnForm will properly execute the code block before any form commands are executed at run-time.

```

prejob { # set up variables needed by merged routines below
# if form title changes per page,
# set up in prepage routine below
form_title$="INVOICE"
}

```

```

prepage {
    # find zip code in city,state,zip line for bar code
    shipzip$=""
    # regular expression of 5 digits on line 15
    x=mask(text$[15],"[0-9][0-9][0-9][0-9][0-9]")
    if x>0 then shipzip$=get(x,15,5)

    # mark memo lines for special handling in detail section
below
    # memo start in column 28 with all spaces before
    for i=25 to 56
        if len(text$[i])>27 and trim(text$[i](1,27))=""
then \
        text$[i](1,2)="mL"
    next i
}

```

The pdf driver supports the ability to email the PDF file created using the email command. The commented # email line below provides an example of the command. It requires four arguments, each of which can be a literal string value or a string expression enclosed in braces. In order for the email command to work, the mailcall.ini file must be properly configured for your system.

```

# When run in PDF mode, and if mailcall.ini is configured
properly,# and if the system can communicate with the mail
server, then the
# next line would send the PDF invoice as an attachment to an
email.
# email "someone@somewhere.com","me@mycompany.com", \
# {"A test invoice "+cvs(get(71,5,7),3)}, \
# "Attached is a sample invoice\n"

```

The next group of commands creates a page header with box and text commands. The box commands are given as the cbox variant, which accepts two pairs of numbers as opposite corners of the box. Some of the commands are stored in a different rule set, called "Mrg Form Header". This rule set is also located in the advanced.rul file. The lines in that rule set are merged in here as if they were part of this rule set.

Note that some of the text commands, and also a barcode command, use an expression rather than a literal. An expression is an executable value assignment enclosed in braces. For example, one text command uses an expression {cut(61,5,8,"")}, which cuts out the text at column 61, row 5, for 8 columns, returning the result, while setting those positions to "". The result is printing at position 75,5 what was at position 61,5.

```

# heading sectionconst HFONT=univers,12
    # headingsbox LEFTCOL,1,RIGHTCOL,MAXROWS,5 #
complete page boxmerge "Mrg Form Header"
    # merge std hdr rules# right top ribbonconst
HFONT=univers,11,italic # headings

```

```

const DFONT=cgtimes,11,bold                                # data

# draw info box with internal grid and shading
# horizontal lines at 6 and 8
# vertical line at 74 with shading between 67 and 74
cbox 67,4,RIGHTCOL,10,5,crows=6 8,ccols=74::20
text 68,5,"Date",HFONT
text 68,7,"Invoice",HFONT
text 68,9,"Page #",HFONT
# cut data from old position and place in new
text 75,5,{cut(61,5,8,"")},DFONT
text 75,7,{cut(71,5,7,"")},DFONT
text 75,9,{cut(79,5,2,"")},DFONT

# sold to section
cbox LEFTCOL,10,41,18.5,5
cbox LEFTCOL,10,41,11.25,0,10
text 8,10.75,"SOLD TO",HFONT,bold
cfont 8,12,40,15,DFONT                                     # sold to address
if copy 1
    barcode 8,16,{shipzip$},900,10.1,2
end if
text 2,18,{"Your customer code is "+cut(9,11,6,"")
+"."},8,cgtimes

# ship to section
cbox 41,10,RIGHTCOL,18.5,5
cbox 41,10,RIGHTCOL,11.25,0,10
text 48,10.75,"SHIP TO",HFONT,bold
# cut ship to address and place in new position
text 48,12,{mcut(51,12,30,4,"","Y","Y")},DFONT
text 43,18,{"Your ship to code is "+cut(55,11,6,"")
+"."},8,cgtimes

```

This section draws order detail boxes and headings. The first cbox command draws a grid, using the internal crows and ccols options. In addition to the boxes and headings, the font used for the data from the input stream is changed using a series of cfont commands, one for each section.

```

# ribbon sectionconst L1=19const L2=20# draw info box with
internal grid and shading# horizontal line at 20.5 with
shading between 18.5 and 20.5# vertical lines at 9, 18, 25,
and 65cbox
LEFTCOL,18.5,RIGHTCOL,22.5,5,crows=20.5::20,ccols=9 18 25 65
# special internal grid in ribbon box
cbox 29,18.5,65,21.5
cbox 42,18.5,56,21.5
# ribbon headings
text 1,L1,"Order",HFONT,right,cols=8
text 1,L2,"Number",HFONT,right,cols=8
text 10,L1,"Order",HFONT,center,cols=8

```

```

text 10,L2,"Date",HFONT,center,cols=8
text 19,L1,"Cust.",HFONT
text 19,L2,"Number",HFONT
text 26,L1,"Sls",HFONT
text 26,L2,"Prs",HFONT
text 30,L1,"Purchase",HFONT
text 30,L2,"Order No.",HFONT
text 43,L2,"Ship Via",HFONT
text 57,L1,"Ship",HFONT,center,cols=8
text 57,L2,"Date",HFONT,center,cols=8
text 66,L2,"Terms",HFONT
# ribbon data
cfont 1,21,8,21,DFONT,right           # order #
cfont 10,21,17,21,DFONT,center         # order date
cfont 19,21,24,21,DFONT                # cust #
cfont 26,21,28,21,DFONT                # sls prs
code
cfont 26,22,64,22,DFONT                # sls prs
name
cfont 30,21,41,21,DFONT                # po #
cfont 43,21,55,21,DFONT                # ship via
cfont 57,21,64,21,DFONT,center         # ship date
cfont 66,21,MAXCOLS,22,DFONT           # terms

```

This section of lines controls the formatting of the invoice detail lines. A grid is drawn around the column headers and detail lines. The column headers are shaded. Item detail lines are fonted using a series of font commands that look for the pattern "~\.[0-9][0-9][0-9]" which is a period followed by 4 digits. Wherever that occurs, font changes are made relative to that position. Similarly, the memo lines identified by the prepage code block and marked with the text marker "mL" are fonted with a different column structure. In addition to the font command, an erase command is used to remove the text markers.

```

# detail section# detail headingsconst L1=23const L2=24# draw
info box with internal grid and shading# horizontal line at
24.5 with shading between 22.5 and 24.5
# vertical lines at 5, 10, 16, 51, 55, and 67
cbox LEFTCOL,22.5,RIGHTCOL,56.5,5,crows=24.5::10, \
ccols=5 10 16 51 55 67
text 1,L1,"Qty",HFONT,right,cols=4
text 1,L2,"Ord",HFONT,right,cols=4
text 6,L1,"Qty",HFONT,right,cols=4
text 6,L2,"Ship",HFONT,right,cols=4
text 11,L1,"Qty",HFONT,right,cols=5
text 11,L2,"Bkord",HFONT,right,cols=5
text 20,L2,"Item & Description",HFONT
text 52,L2,"U/M",HFONT,center,cols=3
text 56,L1,"Unit",HFONT,right,cols=11
text 56,L2,"Price",HFONT,right,cols=11
text 68,L1,"Extended",HFONT,right,cols=12
text 68,L2,"Price",HFONT,right,cols=12

```

```

# detail data
# Modify fonts for lines. As comments may be present in the
same rows,
# use a pattern to locate the .nnnn in the price column,
# which indicates a part number line.
# Use a prepage routine to find the comments and change their
font.
font "~\.[0-9][0-9][0-9][0-9]",-61,0,4,1,DFONT,right # qty
ord
font "~\.[0-9][0-9][0-9][0-9]",-56,0,4,1,DFONT,right # qty
shipped
font "~\.[0-9][0-9][0-9][0-9]",-50,0,4,1,DFONT,right # qty
b/o
font "~\.[0-9][0-9][0-9][0-9]",-42,0,30,2,DFONT          # item #
& desc
font "~\.[0-9][0-9][0-9][0-9]",-10,0,3,1,DFONT,center      #
u/m
font "~\.[0-9][0-9][0-9][0-9]",-6,0,11,1,DFONT,right # unit
price
font "~\.[0-9][0-9][0-9][0-9]",6,0,12,1,DFONT,right # ext
price

# handle memo lines
# inserted 'mL' in prepage above
font "mL@1,25,2,56",10,0,63,1,HFONT
erase "mL@1,25,2,56",0,0,2,1

```

Watermark text is placed in the middle of the detail lines. This text is centered between column 1 and MAXCOLS, is rendered at 120 points, and is printed at 20% gray shading.

```

# watermark - large font with light shading
text 1,52,{form_title$},cgimes,120,shade 20,center,cols=MAXCOLS

```

The totals section is formatted like the other sections, with a grid, text headings, and font changes that apply to the input stream text.

```

# totals section# draw info box with internal grid and
shading# horizontal lines at 59 and 63# vertical line at 69
with shading between 58 and 69cbox
58,57,RIGHTCOL,65,5,ccols=69::20,crows=59 63
text 59,58,"Sales Amt",HFONT
text 59,61,"Sales Tax",HFONT
text 59,62,"Freight",HFONT
text 59,64.25,"TOTAL",HFONT,bold,14
cfont 59,60,68,60,HFONT          # disc
hdr
cfont 70,58,MAXRCOLS,65,DFONT,14,decimal          # totals

```

These text lines simply demonstrate some of UnForm's paragraph features. The first text command forces the longest line in the paragraph to fit within the number of defined columns. The maximum point size is 12, but that may be adjusted down to accommodate the longest line. Lines are delimited by the \n

character sequence, or by a CHR(10) within an expression. Line spacing is determined by the final point size, and may be adjusted with the spacing option. For example, if the rendered size is 8 point, then the spacing of 1 will result in 9 lines per inch ($9 \times 8=72$), while spacing of 1.5 would result in 6 lines per inch ($9/1.5=6$).

The second example will force use the defined point size to render the text, but any lines wider than the specified columns will be word-wrapped.

The third example shows how to use a specified ASCII value in a text command. The ASCII value 174, when printed using the symbol set 9J, is a trademark symbol. This technique can be used to print Latin characters and special symbols. The symbol set determines what any given character value prints as. The 9J symbol set is the default. See the -testpr command line option for viewing printed tables of different symbol sets.

```
# footer section# These lines show fitting and wrapping of
text
```

```
text 2,60,"This sample message text, which contains\nline
breaks, \
      is sized to fit in 20 columns.",cols 20,cgtimes,12, \
      fit,spacing 1
```

```
text 28,60,"This sample message text is word wrapped to not
exceed \
      20 columns, while retaining the specified 12 point
size.",\
      cgtimes,cols 20,12,wrap,spacing 1
```

```
text 2,64,"This sample was generated by
UnForm<174>.",7,cgtimes, \
      symset 9J,blue
```

This set of commands places the phrase "Customer Copy" on copy 1, and "Remittance Copy" on copy 2. The text is placed at row 65.5, and is centered within the columns defined at column 1 and the constant MAXCOLS, which represents the whole page width.

```
# copy name sectionconst ROW=65.5if copy 1 text
1,ROW,"Customer Copy",HFONT,bold,center,cols=MAXCOLS
end if
if copy 2
    text 1,ROW,"Accounting
Copy",HFONT,bold,center,cols=MAXCOLS
end if
```

14.2.2 STATEMENT - 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 statements may contain a single page, others two or more pages.

The trick here is to get UnForm to produce two formats based on the content of each page. In order to accomplish this, we define the job to produce multiple copies, and assign certain copies to certain formats. Using a precopy{} code block, we can then control the printing of the different formats.

```
uf101c -i sample2.txt -f advanced.rul -p pdf -o client:statement.pdf
```

This statement header identifies this rule set.

```
[Statement]
```

The word STATEMENT appears at column 34, row 2, and a date appears at column 65, row 7. To further clarify, a date format is matched at position 65, 7.

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

The page dimensions are 66 rows and 75 columns. The text input to UnForm doesn't contain any form-feeds to indicate the end of a page, so the command "page 66" tells UnForm to consider each 66 lines to be a page.

Pcopies 4 is used to tell UnForm to print 4 copies of each page, with copies following each other in sequence for each page (collated). You will find later that UnForm doesn't actually print all copies of each page, but instead simply prints selected copies, depending on the format required. As each page is processed, if the page contains aging totals, UnForm prints 2 copies of that format, and if it does not contain aging totals, then UnForm prints 2 copies of the second format.

```
# set up document constants
const MAXCOLS=75 # max cols to output
const MAXRCOLS=74 # MAXCOLS-1
const LEFTCOL=1 # use 1 if empty
const RIGHTCOL=75 # MAXCOLS for symmetry
const MAXROWS=66 # max rows to output

portrait
dpi 300
gs on # graphical shading
cols MAXCOLS # max output columns
rows MAXROWS # max output rows
page MAXROWS # no form-feeds used

# to print more copies, increase value and add copy titles in
prejob
# Copy 1,2 Statement with aging totals
# Copy 3,4 Statement w/o aging totals
pcopies 4 # max # of copies
```

If this rule set is used to produce a PDF document, then the title "Statement Sample" will be added to the PDF file. For laser output, the title command is ignored.

```
title "Statement Sample"           # view in PDF
properties
```

The prejob command defines a string variable form_title\$, assigning it the value "STATEMENT". This variable is used later in the rule set for a page heading and also in a watermark.

```
prejob {
  # set up variables needed by merged routines below
  # if form title changes per page,
  # set up in prepage routine below
  form_title$="STATEMENT"
}
```

The prepage code block performs 2 functions. It checks the input data for the word "CONTINUED" at position 66, 64. If that word is present, then a variable continued\$ is assigned to the phrase "Continued"; otherwise it is set to null. In addition, at three individual lines (16, 62, and 64), there may be single ! characters used as character-mode vertical lines in the input data. Elsewhere in the rule set is a 'vline "!!", erase' command, which erases instances of 2 or more ! characters vertically on the page. This code takes care of the single-row instances.

```
prepage {
  # get continued if it exists
  continued$=get(66,64,9)
  if continued$="CONTINUED" then continued$="Continued" \
    else continued$=""

  # remove single ! from line draw regions
  x=pos("!="=text$[16]); \
  while x>0; text$[16](x,1)
  ="" ,x=pos("!="=text$[16]);wend

  x=pos("!="=text$[62]); \
  while x>0; text$[62](x,1)
  ="" ,x=pos("!="=text$[62]);wend

  x=pos("!="=text$[64]); \
  while x>0; text$[64](x,1)
  ="" ,x=pos("!="=text$[64]);wend
}
```

The precopy code block is executed as each of the 4 copies are about to be printed. The logic here indicates the copies 1 and 2 are for pages that do not contain the word "CONTINUED" (remember the prepage code block?), and copies 3 and 4 do contain that word. By setting the variable skip to a non-zero value, the copy being processed is skipped. Only 1 of the 2 formats is printed, depending on the content of the page.

```
precopy {
    if copy=1 or copy=2 then if continued$="Continued" then
skip=1
    if copy=3 or copy=4 then if continued$<>"Continued" then
skip=1
}
```

The following lines remove most of the existing character-mode line drawing elements from the input data. The hline and vline commands scan for places where at least the indicated number of characters, horizontally or vertically, occur on the page. The erase option removes them rather than replacing them with graphical lines.

```
#remove existing lines
hline "--",erase
hline "=",erase
vline "!!",erase
cerase 1,1,1,MAXROWS           # erase all 1st
column
cerase MAXCOLS,1,MAXCOLS,MAXROWS # erase all last
column
```

The following lines draw the page headings. Some of the commands are stored in another rule set, "Mrg Form Header", which is merged as the rule set is parsed. The headings already exist, and are moved and fonted with text commands using expressions, such as {cut(66,5,4,"")}.

```
# heading section
const HFONT=univers,12           # headings
cerase 1,1,MAXCOLS,10
cbox LEFTCOL,1,RIGHTCOL,MAXROWS,5 # complete page
box
merge "Mrg Form Header"         # merge std hdr
rules

# right top ribbon section
const HFONT=univers,11,italic     # headings
const DFONT=cgtimes,11,bold       # data
# draw info box with internal grid and shading
# horizontal line at 6
# vertical line at 68 with shading between 63 and 68
```

```

cbox 63,5,MAXCOLS,9,5,crows=7,ccols=68::20
text 64,6,{cut(66,5,4,"")},HFONT # page #
text 64,8,{cut(59,7,4,"")},HFONT # date
text 69,6,{trim(cut(71,5,3,""))},DFONT # page #
text 69,8,{trim(cut(65,7,8,""))},DFONT # date

# customer section
# draw info box with internal grid and shading
# vertical line at 10 with shading between 1 and 10
cbox LEFTCOL,10,MAXCOLS,15,5,ccols=10::10
text 2,11,{cut(2,10,2,"")},HFONT # to
text 4,13,{trim(cut(15,10,10,""))},DFONT # cust code
cfont 12,11,MAXCOLS,14,DFONT # address

```

The detail section contains several columns of information. There are fewer detail lines on pages with the aging data, so the grid drawing is made specific to particular formats with "if copy 1,2" and "if copy 3,4" sections. Then two groups of font changes are used, first for the column headings and then for the data columns.

```

# detail section
# detail headings
# draw info box with internal grid and shading
# horizontal line at 6
# vertical line at 68 with shading between 63 and 68
if copy 1,2
    cbox LEFTCOL,15,MAXCOLS,56,5,crows=17::20, \
        ccols=10 18 27 39 48 60 63
end if
if copy 3,4
    cbox LEFTCOL,15,MAXCOLS,61,5,crows=17::20, \
        ccols=10 18 27 39 48 60 63
end if
const ROW=16
cfont 2,ROW,9,ROW,HFONT,center # date
cfont 11,ROW,17,ROW,HFONT # inv #
cfont 19,ROW,26,ROW,HFONT,center # due date
cfont 28,ROW,38,ROW,HFONT,right # due amt
cfont 40,ROW,47,ROW,HFONT,center # pmt date
cfont 49,ROW,59,ROW,HFONT,right # pmt amt
cfont 61,ROW,62,ROW,HFONT,center # type
cfont 64,ROW,MAXRCOLS,ROW,HFONT,right # balance
# detail data
const DFONT=cgtimes,11 # data
cfont 2,18,9,60,DFONT,center # date
cfont 11,18,17,60,DFONT # inv #
cfont 19,18,26,60,DFONT,center # due date

```

```

cfont 28,18,38,60,DFONT,right           # due
amt
cfont 40,18,47,60,DFONT,center           # pmt date
cfont 49,18,59,60,DFONT,right           # pmt
amt
cfont 61,18,62,60,DFONT,center           # type
cfont 64,18,MAXRCOLS,60,DFONT,right,BOLD # balance

```

A watermark prints the form title as large, lightly shaded text. Its position depends upon the format, hence the use of if copy blocks.

```

# watermark - large font with light shading and rotation
if copy 1,2
    text 39,56,{form_title$},cgtimes,75,shade 20,center, \
        cols=MAXCOLS,rotate 90
end if
if copy 3,4
    text 44,61,{form_title$},cgtimes,85,shade 20,center, \
        cols=MAXCOLS,rotate 90
end if

```

The footer section differs considerably between the two formats. Copies 1 and 2 are associated with pages that have aging data, so you see the fonting of the aging columns defined there. Copies 3 and 4 are printed when the word "CONTINUED" appears, and that word is printed below, though as the value stored in continued\$ ("Continued").

```

# footer section
# remarks
if copy 1,2
    cbox LEFTCOL,56,RIGHTCOL,61,5
    cfont 2,57,MAXRCOLS,60,HFONT
endif
# totals
const DFONT=cgtimes,11,bold           # data
if copy 1,2
    cbox LEFTCOL,61,RIGHTCOL,64.5,5,crows=63::20, \
        CCOLS=14 26 38 50 62
    const ROW=62
    cfont 1,ROW,13,ROW,HFONT,right     #
current
    cfont 15,ROW,25,ROW,HFONT,right     # 1-15
    cfont 27,ROW,37,ROW,HFONT,right     # 16-30
    cfont 39,ROW,49,ROW,HFONT,right     # 31-45
    cfont 51,ROW,61,ROW,HFONT,right     # over
45

```

```

        cfont 63,ROW,MAXRCOLS,ROW,HFONT,right,bold,12      # total
due
    const ROW=64
    cfont 1,ROW,13,ROW,DFONT,right                          #
current
    cfont 15,ROW,25,ROW,DFONT,right                          # 1-15
    cfont 27,ROW,37,ROW,DFONT,right                          # 16-30
    cfont 39,ROW,49,ROW,DFONT,right                          # 31-45
    cfont 51,ROW,61,ROW,DFONT,right                          # over
45
    cfont 63,ROW,MAXRCOLS,ROW,DFONT,right,bold,12          # total
due
endif

if copy 3,4
    cerase 1,62,MAXCOLS,66
    text 1,65,{Continued$},HFONT,right,cols=MAXRCOLS
endif

```

Finally, within the two formats are two physical copies. Each of these copies is either for the customer to keep or for the customer to return with their payment. Copy 1, the first page of format 1, and copy 3, the first page of format 2, get the "Customer Copy" footer. The others get the "Remittance Copy" footer.

```

# copy name section
const ROW=65.5
if copy 1,3
    text 1,ROW,"Customer
Copy",HFONT,bold,center,cols=MAXCOLS
end if
if copy 2,4
    text 1,ROW,"Remittance
Copy",HFONT,bold,center,cols=MAXCOLS
end if

```

14.2.3 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.

```
uf101c -i sample3.txt -f advanced.rul -p pdf -o client:aging.pdf
```

This statement header identifies this rule set.

```
[AgingReport]
```

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

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

These constants are used throughout the rule set.

```
# set up document constants
const MAXCOLS=131                # max cols to output
const MAXRCOLS=130              # MAXCOLS-1
const LEFTCOL=.5                # use 1 if empty
const RIGHTCOL=131.5            # LEFTCOL for symmetry
const MAXROWS=66                # max rows to output
```

This report should print in landscape orientation, rather than the default portrait. UnForm will scale the columns and rows to 131 by 66.

```
landscape
dpi 1200
gs on                            # graphical shading
cols MAXCOLS                     # max output cols
rows MAXROWS                     # max output rows

pcopies 1                        # max # of copies
```

The title "Aging Sample" will appear in PDF document properties. It is ignored for laser output.

```
title "Aging Sample"            # view in PDF
properties
```

The following prejob code demonstrates the use of sdOfficeÔ to mine data from this report and export it to Microsoft ExcelÔ. SdOffice can be running anywhere on your network on a system with Excel. The code relies on your setting two variables correctly. First, the sdo\$ variable should be set to the path to the sdOffice client program sdojc_e.bb. In addition, the value of gbl("\$sdhost") needs to be set to the address or hostname of the system running sdOffice. An optional way of doing this is to define an environment variable prior to running UnForm, called SDHOST. If you use that alternative, then comment out the x\$=gbl("\$sdhost") line.

The code here contains enough error handling to ignore the code if sdOffice isn't present or fails to execute.

```

prejob {
    # set up sdOffice export to Excel
    # set to path to your sdoffice *.pv programs
    sdo$="/u0/sdofc/sdofc_e.pv"

    # You can set the environment variable SDHOST, or use
this
    # stbl function to define the sdOffice server address
    x$=gbl("$sdhost","bcj")

    # initialize excel
    call sdo$,err=prejob_done,"newbook","",errmsg$
    if errmsg$>"" then goto prejob_done
    sdofc_init=1
    call sdo$,"show","",""
    call sdo$,"setdelim |","",""
    call sdo$,"writerow ID|Name|Phone|Over 60|Total","",""
    call sdo$,"format row=1,font=Arial,size=12,bold","",""
prejob_done:
}

```

The prepage code block starts with code that exports data to Excel, but only if the prejob code block successfully initializes the sdOffice connection. In addition to that code, it also sets two numeric variables, colw and scol, based upon positions and widths of report column headers. These values are used later in the rule set for fonting and line drawing.

```

prepage{
    # if prejob hasn't initialized sdoffice, skip this code
    if sdofc_init<>1 then goto sdofc_complete

    for row=1 to 66
        ln$=text$[row]

        # customer heading row contain phone numbers
        x=mask(ln$,"\\( ...-...-....\\)")
        while x
            custid$=mid(ln$,1,6)
            custname$=trim(mid(ln$,8,30))
            custphone$=trim(mid(ln$,38,14))
            x=0
        wend

        # totals - 50 plus spaces followed by digit-.-
digit-digit
        x=mask(ln$,"^"+fill(50)+".*[0-9]\\.[0-9][0-9]")
        while x
            amount60=cnum(mid(ln$,87,11))

```



```

        amount90=cnum(mid(ln$,98,11))
        amount120=cnum(mid(ln$,109,11))
        over60=amount60+amount90+amount120
        total=cnum(mid(ln$,120,11))

export$=custid$+"|"+custname$+"|"+custphone$+"|"
        export$=export$+str(over60)+"|"+str(total)
        call sdo$,"writerow "+export$,"",""
        x=0
    wend

    next row
sdofc_complete:

    # Now for some tricky code.
    # Agings can have different headings and column widths
    # To use version 5 features allowing variable columns
and rows,
    # the following code will calculate starting positions
    # and column widths. It assumes a consistency in column
widths,
    # 1 char negative, and 1 blank space between each column
    hdl$=text$[7]                # temp heading line with
agings
    x=pos("Type"=hdl$)
    xhdl$=trim(hdl$(x+4))        # remove all except agings
    x=pos(" "=xhdl$)
    x$=xhdl$(1,x-1)             # get first column header
    xhdl$=trim(xhdl$(x))
    x=pos(x$=hdl$)              # find true position
    x1=x+len(x$)-1              # get end of first column
    # now find end of 2nd column
    x=pos(" "=xhdl$)
    x$=xhdl$(1,x-1)             # get second column header
    x=pos(x$=hdl$)
    x2=x+len(x$)-1              # get end of second column
    # now calculate mask width less space between columns
    colw=x2-x1-1
    # now calculate start of first field
    scol=x1-colw+2
}

```

The postjob code block performs some closing formatting control if the job is exporting data to Excel. If sdOffice is not being used, based upon the attempt to initialize it in the prejob code block, then this code is skipped.

```

postjob{
    # if prejob hasn't initialized sdoffice, skip this code
    if sdofc_init<>1 then goto sdofc_complete2

    call sdo$,"leaveopen","", ""
    call sdo$,"format autofit","", ""
    call sdo$,"format col=1,numberformat=@","", ""
    call sdo$,"format
col=4,numberformat=""###,##0.00""", "", ""
    call sdo$,"format
col=5,numberformat=""###,##0.00""", bold", "", ""

    call sdo$,"insertrow 1","", ""
    call sdo$,"mergecells range=A1:E1","", ""
    call sdo$,"writecell range=A1,value="+$22$+ \
        "Over 60 Aging Values as of "+date(0)+$22$","", ""
    call sdo$,"format range=A1:E1,center,size=15,bold","", ""
sdofc_complete2:
}

```

Here, finally, are the commands to enhance the formatting of the report. The initial commands use text commands with cut expressions to move the report header data around and change the fonting.

```

# heading section
const BLFONT=univers,10,bold,italic
const BSFONT=univers,9,bold,italic
cbox .5,.5,RIGHTCOL,5,5,30
# line 1
text 2,1.25,{trim(cut(1,1,10,""))},BSFONT #
date
text 1,1.25,{trim(cut(20,1,100,""))},BLFONT,center, \
    cols=MAXRCOLS # comp name
text 1,1.25,{trim(cut(121,1,15,""))},BSFONT,right, \
    cols=MAXRCOLS # page #
# line 2
text 2,2.35,{trim(cut(1,2,10,""))},BSFONT #
time
text 1,2.35,{trim(cut(20,2,100,""))},BLFONT,center, \
    cols=MAXRCOLS # rpt title
# line 3
text 1,3.45,{trim(cut(20,3,100,""))},BSFONT,center, \
    cols=MAXRCOLS # sub heading
# line 4
text 1,4.45,{trim(cut(20,4,100,""))},BSFONT,center, \
    cols=MAXRCOLS # sub heading

```

This section formats the column headings. The left portion is drawn with text commands, while the aging columns are fonted with font commands, which use the positions from the values calculated in the prepage code block.

```
# detail heading section
const HFONT=univers,10,italic
cbox LEFTCOL,5.25,RIGHTCOL,7.5,1,20
# line 1
cerase 1,6,MAXCOLS,6
text 1,6,"Customer # & Name",HFONT
text 38,6,"Phone #",HFONT,center,cols=14
text 54,6,"Contact",HFONT

# line 2
cerase 1,7,49,7
text 3,7,"Invoice #",HFONT
text 12,7,"Due Date",HFONT,center,cols=8
text 21,7,"P/O #",HFONT
text 32,7,"Order #",HFONT
text 39,7,"Terms",HFONT,center,cols=5
text 45,7,"Type",HFONT,center,cols=4
# using variables from prepage, enhance aging headings

font {scol},7,{colw-1},1,HFONT,right
font {scol+1*(colw+1)},7,{colw-1},1,HFONT,right
font {scol+2*(colw+1)},7,{colw-1},1,HFONT,right
font {scol+3*(colw+1)},7,{colw-1},1,HFONT,right
font {scol+4*(colw+1)},7,{colw-1},1,HFONT,right
font {scol+5*(colw+1)},7,{colw-1},1,HFONT,right
font {scol+6*(colw+1)},7,{colw},1,HFONT,right,bold
```

The report body is enhanced using UnForm's ability to scan for patterns and anchor enhancements to those patterns. The first series of font commands scan for two spaces in the region from column 1, row 9 through column 2, row 66 (defined as the constant MAXROWS above). At each point in that search region, if the two spaces are found, a font command is issued relative to the location. This changes the font of the input data at that location.

The second series of font commands looks for customer heading line types, by searching for any alpha or digit character in the region 1,9 though 2,66. A different set of font commands is then issued for those positions.

```
# detail data section
const BDFONT=cgtimes,10,bold
const DFONT=cgtimes,10
# invoice line
font " @1,9,2,MAXROWS",2,0,8,1,DFONT
font " @1,9,2,MAXROWS",11,0,8,1,DFONT,center
```

```

font " @1,9,2,MAXROWS",20,0,10,1,DFONT
font " @1,9,2,MAXROWS",31,0,7,1,DFONT
font " @1,9,2,MAXROWS",38,0,5,1,DFONT,center
font " @1,9,2,MAXROWS",44,0,4,1,DFONT,center
# using variables from prepage, enhance agings
font " @1,9,2,MAXROWS",{scol},0,{colw},1,DFONT,decimal
font " @1,9,2,MAXROWS",{scol+1*(colw+1)},0,
{colw},1,DFONT,decimal
font " @1,9,2,MAXROWS",{scol+2*(colw+1)},0,
{colw},1,DFONT,decimal
font " @1,9,2,MAXROWS",{scol+3*(colw+1)},0,
{colw},1,DFONT,decimal
font " @1,9,2,MAXROWS",{scol+4*(colw+1)},0,
{colw},1,DFONT,decimal
font " @1,9,2,MAXROWS",{scol+5*(colw+1)},0,
{colw},1,DFONT,decimal
font " @1,9,2,MAXROWS",{scol+6*(colw+1)},0,
{colw+1},1,BDFONT,decimal

# cust line
font "~[A-Z0-9]@1,9,2,MAXROWS",0,0,6,1,BDFONT
font "~[A-Z0-9]@1,9,2,MAXROWS",7,0,28,1,BDFONT
font "~[A-Z0-9]@1,9,2,MAXROWS",37,0,14,1,BDFONT,center
font "~[A-Z0-9]@1,9,2,MAXROWS",53,0,36,1,BDFONT
shade "~[A-Z0-9]@1,9,2,MAXROWS",0,-.15,{RIGHTCOL-1.5},1,20

```

The following commands look for sequences of dashes, which indicate sub total lines. Wherever a sequence of 50 dashes occurs, a box is drawn and input data is bolded. In addition, the original dashes are removed with the hline command.

```

# customer totals
hline "---",erase
# example of UnForm command with continuation to next line
box "-----", \
    -1,.25,{RIGHTCOL-53},1.25
bold
"-----",0,1,120,
1

```

Finally, grand total lines are treated with special fonting and a box.

```

# grand totals
const DFONT=cgtimes,11,bold
# sample of box command with increased thickness and double
lines
box "Grand Total:",-9.5,-1.25,MAXRCOLS,2.25,5,30,dbl 9

```

```
font "Grand Total:",0,0,12,1,BDFONT,13
font "Grand Total:",{scol-10},0,{colw},1,DFONT,decimal
font "Grand Total:",{scol-10+1*(colw+1)},0,
{colw},1,DFONT,decimal
font "Grand Total:",{scol-10+2*(colw+1)},0,
{colw},1,DFONT,decimal
font "Grand Total:",{scol-10+3*(colw+1)},0,
{colw},1,DFONT,decimal
font "Grand Total:",{scol-10+4*(colw+1)},0,
{colw},1,DFONT,decimal
font "Grand Total:",{scol-10+5*(colw+1)},0,
{colw},1,DFONT,decimal
font "Grand Total:",{scol-10+6*(colw+1)},0,
{colw+1},1,DFONT,decimal
```

14.2.4 LABELS – text labels to laser labels

UnForm is capable reading rows of input, parsing those rows into logical pages, and reproducing the output with different dimensions. A typical situation that can take advantage of this is if your application is designed to print mailing labels on continuous label stock on dot matrix printers. The labels can be 1-up, 2-up, or any other dimensions. As long as each label has a consistent number of rows and columns, UnForm can parse each label and treat each label as a logical page with the across and down commands. To use this sample, you must add "-r labels" to the command line.

```
uf101c -i sample4.txt -f advanced.rul -r labels -p pdf -o client:labels.pdf
```

This statement header identifies the rule set. The name is used in the -r command line option.

```
[labels]
```

Each label "page" is 35 columns and 6 rows of input text. If each line is 106 to 140 characters wide, then four labels are parsed from the columns. When the output is produced, each label will be 30 columns by 6 rows. The labels will be arranged 3 rows across and 10 down the page. UnForm will actually print 3x30=90 columns and 10x6=60 rows on each physical page.

Most laser label stock has ½ inch top and bottom margins. The margin command adds 75 dots (¼ inch) to the standard UnForm top and bottom margins, which default to ¼ inch.

In this sample, the text of the labels is printed from lines 1 to 4. By using the vshift 1 command, UnForm will move the text to lines 2 through 5. The shift command moves the text to the right.

```
page 35,6
rows 6
cols 30
across 3
down 10
font 1,1,40,6,cgtimes,12
margin 0,0,75,75
vshift 1
```

```

shift 2
# manual feed tray is usually 2
# tray 2

```

The barcode command supports both 5 and 9-digit formats of the postnet barcode. To get either to print, the prepage code block sets one or the other variable (zip\$ or zip9\$), and both commands are issued. A null value is not barcoded. The prepage code extracts the zip code from line 3 or 4 of the label. It then determines the length and sets zip\$ or zip9\$ appropriately.

```

barcode 2,6,{zip$},900,11.0,2
barcode 2,6,{zip9$},905,11.0,2

prepage{
# get zip code from line 3 or 4
zip$="",zip9$="",zipline$=""
if trim(text$[4])>" " then zipline$=trim(text$[4])
if zipline$="" then if trim(text$[3])>" " then
zipline$=trim(text$[3])
while zipline$>" "
    x=mask(zipline$,"[0-9][0-9][0-9][0-9][0-9]")
    if x>0 zip$=zipline$(x)
    zipline$=""
wend
# remove possible hyphen and validate length
x=pos("-"=zip$); if x=6 then zip$=zip$(1,5)+zip$(7)
if len(zip$)<>5 and len(zip$)<>9 then zip$=""
if len(zip$)=9 then zip9$=zip$,zip$=""
}

```

14.2.5 132x4 – multi-up and 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.

```
uf101c -i sample3.txt -f advanced.rul -r 132x4 -p pdf -o client:132x4.pdf
```

The rule set header identifies the name.

```
[132x4]
```

The page dimensions are defined as 132 columns by 66 rows. UnForm will scale each page to fit 2 across and 2 down on a physical page (264 columns and 132 rows). The report is printed in landscape orientation. A box is drawn around each page, and the hline command will convert all occurrences of 3 or more dashes to horizontal lines.

```

cols 132
rows 66

```

```

across 2
down 2
landscape
cbox .5,.5,132.5,66.5
hline "---"

```

14.2.6 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 it does for laser printers. Some key differences are: fonts are identified differently and are limited in scalability, shading is either 100% (black) or 0% (white), and the **barcode** command is more extensive and capable than the laser printer **barcode** command.

When executing a Zebra run, it is critical to tell UnForm how large the labels are. This is done with a special syntax on the "-paper" command line option. Also, UnForm needs to know what print density is used by the printer. This is determined by the "-p zebran" option, where *n* is either 6, 8, or 12 dots per millimeter. You may need to adjust this sample command line to match your Zebra printer, as it assumes an 8 dpmm printer and 3.25 by 5.5 inch label stock.

```
uf101c -i samplez.txt -f advanced.rul -p zebra8 -paper 3.25x5.5 -o zebra-device
```

This label is scaled to 40 columns and 35 rows.

```

[zebra label]
detect 0,1,"Zebra Barcode"
cols 40
rows 35

```

The prepage code block gets the PO number, setting it into a variable po\$, and removing the PO number from the text with a set() function.

```

prepage{
po$=""
po$=cvs(get(2,16,10),3)
trash$=set(2,16,10,"")
}

```

The From and To sections draw boxes, change fonts, and re-allocate the lines of text from row 10 to row 14 with a series of text commands followed by an erase command.

```

# From section
box 1,1,39,8,3
text 2,2,"From:",font A
font 2,3,35,6,font 0,9

# To section
box 1,9.75,39,10.5,5

```

```
#text 2,10.6,"To:",font 0
text 3,11,{get(2,11,30)},font 0,12
text 3,12.25,{get(2,12,30)},font 0,12
text 3,13.5,{get(2,13,30)},font 0,12
text 3,14.75,{get(2,14,30)},font 0,12
text 3,16,{get(2,15,10)},font 0,12
erase 2,11,30,5
```

This group of commands prints three different barcodes on the label. First, a postnet code is printed from the zip code located at column 2, row 15, for up to 10 characters. Then a UPS maxicode barcode is printed with SDSI's address. Last, a "3 of 9" barcode is printed using the variable po\$, derived in the prepage{} code block above.

```
# bar codes
barcode 10,18.25,{trim(get(2,15,10))},Z,33

text 2,24,"Maxicode",font 0,10
barcode 2,25,{"999840956820000" + $0a$ + "SDSI" + $0a$ + "2195
Talon Drive" + $0a$ + "Latrobe, CA 95682"},D

box 17,25,22,12,3
text 18,25.75,"Our PO No (in code 39):",font A,21
barcode 20,28,{po$},3,120,2,text above
```

14.2.7 OUTLINE - PDF outline

UnForm supports PDF outlines (or bookmarks) when using the pdf driver. Outlines can be multiple levels, and each outline tree can be different levels deep. UnForm assumes each outline branch points to a page. To control the text shown in the outline, you set the variable `outline$` in a prepage or precopy code block. This variable is parsed as each page is printed. Multi-level entries are created by delimiting the text of the levels with a vertical bar (|) within the contents of the variable.

The file `sample5.txt` contains the contents of a 14-page report featuring two sort and subtotal levels, as well as grand totals and a recap page. The outline tree for this report will be based on the salesperson (outer sort) and class code (inner sort), along with specific page entries for the report total and recap page. As there are no detect statements, you need to specify the `-r` option on the command line, as shown.

```
uf101c -i sample5.txt -f advanced.rul -r outline -p pdf -o client:outline.pdf
```

```
[outline]
```

Set the page dimensions and turn on the outline feature with the outline keyword. The default outline title for each page is simply "Page n", but a code block can override the outline text by setting the variable `outline$`.

```
cols 132
rows 66
outline
```


The prepage code block looks on each page for the following cases, in order:

- A 3-digit salesperson number at the first column on line 7
- A salesperson subtotal heading on line 8
- A report total heading on line 8
- A recap page heading on line 2

For the first two types of pages, a two level outline entry is created (level 1|level 2 structure). For the report total and recap pages, a single level outline entry is created.

```
prepage{
# default outline setting matches prior page
outline$=lastoutline$

# if line 7 starts with 3 digits, set 2-level outline
slsp+class
if mask(get(1,7,3),"[0-9][0-9][0-9]") then \
  outline$="Slsp "+get(1,7,3)+"|Class "+get(13,7,2)

# if line 8 contains this, it is a salesperson subtotal
if pos("SALESPERSON: "=text$[8])>0 then \
  outline$="Slsp "+get(14,8,3)+"|Totals"

# if line 8 contains this, it is a report title
if pos("*Report "=text$[8])>0 then \
  outline$="Report Total"

# if line 2 contains this, it is the recap page
if pos("RECAP PAGE "=text$[2])>0 then \
  outline$="Recap Page"

lastoutline$=outline$

}
```

14.3 Additional Sample Rule Files

The following table describes several other sample rule files that have been designed to demonstrate specific techniques. There are two types of rule files provided:

- Cmd – adds to the documentation for specific commands
- Tool – helps an integrator to add new features to their own rule files

Rule file name	Commands	Functions	Variables	Comments
SampleCmdAFO.rul		gtextcount, gtextitem, gtextfind		How to work with Application Formatted Output

SampleCmdBarcode.rul	barcode	exec		Includes relative barcode
SampleCmdBox.rul	box, cbox	exec		
SampleCmdCircle.rul	circle	inchtocols		
SampleCmdConst.rul	const, global, local			
SampleCmdCopy.rul	copies, pcopies, if copy/end if, attach		Copy	Multi-copy, multi-format, variable number of copies
SampleCmdDetect.rul	detect			Many examples with explanations
SampleCmdDuplex.rul	duplex			
SampleCmdErase.rul	erase, notext, hline, vline	sub		
SampleCmdFont.rul	font, cfont			
SampleCmdImage.rul	image, attach			image justification
SampleCmdImages.rul	images			1 sample for archived images
SampleCmdLine.rul	line, hline, vline	exec		nice use of multiple cmds within 1 exec. Saves time.
SampleCmdMicr.rul	micr			
SampleCmdMove.rul	move, cmove			Includes relative moves
SampleCmdShade.rul	shade, cshade	exec		Includes relative shading and greenbar look
SampleCmdText.rul	text			Shaded, wrapped, fit, rotated, justified
SampleCmdTrayBin.rul	tray, bin			

Rule file name	Commands	Functions	Variables	Comments
----------------	----------	-----------	-----------	----------

SampleToolArchive.rul	archive			will do logging with Base Logging below
SampleToolChkContinued.rul	text	exec, get	pagenum, uf.maxpage	look ahead with get
SampleToolDispAFOText.rul	text			AFO field positions
SampleToolDocTitle.rul	text			
SampleToolFullBox.rul	cbox			
SampleToolLogging.rul		log		Base Logging
SampleToolLogo.rul	image			can use justification
SampleToolMultCpy.rul	copies, pcopies			
SampleToolPageSplitter.rul		getpage, putpage, delpage		
SampleToolPageXofY.rul	text	exec, get	pagenum, uf.maxpage	look ahead with get
SampleToolPostnet.rul	barcode	exec, mid		
SampleToolPrint.rul			skip	logging with Base Logging above
SampleToolScanBcd.rul	barcode	exec		barcode
SampleToolWatermark.rul	text	exec		shaded text

15 PROGRAMMING CODE BLOCKS AND JOBS

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 to interact with other applications and devices, or with the operating system.

The Image Manager also supports custom coding using the same dialect. This code can be added to custom job scripting, filters, lookups, and validations.

An experienced Business BASIC programmer (PxPlus is the actual dialect used) 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 web site: www.pvxplus.com. In this manual, we have provided some basic (no pun intended) information that will assist developers experienced in other programming environments.

It should be noted that an UnForm job is not a stand-alone pxplus program. Instead, it is a combination of a an UnForm job wrapper written in static pxplus code, and user-defined code blocks, which are executed at particular times as subroutines while the job progresses. Therefore, code blocks act as subroutines and not full programs. In addition, some pxplus syntax has been overridden to ensure compatibility with previous versions of UnForm, and many UnForm-specific functions have been added so that code blocks can perform UnForm-specific tasks. Lastly, the user-interface features of px are not available, as UnForm jobs run in background and are not connected to a user screen.

15.1 Basic Syntax

Basic Syntax

Statements

A statement consists of a single verb and any arguments or parameters suitable for that verb. Multiple statements can be placed on a single line by separating them with a semicolon (;). Statements can be preceded by a label, which consists of a label name followed by a colon. Label names must follow the same naming conventions as numeric variables.

Variables

There are two types of variables in Business Basic: string and numeric. Variables that end in a "\$" character are treated as string variables. They can hold any amount of text data, limited only by system memory. Numeric variables can contain any number or integer. UnForm sets precision to 10, so that up to 10 digits to the right of the decimal are maintained accurately.

Variable names can be up to 127 letters, digits, and underscore characters, and must start with a letter. Variables can't start with "fn" and should not start with "uf".

work\$, account01\$, and cust_name\$ are valid string variables.
cust-name\$ is invalid.
amount, period_12, and six are valid numeric variables.

Arrays

Arrays can be defined for both string and numeric variables. Arrays must be defined to a fixed number of elements with a DIM statement, and array elements can then be referenced as variables. Arrays can contain up to three dimensions.

dim amount[12] defines a 13-element array, a[0] ... a[12].
dim x\$[1:6,1:20] defines a 2-dimensional string array. The first dimension ranges from 1 to 6, the second from 1 to 20. x\$[2,20] would be a valid element in this array.

Arrays can be re-dimensioned with the REDIM statement. When an array is re-dimmed, all its existing elements are retained as long as they are part of the new array dimensions.

You can get the first and last indexes of an array with a DIM() function. These examples assume a string array named array\$.

- dim(read min(array\$)) is the lowest index, typically 0

- `dim(read max(array$))` is the highest index
- `dim(read max(array$,2))` is the highest index of the second dimension of the array

Associative Arrays

The language also supports arrays that use string keys rather than indexes. Both text and numeric values can be stored this way. There is no DIM statement required to define such an array. Simply start using the array syntax with string keys, such as:

- `states$["CA"]="California"`
- `state$="CA"`
- `desc$="My state is "+states$[state$]`

If an array key is used that doesn't exist, it is treated as null or 0. An associated array can be cleared with a dim statement and the base variable, such as `"dim states$"`. Beware of using a reference to an array element. This will create an element if it doesn't exist, which is likely not the intention:

```
if states$[state$]>"" then found=1
```

If `state$` is null or otherwise doesn't exist in the array, a null array entry will be created for it.

In addition to a key, associative array elements have a 1-based index. You can get the number of elements in the array with: `dim(read num(array$))`. For example:

```
first=1,last=dim(read num(a$))
for i=first to last
  b$+=a$[i]+$0a$
next i
```

You can find the array index of a given key. This statement will set position to the index number of the array element "mykey". If that key isn't found, position will be set to 0. This is useful for determining if a key exists in an array.

```
position=dim(index a$["mykey"])
```

Likewise, you can find the key of a given index (note this will produce an error 42 if the index value is out of range):

```
keyval$=dim(key a$[2])
```

You can remove a key from an array:

```
position_dropped=dim(drop a$["mykey"])
```

Other Uses of DIM

- The dim statement can be used to initialize strings to a specified length. `Dim a$(12)`, for example, will set `a$` to 12 spaces.
- The dim statement can be used to initialize a string template. `Dim state$:"id:c(2*),name:c(20*),taxrate:n(4*)"` creates a string template with variables `state.id$`, `state.name$`, and `state.taxrate`. The above template describes `state.id$` as a variable-length character field, with a suggested length of 2, a second character field, `state.name$`, and a numeric field, `state.taxrate`.

There are additional capabilities in string templates, but those are beyond what is generally used in UnForm coding. Details about those additional features can be found in the pxplus documentation at pxplus.com.

Functions

Many functions are available in Business Basic. Most will be familiar to a Basic programmer. Functions consist of a word, an opening parenthesis, one or more arguments, and a closing parenthesis. The function returns a string or numeric result, which is typically used as part of an expression, or in an assignment. Wherever a string or numeric value can be used, a string or numeric function can be used. In addition to internal Business Basic functions, UnForm also provides some functions that perform tasks typical to print stream environment in which it runs.

String and numeric representation

Strings are made up of concatenated bytes. They can be represented as literals inside double quotes, such as "Name:", or as hexadecimal strings inside "\$" delimiters, such as \$1B45\$ for Escape-E. They can also be made up of combinations of literals, hex strings, string variables, and functions that return string values. These values are combined using the "+" operator to concatenate each string together. For example, a string containing quotes could be constructed one of these ways: chr(34)+"some text"+chr(34); or \$22\$+"some text"+\$22\$, or quote\$+"some text"+quote\$. Since chr(34) and \$22\$ both represent a quote character, and it would be possible for the variable quote\$ to contain the same, all these expressions can represent the same string.

Substrings can be derived from a string variable with the syntax *stringvar(start [,length])*. For example, if account\$ is "01-567", then account\$(4,3) would return the value "567". Substrings references with positions that aren't in the string result in errors, so care must be used. To avoid the possible errors, the mid() function can be used.

Numbers can be represented as integers or decimal numbers, or, like strings, can be represented as expressions containing literal numbers, numeric variables, and numeric functions. With numbers, there are more operators available to produce the expressions. A literal number is just a series of digits, with an optional decimal point and an optional leading minus sign. 1995.99 and -100.433 are valid numbers. Other punctuation, such as thousands separators or currency symbols, are invalid in a number though they can be added when a number is formatted as a string for output.

Operators

Business Basic has the following standard operators:

+	concatenate strings or add numbers, depending on context
-	subtraction
*	multiplication
/	division
^	exponentiation
=	testing for equality, or assignment, depending on context
>	testing for greater than
>=	testing for greater than or equal to
<	testing for less than
<=	testing for less than or equal to
<>	testing for inequality
()	controlling precedence
and	combining expressions with logical "and" in conditions
or	combining expressions with "or" in conditions

+= appends right-side value to a string or adds to a number
 -= subtracts right-side value from a number
 *= multiplies a number by right-side number
 /= divides a number by right-side number
 ++*var*, *var*++, --*var*, *var*-- increments or decrements a numeric variable by 1, either before or after an operation, depending on the position of the ++ or -- operators.

If Then Else

The structure of IF...THEN...ELSE statements are simple and unblocked. The IF must be followed by an expression to test. The expression can be simple or complex, and must resolve to a single Boolean or numeric result. For numeric results, a 0 is considered false, and anything else is considered true. Once resolved, if true the THEN clause is executed, otherwise the ELSE clause, if present, is executed.

Both the THEN clause and the ELSE clause can contain any statements, including nested IF statements. A closing END_IF after a THEN or ELSE clause will terminate the conditional nature of statements following it.

Here are some examples of IF statements:

```

if amount < 0 then text$="Credit Balance"
if x$="A" then desc$="Acme Rental" else if x$="S" then desc$="Smith & Sons" else desc$="N/A"
if testmode then dummy$=set(1,1,10,"Test Mode") end_if; goto exitsub
  
```

UnForm's code block parser also supports blocked if-then-else syntax, like this (optional elements indicated in square brackets):

```

  If condition [then][:]
      Statement1
      Statement2
  [else
      Statement1
      Statement2
      ... ]
  End if
  
```

The key elements are the condition "if *condition*" line and the closing "end if". The structures can be nested, with additional if statements inside the if or else sections. The trailing colon on the IF line is optional.

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 BREAK verb, the EXITTO label verb, or set variables such that the condition is false before executing the wend verb. To iterate the loop from within, use the CONTINUE 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.

The most common FOR statement identifies a variable, a start value, an end value, and an optional step value. The variable is set to the start value; the loop statements are executed until a NEXT statement is encountered; the variable is incremented by the step value; and, until the end value is exceeded, the loop statements are repeated. To exit the loop before the end value is reached, use the BREAK verb or the EXITTO label verb. To iterate the loop from within, use the CONTINUE 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
```

Another construct parses a string into delimited segments and runs the loop code for each segment. The delimiter is always the last character of the string. For example, many UnForm values are linefeed delimited lists. A linefeed can be represented as hexadecimal \$0a\$, so you could use a loop like this:

```
result$=""
for item$ from items$+$0a$
  if item$="" continue
  if mid(item$,1,1)="" result$+=item$+$0a$
next
```

A final construct loops over the indexes of an array, either numeric for standard arrays or string for associative arrays. This code would convert an associative array into a tab-separated-values string.

```
result$=""
for k$ index states$[all]
  result$+=k$+$09$+states$[k$]+$0a$
next
```

Switch/End Switch

The switch statement provides a concise logical testing capability, replacing potentially a large set of 'if' statements. The switch statement sets a test value, either a variable or expression. Inside the body of the switch are case statements, which can name one or more comma-separated values to compare with the test value. If there is a match, the code below executes until there is a break statement. There can be any number of case statements to test any number of possible conditions. There can also be a default statement, which matches any test value, and can be thought of as an 'else' statement. The switch can test either string or numeric values, though the internal case statements must match the switch value type.

```
switch lcs(state$)
case "az"
  name$="Arizona"
  break

case "ca"
  name$="California"; break

default
```



```
name$="unknown"
break
end switch
```

File Handling

Business Basic has very powerful facilities for handling files. Not only are there intrinsic keyed file types, but also text files and pipes can be used.

If the application with which UnForm is integrated is written in ProvideX, then full native access to the data files is available.

If UnForm is working with a non-Business Basic application (e.g. C, Cobol, Informix, Oracle, etc.), there are additional means to obtain data, via ODBC, pipes, or command lines. There are high level functions to work with external data sources, such as `sqlconnect()` and `sqlexec()`, and objects, such as `http`, `textfile`, `infile`, or `binfile`.

Below are comments about directly working with files.

Opening Files

File access is performed through an open file channel. The OPEN statement opens the file on a numeric channel in preparation for later file access. `Open(99)"customers.dat"` opens the named file on channel 99.

Channel numbers can range from 1 to 32767, though the operating system will typically impose a limit on the number of simultaneous channels that can be opened. Channel numbers must be unique. Once opened, that channel number is no longer available until closed. To avoid conflicts with channel numbers, it is common to use a special function that returns an available channel number, UNT. Here is a typical syntax:

```
cust=unt
open(cust)"customers.dat"
```

After that, file access verbs can use the cust variable to access the "customers.dat" file.

To open a pipe channel, you could do the following:

```
faxlist=unt
open(faxlist)"|sqlexec 'select cust,faxnum from customers'"
read(faxlist)line1$
```

```
labelprt=unt
open(labelprt)">lp -dlabels"
print(labelprt)"To: "+name$
print(labelprt)"  "+address1$
```

Reading Files

There are two verbs used for reading channels: READ, and READ RECORD. The READ verb understands line and field separators, whereas the READ RECORD verb reads blocks of a specified size or whole records, in the case of intrinsic keyed file types. The READ verbs accept several options, including "key=string", "ind=index", "err=linelabel", "end=linelabel", and others. Full details can be found in the language reference manuals. Labels can be actual labels in code (*label:*), or a symbolic label, such as *next, *break, or *continue. UnForm also recognizes err=next as a synonym for err=*next.

To read from an intrinsic keyed file (ProvideX files only), you might use one of these:

```
read(cust,key=custkey$,err=next)*,name$,*,*,*,faxnum$
```

```
read record(cust,key=custkey$,err=next)custrec$
name$=custrec$(7,30),faxnum$=custrec$(112,10)
```

To read from a pipe or a text file, you may not use a key= clause, so you just read sequentially through the file:

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

Writing files

You probably would not want to write to your application files, but you may well want to write to external devices or log files. Writing is performed with these verbs: WRITE or WRITE RECORD and PRINT. Each uses a channel number and arguments to print. PRINT terminate its values with a line-feed character (\$OA\$), 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. WRITE terminates values with an internal field separator, normally \$8A\$, which is not useful when writing files that will be used by other applications, but which is recognized by READ.

```
print (logfile)"Customer: "+custname$+" printed on "+date(0,tim:"%D-%M-%Y:%Hz:%mz")
dim block$(128); block$(1)=custname$,block$(31)=str(amount:"000000.00"); write record(log)block$
```

15.2 Object Oriented Programming

Object Oriented Programming

UnForm supports programming with objects, which are self-contained programming units (called objects) that have data elements (known as properties) and functions (known as methods). Object oriented programming (commonly referred to as OOPS) is a modern technique that has become popular with the rise of languages such as Java, C#, C++, and VB.NET. The definition of an object, that is, its properties and methods, is encapsulated within a program unit called a "class". The terms "class" and "object" are sometimes used interchangeably, but there is a distinction: a class is a description or definition, and an object is a physical, programmable representation of a class.

There are many built-in objects supplied with UnForm, which can be created (or instantiated) and used within code blocks as needed. Additional custom objects can be created as a professional service.

15.2.1 Object Instantiation

Code blocks can create new objects based on a class with the new() function. Each object is a simple numeric variable. The new() function returns an object ID number, and all action on that object is processed through the numeric variable.

There can be any number of objects in memory at a given time. Each object that is created is unique, with its own data.

The syntax of the new() function is:

```
objvar=new("classname"[,arg1[$],...][,err=label]*next)
```

When this function is executed in a code block, a new object is created and assigned to *objvar*. Some objects accept arguments during this initialization step, and the arguments are provided in a comma-separated list after the class name.

If an error occurs during the object creation, the object is not created, and the *err=label* is executed.

Once instantiated, the following read-only properties always exist:

- *objvar'*_obj holds the object number (the value of *objvar*)
- *objvar'*_class\$ holds the class name of the object. This can be useful if one object variable can be used for different types of objects.

15.2.2 Object Access

Once the object is created, *objvar* can be used to reference that object's properties and functions, using the apostrophe (') operator (-> is a synonym for '):

```
objvar'property[$]=var[$] | number | "string" | expression  
var[$]=objvar->property[$]
```

```
objvar->function(arg1[$][,...])  
var[$]=objvar->function(arg1[$][,...])
```

Functions can be accessed as functions, which return a value, or as subroutines, which simply execute the function code.

15.2.3 Object Destruction

To destroy an object, use the 'drop object' statement. Since the object variable no longer references an object, it can be set to 0, reused, or ignored. Once an object is no longer needed, it is good practice to destroy it. For example, beware of objects created repeatedly in loops and not destroyed.

```
drop object objvar
```

15.3 Built In Objects

Built In Objects

The following objects are provided with UnForm and can be instantiated with the new() function. Property and method references follow.

- *addrbook* - address book management
- *binfile* - binary file access
- *collection* - collections of values by index and name
- *compare* - runs script code to compare values, used extensively by the Image Manager
- *date* - date management

- docflow - docflow management, the heart of the DocFlow module
- docflowdoc - represents a single document in a docflow
- doclist - library document lists
- filters - runs script code to filter values, used extensively by the Image Manager
- grid - manages tab-separated-value strings easily
- http - http/https client for interacting with web servers
- inbound - inbound document management, the heart of the Image Manager module
- inbounddoc - represents a single document in an Image Manager library
- json – conversion of data to JSON format for ease of Javascript processing
- inifile - ini file access
- keyfile - keyed file access
- libraries - library lists
- library - library management
- mailattachment - a file attachment in a mail message object
- mailmessage - a mail received by the mailreader object, or sent by the mailsender object
- mailreader - a POP email client
- mailsender - a SMTP email client, providing an object oriented option to the email() function
- marked – marked record management
- memcollection - an in-memory collection object, which offers faster access and larger keys compared with the collection object
- notify - template-based email notifications
- objcollection - a specialized collection whose elements are objects
- rac - remote access codes for documents
- search - library search execution
- system - operating system and file system access
- textfile - text file access
- validations - runs script code to perform value validation, used extensively by the Image Manager
- webapi – creation of web-oriented URL strings for web form and DTC processing
- xmlreader - xml document parsing

Examples of using many of these objects can be found in the rule file samples/objects.rul.

15.3.1 addrbook

addrbook

```
book=new("addrbook",name$[,create])
```

Address book object stores delivery records, identified by an entity ID and document type. Each record is maintained as a string template with the following fields:

entityid\$, doctype\$, entityname\$, contactname\$, sendto\$, combine.

All address books are stored in the addrbks subdirectory under the UnForm server. The files are named as name\$.dat", so the name must be a valid file name for the operating system. If the address book exists, it is opened for use. If not, and the create flag is true (1), it is created. An error occurs if the address book is not found and the create flag is missing or false (0).

Properties

filename\$ is a read-only property that contains the actual disk file used to store the address book records.

template\$ is a read-only property that can be used to dimension an address entry template. i.e. dim rec\$:book'template\$.

Methods

count() returns the number of entries in the addressbook.

deladdress(entityid\$,doctype\$) removes the address book entry. Returns 1 or 0.

getaddress(entityid\$,doctype\$,address\$) fills address\$ template with the requested entry, returns 1 if successful, or 0 for failure. For example, if there is no record found, a 0 is returned, and address\$ will be an empty, but dimensioned, template string.

newaddress(address\$) creates an empty address\$ template that can be filled by a code block before writing. Always returns 0.

putaddress(entityid\$,doctype\$,address\$) updates address book using entityid\$ and doctype\$ identification, writing the data in the template address\$. Returns 1 if successful, or 0 if not. Note that address.entityid\$ and address.doctype\$ are set to entityid\$ and doctype\$.

range\$(start,count[,order[,descending]]) returns a LF-delimited list of TAB-delimited records. Each record contains the same fields as the template above, separated by a TAB character. If start or end are 0, they are the first and last records, respectively. The optional order value can control the sequence of the records: 0 for entity ID/DocType, 1 for entity name, 2 for contact name, or 3 for send to address. The optional descending value can be 0 for ascending sequence, 1 for descending sequence.

15.3.2 binfile

binfile

fl=new("binfile"[,filename\$])

The binfile object provides read/write access to the file in binary fashion, where there is no concept of a record. All access is to specific byte positions in the file, or the file as a whole. The filename\$ argument specifies what file to open or create. If no file name is supplied, then a temporary file will be created. This temporary file will be erased when the UnForm job is complete.

Properties

filename\$ is a read-only property that contains the full path to the file name opened or created.

size contains the file size in bytes. If size is assigned, the file is expanded or truncated to the specified size.

Methods

append(block\$) appends block\$ to the end of the file, and returns the size of the file.

delblock(index,length) removes the specific bytes from the file, and returns the size of the file.

getblock\$(index,length) returns file content at the specified position and length. The position is 1-based, so the first byte is 1.

getfile\$() returns the entire contents of the file.

insblock(index,block\$) inserts block\$ in the file at the 1-based index position. Returns the size of the file.

purge() sets the file length to 0, just like setting the size property to 0.

putblock(index,block\$) writes block\$ to the file at the 1-based index position, replacing data at that position. Returns the size of the file.

putfile(block\$) updates the file contents to block\$, truncating or expanding as necessary. Returns the size of the file.

15.3.3 cirrusprint

cirrusprint

```
o=new("cirrusprint",[server$,login$,password$])
```

The cirrusprint object enables file submission to a CirrusPrint server, designating a location ID and device ID to deliver the file to. CirrusPrint is cloud printing and file delivery product offered by the UnForm publisher. It is useful in situations where UnForm output must be sent to remote printers, file systems, or users, such as when UnForm is hosted in the cloud and send print jobs to office location printers.

Properties

server\$ contains the CirrusPrint server url, starting with http:// or https://, normally ending with a :*port*, such as "<https://cloudprint1.sdsi.cloud:8443>". This value can be passed in the new() function.

login\$, password\$ must be set to login to the CirrusPrint server. The settings must match a company location id or user in the format *locid@company*, or a site user configured with API access by the CirrusPrint administrator. These values can be passed in the new() function.

If the password has the format "store:*name*", the password is retrieved from the secure password store maintained in the admin browser interface.

lasterr, lasterrmsg\$ are set if an error occurs during a method call.

Methods

sendfile(filename\$,locid\$,deviceid\$[,title\$]) sends the specified file from the UnForm server to the CirrusPrint server, and delivers it to the location and device specified. The optional title\$ value is used to assist in delivery naming, such as when a file is delivered to a remote file system. If supplied, the remote file name will be derived from the title rather than a random work file name.

15.3.4 collection

collection

```
coll=new("collection",[noclose])
```

The collection object provides access to a collection of items stored by a key or index. Keys can be up to 127 characters long. Methods are provided to add, update, and remove elements from the collection, to find items in the collection, and to list items in the collection. Collection storage is mapped to disk, so there can be a virtually unlimited number of elements. If you wish to have the disk file remain open for the lifetime of the object, supply the noclose argument as true (non-zero). Note however, that if many collections are maintained during a job, only a limited number can have the noclose option set to true, since operating systems impose a limit on how many files can be open at one time by a process.

Properties

count is a read-only property that holds the number of items in the collection.

Methods

add(ky\$,value\$|value) adds a string or numeric value to the collection, identified by unique key. An error occurs if the collection already contains the specified key.

add(value\$|value) adds the specified string or numeric value at a sequential index position. No key is associated with the item.

addlist(values\$[,dlm\$]) adds several string values identified by sequential index. The list of values is delimited by a linefeed character (\$0A\$), or by the delimiter if supplied.

clear() removes all items from the collection.

copyto(colobject) copies all elements from the this collection to the specified collection object.

exists(ky\$) returns true (1) if ky\$ exists in the collection, false (0) if not.

getitem\$([dlm\$]) returns a delimited list of values from the collection, using the specified delimiter. If no delimiter is specified, a linefeed (\$0A\$) is used.

getkeys\$([dlm\$],[order]) returns a delimited list of keys from the collection, using the specified delimiter. If no delimiter is supplied, then a linefeed (\$0A\$) is used. If order is 0, or not supplied, the keys are returned in the order added to the collection. The keys are returned in ascending sequence if order=1, or descending sequence if order=2.

item\$(ky\$|index) returns the string value identified by the key or sequential index.

item(ky\$|index) returns the numeric value identified by the key or sequential index.

key\$(index) returns the key of the item at the specified sequential index position.

keycur\$() returns the current key, based on the last key operation.

keyfirst\$() returns the first key in the collection (in ascending key sequence order).

keylast\$() returns the last key in the collection.

keynext\$() returns the next key in the collection, relative to the key of the last key operation.

keyprev\$() returns the previous key in the collection, relative to the key of the last key operation.

remove(ky\$|index) removes the item identified by its key or sequential index from the collection.

update(ky\$,value\$|value) updates the item identified by ky\$ with value\$ or value. Adds the key if it doesn't exist.

15.3.5 compare

This object can be used to compare two values, returning true or false (1 or 0, respectively) for each function. It is used by the Image Manager for detection operation, and the object is available in each image manager job.

Methods

Equal(left\$,right\$) returns true if left\$ and right\$ are identical strings

NCEqual(left\$,right\$) returns true if left\$ and right\$ are identical case-insensitive strings ("dog" equals "Dog")

NumEqual(left\$,right\$) returns true if left\$ and right\$ are identical values when converted to numbers ("123.50" equals "123.5000")
NotEqual(left\$,right\$) returns true if left\$ and right\$ are not identical strings
NotNCEqual(left\$,right\$) returns true if left\$ and right\$ are not identical case-insensitive strings
NotNumEqual(left\$,right\$) returns true if left\$ and right\$ are not identical values when converted to numbers
Like(value\$,wildcard\$) returns true if value\$ matches the wildcard, which uses * for any sequence of characters, and ? for any character ("Acme Inc" is like "*Inc")
NCLike(value\$,wildcard\$) returns true if value\$ matches the wildcard, which uses * for any sequence of characters, and ? for any character, without regard to letter case
Match(value\$,regex\$) returns true if value\$ matches the regular expression regex\$ ("12/31/2018" matches "[01][0-9]/[01][0-9]/\d\d\d\d")
NCMatch(value\$,regex\$) returns true if value\$ matches the regular expression regex\$, without regard to letter case
LessThan(left\$,right\$) returns true if the left\$ string is less than the right\$ string, using character comparison ("abc" is less than "xyz")
NumLessThan(left\$,right\$) returns true if the left\$ string is less than the right\$ string, when converted to numbers ("999" is less than "1000")
GreaterThan(left\$,right\$) returns true if the left\$ string is greater than the right\$ string, using character comparison
NumGreaterThan(left\$,right\$) returns true if the left\$ string is more than the right\$ string, when converted to numbers
OneOf(left\$,list\$) returns true if left\$ is found as a value in the comma-separated list\$ ("CA" is one of "WA,OR,CA")
Contains(left\$,right\$) returns true if the value right\$ is found anywhere in the value left\$ ("PC Mac" contains "Mac" - and also contains "C")
NCContains(left\$,right\$) returns true if the value left\$ is found anywhere in the value right\$, without regard to letter case
IsDate(date\$,format\$) returns true if the date string is a valid date using the format specified, mdy, ymd, or dmy, indicating the order of date elements for month, day, and year.

15.3.6 date

date

dt=new("date")

The date object provides date-oriented functionality, including date parsing and formatting. When the date object is created, the datetime property is set to the current date and time. A datetime value is a numeric Julian number, indicating the number of days since January 1, 4713 BC, plus time expressed as a fraction of a day. Methods are provided for parsing a text date into a datetime value, to format a datetime value into a human-readable value, and to calculate elapse time between two datetime values in different increments.

Properties

d is a read-only property that provides the day.
datetime is the date and time value upon which all methods work. The datetime value is initially set to the date and time at the moment the object is created. It can be updated to the current date and time with the update() method, or set to a value via the parsedate() function or setdate() function.
hr is a read-only property that provides the hour (using a 24 hour clock).

m is a read-only property that provides the month.

mn is a read-only property that provides the minute.

se is a read-only property that provides the second.

utcoffset provides the offset from Universal Time for the local time zone. The value is provided as a fraction of a day, so it can be added or subtracted from datetime without any conversion.

y is a read-only property that provides the year.

Methods

days([enddatetime]) returns the number of days, with a fractional amount, between the current date and time and datetime, or the supplied date and time (such as the datetime value of another date object).

format\$([fmt\$]) returns the formatted date and time. If no format is supplied, or it is null or "utc", the date is returned in UTC format in adjusted by the utcoffset property. For example: Fri, 7 Aug 2009 23:17:04 +0000. If fmt\$ is "local", then the format is the same, but reported for the local time zone. For example: Fri, 7 Aug 2009 16:17:04 -0700. If fmt\$ is "ymd", then a 14-byte string is returned in the format yyyyymmddhhmmss. If fmt\$ is "iso" or "isodtm", a date (yyyyymmdd) or date-time (yyyyymmddThhmmssZ) format is returned, using UTC time rather than local time.

Additional formats are custom, using mapping characters that are replaced with appropriate date/time components. The characters are:

- am or pm
- AM or PM
- YYYY or YY (4- or 2-digit year)
- MMMM, MMM or MM (month name, abbreviation, or number)
- DDDD, DDD, or DD (day name, abbreviation, or number)
- HH (24 hour clock)
- hh (12 hour clock)
- mm
- ss
- tzc, tz (timezone offset from UTC, in +/-HH:mm or +/-HHmm format)

Other characters represent themselves. "MM/DD/YYYY" would return a typical US date. DD/MM/YYYY would return a typical Canadian date.

fromutcseconds(utc-seconds) sets the instance's date/time value based on the utc-seconds value supplied, which is a common operating and internet time format of the number of seconds since Jan 1, 1970 in UTC (0000 timezone) time.

hours([enddatetime]) returns the number of hours between the current date and time and datetime, or the supplied date and time.

minutes([enddatetime]) returns the number of minutes between the current date and time and datetime or the supplied date and time.

parsedate(datestr\$, [fmt\$]) parses a human readable date using fmt\$ for parsing rules, sets datetime, and returns datetime. If not supplied, the default format is "utc". The parsing rules are "utc" for UTC format, "ymd" for yyyyymmddhhmmss format (if all digits) or "mdy" or "dmy" or "ymd" for delimited dates, such as 12/31/2009 or 31/12/2009.

seconds([enddatetime]) returns the number of seconds between the current date and time and datetime, or the supplied date and time.

setdate(year, month, day [,hours [,minutes [,seconds]]]) sets the date and time according to the arguments provided, and returns datetime. Only year, month, and day arguments are required.

update() updates datetime to the current date and time, and returns datetime.

15.3.7 docflow

The docflow object is used extensively by the DocFlow component, and some of its methods are useful in custom programming as well. In particular, the createdoc() method is used to start an existing library document in a flow, and run the flow's initialize routine.

Many of its methods are duplicated in the [docflowdoc](#) object, for convenience when scripting with a particular document in a flow.

Properties

lasterrmsg\$ contains the last error message from an operation.

Methods

addnote(flowid\$,doctype\$,docid\$,note\$) adds a note to the specified document.

addstamp(flowid\$,doctype\$,docid\$,stampname\$,left,top) adds the named stamp file to the flow document. The stamp name should be the base filename of a stamp image in the df/stamps directory. Left and top specify the position on the page in inches.

advance(flowid\$,doctype\$,docid\$,errmsg\$) moves the specified document a step forward in the flow, and runs the advance script code. The errmsg\$ variable returns any errmsg\$ set in the script code. This operation is normally performed through the user interface.

attachfile(flowid\$,doctype\$,docid\$,file\$[,newsubid\$[,title\$]]) adds the specified file, found on the uniform server, as an attachment to the document specified. If newsubid\$ is supplied, it is filled with the subid of the file just added. The subid is generated as 'upload*' in order to ensure that the file is copied to the original source document when the document is finalized. The title\$ value, if supplied, creates a title for the attachment in the browser interface. If not supplied, the file name is used.

attachsubdoc(flowid\$,doctype\$,docid\$,fromlib\$,fromdoctype\$,fromdocid\$,fromsubid\$[,newsubid\$]) attaches an existing library archive image to the specified document in a flow. Flowid\$, doctype\$, and docid\$ identify which flow document to modify, and fromlibrary\$, fromdoctype\$, fromdocid\$, and fromsubid\$ identify which existing image to attach. If newsubid\$ is supplied, it is filled with the subid of the attachment just added.

createdoc(flowid\$,fromlib\$,fromdoctype\$,fromdocid\$,fromsubid\$,docprop\$[,additional\$]) adds a new image to the specified flow library. The source image is specified in the fromlibrary\$, fromdoctype\$, fromdocid\$, and fromsubid\$ values. It fills docprop\$ with a document property template of the newly created document in the flow's system library. If additional\$ is provided, it is used to automatically add attachments to the document. If set to 1 all non-@text subids of the source document are attached. If set to 2, all images linked from the source document are attached. If set to 3, both non-@text subids and linked images are added as attachments.

createdoc(flowid\$,fromlib\$,fromdoctype\$,fromdocid\$,fromsubid\$,docprop\$[,links\$]) adds a new image to the specified flow library. The source image is specified in the fromlibrary\$, fromdoctype\$, fromdocid\$, and fromsubid\$ values. It fills docprop\$ with a document property template of the newly created document in the flow's system library. If links\$ is provided, it should be line-feed delimited library image identifiers, which is a pipe- or tab-delimited value of library, doctype, docid, and subid values to extract images from. These are all added as attachments.

delstamp(flowid\$,doctype\$,docid\$,stampname\$) removes stamps of the specified name from the flow document. The stamp name is the base filename of the stamp image from the df/stamps directory. Note the extension of the file is not needed, for example either "approved.jpg" or "approved" work.

finish(flowid\$,doctype\$,docid\$,errmsg\$) completes the document flow, creates the XML document and any attachment or annotation images, and runs the finalize script code. If the script code sets errmsg\$, it is returned, though the finalize actions are not stopped. This operation is normally performed by the user interface, but script code could do so if there is reason to finalize a document without user action.
flowrange\$(id\$,first,count) returns a tab-separated-values list of documents in the specified flow. Each row includes doctype, docid, and timestamp columns.
flowstep\$(flowid\$,step) returns the flow step name of the 1-based step number.
flowsteprole\$(flowid\$,stepname\$ stepnum) returns the role name assigned to the step name or number in the specified flow definition.
flowsteps(flowid\$) returns the number of steps defined in the flow specified.
flowsteps\$(flowid\$) returns a list of linefeed-delimited steps for the specified flow definition. Each step is a tab-delimited row of name, description, and role name.
getdocdata(flowid\$,doctype\$,docid\$,data\$) creates the data\$ template with the following fields: library\$, doctype\$, docid\$, subid\$, started\$, updated\$, step\$, updatehist\$, notes\$, duedate\$, invaliditems\$, annotations\$, stephist\$. These equate to properties in the docflowdoc object, and are further documented there. This information should be considered read-only.
getdocsbysource(library\$,doctype\$,docid\$[,flowid\$]) returns a linefeed-delimited list of flow records found in active flows for the source document specified by library\$, doctype\$, and docid\$. The source document is the original document that contains the subid any given flow record was started from. All subids for the specified document are checked. If the flowid\$ argument is supplied, only records from that flow are returned. The records are in tab-delimited format, with the flowid, doctype, and docid of the flow library record, suitable for creation of a docflowdoc object for further information. If the flowid\$ argument is supplied, the returned list has just doctype and docid values.
getfield(flowid\$,doctype\$,docid\$,name\$) returns the value of the custom field identified by name\$ from the flow document specified.
getfield(flowid\$,doctype\$,docid\$,name\$,value\$) fills value\$ with the value of the custom field identified by name\$ from the flow document specified.
getrole(roleid\$,roleprop\$) fills the rollprop\$ template with role properties, which are: roleid\$, description\$, and members\$, a semi-colon delimited list of user ID's who are members of the role.
getparam(flowid\$,name\$) returns the value of the named parameter in the specified flow definition. It returns null if the flow id doesn't exist or the parameter name doesn't exist.
hasstamp(flowid\$,doctype\$,docid\$,stampname\$) returns true (1) if the named stamp exists on the flow document, or false (0) if not. Note the extension of the file is not needed, for example either "approved.jpg" or "approved" work.
libname\$(id\$) returns the library pathname of the source id\$, used when access to a library object for the library is required.
putfield(flowid\$,doctype\$,docid\$,name\$,value\$) sets the value of the named field for the specified document.
delfield(flowid\$,doctype\$,docid\$,name\$) removes the specified field in the specified document. This doesn't remove it from the flow definition, but just removes the record where it was stored for the document. It has the same effect as putfield with value\$="".
getfields\$(flowid\$) returns a line-feed delimited list of custom field names for the flow definition specified.
nextstep\$(flowid\$,curstep\$) returns the next sequential step name after the specified step name, in the flow specified. If the current step is the last step, "" is returned. Use this function to retrieve the default next step in script code, in cases where the step order is not modified by scripting.
paramexists(flowid\$,name\$) returns true (1) if the named parameter exists in the specified flow definition, false (0) otherwise.
reverse(flowid\$,doctype\$,docid\$,errmsg\$) moves the specified document a step backward in the flow. The errmsg\$ variable is filled if an unexpected error occurs. This operation is normally performed through the user interface.
setstep(flowid\$,doctype\$,docid\$,step stepname\$) sets the current step number of name of the specified document.

signaturefile\$(flowid\$,doctype\$,docid\$) extracts the signature file associated with the flow document and stores it as a JPEG file on disk, and returns the file name. If no signature file exists, null ("") is returned.
signatureid\$(flowid\$,doctype\$,docid\$) returns the subid of a signature image found in the flow document's attachments. It returns null ("") if no signature file exists. The signature file is identified as an attachment with an "upload" prefix and a title of "df-signature.jpg".
transfer\$(fromflowid\$,fromdoctype\$,fromdocid\$,toflowid\$) transfers a flow document to a different flow. It returns the new flow document in a colon-delimited structure <i>flowid:doctype:docid</i> . The from document identification can be found in a docflowdoc object, typically the "doc" variable in flow scripting, as doc'flowid\$, doc'dfdotype\$, and doc'dfdocid\$.
updatehist(flowid\$,doctype\$,docid\$,what\$) adds a message what\$ to the update history of the specified document.

15.3.8 docflowdoc

doc=new("docflowdoc",flowid\$,dfdotype\$,dfdoid\$)

The docflowdoc object provides simplified access to a single document in a document flow. Most methods are mapped to those found in the [docflow](#) object, but don't require specification of the flow and document identification, since that is provided at the time the object is instantiated. This object is automatically provided in [DocFlow scripting](#) routines.

Properties

duedate\$ is the due date for the document. This is a writable property and should be in yyyy-mm-dd format.
flowid\$, dfdoctype\$, dfdocid\$ are document identifiers provided when the object is instantiated.
library\$, doctype\$, docid\$, subid\$ are values identifying the source document and image subid used when starting this document in the flow.
notes\$ is a list of notes in inifile format. Each section is formatted as [yyyymmdd-hhmmss:useridl], and the section content is the note text.
started\$, updated\$ are date values in yyyy-mm-dd hh:mm:ss format (24-hour clock) showing the start time and last-updated time.
step\$ is the current step name.
stephist\$ is a line-feed delimited list of step names, excluding the current step.
title\$ is the title of the document. This is a writable property.
updatehist\$ is a list of updates in tsv format, line-feed delimited updates, each in tab-delimited format with date (yyyy-mm-dd hh:mm:ss format), user, and description items.
url\$ is a read-only property that returns the url for this document. It is the same as the geturl\$() method, with no prefix argument.

Methods

addnote(note\$) adds a note to the document.
addstamp(stampname\$,left,top) adds the named stamp file to the flow document. The stamp name should be the base filename of a stamp image in the df/stamps directory. Left and top specify the position on the page in inches.
advance(errmsg\$) moves the document a step forward in the flow, and runs the advance script code. The errmsg\$ variable returns any errmsg\$ set in the script code.
attachfile(file\$[,newsbid\$[,title\$]]) adds the specified file, found on the uniform server, as an attachment to the document. If newsbid\$ is supplied, it is filled with the subid of the file just added. The

subid is generated as 'upload*' in order to ensure that the file is copied to the original source document when the document is finalized. The title\$ value, if supplied, creates a title for the attachment in the browser interface. If not supplied, the file name is used.
attachsubdoc(fromlib\$,fromdoctype\$,fromdocid\$,fromsubid\$[,newsbid\$]) attaches an existing library archive image to the document in a flow. Flowid\$, doctype\$, and docid\$ identify which flow document to modify, and fromlibrary\$, fromdoctype\$, fromdocid\$, and fromsubid\$ identify which existing image to attach. If newsbid\$ is supplied, it is filled with the subid of the attachment just added.
delstamp(stampname\$) removes stamps of the specified name from the flow document. The stamp name is the base filename of the stamp image from the df/stamps directory. Note the extension of the file is not needed, for example either "approved.jpg" or "approved" work.
finish(errmsg\$) completes the document flow, creates the XML document and any attachment or annotation images, and runs the finalize script code. If the script code sets errmsg\$, it is returned, though the finalize actions are not stopped. This operation is normally performed by the user interface, but script code could do so if there is reason to finalize a document without user action.
getfield(name\$) returns the value of the custom field identified by name\$.
getfield(name\$,value\$) fills value\$ with the value of the custom field identified by name\$.
geturl([scriptprefix\$]) returns a url to display the document in a browser. The script prefix is the first portion of the url, including protocol, host, port, and script name, such as https://demo.example.com:28392/arc. If not supplied, the prefix comes from the external= setting in the [archive] section of uf101d.ini, or if executed in the context of a browser request, such as running a docflow job, it will match the browser's connection.
hasstamp(stampname\$) returns true (1) if the named stamp exists on the flow document, or false (0) if not. Note the extension of the file is not needed, for example either "approved.jpg" or "approved" work.
initinvaliditems() initializes all validation error messages for the document.
iteminvalid(item\$,errmsg\$) sets a validation error message on a field name, using the item naming syntax of "field.name".
nextstep(curstep\$) returns the next sequential step name after the specified step name, in the flow specified. If the current step is the last step, "" is returned. Use this function to retrieve the default next step in script code, in cases where the step order is not modified by scripting.
putfield(name\$,value\$) sets the value of the named field.
delfield(name\$) removes the specified field in the document. This doesn't remove it from the flow definition, but just removes the record where it was stored for the document. It has the same effect as putfield with value\$="".
reverse(errmsg\$) moves the document a step backward in the flow. The errmsg\$ variable is filled if an unexpected error occurs. This operation is normally performed through the user interface.
setstep(step stepname\$) sets the current step number of name of the document.
signaturefile\$() extracts the signature file associated with the flow document and stores it as a JPEG file on disk, and returns the file name. If no signature file exists, null ("") is returned.
signatureid\$() returns the subid of a signature image found in the flow document's attachments. It returns null ("") if no signature file exists. The signature file is identified as an attachment with an "upload" prefix and a title of "df-signature.jpg".
updatehist(what\$) adds a message what\$ to the update history of the document.

15.3.9 doclist

doclist

```
docs=new("doclist",[filename$[,initfile]])
```

A doclist is a list of documents from one or more libraries. Each document is identified by a library, document type, and document ID, called a document record. Doclist objects can be manipulated and

listed. A doclist object is also created by the search object, providing a set of methods for processing the search results.

If filename\$ is supplied, then that file is opened and used for the document lists. If not, a new temporary file is created, which is automatically erased when the object is destroyed. If initfile is provided and true (1), the document list is initialized so no document records are present.

Properties

listfile\$ is a read-only property that provides the name of the document list data file.

Methods

clear() clears the list of all documents.
count() returns the number of records in the list.
count(library\$) returns the number of records in the list for the given library.
count(library\$,doctype\$) returns the number of records in the list for the given library and document type.
deldoc(library\$,doctype\$,docid\$) removes the specified document from the list. Note this does not affect the document stored in the library.
getalldocs\$(first,count [,descending]) returns a list of document library, doc type, and doc ID's. The three fields are delimited by tabs (\$09\$) and the records are delimited by linefeeds. Up to count records are returned, from starting with index indicated by first. If descending is true (1), the list is returned in reverse order.
getdoc(library\$,doctype\$,docid\$[,prop\$]) returns 1 if the document exists in the list. If prop\$ is provided, it creates it as a template and fills it if the document exists. The prop\$ template contains document properties of the document itself, from the archive storage system, using a library object. The following properties are provided: <ul style="list-style-type: none"> • prop.date\$ • prop.time\$ • prop.title\$ • prop.entityid\$ • prop.notes\$ • prop.keywords\$ • prop.categories\$ • prop.links\$
getdocs\$(library\$,doctype\$,first\$ first,count [,descending]) returns a list of document library, doc type, and doc ID's. The three fields are delimited by tabs (\$09\$) and the records are delimited by linefeeds. Within the range of library and document type, up to count records are returned, from starting with the document ID indicated by first\$, or index indicated by first. If descending is true (1), the list is returned in reverse order.
getlibs() returns a linefeed (\$0A\$) delimited list of libraries in the list.
gettypes\$(library\$) returns a linefeed delimited list of document types, within a library name, in the list.

movefirst(library\$,doctype\$,docid\$[,docprop\$]) moves to the first record and returns the library, document type, document ID, and optional document properties template with the following fields: date\$, time\$, title\$, entityid\$, notes\$, keywords\$, categories\$, and links\$, referenced as docprop.name. Returns 1 if successful, 0 if not.
movelast(library\$,doctype\$,docid\$[,docprop\$]) moves to the last record. Returns 1 if successful, 0 if not.
movenext(library\$,doctype\$,docid\$[,docprop\$]) moves to the next record. Returns 1 if successful, 0 if not.
moveprev(library\$,doctype\$,docid\$[,docprop\$]) moves to the previous record. Returns 1 if successful, 0 if not.
moveto(library\$,doctype\$,docid\$) navigates a specific point in the document list. Use this as a seed for subsequent moveprev() or movenext() methods.
putdoc(library\$,doctype\$,docid\$) adds the specified document to the list. Note this does not affect the document stored in the library.

15.3.10 filters

filterobj=new("filters")

The filters object is designed to run filters against text data. Both standard and custom filters can be defined in the Image Manager's Custom Code window. The goal of each filter is to process text data and modify it in some way so that it is more useful. A filter could be as simple as something that formats a raw number into a nice text format, or strips out non-digit characters, or even manipulates line item detail to organize it into proper columns and rows.

Filter definitions are found in the im/filters.ini and im/filters.custom.ini files. Each filter is in a section, headed by the name in brackets. Each section contains a description= line, an optional parameters list, and an optional types list. The types list contains a comma-separated list of field types this filter can apply to, to provide context when looking up filters in job design. Types can include zone types ocr, bcd, grid, col, and field types text, note, number, date, list, and lookup. A filter can be run by code regardless of the types= list. The balance of the filter section is code designed to convert the value in\$ to the value out\$ upon execution with the run() or run\$() methods.

Filters can use parameters, passed into the running code as part of the name or as an array. Parameters can be passed to filters explicitly in an associative array param\$[all] (i.e. p\$["Val1"]="Value 1", passed as p\$[all]), or via a syntax convention in the name (i.e. filteritem(Val1="Value 1",Val2="Value 2") - note quotes are optional if commas are not in values). If a named value is passed both via parenthesized argument and parameter array, the parameter array value is used.

Image Manager jobs use filters in various places, where the design calls for it. Custom code can also use filters, as the filters object is available to any code the UnForm server runs.

Properties

errmsg\$, lasterrmsg\$ can be set by filter code to indicate an error condition.

doc contains an [inbounddoc](#) object when the filter is run in the context of an image manager job, or an [docflowdoc](#) object when run from a docflow job.

Methods

Run(name\$,in\$,out\$,param\$[all]) runs the filter name\$ with a value supplied as in\$. It expects the filter to create out\$. Optionally pass a param\$[all] associative array with parameter names as keys, or use the name syntax described above. The method returns true (1) if the errmsg\$ variable is null.

Run(docobj,name\$,in\$,out\$,param\$[all]) runs the filter name\$ with a value supplied as in\$. It expects the filter to create out\$. Optionally pass a param\$[all] associative array with parameter names as keys, or use the name syntax described above. The docobj should contain an inbounddoc object or a libdoc object that has been pre-instantiated. This is an alternative to setting the doc property. The method returns true (1) if the errmsg\$ variable is null.

Run\$(name\$,in\$,param\$[all]) runs the filter name\$ with a value supplied as in\$. It returns the filter code's out\$ value. Optionally pass a param\$[all] associative array with parameter names as keys, or use the name syntax described above.

Run\$(docobj,name\$,in\$,param\$[all]) runs the filter name\$ with a value supplied as in\$. It returns the filter code's out\$ value. Optionally pass a param\$[all] associative array with parameter names as keys, or use the name syntax described above. The docobj should contain an inbounddoc object or a libdoc object that has been pre-instantiated. This is an alternative to setting the doc property.

15.3.11 grid

gridobj=new("grid")

The grid object is designed to manage tab-separated-values data, enabling creation, manipulation, and formatting of the data in discrete cells, columns, and rows. The object is used extensively by the Image Manager, but is available to other operations and rule set code blocks. Grids can contain subheaders to represent pages of data, which is used by the Image Manager when working with OCR data from multiple pages. A subheader is one with a column 1 format of [pagenum]. Some methods work with pages. When navigating the grid, you can use the 'ispagehdr(row)' method to test if a given row is a page header rather than a data row.

After instantiation, the data in the grid object is populated by the methods addcol(), addrow(), or parse(). Various methods are supplied to format and retrieve data by column, row, or as a tab-separated-values list.

Properties

cols contains the number of columns in the grid. Set this value to quickly modify the number of columns in the grid.

rows contains the number of rows in the grid. Set this value to quickly modify the number of rows in the grid.

dateorder\$ sets the parsing order for date format options in the grid. This can be mdy, dmy, or ymd. When cell values are parsed, date segments (delimited by any non-digit character) are assumed to be in this sequence.

Methods

addcol(col\$,hasheader|header\$)) adds a column to the grid, by parsing the col\$ string for line-feed delimited values. If hasheader is true (non-zero), the first row of data is treated as a header. If header\$ is provided, it is used as a header. If neither hasheader or header\$ is provided, the header for the column is null.

addrow(row\$) adds a row to the grid, parsing the row\$ string for tab-delimited cells. Columns are added if needed.

deletecol(colnum) removes the column specified.

deletecolcell(col,row) removes the cell specified, shifting other rows in that column up.

deleterow(rownum) removes the row specified.

deleterowcell(col,row) removes the cell specified, shifting other columns in that row to the left.

formatdatecol(col[,order\$,format\$]) formats the column as date values. If order\$ is not provided, it uses the dateorder\$ property. If format\$ is not provided, the format used is YYYY-MM-DD. The format\$ string can contain MM, DD, YY, and YYYY values to control the format. Cells that do not have valid dates are set to null.

formatnumcol(col[,mask\$]) formats the column as number values. If mask\$ is provided, it controls the formatting. Otherwise raw numbers are shown. Non-number cells are set to null. Mask characters are replaced with text characters from the number string as follows:

- # with a digit or space
- 0 with a digit or "0"
- , with a thousands separator
- . with a decimal character
- - with a "-" if the number is negative, or space if positive, removing spaces between the - and the first digit if placed on the left of the mask
- + with a "+" if the number is positive, - if the number is negative

"#,###,##0.00-" will display 12345.67 as "12,345.67 ", or -124 as "124.00-".

formatstrcol(col[,mask\$]) formats text data in a column using a string format mask. Mask characters are replaced with text characters as follows:

- 0 with a digit
- A with a letter, which is converted to uppercase
- a with a letter
- X with any character, letters converted to uppercase
- x with any character
- Z with any letter or digit, letters converted to uppercase
- z with any letter or digit

"A0A-0A0" will convert x9za0b to "X9Z-A0B".

getarray(x\${all}) fills x\${all} with the contents of grid. X\${0:cols,0:rows}. Row 0 contains the column headers. Column 0 is empty.

getcell\$(col,row) returns the value of the specified cell.

getcol\$(colnum|colname\$[,withhdr]) returns a line-feed delimited list of values from the specified column, given its column number (1 to cols) or column header value as colname\$. If withhdr is true (non-zero), the first value will be the column header.

getcolnum(name\$) returns the column number of the column name specified. Column names are case-insensitive header values.

getpages\$() returns a line-feed delimited list of page numbers in the grid.

getpagetext\$([pagelist\$,withheader\$]) returns all the text of the specified page numbers, or all pages if there is no pagelist\$ specified or it is null. Each page is prefixed with a [pagenum] header row, and if withheader is true (non-zero), each page's rows includes an initial header row.

getpagetext\$(page\$,page[,withhdr]) returns all text of a single page, specified with a string or number of the page number. No page header is provided. A column header row is provided if withhdr is true (non-zero).

getrow\$(rownum) returns the tab-delimited cells of the specified row.

gettext\$([withhdr[,trim]]) returns all the text of the grid. If withhdr is true, the first row contains column headers. If trim is true, trailing empty rows are removed. There are no page headers included in the response.

initcol(colnum|colname\$) sets all the specified column's cells to null.

initrow(rownum) sets all the specified row's cells to null.

insertcol(col\$,colnum[,hasheader|header\$]) inserts a column at the specified position, and fills the column with line-feed delimited row data from col\$. If hasheader is true, the first row is assumed to be the column header. If header\$ is supplied, it is used as the column header.

insertcolcell(col,row,value\$) inserts a value at the cell specified, shifting rows in that column down.

insertrow(row\$,rownum) inserts a row at the specified position, and fill is with tab-delimited column data from row\$.

insertrowcell(col,row,value\$) inserts a value at the cell specified, shifting columns in that row right.

ispagehdr(row) returns 1 if the row is a page header row.

joinrows([col\$,pattern\$ [,delim\$]]) joins rows together by appending cells to the top row of any row group. Row groups separated by empty rows, where no cell has any data. A row group header row can also be separated by a pattern that occurs in a column or columns. The col\$ value can contain one or more column names separated by commas or linefeeds. A pattern is a literal text phrase, or a regular expression prefixed with ~. Regular expression patterns are case-insensitive. For example, to join rows where the first line of any row has a three-digit number in a LineNo column:
gjoinrows("LineNo","~\d{3}"). This function returns the number of row groups found.

If delim\$ is supplied it separates rows that are joined. If not supplied or null it defaults to a space.

multiplycols(col1,col2,colresult) multiplies numeric values in the col1 and col2 columns, and fills the colresult column with the result.

parse(txt\$,hasheader) parses the tab-separated-values text content into the grid, replacing any former contents. If hasheader is true, the first row of data is used for column headers.

removeemptyrows() removes all rows that have no data.

removepages() removes page headers from the grid. Page headers are typically supplied by an ocr grid column zone, but removing them can make some grid data processing easier.

removerowsabove(search\$[,includefoundrow]) searches rows (top down search) for a match, and if found, removes rows above the matching line. If includefoundrow is true (non-0), the line found is also removed. The search\$ string can be a simple text string, in which case all columns are scanned. It can contain a @collist suffix, where collist is a comma-separated list of column names or numbers to search. Also, if the string begins with ~, then the search is considered a case-insensitive regular expression rather than a simple text value. Use \@ and \~ to override these interpretations, and \\ to represent a backslash anywhere in the search string.

removerowsbelow(search\$[,includefoundrow]) searches rows (bottom up search) for a match, and if found, removes rows below the matching line. If includefoundrow is true (non-0), the line found is also removed. The search\$ string can be a simple text string, in which case all columns are scanned. It can contain a @collist suffix, where collist is a comma-separated list of column names or numbers to search. Also, if the string begins with ~, then the search is considered a case-insensitive regular

expression rather than a simple text value. Use \@ and \~ to override these interpretations, and \\ to represent a backslash anywhere in the search string.

Example: g'remove rows below("Total:@Price",1) would search the Price column for the string "Total:". If a matching row is found, the rows below, and the matching row, will be removed.

setcell(col,row,value\$) sets the value of the specified cell.

splitcol(source|source\$,target|target\$,cols) splits values in a source column into two values, and places the two values in the source and target columns. If cols is positive, the first value is the first cols characters, the second is the remaining characters. If cols is negative, the second value is the last cols characters, and the first value is remaining characters.

Source and target columns can be specified by column number or column name (both must be the same type).

splitcol(source|source\$,target|target\$,pattern\$) splits values in a source column into two values, and places the two values in the source and target columns. If pattern\$ starts with ~, the balance is a case-insensitive regular expression; otherwise, pattern\$ is treated as a delimiter value.

If the regular expression is anchored to the start of the string (^pattern), the first value is the matching string, the second value is the remaining characters. If it is anchored to the end of the string (pattern\$), the first value is the string up to the match, the second value is the matching string. If no anchor is used, the second value is the matching string, the first value is the string with the matched value removed. If the pattern contains a parenthesized segment, the sub-match is used as the second value, and the full match is removed from the first value.

If a delimiter is specified, the first value is the characters before the delimiter, and the second value is the characters after the delimiter.

Source and target columns can be specified by column number or column name (both must be the same type).

subtractcols(col1,col2,colresult) subtracts numeric values in col2 from col1, and fills the colresult column with the result.

sumcol(colnum) returns the sum of all numeric values in the specified column.

sumcols(col1,col2[,col3],colresult) adds numeric values in col1 and col2, and the optional col3, and fills the colresult column with the result.

unstackcols(collist\$,targetcol|targetcol\$[,pagebreaks]) modifies the specified columns, a linefeed- or comma-delimited list of column numbers or names. For each row, if any cell in the column list is null, the non-null cells in that row are appended (with a space) to the previous non-empty row values for that cell. This method can be used to accumulate stacked description columns by using a control column and the description column. Whenever the control column is null, the description column gets appended to the previous non-null description.

If a target column name or number is provided, then the values are appended to the target column rather than the collist\$ column in which they are bound. This can be used to split a stacked column into a two columns, one for the top row and the other for the rows below.

If pagebreaks is true (non-0), any page break resets the last non-empty row, so that each page is treated as an independent entity. If false or not supplied, then continuation lines may be appended to lines of the previous page. This enables stacked columns to cross page boundaries.

15.3.12 http

http

```
httpobj=new("http",[url$])
```

The http object provides access to HTTP servers, more commonly known as web servers. HTTP is the protocol used by web browsers to communicate with web servers. The http object performs the client-side activity of the HTTP protocol, connecting to and requesting actions of a web server. To use the object, set the url or any of its component element properties, optionally add cookies or files to be uploaded, and optionally set the user and password if authentication is required. Then issue a `getrequest()` method to submit the request and obtain a response.

Properties

customheaders\$ can be manipulated directly or updated with the <code>addheader()</code> method. The contents is added to the headers sent to the server. If manipulated, it must be constructed with <i>name: value<crlf></i> sequences, using colon-space breaks between the name and value, and with exactly one CRLF sequence following each header.
follow indicates HTTP redirect responses should be followed. It defaults to true (1), but if set to 0, redirect responses will be returned by the <code>getresponse()</code> method.
header\$ contains the last set of headers from the server.
host\$ is the hostname, or IP address, of the HTTP server.
method\$ sets the server communication method. It must be "get" or "post". To simulate a form or file upload, use the post method. To simulate a link click in a browser, use the get method. Generally, if you expect to send a large amount of data, use post.
password\$ can be specified if the server requires authentication.
path\$ is the portion of the url after the hostname:port and before the ?query string in. This normally represents a name-mapped file or script on the server.
port is the port on which the server is listening. The default ports are 80 for http protocol, and 443 for https protocol, but these values can be configured on the server and may vary.
protocol\$ should be http or https (for a secure server connection).
query\$ is the trailing portion of the url which scripts interpret for variable data. The data follows the path and a "?" delimiter, and is often in name=value pairs. The query string must be URL-encoded. The query string can be generated using the <code>addfield()</code> method, or you can manipulate the string directly using the <code>urlencode()</code> function.
reason\$ contains the last reason text from the server.
response\$ contains the last response body from the server.
status contains the last status code from the server.
tcpoptions\$ can contain a semicolon- or comma-delimited list of extra options for the socket connection to the http server. The options can include: <ul style="list-style-type: none"> • <code>certificates=ignore validate trustreqd</code> (https certificate trust level) • <code>nossl2, nossl3, notls1, notls1.1, notls1.2</code> (disable specific protocols) • <code>tls, tls1.1, tls1.2</code> (force a specific protocol)

For example, `obj.tcoptions$="tls1.2;certificates=trustreqd"` ensures a high level of TLS security, plus a requirement that the server's certificate is issued by an authority trusted by the local operating system.

timeout establishes the timeout for server communication, in seconds. By default, there is no timeout, and the object will wait forever for a server response.

url\$ is the full address of the currently requested web page. It is represented with a protocol (http/https), a host name, a port, path, and query string, such as "http://example.com/uniform.html".

userid\$ can be specified if the server requires authentication.

Methods

addcookie(name\$,value\$) adds a set-cookie header to the http request headers. Some server applications require receipt of a cookie value.

addfield(name\$,value\$) adds a name=value pair to the query string, with proper url-encoding.

addfile(name\$,filename\$) adds a file to the submission

addheader(name\$,value\$) adds a custom header to the request, useful when interacting with services that look for custom headers. This internally appends "*name: value*" plus a CRLF to the `customheaders$` property.

getfields() returns a count of the number of query string fields.

getheader\$(name\$) returns the value of the named header.

getname\$(n) returns the name of the nth query string value.

getresponse([response\$],[headers\$],[status]) submits the request and fills `response$`, `headers$`, and `status` with the server's response. The `response$` value contains the actual content, often an HTML page or XML document. The `headers$` field contains linefeed (\$0A\$) delimited rows of 'name: value' pairs. The status code is the HTTP status of the response, a number whose meaning is defined in the HTTP protocol specification. For example, a status of 200 means OK, and a status of 404 means the requested file wasn't found.

getvalue\$(n) returns the value of the nth query string value.

putfile(file\$,[response\$],[headers\$],[status]) submits the file specified as a PUT request to the server, using the URL path as the server-side file name. Web servers implement strict configuration security for PUT methods, and support for the method is controlled by the web server configuration. A status of 200 or 204 indicates success.

setbody(body\$,[contenttype\$])

Use this to manually set the body submitted to the web server. The body is automatically generated when you add files or fields, but if you want to control the format, for example to submit specific content types (i.e. text/xml), you can use this method.

15.3.13 inbound

inboundobj=new("inbound")

The inbound object is designed to manage documents stored in inbound source libraries. These libraries are supplied documents from configured inbound sources such as email and monitored directories, or can also be supplied documents programmatically or via upload, using the inbound object. The documents are identified by library, type, and doc ID, but these values are system-controlled. Inbound libraries are

staging repositories for documents that will ultimately be transferred to document archive libraries in the UnForm archiving system. Properties are maintained using the 'document data' feature of UnForm libraries, and then used as identification and data values when the documents are transferred.

Many of the document-related methods are duplicated in the inbounddoc object as a convenience.

Properties

lasterr contains an error number if the last method encountered an error to be reported.

lasterrmsg\$ contains an error message if the last method encountered an error to be reported

Inbound Source Methods

These methods are generally used only by the UnForm server internally when maintaining inbound source records.

count() returns the number of inbound source records.

delete(id\$) removes the source record id\$.

exists(id\$) returns true (1) if the source id\$ exists.

get(id\$,prop\$) fills the prop\$ template with the record for source id\$. Property fields include id\$, description\$, inactive\$, type\$, source\$, emserver\$, empassword\$, emmanager\$, emallowfrom\$, emtimeout\$, action\$, actionoption\$.

libname(id\$) returns the library pathname of the source id\$, used when access to a library object for the library is required.

put(id\$,prop\$) adds or updates the source record id\$ with data in the prop\$ template.

range\$(first,count[,flds\$]) returns a tab-delimited range of source records, starting with a 1-based index, for up to count records. Optionally specify a list of fields.

Document Related Methods

These methods related to specific documents in an inbound source library. Where id\$, doctype\$, and docid\$ values are specified, these refer to the working library document identification rather than the ultimate target library identification. The source id\$ refers to the inbound source identifying name rather than the library itself. The doctype\$ and docid\$ values are the auto-generated values used to identify the documents while they reside in the inbound source library. The docid\$ value is visible in the Image Manager interface, as the Inbound ID column, and both doctype and docid values can be seen when browsing the source library through the archive browser interface (these libraries are only visible to administrators). The doctype\$ value is based on the date the document was imported from the inbound source, in yyyy-mmdd format. The docid\$ value is a day value plus time-based sequence values, in the format mmdd-####-####.

assignto(id\$,doctype\$,docid\$[,userid\$]) assigns the specified document to userid\$, or de-assigns it if no userid\$ is supplied. De-assigned documents are in the pool of available documents that image manager users can self-assign.

bcd(id\$,doctype\$,docid\$,x,y,w,h[,page|pages\$][,symbology\$]) returns a barcode value found on the specified page(s) on the coordinates supplied (in inches). If no page is supplied, 1 is assumed. If a string of pages is supplied, such as "1-5", or "1,3,5", or "first-{last-1}", multiple values separated by [page] headers are returned. Optionally specify a symbology that the barcode must be, such as code39, code128, or qrcode.

createdoc(id\$,docprop\$) creates a new document in the source library, and fills docprop\$ with its default properties, so that docprop.doctype\$ and docprop.docid\$ are valid to further set up the inbound document in the source library.

deldoc(id\$,doctype\$,docid\$) removes a document from the inbound source library.
delfield(id\$,doctype\$,docid\$,name\$) removes a specific custom data field from the document.
getdoc(id\$,doctype\$,docid\$,prop\$) loads prop\$ with the underlying library document properties, using the library object. Note this data differs from the getdocdata() method, which produces document identification data to be used when transferring to an archive library. For example the doctype and docdata properties from this method hold inbound source library document type and id, which are auto-generated by the createdoc() method as documents arrive from their respective inbound sources.
getdocdata(id\$,doctype\$,docid\$,data\$) loads data\$ with the document identification properties that will be used when the document is transferred to an archive library. Template values include library\$, doctype\$, docid\$, subid\$, subtitle\$, title\$, notes\$, categories\$, keywords\$, and links\$.
getdocdataitem(id\$,doctype\$,docid\$,item\$) returns the value of one document data identifier. Items can be library, doctype, docid, subid, subtitle, title, notes, categories, keywords, and links.
getdocdataitem(id\$,doctype\$,docid\$,item\$,value\$) fills value\$ with the value of a specified document data item. Returns 1 if the item is found.
getfield(id\$,doctype\$,docid\$,name\$) returns the custom field data named name\$.
getfield(id\$,doctype\$,docid\$,name\$,value\$) fills value\$ with the custom field name\$ data.
getfields(id\$,doctype\$,docid\$) returns a tab-separated values list of custom field names and values.
getmeta(id\$,doctype\$,docid\$,name\$) returns the metadata value of name\$. Metadata items are read-only values created at the time a document is imported from a source, such as subject and from addresses of email sources, or file names of directory sources. Each name is prefixed with "@".
getmeta(id\$,doctype\$,docid\$,name\$,value\$) fills value\$ with the metadata value for name\$, such as @subject, or @filename.
getmetas(id\$,doctype\$,docid\$) returns a tab-separated values list of metadata names and values.
getpages(id\$,doctype\$,docid\$) returns the number of image pages in the document.
getsubdoc(id\$,doctype\$,docid\$,subid\$,prop\$) fills the library object subdoc properties in prop\$.
getsubdoc(id\$,doctype\$,docid\$,subid\$,prop\$,imagefile\$) fills the library object subdoc properties in prop\$, and also creates a file with the image of the subdoc and provides the filename in imagefile\$.
gridzone(id\$,doctype\$,docid\$,coldefs\$[,pages\$ page]) returns a tab-delimited-values list of OCR word data from the coldefs\$ specification. If no page is supplied, 1 is assumed. If a string of pages is supplied, such as "1-5", or "1,3,5", or "first-{last-1}", columns contain [page] headers. The coldefs\$ variable must be a tab-separated-values where each column is defined as a name, position, pages, and filter. Position is a comma-separated list of left, top, width, height in inches. Column filters are applied, and supplied with an inbounddoc object obtained with the id\$, doctype\$, and docid\$ fields.
initinvaliditems(id\$,doctype\$,docid\$) initializes all validation error messages for the document.
initinvaliditems(id\$,doctype\$,docid\$,item\$) initializes the validation error message for a single item\$, which can be a docdata property name or a custom field name.
joindocs(id\$,doclist\$) joins the images and properties of the documents in doclist\$. The doclist\$ value should be tab-separated-value list of doctypes and docids in the inbound source library. The second and later documents in the list are added to the first, so that images are appended as additional pages, notes are appended, and categories, keywords, and links are merged. The function returns tab-separated doctype and docid of the combined document, which is simply the first document type/id in doclist\$.

movetosource(id\$,doctype\$,docid\$,tosrcid\$[,newdoctype\$,newdocid\$]) transfers a document from the source *id\$* to a different source *tosrcid\$*. The document is removed from the original source, and begins the normal processing defined for the new source, such as OCR processing and job execution. If the *newdoctype\$* and *newdocid\$* arguments are provided, they will return the values in the new source library.

ocr(id\$,doctype\$,docid\$,x,y,w,h[,page|pages\$][,trim]) returns the OCR word values found on the specified page(s) on the coordinates supplied (in inches). If no page is supplied, 1 is assumed. If a string of pages is supplied, such as "1-5", or "1,3,5", or "first-{last-1}", multiple values separated by [page] headers are returned.

ocrconfidence(id\$,doctype\$,docid\$,x,y,w,h[,page|pages\$]) will return the lowest OCR confidence value for words in the region specified, a value from 1 to 100. If no values are available, this returns 100. Confidence values are returned by an UnForm OCR service.

pageswap(id\$,doctype\$,docid\$,page1,page2) swaps two image pages in a document.

putdoc(id\$,doctype\$,docid\$,prop\$) updates document properties in the underlying library, using the library object. Note this differs from the `putdocdata()` method, which updates properties related to document identification when a document is transferred to an archive library.

putdocdata(id\$,doctype\$,docid\$,data\$) updates document identification data with values in the *data\$* template. Template values include *library\$*, *doctype\$*, *docid\$*, *subid\$*, *subtitle\$*, *title\$*, *notes\$*, *categories\$*, *keywords\$*, and *links\$*.

putdocdataitem(id\$,doctype\$,docid\$,item\$,value\$) updates a specific docdata item, such as library or title.

putfield(id\$,doctype\$,docid\$,name\$,value\$) updates a custom field *name\$* with the supplied *value\$*.

removepage(id\$,doctype\$,docid\$,page) removes the specified page number from the document, shifting later pages up. Returns 1 if successful.

reversepages(id\$,doctype\$,docid\$) reverses the page order of the document.

rotate(id\$,doctype\$,docid\$,degrees) rotates all pages and re-processes them for text and page extraction.

splitdoc\$(id\$,doctype\$,docid\$) splits all images of a document into new single-page documents. It returns a tab-separated-values list of the added doc types and doc ids (note no trailing linefeed character). All document data and field properties are duplicated.

splitdoc\$(id\$,doctype\$,docid\$,atpage) splits the document into two, with pages starting with *atpage* removed from the first document and added to the second. It returns the added doc type and doc id, separated by a tab character. All document data and field properties are duplicated.

splitdoc\$(id\$,doctype\$,docid\$,zonename\$[,jobname\$]) splits the document at pages where the specified zone value changes. If no *jobname\$* is specified, the job currently assigned to the document is used. Both OCR and barcode zone types are supported. If the zone returns a value, and that value differs from the previous page, the document is split at that page. The function returns a tab-separated list of added doctypes and doc ids (note no trailing linefeed character). All document data and field properties are duplicated, so this process is normally followed by job execution on the existing and new documents in order to recalculate the data.

transfer\$(id\$,doclist\$) transfers the documents specified in *doclist\$* to their respective target libraries with the document properties defined as document data and fields. The *doclist\$* value should be a line-feed delimited list of tab-separated *doctype* and *docid* values in the inbound source library. Each *doctype/docid* pair identifies an inbound document to transfer. Only documents that have no validation errors are transferred. Before each document is transferred, if a job is assigned to the document, its preupload code is executed. Likewise, after the transfer, its postupload code is executed.

The method returns a log of the transfer, with a line for each document.

validatedoc(id\$,doctype\$,docid\$) performs all validation tests on the document's identification data and fields. If there is a job assigned to the document, its validation rules are used. Otherwise, the minimum validation required is performed, requiring library and doc type data.

zoneconfidence(id\$,doctype\$,docid\$,zonename\$[,jobname\$]) returns the lowest OCR confidence value for words in the named zone, from 1 to 100. If no jobname\$ is specified, the job currently assigned to the document is used. If no values are available, this returns 100. Confidence values are returned by an UnForm OCR service. If zonename\$ is null (""), all zones are checked.

15.3.14 inbounddoc

obj=new("inbounddoc",sourceid\$,doctype\$,docid\$)

The inbounddoc object provides methods to manage a single document in an inbound library. It provides identification properties and some image manager-specific values as properties rather than methods, plus offers simpler syntax for some additional methods mirrored in the inbound object.

Properties

exists is set to true (1) if the document specified in the new() instantiation function exists.

ibsrc\$, ibdoctype\$, ibdocid\$ are three properties that identify the document in the inbound source library. They are given these property names to distinguish them from the library\$, doctype\$, and docid\$ properties that are used when the document is transferred to a document archive library.

invaliditems\$ is a list of field and identification names and associated error messages. It should not be maintained directly, but rather through the initinvaliditems() and iteminvalid() methods. The list is in tab-separated-values format.

jobid\$ is the job id of the last job run against this document. Additional details about the job can be accessed in the browser interface.

jobname\$ is the name of the job assigned to this document, through Image Manager operations or directly via assignment of this property. Note that assigning the property does not run the job.

user\$ is the UnForm user id assigned to this document. This should be an image manager user. Documents assigned to a user show in the Image Manager pending list for that user. If user\$ is null (""), it is available for self-assignment.

The following additional properties can be assigned as identification and standard archive properties.

- library\$
- doctype\$
- docid\$
- subid\$
- subtitle\$
- title\$
- entityid\$
- date\$ (should be in yyyyymmdd format)
- categories\$ (pipe-delimited segments, semi-colon delimited category indexes)
- keywords\$ (semi-colon delimited words)
- links\$ (semi-colon delimited urls or pipe-separated library|doctype|docid[[subid] strings)

Methods

assignto([,userid\$]) assigns the document to userid\$, or de-assigns it if no userid\$ is supplied. De-assigned documents are in the pool of available documents that image manager users can self-assign.
bcd\$(x,y,w,h[,page pages\$][,symbology\$]) returns a barcode value found on the specified page(s) on the coordinates supplied (in inches). If no page is supplied, 1 is assumed. If a string of pages is supplied, such as "1-5", or "1,3,5", or "first-{last-1}", multiple values separated by [page] headers are returned. Optionally specify a symbology that the barcode must be, such as code39, code128, or qrcode.
delfield(name\$) removes a specific custom data field from the document.
getfield\$(name\$) returns the custom field data named name\$.
getfield(name\$,value\$) fills value\$ with the custom field name\$ data.
getfields\$() returns a tab-separated values list of custom field names and values.
getmeta\$(name\$) returns the metadata value of name\$. Metadata items are read-only values created at the time a document is imported from a source, such as subject and from addresses of email sources, or file names of directory sources. Each name is prefixed with "@".
getmeta(name\$,value\$) fills value\$ with the metadata value for name\$, such as @subject, or @filename.
getmetas\$() returns a tab-separated values list of metadata names and values.
getpages() returns the number of image pages in the document.
gridzone\$(coldefs\$,pages\$[page]) returns a tab-delimited-values list of OCR word data from the coldefs\$ specification. If no page is supplied, 1 is assumed. If a string of pages is supplied, such as "1-5", or "1,3,5", or "first-{last-1}", columns contain [page] headers. The coldefs\$ variable must be a tab-separated-values where each column is defined as a name, position, pages, and filter. Position is a comma-separated list of left, top, width, height in inches. Column filters are applied.
initinvaliditems() initializes all validation error messages for the document.
iteminvalid(item\$,errmsg\$) sets a validation error message on a field or identification name, using the item naming syntax of "field.name" or "ident.name".
<p>ocr\$(x,y,w,h[,page pages\$][,trim]) returns the OCR word values found on the specified page(s) on the coordinates supplied (in inches). If no page is supplied, 1 is assumed. If a string of pages is supplied, such as "1-5", or "1,3,5", or "first-{last-1}", multiple values separated by [page] headers are returned.</p> <p>This method is used by auto-generated code for ocr zone regions in Image Manager jobs, but can be useful outside of automatic zone assignments. For example, to convert text on page 1 to keywords, custom script code like this could be used:</p> <pre>doc'keywords\$=texttokeywords(doc'ocr\$(0,0,8.5,11,"1"))</pre>
ocrconfidence(x,y,w,h[,page pages\$]) will return the lowest OCR confidence value for words in the region specified, a value from 1 to 100. If no values are available, this returns 100. Confidence values are returned by an UnForm OCR service.
putfield(name\$,value\$) updates a custom field name\$ with the supplied value\$.
removepage(page) removes the specified page number from the document, shifting later pages up. Returns 1 if successful.
reversepages() reverses the page order of the document.
rotate(degrees) rotates all pages and re-processes them for text and page extraction.

splitdoc\$() splits all images of a document into new single-page documents. It returns a tab-separated-values list of the added doc types and doc ids (note no trailing linefeed character). All document data and field properties are duplicated.

splitdoc\$(atpage) splits the document into two, with pages starting with atpage removed from the first document and added to the second. It returns the added doc type and doc id, separated by a tab character. All document data and field properties are duplicated.

splitdoc\$(zonename\$[,jobname\$]) splits the document at pages where the specified zone value changes. If no jobname\$ is specified, the job currently assigned to the document is used. Both OCR and barcode zone types are supported. If the zone returns a value, and that value differs from the previous page, the document is split at that page. The function returns a tab-separated list of added doctypes and doc ids (note no trailing linefeed character). All document data and field properties are duplicated, so this process is normally followed by job execution on the existing and new documents in order to recalculate the data.

validate() performs all validation tests on the document's identification data and fields. If there is a job assigned to the document, its validation rules are used. Otherwise, the minimum validation required is performed, requiring library and doc type data.

zoneconfidence(zonename\$[,jobname\$]) returns the lowest OCR confidence value for words in the named zone, from 1 to 100. If no jobname\$ is specified, the job currently assigned to the document is used. If no values are available, this returns 100. Confidence values are returned by an UnForm OCR service. If zonename\$ is null (""), all zones are checked.

15.3.15 inifile

iniobj=new("inifile"[,filename\$])

This class manages sections and items in a text file in "ini" format, where sections in the file have a [name] header, and items are stored in name=value lines. The ini file format is common in many applications, including UnForm. Both the uf101d.ini and ufparam.txt file use this format.

If filename\$ does not exist and not null, it is interpreted as a string version of INI file content. It must have headers and name=value pairs under those headers. You can use \n sequences to represent line feeds, and \\ to represent backslashes.

If filename\$ is not provided, a temporary file is created that is erased when the object is destroyed.

Properties

content\$ is the read-only content of the entire file.

filename\$ is the read-only name of the file being managed.

Methods

getitem\$(section\$,name\$) returns the value of the named line in the named section.

getitems\$(section\$) returns a line-feed delimited list of item names in the named section.

getname\$(section\$,value\$) returns the first name in the section that has the value specified.

getnames\$(section\$) returns a list of names in a section, delimited by linefeeds (\$0A\$).

getsection\$(section\$[,alltext]) returns all lines in the section, delimited by linefeeds (\$0A\$). If alltext is supplied and true (non-zero), then all lines, including blank and comment lines, are included.

putitem(section\$,name\$,value\$) replaces or adds a line in the section, in name=value format.

putsection(section\$,lines\$) replaces or adds an entire section with the contents of lines\$, which should be a series of name=value lines separated by linefeeds (\$0A\$).

removeitem(section\$,name\$) removes the line identified by name\$ from section\$

removesession(section\$) removes the entire section.

15.3.16 jobdef

obj=new("jobdef"[,jobname\$])

The jobdef object can be used to maintain a job definition. Jobs are composed of several elements, including many that are lists of elements, such as zones and fields. If *jobname\$* is supplied, that job is loaded at the time the object is created. You can also use the get() method to load a job at any time.

The object instance maintains an in-memory copy of the job as methods are used to manipulate it. Once all modifications are complete, use the put() method to write the changed job to the job definition file.

Properties

codeinitialize\$, codefinalize\$, codezones\$, codedetect\$, codefields\$, codeidentification\$, codepreupload\$, codepostupload\$ contain custom script code for the respective sections. Code lines are separated with linefeeds (\$0A\$).

createdby\$, createdon are read-only values indicating the job's creating user and julian creation time.

enablenewlib is a boolean (1=true, 0=false) indicating the job can create a new library when transferring.

enableoverwrite is a boolean indicating the job can overwrite existing document images when transferring to a library.

inactive is a boolean indicating the job can't be run.

jobdesc\$ is the job's description.

jobname\$ is the job name, which identifies the job. It can be up to 20 alpha-numeric characters.

jobtype\$ is the type of job, such as Pick Ticket or AP Invoice. In the job user interface, the list of jobs is ordered by type, and detect and run actions can work with a single type of job.

lasterrmsg\$ will contain an error message when a method is unsuccessful.

singlepage is a boolean indicating when this job runs, all document pages are split into individual files automatically, and all zones are single page zones without page headers.

Methods

delldetect(name\$) deletes the named detection row.

delldfield(name\$) deletes the named data field row.

delldparam(name\$) deletes the named parameter row.

delldzone(name\$) deletes the named zone row.

get(name\$) loads the specified job, filling all properties and enabling method operations on its structure.

getlddetect(name\$,prop\$) creates the prop\$ template with the named detection item. Fields in the template are zone\$, operator\$, and value\$

getlddetect(seq,prop\$) creates the prop\$ template with the row sequence detection item.

getdetects\$([detail]) returns a linefeed-delimited list of detection row names. If detail is true (non-0) the, all properties for each row are returned tab-separated.

getdetects() returns the number of detection rows.

getfield(name\$,prop\$) creates the prop\$ template with the named field item. Fields in the template are name\$, description\$, type\$, value\$, filter\$, validation\$, autofill\$. Multiple filters and validations are separated by semicolons. Autofill\$ can be "1" or null. Valid types are text, longtext, note, number, date, checkbox, radio, list, lookup, grid, label, hidden, and link. Special formats:

- list=*item1;item2;...*
- radio=*value1;value2;...*
- checkbox=*caption*
- lookup=*lookup-name*
- label=*label-text*
- link=*link-text* (the value\$ field contains the actual url, while *link-text* is the visible text, if a different value is needed)

You can also see how these values are structured in the image manager job editor, as the fields match the screen presentation.

getfield(seq,prop\$) creates the prop\$ template with the row sequence field item.

getfields\$([detail]) returns a linefeed-delimited list of field row names. If detail is true (non-0) the, all properties for each row are returned tab-separated.

getfields() returns the number of field rows.

getident(name\$,prop\$) creates the prop\$ template with the named identification item. Fields in the template are name\$, description\$, value\$, validation\$, autofill\$. Multiple validations are separated by semicolons. Autofill\$ can be "1" or null. The names can be library, doctype, docid, subid, subtitle, title, entityid, date, categories, keywords, links, and notes.

getidents\$([detail]) returns a linefeed-delimited list of identification row names. If detail is true (non-0) the, all properties for each row are returned tab-separated.

getidents() returns the number of identification rows.

getparam(name\$,prop\$) creates the prop\$ template with the named field item. Fields in the template are name\$, description\$, and value\$.

getparam(seq,prop\$) creates the prop\$ template with the row sequence parameter item.

getparams\$([detail]) returns a linefeed-delimited list of parameter row names. If detail is true (non-0) the, all properties for each row are returned tab-separated.

getparams() returns the number of parameter rows.

getzone(name\$,prop\$) creates the prop\$ template with the named zone item. Fields in the template are name\$, type\$, position\$, pages\$, and filter\$. Multiple filters are separated by semicolons. Valid types are bcd, ocr, grid, and col.

getzone(seq,prop\$) creates the prop\$ template with the row sequence zone item.

getzones\$([detail]) returns a linefeed-delimited list of zone row names. If detail is true (non-0) the, all properties for each row are returned tab-separated.

getzones() returns the number of zone rows.

movefield(name\$,position) moves the named field to the specified row position.

moveparam(name\$,position) moves the named parameter to the specified row position.

movezone(name\$,position) moves the named zone to the specified row position.

put() updates the current job. Returns 1 for success, 0 for failure.

putdetect(name\$,prop\$) updates the named detection row with the property template values.

putfield(name\$,prop\$) updates the named field row with the property template values.

putident(name\$,prop\$) updates the named identification row with the property template values.

putparam(name\$,prop\$) updates the named parameter row with the property template values.

putzone(name\$,prop\$) updates the named zone row with the property template values.

15.3.17 jobdefs

The jobdefs object is used by the Image Manager to both configure and run jobs.

Methods

exportjobs\$(jobnames\$) returns the name of a work file, whose contents are built from the jobs named in the linefeed delimited list of image manager job names supplied. The work file is a structured data file (treat it as binary rather than text) containing both the job definitions and well as script library definitions used by the jobs, including filters, validations, and lookups. This file can be imported by another UnForm installation, in order to deploy jobs between sites.

getparam\$(jobname\$,name\$) returns the value of the named parameter in the specified job definition. It returns null if the job name doesn't exist or the parameter name doesn't exist.

importjobs(jobsfile\$[,log\$]) imports jobs and script library definitions from the supplied jobs file. This file must have been created by the exportjobs\$() method. Before importing the records, a dated backup zip file is generated in im/backups containing the jobs master file and the im/* script library ini files. If supplied, log\$ will contain a plain text log of actions performed.

logmsg(jobid\$,message\$) updates the log for the running job. The jobid\$ value is maintained by the job execution code and can be used 'as is' when your code needs to add logging data. It is also available in the doc object automatically instantiated in filter, validation, and lookups objects.

Note this differs from the system-level log() function, which writes lines to the UnForm server log file.

paramexists(jobname\$,name\$) returns true (1) if the named parameter exists in the specified job definition, false (0) otherwise.

searchjobs\$(text\$) returns a linefeed delimited list of job names, of jobs that contain the text supplied. If text\$ starts with ~, a regular expression match is performed. The entire job definition is searched.

15.3.18 json

json

jsonobj=new("json")

The json object provides functionality to convert data from different file structures often found in UnForm server operations into JSON format used often in Javascript code. The common use for this function is for custom web form development, where data from a server-run rule set can be obtained in Javascript at runtime using the runRuleSet function found in common.js and available in the browser interface to document management archives.

All text is assumed to be encoded as ISO-8859-1.

Methods

fromdlm\$(fileorstring\$ [,delim\$ [,quotes [,header\$]]]) returns a JSON representation of a delimited file, such as a CSV file or tab-delimited file. If no delim\$ character is provided, a tab (\$09\$) character is assumed. Quotes can be 0 or 1; 1 indicates that data that might contain the delimiter will be quoted. The first line of the file is assumed to contain column header names, and these names are used as the

JSON object names (after replacing invalid name characters with underscores). Some delimited files do not contain a header row, in which case you can supply a delimited list of column names in `header$`.

The response is an array of objects, where each file row is an array element, and each row is represented as an object with *name:value* pairs.

If the file doesn't exist, the value of the argument is used as if it was file content.

fromcsv\$(fileorstring\$,header\$) is equivalent to `fromdlm$(fileorstring$,"",1,header$)`

fromini\$(fileorstring\$) parses a file or string in INI format, with section headers in square brackets (i.e. [main]) and section data made up of lines of *name=value* pairs. Each section becomes an object whose value is another object.

fromtpl\$(template\$) returns a JSON object representing fields and data in a string template. For example, a library object can obtain properties of a document in a template: `obj'getdoc(doctype$,docid$,tpl$)`, and the returned `tpl$` value can be converted to JSON format using this method. Numeric and string types are maintained during this conversion.

jsclean\$(str\$) returns a string that has been modified to fix known JSON syntax incompatibilities with the internal JSON parsing engine. These include replacing empty objects with `""`, and converting exponential notation values (`n.nnnne+/-n`) with standard numbers.

jsencode\$(str\$) returns a string encoded to be safe to include in a quoted JSON string object, by backslash-escaping backslashes, quotes, linefeeds, and other binary and control characters.

jstoaarray(json\$,array\$[all]) attempts to convert the `json$` string to an associative (keyed) array. It first runs the `json` through the `jsclean$()`, then creates `array$[all]` from the `json` string, where each key is an object name, using `"."` separators between nested levels, and `.n` digit levels for arrays (1-based numbers). It returns 1 on success, 0 if it fails, in which case `array$` will be empty.

totpl(json\$,tpl\$) parses `json$` string and return a string template in `tpl$` with field names matching the name elements in the `json` structure. Arrays are converted to `name.n` notation. All template variables are strings, so must be accessed as `tpl.name$`. Some complex or nested `json` structures are not convertible. Returns 1 on success, 0 if it fails, in which case `tpl$` will be null.

A CSV to JSON example:

```
"First Name","Last Name",Age
"Joe","Smith",40
"Sue","Smothers",45
```

```
[ {First_Name:"Joe", Last_Name:"Smith", age:40} ,
  {First_Name:"Sue", Last_Name:"Smothers", age:45}
]
```

An INI to JSON example:

```
[Section 1]
Name1=Value 1
Name2=Value 2
[Section 2]
Name1=Value 1
Name2=Value 2
```

```
{
  Section_1: {Name1="Value 1", Name2="Value 2" },
```

```
Section_2: {Name1="Value 1", Name2="Value 2"}
}
```

A jstocarray/jsfields example:

Given json\$={ "names": [{ "First_Name": "Joe", "Last_Name": "Smith", "age": 40 } , { "First_Name": "Sue", "Last_Name": "Smothers", "age": 45 }] }

o'jstocarray(json\$,a\$[all]) will create an associative array a\$.

Return a list of elements in the a\$ array:

```
elements$=o'jsfields$(a$[all])
```

```
names.1.First_Name=Joe
names.1.Last_Name=Smith
names.1.age=40
names.2.First_Name=Sue
names.2.Last_Name=Smothers
names.2.age=45
```

Return a filtered list of elements:

```
elements$=o'jsfields$(a$[all],"names.2")
names.2.First_Name=Sue
names.2.Last_Name=Smothers
names.2.age=45
```

15.3.19 keyfile

keyfile

```
kfobj=new("keyfile"[,filename$])
```

A keyfile object provides access to a disk file that stores records based on a unique string key. Keys can be up to 127 characters long. Records can be of any size and format. If filename\$ is not supplied, a temporary file is created that is erased when the object is destroyed.

Properties

chan is a read-only property that stores the underlying channel number that the file is open on.
count is a read-only property that contains the number of records in the keyfile.
filename\$ is a read-only property that contains the name of the file being managed.
found is a read-only collection object handle that is filled by methods that find records or return ranges of records from the file.

Methods

delete(ky\$) removes the record identify by ky\$. If the key doesn't exist, an error occurs.
exists(ky\$) returns true (1) if ky\$ exists in the file, false(0) otherwise.

find(search\$,nocase[,invert]) initializes the found collection object, then fills it with keys and records whose records contain the text search\$. If nocase is true (1), then the search is case-insensitive. If invert is true (1), then records that do not contain search\$ are added, rather than those that do. The function returns the number of records found.

findreg(regex\$,nocase[,invert]) initializes the found collection object, then fills it with keys and records whose records match the regular expression regex\$. If nocase is true (1), then the search is case-insensitive. If invert is true (1), then records that do not match regex\$ are added, rather than those that do. The function returns the number of records found.

findwhere(whereexpr\$,dlm\$,quotes) initializes the found collection, then searches records that match the where expression. For this search, records are assumed to be in delimited format, with the supplied dlm\$ value as the delimiter. If quotes is true (1), then fields are parsed assuming they may be quoted to protect delimiter values in field values.

The structure of the where expression is of a Boolean expression using fields, values, comparison operators, parentheses, and AND or OR, using *#number* syntax to represent field numbers. All fields are assumed to be string data, but you can use num() to convert strings to numbers, so long as the string is an unpunctuated numeric value. Here are some examples:

#2<>"" - field 2 not null

#2="100" and (num(#3)>=0 and num(#3)<10000) - field 2 is "100" and field 3 is between 0 and 10000

keycur() returns the key of the last accessed key accessed.

keyfirst() returns the first key in the file in key sequence.

keylast() returns the last key in the file in key sequence.

keynext() returns the next key relative to the last key accessed.

keyprev() returns the previous key relative to the last key accessed.

keyval(recno) returns the key of the record number specified. The record number is a sequential value in key order.

range(first\$,count[,descending]) initializes the found collection, then fills it with keys and records starting from the key first\$, and adding up to count records. If descending is true (1), then records are returned in reverse key sequence. The number of records found is returned.

range(first,count[,descending]) initializes the found collection, then fills it with keys and records starting from the record number first, and adding up to count records. If descending is true (1), then records are returned in reverse key sequence. The number of records found is returned.

range(key1\$,key2\$) initializes the found collection, then fills it with keys and records from the key range provided. For example, to get all records with keys starting with "100", you might use range("100","100Z"). The two keys do not have to exist in the file. The number of records found is returned.

read\$([key\$][recno]) returns a record. If a key or record number are supplied, the specified record is returned. Otherwise, the next record in ascending key sequence is returned, relative to the last record accessed. If the key or record number does not exist, an error is generated.

readlock(key\$[,timeout]) reads and locks the record. If timeout is provided, then if the record is not available (locked by another task), the system will wait for up to timeout seconds before returning an error. If the key doesn't exist, or the record is locked, an error is generated.

write(ky\$,rec\$) writes the record identified by ky\$. If the key already exists, the record is replaced. If not, it is added. If the key is locked by another task, an error occurs.

15.3.20 libdoc

obj=new("libdoc",library\$,doctype\$,docid\$)

The libdoc object provides methods to manage a single document in an archive library. It provides identification properties and some methods designed to mirror the inbounddoc object used by the image manager. There are differences, however, since once a document exists in an archive library, it's identification properties are true document properties and not document data values. When transferred, the custom field items are transferred to document data values, and metadata values are transferred as document data values with names prefixed with "\$".

Properties

exists is set to true (1) if the document specified in the new() instantiation function exists.

libobj is the underlying library object for the library this document is stored in.

The following additional properties can be assigned as identification and standard archive properties. They are updated using the inbound object's putdocdataitem() method:

- library\$
- doctype\$
- docid\$
- subid\$
- subtitle\$
- title\$
- entityid\$
- date\$ (should be in yyyyymmdd format)
- categories\$ (pipe-delimited segments, semi-colon delimited category indexes)
- keywords\$ (semi-colon delimited words)
- links\$ (semi-colon delimited urls or pipe-separated library|doctype|docid[[subid] strings)

Methods

delfield(name\$) removes a specific custom data field from the document.

getfield\$(name\$) returns the custom field data named name\$. This utilizes the library object's getdocdata() method.

getfield(name\$,value\$) fills value\$ with the custom field name\$ data.

getmeta\$(name\$) returns the metadata value of name\$. Metadata items are read-only values transferred from inbound source libraries by the Image Manager, and represent data about the imported document, such as its original file name, or a from address or subject of an email source.

getmeta(name\$,value\$) fills value\$ with the metadata value for name\$.

getmetas\$() returns a tab-separated values list of metadata names and values.

putfield(id\$,doctype\$,docid\$,name\$,value\$) updates a custom field name\$ with the supplied value\$, using the library object's putdocdata() method.

15.3.21 libraries

libraries

```
libs=new("libraries")
```

The `libraries` object provides access to the list of libraries that are known on a system. Lists are returned in linefeed-delimited (`$0A$`) format, and each list row can be in one of two formats: just library path names, or all library details. A template can be obtained that describes the library information that is returned, so it is possible to develop routines that parse a list and obtain specific library information. A name-only template contains one field: `pathname$`. A detail template contains the following fields, delimited by a tab (`09`) or other character, if specified:

- `basename$` - the base filename of the library path.
- `dirname$` - the directory portion of the library path.
- `path$` - the full library path.
- `category$` - the category, if any, of the library.
- `description$` - the library description
- `created$` - the library creation date, in `yyyymmdd` format
- `defperms$` - default permissions (r, w, and/or d) in semicolon-delimited format
- `forceseq` - true (1) to force sequencers on sub IDs
- `inactive` - true (1) if the library is inactive and no longer can be updated
- `count` - the number of document records in the library

Methods

createlib(library\$,errmsg\$) - creates a new library, fills <code>errmsg\$</code> if an error occurs.
dellib(library\$) - remove library from library list (not from disk). Admin permissions required.
doccount(library\$) returns the number of documents in a library.
exists(library\$) returns true (1) if the library exists, false (0) otherwise.
getall([detail[,delim\$]) returns the names of all libraries, including inactive ones. If <code>detail</code> is true (1), then lines of detail are returned; if not, only path names are returned. If <code>delim\$</code> is not null, that is used as a template delimiter character; if it is null, a tab (<code>\$09\$</code>) delimiter is used.
getdelete(userid\$,detail[,delim\$]) returns the names of all libraries the specified user can delete from. If <code>detail</code> is true (1), then lines of detail are returned; if not, only path names are returned. If <code>delim\$</code> is not null, that is used as a template delimiter character; if it is null, a tab (<code>\$09\$</code>) delimiter is used.
getlib(library\$,prop\$) fills the template <code>prop\$</code> with library properties. The properties are as follows: <ul style="list-style-type: none"> • <code>prop.basename\$</code> - the base name (no path information) • <code>prop.dirname\$</code> - the path information to the library • <code>prop.path\$</code> - the full path, including <code>dirname</code> and <code>basename</code> • <code>prop.category\$</code> - a category of the library, used to organize libraries in the browser interface • <code>prop.description\$</code> • <code>prop.created\$</code> - <code>yyyymmdd</code> creation date • <code>prop.defperms\$</code> combination of r, w, and d, semicolon delimited • <code>prop.forceseq</code> - true (1) to force all sub ID's to have sequencing • <code>prop.inactive</code> - true (1) to make the library inactive, so no updates are allowed to the library • <code>prop.count</code> - the number of documents in the library • <code>prop.aliases\$</code> - semicolon delimited list of library alias names

getnames\$([detail[,delim\$]) returns the names of all active libraries. If detail is true (1), then lines of detail are returned; if not, only path names are returned. If delim\$ is not null, that is used as a template delimiter character; if it is null, a tab (\$09\$) delimiter is used.

getread\$(userid\$,[detail[,delim\$]) returns the names of all libraries the specified user can read from. If detail is true (1), then lines of detail are returned; if not, only path names are returned. If delim\$ is not null, that is used as a template delimiter character; if it is null, a tab (\$09\$) delimiter is used.

gettemplate\$(detail[,delim\$]) returns a string template definition for detail or non-detail return lines, optionally with the specified delimiter. If no delimiter is specified, a tab delimiter (\$09\$) is assumed.

getwrite\$(userid\$,[detail[,delim\$]) returns the names of all libraries the specified user can write to. If detail is true (1), then lines of detail are returned; if not, only path names are returned. If delim\$ is not null, that is used as a template delimiter character; if it is null, a tab (\$09\$) delimiter is used.

putlib(library\$,prop\$) update library with template prop\$ (see getlib() for format). Admin permissions required. Note a library can't be renamed this way, only information properties such as category and description are updated.

The following code fragment illustrates how one could parse a library list.

```
o=new("libraries")
dim row$:o'gettemplate$(1)
rows$=o'getnames$(1)
x=pos($0a$=rows$)
while x>0
    row$=rows$(1,x-1)
    rows$=rows$(x+1)
    libname$=row.basename$
    libpath$=row.dirname$
    libcount=row.count
    x=pos($0a$=rows$)
wend
drop object o
```

15.3.22 library

library

```
o=new("library",libname$[,create])
```

The library object provides extensive access to the documents in a specific library, specified in libname\$, including document listings and manipulation of individual document and sub-image properties, as well as general library information. If the libname\$ provided is not found, a new library is created if the create field is true (1).

The library object enforces library security, and will generate appropriate errors if the active user login (specified with the setlogin() code block function) does not have rights to perform requested operations. Note that secure passwords can be configured in the browser interface and referenced in the setlogin() function.

Properties

All properties are read-only.

datecreated\$ is the date the library was created, in yyymmdd format.
forceseq is true (1) if sub IDs are auto-sequenced to prevent overwriting.
inactive is true (1) if the library is locked from further updates.
lasterrmsg\$ contains the last known error message related to methods.
libname\$ is the name supplied to the object.
pathname\$ is the full system path to the library.
permission\$ is the default permission setting for users that do not have specific permissions allocated. It is a semicolon-delimited list of "r", "w", or "d".
title\$ is the library title, or description.

Methods

candelede(userid\$) returns true (1) if the specified user can delete images or documents from the library.
canread(userid\$) returns true (1) if the specified user can read the library.
canwrite(userid\$) returns true (1) if the specified user can write to the library.
catcount() returns the number of category indexes in the library.
copydoc(doctype\$,docid\$,todotype\$,todocid\$[,library\$,[overwrite[,toobject]]) copies a document, identified by doctype\$ and docid\$, including all its subdocuments, to the specified todotype\$ and todocid\$. If library\$ is specified, the document is copied to a different library. Returns 1 if successful. Returns 0, and fills lasterrmsg\$, if not. The overwrite flag can be 0 to skip duplicates, 1 to overwrite duplicates (removes target first), 2 to compare update dates before replacing document and sub ID records, and 3 to do a full merge of properties and sub ID's. Note that unlike version 8.0, to overwrite a document when using this method, the overwrite flag must be supplied. If you will be copying a number of documents to the same library, you can create a library object for the target library, and supply that object in as the toobject value. Otherwise, a new target library object is created and destroyed for each copy operation.
copysubdoc(doctype\$,docid\$,subid\$,todotype\$,todocid\$,tosubid\$[,library\$]) copies a subdocument, identified by doctype\$, docid\$, and subid\$, including its image data, to the specified todotype\$, todocid\$, and tosubid\$. If library\$ is specified, the subdocument is copied to a different library. Returns 1 if successful. Returns 0, and fills lasterrmsg\$, if not.
countcats([seed\$]) returns the number of category indexes within the given seed. If no seed\$ is provided, a count of initial segments is returned. If seed\$ is provided, the count is the number of segments at the next level. Seed\$ is pipe-delimited. For example, countcats("Customers ByName") would return the number of third-level segments within Customers ByName.
countdocs(doctype\$[,idseed\$]) returns the number of documents in the specified document type. If an ID seed is provided, the count only includes doc ID's that start with the seed.
countdocsbycat(seed\$) returns the number of documents for the specified category index, which is in a pipe-delimited segment series format.
countdocsbydate(date\$) returns the number of documents for the specified date. Date\$ should be in yyymmdd format.

countdocsbykeyword(keyword\$) returns the number of documents indexed by the keyword specified.
countdocsbytype(date\$type\$,date\$) returns the number of documents for the specified type and date. Date should be in yyyyymmdd format.
countdocsbytypeupdated(type\$type\$,date\$) returns the number of documents for the specified type and updated date. Date should be in yyyyymmdd format.
countdocsbyupdated(date\$) returns the number of documents for the specified updated date. Date\$ should be in yyyyymmdd format.
countdocsbyentity(entityid\$) returns the number of documents for the given entity ID.
delcategory(doctype\$,docid\$,segs\$) removes the category segment defined in segs\$ from the document. The segs\$ value is a pipe-delimited list of category segments. The value is removed from the categories property and from the category index file.
deldoc(doctype\$,docid\$) removes the specified document, and any of the document's sub IDs , from the library.
deldocdata(doctype\$,docid\$,name\$) removes the docdata item name\$ from the document doctype\$, docid\$.
delkeyword(doctype\$,docid\$,keyword\$) removes the keyword and its index from the specified document.
dellink(doctype\$,docid\$,link\$) removes the link string from the specified document. Link strings can be urls, or pipe-delimited values of three or four values indicating library, doctype, docid, and optional subid.
delsubdoc(doctype\$,docid\$,subid\$) removes the specified subdocument from the library.
doccount() returns the number of documents in the library.
docdatacount() returns the number of document data records are in the library.
docdataexists(doctype\$,docid\$,name\$) returns true (1) if the docdata item name\$ is found in document doctype\$, docid\$.
docexists(doctype\$,docid\$) returns true (1) if the document type and ID exist in the library, false (0) otherwise.
<p>getcats([seed\$,first first\$,count[,descending]]) returns a linefeed delimited list of category indexes found in the library. Each index has pipe () segment delimiters. If seed\$ is provided, the list is returned within the prefix found in seed\$, which is a pipe-delimited list of segments. In this way, it is possible to get a list of indexes, sub-indexes, sub-sub-indexes, and so forth. The list starts at the record number specified by first, or the sub-segment value specified in first\$. Up to count records are returned. If descending is true (1), the list produced in descending order.</p> <p>getcats\$() returns a list of initial category segments.</p> <p>getcats\$("Customers") returns a list of second segments within the "Customers" initial segment.</p> <p>getcats\$("Customers ByName","B",1000) returns a list of up to 1000 category indexes within the Customers, ByName seed, listing values of the third segment, starting with "B".</p>
getcatsbyprefix(seed\$,prefix\$,first,count) returns a linefeed delimited list of category segments for the given category seed\$ and starting value. The list starts at <i>first</i> record, for <i>count</i> records. The seed value is a pipe-delimited list of segments that precede the desired segment the list is generated from. The prefix\$ value is the starting value of the desired segment, used to seed the starting point of that segment.

getcattypes\$(seed\$) returns a linefeed delimited list of document types for the given category index, specified by the a pipe-delimited list of category segments.

getdates\$([type\$],[year[,month]]) returns a linefeed delimited list of document dates found in the library. Dates are returned in yyyyymmdd format. If the year, or year and month, are provided, only dates for that period are returned. If type\$ is supplied, then dates within that document type are returned.

getdoc(doctype\$,docid\$,doc\$) fills the doc\$ property template with document properties, and returns 1 if successful. The template contains the following fields:

- doctype\$
- docid\$
- date\$ (yyyyymmdd)
- time\$ (hhmmss)
- dateupdated\$
- timeupdated\$
- title\$
- entityid\$
- notes\$
- keywords\$
- categories\$
- links\$

getdocdata(doctype\$,docid\$,name\$,value\$) fills value\$ with the docdata value for name\$, for the document doctype\$, docid\$.

getdocdatanames\$([first,count]) returns a list of document data names found in the library, either all names or the range specified.

getdocdata\$(doctype\$,docid\$[,name\$]) returns a tab-separated-values list of document data names and values for the specified document. If name\$ is supplied, the returned value is instead just the value of that docdata item. Name\$ is not case sensitive.

getimage(doctype\$,docid\$,subid\$,filename\$) extracts the image data to the specified filename\$. If filename\$ is null, then a temporary file is generated with an appropriate extension and returned in filename\$. The temporary file is automatically deleted with the job is complete. A user-supplied file name is not deleted. Returns 1 if successful. If not successful, lasterrmsg\$ will contain an error message.

getkeywords\$([seed\$],[first,count]) returns a list of *count* keywords, starting with index *first*. If *seed\$* is provided, keywords must start with the seed value. Keywords are case-insensitive.

getkeywordseeds\$([prefix\$],[length]) returns a list of unique keyword prefixes, up to *length*+1 characters. If *prefix\$* is supplied, only keyword prefixes starting with that value are returned. Keyword seeds are not keywords themselves, but keywords prefixes. Base keywords prefixes, with no *prefix\$* and a length of 1, would be a list of 1-character values that start all keywords (i.e. a, b, d, 0, 1, etc.).

getrange\$(doctype\$,first|first\$,count[,descending[,delim\$[,skip]]) returns a linefeed delimited list of document types and IDs within the specified document type. The list starts at the record number specified by first, or the document ID specified by first\$. Up to count records are returned. If descending is true (1), then the list is returned in descending order. If delim\$ is specified, the type and ID are delimited by that character. If no delim\$ is specified, then they are delimited by a tab (\$09\$). If skip is specified, that many records are skipped before starting the list.

An addition argument structure limits the list to a document type and ID values starting with a seed value.

getrange\$(doctype\$,first,count,descending,delim\$,idseed\$)

getrange\$(first,count[,order[,descending[,delim\$]]) returns a linefeed (\$0A\$) delimited list of document types and IDs. The list range is in sequence based on the value of order: 0=type/ID, 1=date/time created, 2=title. The range is specified with first and count, indicating the starting document, and the maximum number to return. If descending is true (1), then the list is returned in descending order. If delim\$ is specified, the type and ID are delimited by that character. If no delim\$ is specified, then they are delimited by a tab (\$09\$).

getrangebycat\$(seed\$,first,count[,descending[,delim\$]]) returns a linefeed delimited list of document types, IDs, and category indexes, in category index order, within the specified seed, which is provided as pipe-delimited segments. The list starts at the record number specified by first. Up to count records are returned. If descending is true (1), then the list is returned in descending order. If delim\$ is specified, the type, ID, and category index are delimited by that character. If no delim\$ is specified, then they are delimited by a tab (\$09\$).

getrangebydate\$(date\$,first|first\$,count[,descending[,delim\$]]) returns a linefeed delimited list of document types, IDs, and date/times, in date/time sequence, within the specified date, which is provided in yyyyymmdd format. The list starts at the record number specified by first, or the time specified by first\$ (in hhmmss format, 24 hour clock). Up to count records are returned. If descending is true (1), then the list is returned in descending order. If delim\$ is specified, the type, ID, and date/time are delimited by that character. If no delim\$ is specified, then they are delimited by a tab (\$09\$).

getrangebydocdata\$(name\$[,first[,count[,justvalues]]) returns a tab-separated-values list of doctype, docid, and docdata values for the given docdata name. The first and count values are used to get a range of records. If justvalues is true (non-0) then a range of values for the specified name, from the whole library, is returned.

getrangebydocdata\$(name\$,value\$[,first[,count]]) returns a tab-separated-values list of doctype and docids for the given docdata name and value. The first and count values are used to get a range of records.

getrangebyentity\$(entityid\$,first,count[,descending[,delim\$]]) returns a linefeed delimited list of document types and IDs for the given entity ID. The list starts at the record number specified by first. Up to count records are returned. If descending is true (1), then the list is returned in descending order. If delim\$ is specified, the type, ID, and category index are delimited by that character. If no delim\$ is specified, then they are delimited by a tab (\$09\$).

getrangebykeyword\$(keyword\$[,first,count]) returns a list of document types and IDs for the given keyword. If first and count are supplied, *count* records are returned, starting with index *first*.

getrangebytypedate\$(type\$,date\$,first,count[,descending[,delim\$]]) returns a linefeed delimited list of document types, IDs, and date/times, in type and date/time sequence, within the document type and specified date, which is provided in yyyyymmdd format. The list starts at the record number specified by first. Up to count records are returned. If descending is true (1), then the list is returned in descending order. If delim\$ is specified, the type, ID, and date/time are delimited by that character. If no delim\$ is specified, then they are delimited by a tab (\$09\$).

getrangebytypeupdated\$(type\$,date\$,first,count[,descending[,delim\$]]) returns a linefeed delimited list of document types, IDs, and updated date/times, in type and last updated date/time sequence, within the document type and specified date, which is provided in yyyyymmdd format. The list starts at the record number specified by first. Up to count records are returned. If descending is true (1), then the list is returned in descending order. If delim\$ is specified, the type, ID, and date/time are delimited by that character. If no delim\$ is specified, then they are delimited by a tab (\$09\$).

getrangebyupdated\$(date\$,first|first\$,count[,descending[,delim\$]]) returns a linefeed delimited list of document types, IDs, and updated date/times, in last updated date/time sequence, within the specified date, which is provided in yyyyymmdd format. The list starts at the record number specified by first, or the time specified by first\$ (in hhmmss format, 24 hour clock). Up to count records are returned. If descending is true (1), then the list is returned in descending order. If delim\$ is specified, the type,

ID, and date/time are delimited by that character. If no `delim$` is specified, then they are delimited by a tab (`09`).

getrecentdates\$([type\$,]count) returns a linefeed delimited list of the most recent dates in the library. The count value specifies the number of dates to return. Dates are returned in `yyyymmdd` format. If `type$` is supplied, then dates within that document type are returned.

getrecentupdated\$([type\$,]count) returns a linefeed delimited list of the most recent updated dates in the library. The count value specifies the number of dates to return. Dates are returned in `yyyymmdd` format. If `type$` is supplied, then dates within that document type are returned.

getsubdoc(doctype\$,docid\$,subid\$,subdoc\$) fills the `subdoc$` property template with subdocument properties, and returns a 1 if successful. You can reference properties as `subdoc.doctype$`, `subdoc.docid$`, `subdoc.subid$`, `subdoc.title$`, `subdoc.date$`, `subdoc.time$`, `subdoc.type$`, `subdoc.notes$`, and `subdoc.size`. The `subid` value can end in `"-<"` or `"->"` to return the first or last sub ID records, respectively, given a starting seed value that precedes the `-<` or `->` sequence. For example `scan->` will return the last sequence of sub ID values that start with "scan".

getsubids\$(doctype\$,docid\$[,notext[,currentid\$[,withdata]])] returns a linefeed (`$0A$`) delimited list of sub IDs on file for the given document. If `notext` is true (non-zero), don't return `@text*` subids. If `currentid$` is provided, don't return that id. If `withdata` is provided, a value of 1 returns a space delimited set of data for each, and if 2, returns a tab-delimited set of data for each. The detail is `subid`, title, date, type, and notes, where notes white space is normalized to spaces.

gettypes\$() returns a linefeed delimited list of document types found in the library.

getupdated\$([type\$],[year[,month]]) returns a linefeed delimited list of document update dates found in the library. Dates are returned in `yyyymmdd` format. If the year, or year and month, are provided, only dates for that period are returned. If `type$` is supplied, then dates within that document type are returned.

getyears\$([withmonths[,updated]]) returns a linefeed delimited list of years found in the library. If `withmonths` is true (1), years and months are returned, and if `updated` is true, the list is based on the last updated dates.

imgcount() returns the number of images (sub IDs) in the library.

kwcount() returns the number of keywords in the library.

moveNEXT(doctype\$,docid\$[,docprop\$]) returns information about the next document in sequence. `Doctype$`, `docid$`, and if supplied, the `docprop$` document properties template, are filled. See the `getdoc()` method for information about fields in `docprop$`. The method returns 1 if successful, or 0 otherwise (such as an end of file error). The starting position of a series of move commands can be set with the `moveto()` method.

moveNEXTcat(categories\$,doctype\$,docid\$[,docprop\$]) returns information about the next category and its associated document in sequence. `Categories$`, `doctype$`, `docid$`, and if supplied, the `docprop$` document properties template, are filled. `Categories$` is filled with a pipe-delimited list of category segments (note this is case-sensitive, returning segments as stored). See the `getdoc()` method for information about fields in `docprop$`. The method returns 1 if successful, or 0 otherwise (such as an end of file error). The starting position of a series of category move commands can be set with the `movetocat()` method.

moveprev(doctype\$,docid\$[,docprop\$]) returns information about the previous document in sequence. `Doctype$`, `docid$`, and if supplied, the `docprop$` document properties template, are filled. See the `getdoc()` method for information about fields in `docprop$`. The method returns 1 if successful, or 0 otherwise (such as an end of file error). The starting position of a series of move commands can be set with the `moveto()` method.

moveprevcat(categories\$,doctype\$,docid\$[,docprop\$]) returns information about the previous category and its associated document in sequence. Categories\$, doctype\$, docid\$, and if supplied, the docprop\$ document properties template, are filled. Categories\$ is filled with a pipe-delimited list of category segments (note this is case-sensitive, returning segments as stored). See the getdoc() method for information about fields in docprop\$. The method returns 1 if successful, or 0 otherwise (such as an end of file error). The starting position of a series of category move commands can be set with the movetocat() method.

moveto(doctype\$[,docid\$]) seeds the next movenext or moveprev methods to the position specified by doctype\$ and docid\$. To seed to the start of the library, provide a doctype\$ of null. To seed to the end, provide a high doctype\$, such as \$ff\$. Likewise, to seed to the end of a given document type, seed the docid\$ with a high value like \$ff\$.

movetodate(ymd\$[,hms\$]) seeds the next movenext or moveprev to navigate the library in date/time order, beginning at the supplied date and optional time. The date must be in yyyyymmdd format, such as "20091231". If the time is supplied, it must be in hhmmss format using a 24-hour clock, such as "150000" for 3:00 PM.

movetocat(seed\$) seeds the next movenextcat or moveprevcat methods to an index position. Seed\$ must be provided as a series of pipe-delimited category segments, such as "Vendors|000999|PurchaseOrders".

newdoc(doc\$) fills an empty document property template as doc\$. The property template fields are named in the getdoc method, above.

newsubdoc(subdoc\$) fills an empty subdoc property template as subdoc\$. The property template fields are named in the getsubdoc method, above.

putcategory(doctype\$,docid\$,segs\$) updates the category index specified in segs\$, which must be a pipe-delimited list of segments for a single category index. Note there can be more than one set of category segments per document, but this method is used to update a single segment set. The categories property is updated, as well as the category index file.

putdoc(doctype\$,docid\$,doc\$ [,retainupdated]) updates the specified document properties. If the document does not exist, it will be added. Note that this method does not add any image files to the library. The doc\$ template contains properties described in the getdoc method. The doctype\$ and docid\$ values override the type/id values in the doc\$ properties template. If docid\$ is null, it is assigned an automatic date-sequence ID value. If retainupdated is true (provided and non-0), and the dateupdated\$ property is not null, date and time updated values are retained.

putdocdata(doctype\$,docid\$,name\$,value\$) updates the specified docdata name\$ with value\$ for the document doctype\$, docid\$.

putkeyword(doctype\$,docid\$,keyword\$) adds the keyword and its index to the document specified.

putlink(doctype\$,docid\$,link\$) adds the link string to the specified document. Link strings can be urls, or pipe-delimited values of three or four values indicating library, doctype, docid, and optional subid. Link strings can contain references to names in the uf101d.ini [linksubs] section, using square brackets, such as "[myserver]/doc/1000/index.html", which at view time would replace "[myserver]" with the value of myserver=value in [linksubs].

putsubdoc(doctype\$,docid\$,subid\$,subdoc\$[,filename\$]) updates subdocument properties, and updates the image file as well, if filename\$ is supplied. If the specified subdocument does not exist, it is added. The subdoc\$ template contains properties as described in the getsubdoc method. The title, date, and time can be updated; other properties are ignored. Note that if you set the date and time values to null (""), they will be updated with the current date and time.

subdocexists(doctype\$,docid\$,subid\$) returns true (1) if the document type, ID, and sub ID exist in the library, false (0) otherwise.

15.3.23 lookups

```
lookupobj=new("lookups")
```

The lookups object is designed to run lookups to retrieve data from an external source, such as a SQL connection or an external API. When run, the lookup code should return a tab-separated-values list of data.

Lookup definitions are found in the im/lookups.ini and im/lookups.custom.ini files. Each lookup is in a section, headed by the name in brackets. Each section contains a description= line, and an optional parameters= list, a cols=list, and a value=name. The balance of the lookup section is code designed to set rows\$ to a tab-separated-values list matching the columns named in the cols=list.

The cols=list value contains a comma-separated list of column headers that should align with the rows\$ data returned by the lookup.

The value=name specifies which column has the actual data to be extracted when a user performs a lookup in the Image Manager.

Lookups can use parameters, passed into the running code as part of the name or as an array. Parameters can be passed to validations explicitly in an associative array param\$[all] (i.e. p\$["Val1"]="Value 1", passed as p\$[all]), or via a syntax convention in the name (i.e. verifyitem(Val1="Value 1",Val2="Value 2") - note quotes are optional if commas are not in values). Parameters are very important when dealing with potentially large sets of data, using them as filters when requesting data from external sources. If a named value is passed both via parenthesized argument and parameter array, the parameter array value is used.

Lookups are used by "lookup" fields in job definitions. When a lookup is performed by the Image Manager, it displays the result in a grid. It uses the cols= list to create columns for the grid, and displays the rows returned by the lookup in cells of the grid. When the user selects a row, it uses the column specified in the value= definition line to determine what data to paste into the associated data field.

If a doc object is present, it is possible for a lookup run to update document properties and fields via that object.

Properties

errmsg\$, lasterrmsg\$ is set by validation code to indicate an invalid value

doc contains an [inbounddoc](#) object when the validation is run in the context of an image manager job, or a [docflowdoc](#) object when run from a docflow job. It can also contain a [libdoc](#) object if run from actions in the archive browser interface.

Methods

Run(name\$,rows\$[,param\$[all]]) runs the lookup name\$. It expects the lookup to create rows\$ with tab-separated values that match the columns specified in the cols= line of the definition. Optionally pass a param\$[all] associative array with parameter names as keys, or use the name syntax described above. It may set errmsg\$ if an error occurs. The method returns true (1) if the errmsg\$ variable is null.

Run\$(name\$[,param\$[all]]) runs the lookup name\$. It returns the value of rows\$ created by the lookup's code, with tab-separated values that match the columns specified in the cols= line of the

definition. Optionally pass a param\$[all] associative array with parameter names as keys, or use the name syntax described above. It may set errmsg\$ if an error occurs. The method returns true (1) if the errmsg\$ variable is null.

15.3.24 mailattachment

```
o=new("mailattachment",file$,name$)
```

A mail attachment object represents a file attachment to an email. It is typically used as a child object of a message, as part of the attachments collection of a [mailmessage](#) object. When creating a mail attachment, you can supply a file and optional name, where the name is used by the recipient as a suggested file name. The file content itself is stored in a work file while the object exists.

charset\$	The character set of the attachment, usually for text-oriented attachments.
content\$	The binary content of the attachment.
contentid\$	The content ID, used with generating a MIME attachment for sending email. By specifying a content ID, an HTML message body can reference the attachment with a "cid:content/ID" reference, such as in a src= value of an HTML tag.
encoding	1 for base64 encoding, 2 for quoted-printable encoding
filename\$	When set, loads the specified file into the attachment.
name\$	The name used for MIME encoded headers, typically a suggested file name.
type\$	The MIME type of the attachment, such as text/plain or application/pdf.

loadfile(filename\$ [,type\$ [,encoding [,name\$ [,charset\$ [,contentid\$]]]]])	Loads the specified file into the attachment, optionally specifying additional parameters.
setcontent(value\$ [,type\$ [,encoding [,name\$ [,charset\$ [,contentid\$]]]]])	Loads the provide content value into the attachment, optionally specifying additional parameters.

15.3.25 mailmessage

```
o=new("mailmessage",to$,from$,subject$,body$,attach$,cc$,bcc$)
```

A mail message object represents an email message, including all its header and message content, as well as all its attachments. Attachments are found in the attachments property, which is an object collection object holding [mailattachment](#) objects. When creating a new mail message, all parameters are optional, but are positional. Properties can also be set once the object is created.

Mail message objects are applicable to both the mailreader and mailsender objects. A mailreader object creates message objects when it retrieves mail from a mail server. The mailsender object can send messages provided as mailmessage objects, so a new mailmessage can be created and populated, then sent, as an alternative technique to the email() or deliver() functions.

Properties

to	The "to" property is a memcollection object, which can be read using memcollection methods, or added to with the addto() method. Each item in the collection is either a simple address, or a "name" <address> string.
cc	The "cc" property is similar to the "to" property, but stores carbon copy addresses.
bcc	The "bcc" property is similar to the "cc" property, but stores blind carbon copy addresses.
from\$	Stores the From address, or "Name" <address> string.
replyto\$	Stores a "Reply-to" value, similar to a From address.
date\$	Stores a UTC date string in Internet standard format.
datetime	Stores a julian date/time value derived from the date\$ property.
subject\$	Stores the message subject.
body\$	Stores the message text or html content.
bodytype\$	Stores the MIME type of the message body.
bodycharset\$	Stores the character set of the message body.
retrec\$	Stores a return receipt email address.
importance\$	Stores the "importance" header information.
otherheaders	Stores additional headers. Use the addheader() method to add custom headers before sending a mailmessage.
attachments	An objcollection object that stores mailattachment objects associated with the message.
popid\$	When a mail message is read from a POP server, this is the unique ID assigned to the message by the mail server. This ID is used when the mailreader retrieves a message.
popsiz	The message size reported by the POP mail server.

Methods

createreply([replyhdr\$])	This method returns a new mailmessage object, created from the current object, with the To address being the Reply-to or From address of the current message. The message body is copied into the new message, with the specified replyhdr\$ value above it. If not supplied, a From, Date, and Subject reply header is generated. Also, the subject line in the new message is prefixed with "Re:".
createreplyall([replyhdr\$])	Like the createreply() method, except the reply is generated with all To and CC addresses copied to the new message.

createforward()	Like the createreply() method, but no To or CC addresses are generated.
addto(addr\$,name\$)	Adds the address and optional name to the the To collection.
addcc(addr\$,name\$)	Adds the address and optional name to the the Cc collection.
addbcc(addr\$,name\$)	Adds the address and optional name to the the Bcc collection.
addattachment(filename\$)	Adds the file specified to the attachments collection.
addheader(line\$)	Adds a header line, which must be "Header: Value" syntax.
addheader(name\$,value\$)	Adds a header line with the header name and value property formatted.
setfrom(addr\$,name\$)	Sets the from address to the address and optional name.
setreplyto(addr\$,name\$)	Sets a reply-to address.
setretrec(addr\$,name\$)	Sets a return receipt request address.
getheader\$(name\$)	Returns a header value of the specified name, or null if the name does not exist. The name is case-insensitive. The to, cc, and bcc headers return comma-separated addresses.
parsefile(filename\$)	Parses an email content file, which must be in RFC822 format, as is typically received by a mail client from a mail server. Such files are often stored in archive libraries with a "eml" file type. The current object instance data is populated from the file data. Returns 1 if successful, 0 otherwise.
parsesubid(library\$,doctype\$,docid\$,subid\$)	Parses email content from a stored archive record, which must be in RFC822 format, typically with a type of "eml". The current object instance data is populated from the file data. Returns 1 if successful, 0 otherwise.

15.3.26 mailreader

```
o=new("mailreader",server$,port,timeout,ssl,login$,password$)
```

The mailreader object provides functionality to read emails from a POP mail server. Email content is provided via "messages" object collection, with each object of that collection being a [message object](#). All parameters are optional, but are positional. Each value can also be set by assigning a property value, and defaults are defined in the email.ini file in the [mailreader] section.

When reading messages from a POP mail server, each message is given a unique ID by the server. This ID is provided in message lists as the popid\$ property of any given message object when messages are listed. The ID is used for message retrieval or deletion.

An example of using the mailreader object is found in the samples/emailarchive.rul file.

Properties

apop	Set to true (1) to use APOP security, or false to not. The default is true.
------	---

errmsg\$	The last error message.
logdetail	Set to 1 to enable detailed logging to logfile\$
logfile\$	Sets the log file name.
login\$	The POP user login.
messages	A collection object that holds the most recent collection of messages read. Read-only.
password\$	The POP user password.
port	The port to connect to, which defaults to 110 for standard POP, or 995 for SSL.
server\$	The POP mail server name or IP address
ssl	Set to 1 for POPS (POP or SSL protocol)
timeout	Timeout in seconds for server communication. Defaults to 30 seconds.

Methods

getstatus(count,size)	Requests status from the mail server, and fills in the count and size values with the number of pending messages and the total size in bytes of those messages.
getlist([toplines])	Fills the messages property with basic message content, including header information and up to <i>toplines</i> message lines. No attachments are downloaded. Returns the number of messages downloaded. Message objects each include the popid\$ property, which can be used to retrieve full message content.
getmsgs([msglist msgstr\$])	Fills the messages property with messages requested. If no list is provided, then all messages are downloaded. The message list can be either an object collection whose keys are message IDs (the popid\$ property from a message object), or a string of space-separated IDs. The message objects added to the messages collection are fully populated with data, including attachments.
delmsgs(msglist msgstr\$)	Removes the messages from the server. The message list can be either an object collection whose keys are message IDs, or a string of space-separated IDs.

15.3.27 mailsender

```
o=new("mailsender",server$,port,timeout,ssl,login$,password$)
```

The mailsender object provides functionality to send email constructed with [mailmessage](#) objects. Like the related email and deliver commands, and the email() code block function, it supports SMTP and SMTPS servers, with or without authentication.

Properties

errmsg\$	The last error message.
logdetail	Set to 1 to enable detailed logging to logfile\$
logfile\$	Sets the log file name.

login\$	The SMTP user login.
password\$	The SMTP user password.
port	The port to connect to, which defaults to 25 for standard SMTP, or 465 for SSL.
server\$	The SMTP mail server name or IP address
ssl	Set to 1 for SMTPS (SMTP over SSL protocol)
timeout	Timeout in seconds for server communication. Defaults to 30 seconds.

Methods

send(message[,errmsg\$])	Sends the email specified, and returns true (1) on success. If an error occurs, it returns false and sets errmsg\$ to an error message. The message parameter must be a mailmessage object .
sendeach(message[,errmsg\$])	Sends individual email messages to each To address specified in the message's "to" collection property. This differs from the send method, which leaves multiple To recipients as a list, allowing each recipient to see the others who were addressed. Internally, it creates a new message for each To address and uses the send method. If all sends are successful, the method returns true (1), otherwise false (0). The errmsg\$ value will contain the error message of the last send that failed.

15.3.28 marked

```
a=new("marked",[sessionid$])
```

The marked records object provides interaction with a browser sessions' marked records. A list of marked records is typically selected manually by the user while browsing document, and is maintained during the life of the user's browser session. The marked object allows adding to, deleting from, and navigating through the list of marked records.

A "marked" object is typically used in custom browser forms, when a user has marked one or more records and a related form rule set is designed to process that list. In addition, such a rule set can add or remove items from the marked record list. For example, it would be possible for a user to mark one record, then execute a form that would locate related documents and mark them as well.

Properties

sesid\$ is a read-only value that returns the current session ID for the browser interface user.

Methods

add(library\$,doctype\$,docid\$,subid\$) adds a record to the marked records list. Returns 1 if successful, 0 if not (for example, if the record already exists in the list).
count() returns the number of records in the session's marked records list.
delete(library\$,doctype\$,docid\$,subid\$) removes the record from the session's marked records. Returns 1 if successful, 0 if not.
getmarked\$() returns a list of all marked records as a string. The string is structured with linefeed (\$0A\$) delimited records, and tab (\$09\$) delimited fields. Each record consists of the library, doctype,

doc ID, and sub ID fields.
movefirst(library\$,doctype\$,docid\$,subid\$) fills the four arguments with values from the first marked record. The list is sorted in library, doc type, doc ID, and sub ID order.
movelast(library\$,doctype\$,docid\$,subid\$) fills the four arguments with values from the last marked record.
movenext(library\$,doctype\$,docid\$,subid\$) fills the four arguments from the next marked record in sequence. Returns 1 if successful, 0 if not (such as at the end of the list).
moveprev(library\$,doctype\$,docid\$,subid\$) fills the four arguments from the previous marked record in sequence. Returns 1 if successful, 0 if not.
moveto(library\$[,doctype\$[,docid\$[,subid\$]]]) moves the record position based on the values provided. Subsequent movenext and moveprev methods will be relative to this location.

15.3.29 memcollection

A memory collection object is like a [collection object](#), except:

- Stored in memory
- Key sizes are not limited
- No key navigation methods are available

15.3.30 notify

The notify object can be used to send messages to email addresses, users who have email addresses, or DocFlow role members who have email addresses. The object works with the notify.ini file (and notify.custom.ini) to generate email subject and message bodies based on named templates. Data can be supplied to replace expressions within the templates at the time a message is sent.

Note that notify.ini is replaced by updates, and as shipped contains sample notification settings. For user-defined notifications, create notify.custom.ini in the UnForm server installation directory.

The format of each section of notify.ini and notify.custom.ini is:

```
[msgid]
# comment
name=value
name2=value2
...
message line 1
message line 2
...
```

Comments are ignored. The first line that is not a name=value format begins the message content. Specific names that are important are subject, mimetype, cc, and bcc. All values and message lines support embedded expressions in the format {name\$} (note the \$ suffix is required in the expression), and they will be replaced with the *value*, from the list of names in the section, or of more benefit, name-value pairs supplied in a data\$ argument. *Data\$* can be either a string template or a line-feed delimited list of name=value pairs. For name=value pairs, the name doesn't require a \$ suffix, though it is allowed. For

string template-based expressions, you can specify the base variable name or `data.name$` in expressions.

You can also use object properties and methods from the `doc` object, if it has been assigned. Properties are referenced as `{doc'propertyname$}` for string properties, or `{str(doc'propertyname)}` for numeric properties. A method is invoked using parentheses, such as `{doc'geturl$()}`.

Note that when sending HTML email, data values may need to be entity-encoded if they will appear in message body content. If required, you can use the `entityencode()` function as part of the expression syntax: `{entityencode(name$)}` for example.

Emailing requires configuration of SMTP email settings in the server manager or configuration files.

notify=new("notify",[object])

Properties

doc is a object, initially assigned by the optional 'object' argument during instantiation, or set to an object identifier before executing any of the emailing methods. This object is typically a `inbounddoc` or `docflowdoc` object, and named based on this assumption, but it in fact can be any object with properties. **lasterrmsg\$** contains an error message if any of the methods fails.

Methods

email(to\$,msgid\$,data\$,cc\$,bcc\$,attachfile\$)) sends an email to the `to$` address specified (multiples separated by commas). The `notify.ini` section specified by `msgid$` is used to create the email content. The `data$` argument can be a string template or a linefeed-delimited list of `name=value` pairs. Each template variable or name can be referenced in an embedded string expression `{name$}` in the section's values and message content. If supplied, `attachfile$` should contain the name of a file to attach to the email, typically a PDF or image file.

emailuser(userid\$,msgid\$,data\$,cc\$,bcc\$,attachfile\$)) sends an email to the named UnForm user, as long as that user has an email address configured.

emailrole(roleid\$,msgid\$,data\$,cc\$,bcc\$,attachfile\$)) sends an email to the notify-enabled users in the DocFlow role specified, as long as each user has an email address configured.

15.3.31 objcollection

A object collection object is like a [collection object](#), except:

- Stored in memory
- Key sizes are not limited
- No key navigation methods are available
- All values should be objects, and when they are removed from the collection, the objects are dropped as well

15.3.32 rac

rac

```
a=new("rac")
```

Remote Access Control object for simplified document access from public users. The rac object allows creation and management of RAC codes, which are random codes linked to specific document images. RAC codes are very secure in that there are an extremely large number of possible codes (about $1.15e+77$ combinations), so the potential for guessing a valid code and gaining access to a document is extremely small.

A remote access URL is in the form `http://server/script?rac=code`, where the code is a 44-byte encoded string previously generated for a specific document image. These URL's can be emailed to users, who can open the link and view the single file associated with that document image. Such a link does not require a login or password, so it useful in cases where a site doesn't wish to manage external user logins to provide full archive library access. A code expires after a certain number of days (default defined with `racdays` in `uf101d.ini`), as specified in the rac object's `create$()` method.

If the `uf101d.ini` `[archive]` `external=` value is set to a path to a public access cgi script (or direct to the internal web server, though that is not typically done), then the browser interface document email screen can generate or delete an RAC url. Note if this is not configured, rule sets can still generate a code and a url, by providing a script prefix to the `geturl$()` method.

Methods

count() returns the number of documents in the RAC table.
create\$(lib\$,doctype\$,docid\$,subid\$[,validdays\$[,force\$]]) generates code for the given document image, valid for the number of days (default - <code>uf101d.ini</code>). If a code has been previously generated, that code is returned and a new expiration date is calculated. To force the generation of a new code, use a force argument of 1.
deldoc(lib\$,doctype\$,docid\$,subid\$) removes the RAC code, if any, associated with the given document. This allows a rule set to explicitly revoke remote access for a given document.
getcode\$(lib\$,doctype\$,docid\$,subid\$) returns the 44-byte code associated with the document image specified, or null if no code has been created.
getdoc(code\$,lib\$,doctype\$,docid\$,subid\$) given a 44-byte RAC code, this fills the associated lib\$, doctype\$, docid\$, and subid\$ values. It returns a 1 if the document is found, or a 0 if not.
getexpire\$(lib\$,doctype\$,docid\$,subid\$) returns the expiration of the RAC code associated with the given document. The format is <code>yyyymmdd</code> . If the document has no RAC code available, a null string is returned.
geturl\$(lib\$,doctype\$,docid\$,subid\$[,scriptpath\$]) returns full URL for a given document, which can be used to retrieve the document. If <code>scriptpath\$</code> is not supplied, then the 'external=' path is used from the <code>uf101d.ini</code> <code>[archive]</code> section. This script is the <code>http:</code> path used by a public user (not generally the internal http server) to access the archive server. Note this generally requires configuring a public-facing web server to use one of the CGI scripts supplied with UnForm.

15.3.33 search

search

```
o=new("search"[,searchid$])
```

The search object provides programmable control over the library search function offered by the browser interface and the command line `-arcsearch` option. By setting selected properties to values that follow the

search syntax requirements, and running the search, a document list object is created that provides access to the results of the search. In addition, the results can be placed on a user's search results page, so a rule set can execute a search on behalf of a user, and the results can be displayed in the browser interface.

If a `searchid$` is supplied, then the search object opens an existing search result, and the `doclist` object is attached to those results. This feature is useful when using custom browser forms driven from a search screen, as the search ID is provided and the search results the user generated can be accessed in a rule set. The search ID is the "sid" field value found in the URL of a search result screen, and is always a 10-digit value.

Each of the properties can be set to a wildcard (i.e. `"*value*" or "value*"`), an exact value (i.e. `"12345"`), a range (i.e. `"12/1/2007--12/31/2007"` - two dashes), or a regular expression with a tilde prefix (i.e. `"~[0-9][A-Z]"`). You can use "not" or "!" to look for archives that do not match a criteria, and "and" (or a semicolon) or "or" (or a comma), to search for multiple values or alternate values. All properties that are set must evaluate to true for a document to be added to the document list.

Searches are optimized when possible. The best optimizations are document IDs, entity IDs, small date ranges, and multi-level categories.

Note that document types and document IDs are case-sensitive when optimized.

Properties

docdata\$ specifies one or more name=value settings, separated by semi-colons (for AND logic) or commas (for OR logic).
docid\$ specifies document ID(s) to search for. Note that document IDs are case-sensitive.
doclist is a document list object that is filled with search results when the run method is executed.
doctype\$ specifies document type(s) to search. Note that document types are case-sensitive.
filter\$ is compared to all document properties.
library\$ contains one or more library names to search. Multiple libraries can be separated by semicolons.
recsread is a read-only property that stores the number of records read when the search was last run.
selected is a read-only property that stores the number of records selected and added to the document list object when the search was last run.
subids\$ searches for documents that contain the subid(s) specified.
text\$ searches the contents of the @text subdocument, if found, for the text\$ search criteria.
title\$, date\$, dateupdated\$, entityid\$, keywords\$, notes\$, categories\$, and links\$ specify criteria for a given document property. Dates can be specified in <code>yyyymmdd[hmmss]</code> format, or in local delimited date format, such as <code>"12/31/2009"</code> , where the mdy order is configured with <code>datefmt=xxx</code> in <code>uf101d.ini</code> .

Methods

makesearch(userid\$,description\$,errmsg\$) saves the search results in the user's search results table, so the browser interface will have access to the results. Optionally specify the description, or an error message, which will appear in the table.
--

run([append]) runs the search and fills the doclist object. If append is true (1), then the doclist object is not cleared before running. This allows multiple searches to produce a combined document list.

15.3.34 system

system

```
o=new("system")
```

The system object provides access to operating system information and services. Note that the system object is a static object, meaning there is only one object instantiated for any and all new() functions executed.

Properties

All properties are read-only.

home\$ is the path of the UnForm server's home directory.

networkid\$ is the server's hostname or IP address.

osname\$ is the specific operating system name, such as "MS-WINDOWS", or "UNIX-Linux-RedHat".

ostype\$ is "win" on Windows, and "*nix" on Unix or Linux-like systems.

user\$ is the user account running the UnForm server.

Methods

copyfile(from\$,to\$[,errmsg\$]) copies the from\$ file to the to\$ file. If an error occurs, such as a permission violation or if the to\$ file already exists, errmsg\$ is filled with a message. The function returns true (1) on success, false (0) otherwise.

deletefile(filename\$[,errmsg\$]) erases the specified file. It returns true (1) if successful, or false (0) and fills errmsg\$ if not.

getcmd\$(cmd\$[,output\$]) executes the specified command, captures its standard output, and returns the contents of the standard output. If output\$ is provided, it must be a file name that will also contain the standard output result.

newfile\$(filename\$[,pad [,content\$]]) creates a new file, with a sequencer before the file extension to ensure uniqueness. Directory paths are created if needed, then the file name is created if it does not exist. If it does exist, a sequencer is added until a unique name is created. If the pad option is supplied, the sequencer is a 0-filled value of *pad* digits. In the case of a pad value, the sequencer is always used. If pad is 0 or not supplied, and the initial file does not exist, the initial file without a sequencer is created. When used, the sequencer value starts with 1, and is always prefixed with a dash (i.e. /tmp/a/b-1.txt" or "/tmp/a/b-0001.txt"). If sequences exceed the capacity of the pad size, the pad size will be exceeded with extra digits as necessary.

If content\$ is supplied, that content is written to the file.

The method returns the filename created.

renamefile(from\$,to\$[,errmsg\$]) renames the from\$ file to the name specified in to\$. It returns true (1) if successful, or false (0) and fills errmsg\$ if not.

runcmd(cmd\$) executes the specified command and returns its exit code.

tempfile\$(**[extension\$]**,**[retain]**) creates a temporary work file that is erased at the end of the current job. The file name is returned. If **extension\$** is provided, the file will have that extension. Otherwise, a .tmp extension is used. The file must be opened to work with it. This function can be used to generate an empty file to use with other operating system commands. Normally work files are marked for deletion at the end of a job, but if the retain option is supplied and true (<>0) the file is not so marked.

tempfile(**[extension\$]**,**[retain]**) creates and opens a temporary work file that is erased at the end of the current job. The channel number of the opened file is returned. If **extension\$** is provided, the file will have that extension. Otherwise, a .tmp extension is used. Use **PTH(chan)** to obtain the file name. Normally work files are marked for deletion at the end of a job, but if the retain option is supplied and true (<>0) the file is not so marked.

winprtprinters\$() returns a linefeed (\$0A\$) delimited list of printer names available for *winprt* or *windev* printing on a Windows UnForm installation.

winprttrays\$(printer\$) returns a list of tray numbers and descriptions for the specified Windows printer. Windows print driver vendors often use user-defined tray numbers above 256, rather than tray numbers that match traditional PCL values. The list contains linefeed (\$0A\$) delimited tray lines, where each tray line contains a tab (\$09\$) delimited tray number and description.

15.3.35 textfile

textfile

```
o=new("textfile"[,filename$])
```

The textfile class provides line-oriented text file processing capabilities. Each line in the file is considered a record, and lines are separated by linefeeds (\$0A\$) on Unix-like systems, and carriage return-linefeed (\$0D0A\$) sequences on Windows systems.

If **filename\$** is provided, the file's existing content is loaded, or it is created if necessary. If no **filename\$** is provided, a temporary file is created that is erased when the object is destroyed.

Properties

All properties are read-only.

filename\$ contains the file name of the text file being managed.

lines contains the number of lines in the file.

found is an unkeyed collection object that holds the results of find operations. Each element of the collection is a matching line from the file.

Methods

append(row\$) appends the line **row\$** to the end of the file.

delline(line) removes the line specified from the file, shifting remaining rows up.

find(search\$[,nocase[,invert]]) initializes the found collection object, then fills it with text lines whose data contain the text **search\$**. If **nocase** is true (1), then the search is case-insensitive. If **invert** is true (1), then records that do not contain **search\$** are added, rather than those that do. The function returns the number of records found.

findreg(regex\$,nocase[,invert]) initializes the found collection object, then fills it with text lines whose data match the regular expression regex\$. If nocase is true (1), then the search is case-insensitive. If invert is true (1), then records that do not match regex\$ are added, rather than those that do. The function returns the number of records found.

findwhere(whereexpr\$,dlm\$,quotes) initializes the found collection, then searches records that match the where expression. For this search, records are assumed to be in delimited format, with the supplied dlm\$ value as the delimiter. If quotes is true (1), then fields are parsed assuming they may be quoted to protect delimiter values in field values.

The structure of the where expression is of a Boolean expression using fields, values, comparison operators, parentheses, and AND or OR, using *#number* syntax to represent field numbers. All fields are assumed to be string data, but you can use num() to convert strings to numbers, so long as the string is an unpunctuated numeric value. Here are some examples:

```
#2<>" " - field 2 not null
```

```
#2="100" and (num(#3)>=0 and num(#3)<10000) - field 2 is "100" and field 3 is between 0 and 10000
```

getline\$(line) returns the line number specified.

insline(line,row\$) inserts the line row\$ into the file at the line specified. Lines are 1-based, so insline(1,"text line") inserts "text line" at the beginning of the file. If line is more than the number of lines in the file, new lines are added to ensure the file has at least that many lines.

purge() removes all lines from the file.

putline(line,row\$) replaces the line specified with the contents in row\$. If the line exceeds the number of lines in the file, lines are added as necessary.

15.3.36 validations

validationobj=new("validations")

The validations object is designed to run validations against a text value. Both standard and custom validations can be defined in the Image Manager's Custom Code window. The goal of each validation is to analyze a text value and set an errmsg\$ variable if there is a problem with the value. The error message serves two purposes: to indicate the validity of the value, and to provide an information message to the user if the value is invalid.

Validation definitions are found in the im/validations.ini and im/validations.custom.ini files. Each validation is in a section, headed by the name in brackets. Each section contains a description= line, an optional parameters list, and an optional types list. The types list contains a comma-separated list of field types this validation can apply to, to provide context when looking up validations in job design. Types can include zone types ocr, bcd, grid, col, and field types text, note, number, date, list, and lookup. A validation can be run by code regardless of the types= list. The balance of the validation section is code designed to set errmsg\$ to some value if the value supplied is invalid.

Validations can use parameters, passed into the running code as part of the name or as an array. Parameters can be passed to validations explicitly in an associative array param\$[all] (i.e. p\$["Val1"] = "Value 1", passed as p\$[all]), or via a syntax convention in the name (i.e. verifyitem(Val1="Value

1", Val2="Value 2") - note quotes are optional if commas are not in values). If a named value is passed both via parenthesized argument and parameter array, the parameter array value is used.

If a doc object is present, it is possible for a validation run to update document properties and fields via that object.

Image Manager jobs use validations in various places, where the design calls for it. Custom code can also use validations, as the validations object is available to any code the UnForm server runs.

Properties

errmsg\$, lasterrmsg\$ is set by validation code to indicate an invalid value

doc contains an [inbounddoc](#) object when the validation is run in the context of a job. It can also contain a [libdoc](#) object if run from actions in the archive browser interface, or a [docflowdoc](#) object if run from a DocFlow script.

Methods

Run(name\$,value\$[,param\$[all]]) runs the validation name\$ with a value supplied. It expects the validation to analyze value\$ and set the errmsg\$ property if the value is not valid. Optionally pass a param\$[all] associative array with parameter names as keys, or use the name syntax described above. The method returns true (1) if the errmsg\$ variable is null.

Run(docobj,name\$,value\$[,param\$[all]]) runs the validation name\$ with a value supplied. The docobj variable should contain a inbounddoc, docflowdoc, or libdoc object. It expects the validation to analyze value\$ and set the errmsg\$ property if the value is not valid. Optionally pass a param\$[all] associative array with parameter names as keys, or use the name syntax described above. The method returns true (1) if the errmsg\$ variable is null.

15.3.37 webapi

webapi

```
o=new("webapi")
```

The webapi object is used to generate URL strings appropriate for browser interaction with UnForm's document management web interface. These strings can be used when constructing responses to the [UnForm Desktop Client](#) or the [Web Extension](#), or for email notification content.

URL strings are built with a protocol, server, and script path prefix (a "script prefix"), such as http://myserver:27402/arc. This prefix is derived automatically when the webapi object is instantiated by a rule set in response to a browser form or other web-based requests. However, if the object is instantiated outside this environment, such as by a print job, a script prefix must be created dynamically. At such times, the default value comes from the "external=value" setting in the [archive] section of uf101d.ini. It is also possible to override the value by passing a scriptpfx\$ string argument to the various methods.

This is important because the UnForm web server may need to be accessed in different ways depending on if the user is inside or outside the local network. In some cases, it may be necessary to create two links so the user can select which is appropriate for their location at the time the link is clicked.

If the scriptpfx\$ is passed as "", then it will always attempt to use the [archive] section external= value regardless of the context in which it is run. This can be useful in situations where UnForm is accessed via

a proxy server, so that the browser environment requires a different server and path than what the web server actually provides (the UnForm web server is serving the proxy server, not the browser).

Methods:

browseurl\$(lib\$,order\$,segments\$,scriptpfx\$) returns a URL to display a browse page in a particular order, optionally seeded with specific values based on the order. The segments are pipe-delimited. Segment dates must be in yyyyymmdd order, such as "20141231".

Valid orders are:

- type - segments\$ is *doctype*
- date - segments\$ is *yyyyymmdd*
- updated - segments\$ is *yyyyymmdd*
- type-date - segments\$ is *doctype|yyyyymmdd*
- type-updated - segments\$ is *doctype|yyyyymmdd*
- keyword - segments\$ is *keyword*
- docdata - segments\$ is *name|value*

Use the catlist\$() method for category browsing.

catlist\$(lib\$,segments\$,scriptpfx\$) returns a URL to display a category browse page, as if the user had clicked through category segments until a list of documents is displayed. Segments are pipe-delimited.

docflowdoc\$(flowid\$,doctype\$,docid\$,scriptpfx\$) returns a url to load the DocFlow browser interface and preselect the specified flow and document.

documenturl\$(lib\$,doctype\$,docid\$,subid\$,nowrapper[,scriptpfx\$]) returns a document URL. No subid will result in a document level view. A "@" subid will result in a conditional view, document if multiple sub ids are available, or @unform if just that is available. Otherwise, use a valid sub id. If nowrapper is 1, an image is shown without the standard UnForm document wrapper menus.

multiview\$(doclist\$,scriptpfx\$) returns a multi-view url, where documents can be viewed side by side. The doclist\$ variable contains a linefeed delimited list of document identifiers or urls in different formats:

Tab- or pipe-delimited document or image identification:

```
library|doctype|docid
library|doctype|docid|subid
```

Or a URL:

```
https://somewhere.com/images/1234.pdf
```

searchtpl(tpl\$) returns search template tpl\$ with the following fields:

- library\$
- doctype\$
- docid\$
- date\$ (in yyyyymmdd format)
- title\$
- keywords\$

- notes\$
- categories\$ (pipe-delimited segments)
- subids\$
- text\$
- links\$
- dateupdated\$ (in yyyyymmdd format)
- entityid\$
- filter\$
- docdata\$ (name=value pairs)

Populate these fields as you would in a browser search, and pass this template to the searchurl\$ method. Use of wildcards is supported with "*" or "?" characters. For multiple settings of a given field, you can use comma or semicolon delimiters to represent "or" or "and" logic, or use the words " or " or " and ".

searchurl\$(tpl\$[,scriptpfx\$]) returns search url with tpl\$ based criteria. Note lots of criteria could exceed http GET size limits, which are typically a few hundred characters.

webform\$(formname\$,lib\$[,doctype\$[,docid\$[,scriptofx\$]]]) displays the indicated web form, with the library and optional document type and doc id populated on the web form.

15.3.38 xmlreader

xmlreader

```
o=new("xmlreader"[,filename$|content$][,prepare])
```

The xmlreader class provides XML parsing capabilities to rule set code blocks. XML is a common file format for transmitting data between applications. It is a tag-based data format, where tags are used to identify data elements, as well as data attributes. Data is arranged in hierarchical fashion, where content contains other content, such as a report, which contains customers, which contain invoices, which contain shipping and order details, and so on.

The xmlreader navigates through this hierarchy using slash delimiters, similar to a file system directory, return data for a particular tag sequence. The data may be atomic data, a complete element, an element attribute, or it may be addition XML fragments that can be further parsed. For example, if the root element is "documents", and it contains a number of "document" elements, each of which contains an "invno" tag, you can navigate to an invoice number as "/documents/document/invno". If there are multiple occurrences of a particular element, you can get a count, and retrieve a particular element using a sequence number.

Elements with sequences can be referenced using a sequence number in many methods (referring to the last element tag), or can be referenced with [n] suffixes anywhere in the path chain. For example, "/documents/document[2]/invno" refers to the second document element's invno tag.

Element references can also be designed to match attributes or values, using one of two syntaxes:

- [attribute=value]
- [<tag>=value]

For example, `"/documents/document[OrderID=12345]/Address[<Type>=ShipTo]/City"` would locate the document with an attribute "OrderID" of "12345", then find the Address element of that document with a tag `<Type>` whose value was "ShipTo", then locate the Address element's City tag.

If a valid `filename$` value is supplied, the file is loaded when the object is first instantiated. Alternatively, the argument can be an XML document string (if the value doesn't exist as a file, it is assumed to be XML content). At any time, a new file can be loaded or the XML content provided directly as a string.

If the `preparse` option is supplied and true (not 0), the entire document is parsed into a keyed file structured, which is used for direct element retrieval (not for attribute or value searches). In cases where the document will be fully accessed during its lifetime, this can provide a significant performance improvement, depending on how frequently elements are retrieved, particularly those in deeply nested or large structures.

Properties

body\$ contains the non-header portion of the file, with any comments removed.
content\$ is the string representation of the XML data. When a file is loaded, <code>content\$</code> is filled with its data.
encoding\$ contains the character encoding of the file. The default encoding is utf-8, which is a compressed version of unicode, but files may be encoded using other character sets, such as unicode or iso8859-1. Since UnForm's normal character set is iso8859-1 (or the 9J symbol set), it is often necessary to specify a target encoding when retrieving values for UnForm printing.
filename\$ is the name of the most recently loaded file.
header\$ contains the header portion of the file, similar to <code>"<?xml version="1.0"?>"</code> .
root\$ contains the name of the root element. All XML documents have a root element that is the parent of all other elements.
version\$ contains the XML version number from the header.

Methods

<p>flatten(arr\$[all] [,nocase [,toencoding\$]]) fills the associative array <code>arr\$[all]</code> with a flattened (one dimensional) version of the xml content, where each key to the array is an element path starting from root, and the associated values are the most nested element values. Keys use <code>"[index]"</code> suffixes to indicate a 1-based repeating element array. Attributes are added to <code>arr\$</code> with keys suffixed with <code>"[attr\$]"</code>. Values are entity-decoded automatically.</p> <p>If <code>nocase</code> is true (non-0), all keys are forced to lowercase. In rare cases, this could result in overwriting of keys and lost data, since xml element names and attribute names are case sensitive.</p> <p>If <code>toencoding\$</code> is specified, values are converted to the defined encoding, such as "cp1252" or "8859-1".</p> <p>Note the order of elements in <code>arr\$</code> is undefined. An index loop will not return results in any particular order. Typically, loops look for the existence of a key in the array, and exit if the key is not found. Details about associative arrays can be found in the Basic Syntax chapter.</p> <p>See the example below.</p>
<p>getattr(element\$,sequence,attrname\$) returns the attribute value of the specified element. If a sequence is provided, the specified occurrence of the element is used.</p>

getattr\$(element\$,sequence) returns a list of element attributes, which are name-value pairs. Each pair is delimited by a linefeed (\$0A\$), and the name and value are delimited by a tab (\$09\$). If a sequence is provided, the specified occurrence of the element is used.

getchild\$(element\$,sequence,child) returns a specified child element (which may itself contain nested child elements). If the sequence is provided, then it applies to the selection of the parent element, not the child element. The child argument is used to determine which child element to return. `getchild$("/documents/document",5,1)` returns the 1st child element of the 5th documents/document element.

getchildname\$(element\$,sequence,child) returns a specified child element name. If the sequence is provided, then it applies to the selection of the parent element, not the child element. The child argument is used to determine which child element to return. `getchild$("/documents/document",5,1)` returns the 1st child element of the 5th documents/document element.

Note this method was added in 9.0.21 to assist with updates due to a former bug in the `getchild$()` method, which was returning a name, rather than the correct, and documented, full element. If you relied on `getchild$()` to return just a name, use `getchildname$()` instead.

getchildren\$(element\$,sequence) returns a linefeed (\$0A\$) delimited list of element names that are children of the element provided. If there are no children, then null is returned, and the value of the element is an atomic data value. If a sequence is provided, the specified occurrence of the element is used.

getchildren(element\$,sequence) returns the number of child elements that are found under the element. If a sequence is provided, the specified occurrence of the element is used. The sequence applies to the element, not the children. `getchildren("/documents/document",5)` returns the number of child elements found under the 5th document element under the documents root element.

getcount(element\$) returns the number of times a given element occurs. This can be used to determine how many sequences of a particular element are available.

getelement\$(element\$,sequence) returns element and enclosed content specified by element\$. If a sequence is provided, the specified occurrence of the element is used.

getns\$(element\$,sequence,uri\$) scans the attributes of an element for an XML namespace declaration that matches the uri\$ provided. It returns the namespace name, which can then be used to access elements that require a namespace prefix that associates a name with a URI.

getvalue\$(element\$,sequence[,toencoding\$[,entitydecode]]) returns the inner value of the specified element. If the element holds atomic data, the result is a string data value. Note that an element may contain sub-elements. If a sequence is provided, the specified occurrence of the element is used. If `toencoding$` is provided, then the value is translated from the XML document's encoding to the specified encoding. A common `toencoding$` value might be "iso8859-1" or "9j" to translate UnForm's default 8-bit text encoding.

If `entitydecode` is provided and true (non-0), the value is entity-decoded.

loadfile(filename\$) sets the XML content by reading filename\$

parseattribute\$(element\$,name\$) returns the entity-decoded value of the named attribute from the element string. If the attribute doesn't exist, null is returned.

parseelement(element\$,tag\$,value\$,attributes\$) parses the element string into components and fills the tag\$, value\$, and attributes\$ values with the element's tag name, value, and attribute string, which is a linefeed-delimited list of tab-delimited name-value pairs as found in the element's main tag. The attribute values are entity-decoded.

The value string can be empty, a text value, or a child element. Returns 1 on success, 0 on failure. The value is not entity-decoded, to retain child element encoding.
preparse() parses the current XML content into an internal table for faster element retrieval, at the cost of the initial parsing time, roughly linear to the number of elements in the entire document.
setContent(content\$) sets the XML content to content\$.

Example of flatten method

Given this xml:

```
<Rows>
  <Row id="row1">
    <Col>Val 1.1</Col>
    <Col>Val 1.2</Col>
  </Row>
  <Row id="row2">
    <Col>Val 2.1</Col>
    <Col>Val 2.2</Col>
    <Col>Val 2.3</Col>
  </Row>
</Rows>
```

These are the keys and values of the array, sorted for clarity:

```
Rows/Row[1][id]=row1
Rows/Row[1]/Col[1]=Val 1.1
Rows/Row[1]/Col[2]=Val 1.2
Rows/Row[2][id]=row2
Rows/Row[2]/Col[1]=Val 2.1
Rows/Row[2]/Col[2]=Val 2.2
Rows/Row[2]/Col[3]=Val 2.3
```

15.4 Internal Variables

Internal Variables

In addition to your own variables, UnForm provides a list of variables that you can use, or in many cases, set to a desired value.

across\$	Can be set to values described in the across command. Available only in prepage and precopy.
bin\$	Can be set to values described in the bin command. Available only in prepage and precopy.
cols\$	Can be set to values described in the cols command. Available only in prepage and precopy.

copies pcopies	Can be set to the number of copies to generate for a page. You can change this value to dynamically adjust the number of copies. If the number you specify is higher than the number specified by the rule set, then that highest defined copy's text and enhancements will be repeated until your specified copies are complete. This value is reset after each page to the rule set default, so you can't set it in the prejob routine. If you set pcopies, that is also honored like the pcopies command.
copy	Contains the current copy number in precopy. Generally you shouldn't modify this value. If you need to skip printing of a copy, use the skip variable instead.
coverset\$, coverfile\$, coverargs\$	If coverset\$ is set to a rule set name\$, a cover page is generated for the current document, using the rule set specified. Additionally, coverfile\$ can specify a rule file, if the rule set is not in the current rule file, and coverargs\$ can be set to command line arguments to use when running the subjob that generates the cover page. The cover page values must be set before the first page of the document is generated, so typically these values will be assigned in a prejob code block. However, if different cover page details are needed as the output device changes, you can set them in prepage or precopy code blocks.
crosshair\$	Can be set to "Y" or "y" to enable crosshair grid printing over the output (laser and PDF output only).
down\$	Can be set to values described in the down command. Available only in prepage and precopy.
driver\$	Stores the current driver as "laser", "ps", "pdf", or "zebra". The win and winpw drivers are considered variants of PDF, and driver\$ is set to "pdf" when used. This variable should not be changed.
duplex\$	Can be set to values described in the duplex command. Available only in prepage and precopy.
gs\$	Can be set to the values described in the gs command. Available only in prepage and precopy.
lcopies\$, ldarkness\$, lspeed\$	Set any of these values in a prepage code block to mimic the zcopies, zdarkness, or zspeed commands that can be used to control specific aspects of ZPL label printing.
margin\$	Can be set to values described in the margin command. Available only in prepage and precopy.
noarchive	Set to 1 in prejob to turn off all archiving for the job, or in prepage to turn off archiving of just the current page.
nocover	Set to 1 to turn off cover page generation, before the first page of a document.
nodeliver	Set to 1 to in prejob to turn off all deliver commands for the job, or in prepage to turn off deliver for just the current page.
noemail	Set to 1 to turn off the execution of the email command. Note: this does not affect the email() code block function or direct calls to mailcall.

nohpgl	Set to 1 in prejob to turn off HP/GL formatting, as if the -nohpgl command line argument had been used. This variable is ignored in code blocks other than prejob.
nooverlay	Set to 1 to suppress an AFO overlay on a page, in either prepage or precopy.
orientation\$	Can be set to "landscape", "portrait", "rlandscape", or "rportrait". It can also be set to a literal digit: "0"=portrait, "1"=landscape, "2"=reverse portrait, or "3"=reverse landscape.
outline\$	Can be set to an outline string used when the PDF outline feature is turned on, by use of the outline command. Multiple levels of outlines can be defined by delimiting levels with vertical bars, such as outline\$="Customer type "+get(1,6,4)+" Page "+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.
output\$	<p>In laser output, this can be changed in prejob, prepage, or precopy, and is tracked by copy. Set it to the device or file name desired for output on the server. If it changes for a given copy in the middle of a laser job, UnForm will close the prior output channel and reopen the new one. This can be used to send a copy to a different printer, or to a fax device. You can set the value to any file, or a pipe or redirect (Linux/Unix), such as ">vfx -n "+faxnum\$. When using a redirect or pipe, be sure to add quote characters (CHR(34)) around any data that might contain ampersands (&) or other shell-aware characters.</p> <p>For PDF output, you can set this value in the prejob code block to override any -o command line setting. Setting this value in any other code block is ignored.</p>
pagenum	Can be referenced as the current page number. The value should not be changed.
paper\$	Can be set to values described in the paper command. Available only in prepage and precopy.
rows\$	Can be set to values described in the rows command. Available only in prepage and precopy.
showimages	Can be set to a non-zero value to execute an images command even if the skip variable is true. This is only honored in the precopy code block.
skip	Can be set to a non-zero value in prepage or precopy, to skip printing of that page or copy, respectively. If set non-zero in prejob, the entire job is skipped.
text\$[all] textpage\$[all] textjob\$[all]	<p>Stores the text for the page as a one-dimensional array. For example, text\$[2] is the second line of text on the page. In prejob, it contains the content of the first page. In prepage and precopy, it contains the content of each page in sequence. You can use the array directly in code, or you can use the built in get(), mget(), set(), cut(), and mcut() functions to retrieve or manipulate its contents.</p> <p>Textpage\$[all] contains the text of the current page before any manipulations by code blocks.</p> <p>Textjob\$[all] contains the text of all pages in the job. Each line of each page, up through the last non-blank line of each page, is appended to the array when the input stream is initially parsed for the job. This array can be used in a prejob code block to analyze the full report content.</p>

	PostScript input not supported.	
tray\$	Can be set to values described in the tray command. Available only in prepage and precopy.	
uf.xxx\$	A string template or composite string that can provide access to many attributes of the UnForm environment and command line.	
	uf.afo2	Set to true (1) if the -afo2 command line option is used.
	uf.arcjob	Set to true (1) if the current job is a subjob for an archive command. This will be 0 otherwise.
	uf.arcenabled	Set to true (1) if archiving is licensed.
	uf.clientip\$	The IP address of the connecting uf101c client. In the case of the Unix perl-based client, an attempt is made to get the connecting terminal's IP address, whether by telnet or ssh. If the command is used, which may provide a name from /etc/hosts rather than an IP address, but it will still be locally resolvable. In the case of jobs submitted via direct TCP/IP ports, the IP address of the computer that connected to the server's listening raw port. If this is a server-based printer share, the IP address returned will be the server and not the originating computer.
	uf.clientoutput\$	The original -o argument value passed by the client to the server including any "client:" prefix. On Unix, if the client does not specify the option (or doesn't quote it properly), output goes to its standard output and this value is null.
	uf.cols	Columns for the current page.
	uf.copies	Copies defined for the job.
	uf.deljob	Set to true (1) if the current job is a subjob for a delivery command. This will be 0 otherwise.
	uf.dfrule\$	Default rule file from the environment.
	uf.driver\$	Driver for the current job.
	uf.dsrun	Set to true (1) if this execution is for a design tool preview. Use need to enable or disable processing when a rule set is running design tool.
	uf.emattach\$	Command-line --emattach value.
	uf.embcc\$	Command-line --embcc value.
	uf.emcc\$	Command-line --emcc value.
	uf.emfrom\$	Command line --emfrom value.
	uf.emlogin\$	Command line --emlogin value.
	uf.emmsgtxt\$	Command line --emmsgtxt value.
	uf.emoh\$	Command line --emoh value.
	uf.empswd\$	Command line --empswd value.
	uf.emsubject\$	Command line --emsubject value.

uf.emto\$	Command line –emto value.
uf.errfile\$	Command line –e file value (dynamically determined by the server).
uf.home\$	Home directory of the UnForm server.
uf.inputfile\$	Command line –i file value (dynamically determined by the server).
uf.job	Current job number.
uf.jobexecerr\$	If an error occurs while running a subjob with the jobexec() command, the error message will be in this variable.
uf.localclient\$	Path to the local uf101c client that resides on the server.
uf.login\$	Contains the login name from a -arclogin command option.
uf.maxdatacols	Maximum column with data on any page in the job.
uf.maxdatarows	Maximum row with data on any page in the job.
uf.maxpagecols	Maximum column with data in the current page.
uf.maxpagerows	Maximum row with data in the current page.
uf.maxpage	The maximum page number for the job. This value is calculated at the start of the job, and may be adjusted if pages are inserted or removed.
uf.model\$	Command line –m model value.
uf.outputfile\$	Command line –o file value. For server-based output, this is the option sent by the client. For client-based output, this is dynamically determined by the server.
uf.page	Number of input lines per page. Do not confuse this with the page variable, which holds the current page number.
uf.paper\$	Paper size name.
uf.parent	The job ID of the parent job when a subjob is running.
uf.pcopies	Pcopies defined for the job.
uf.pdfauthor\$	Command line –pdfauthor value.
uf.pdfkeywords\$	Command line –pdfkeywords value.
uf.pdfprotect\$	Command line –pdfprotect value.
uf.pdfsubject\$	Command line –pdfsubject value.
uf.pdftitle\$	Command line –pdftitle value.
uf.prm\$	Command line –prm value.
uf.rows	Rows for the current page.
uf.rulefile\$	Command line –f rule file value.
uf.ruleset\$	Selected rule set for the current job.
uf.shift	Horizontal shift value.
uf.subjob	Set to 1 (can be treated as a Boolean) if this is a sub-job executing the jobexec() function.

	<table> <tr> <td>uf.subst_file\$</td><td>Command line -s file value.</td></tr> <tr> <td>uf.version\$</td><td>The version of the UnForm server.</td></tr> <tr> <td>uf.vshift</td><td>Vertical shift value.</td></tr> <tr> <td>uf.warn\$</td><td>Job warning messages, delimited by line-feeds. For example, to set your own message: uf.warn\$=uf.warn\$+"My message"+chr(10).</td></tr> </table>	uf.subst_file\$	Command line -s file value.	uf.version\$	The version of the UnForm server.	uf.vshift	Vertical shift value.	uf.warn\$	Job warning messages, delimited by line-feeds. For example, to set your own message: uf.warn\$=uf.warn\$+"My message"+chr(10).
uf.subst_file\$	Command line -s file value.								
uf.version\$	The version of the UnForm server.								
uf.vshift	Vertical shift value.								
uf.warn\$	Job warning messages, delimited by line-feeds. For example, to set your own message: uf.warn\$=uf.warn\$+"My message"+chr(10).								
zcopies\$, zdarkness\$, zspeed\$	Set any of these values in a prepage code block to mimic the zcopies, zdarkness, or zspeed commands that can be used to control specific aspects of ZPL printing.								
%c, %s	These are global convenience variables that are an inifile object for uf101d.ini, and a system object, respectively.								
%linux, % windows, % unix	Global convenience variables to indicate the server operating environment. %windows will be true (1) on Windows, or false (0) on Linux environments. Likewise, %linux will be true (along with the %unix synonym) on Linux, false on Windows.								

15.5 Internal Functions

In addition to the intrinsic functions available in the run-time Business Basic engine, the most common of which are documented later in this chapter, UnForm provides a set of functions specific to its operating environment. Some functions are macros that perform an action, rather than return a value.

arrtostr(arr\$[all],str\$,dlm\$)	Converts array arr\$[all] to delimited string str\$, using dlm\$ as delimiter.
arrset(arr\$[all],col,row,cols,val\$)	Sets val\$ into the one-dimensional array at the position specified. If val\$ contains linefeeds, it is treated as a multi-row value, and the row increments for each additional row in val\$. The array is redimensioned as necessary to accommodate the row and position.
basename(file\$)	Returns the base name, without path information, from the file name specified. See also dirname().
bbxread(file\$,key\$,rec\$,errcode)	Executes an instance of BBx, configured with the bbpath=path line in uf101d.ini, and obtains the record specified by key\$ in file\$. If an error occurs in the BBx instance, it is returned in errcode. An errcode value of -1 indicates no error occurred. The variable rec\$ can be DIMed as a string template, but be sure to use '='10' to define field separators, as the default separator in the ProvideX engine is a hex 8A rather than the BBx default hex 0A. If it is not

	<p>defined as a template, the raw record data is returned and may be parsed.</p> <p>Here is an example:</p> <pre> prepage{ ky\$=get(65,5,6) dim rec\$:"id:c(5*=10), *:c(1*=10), ..., fax:c(9*=10)" bbxread("/u/data/CUSTOMER",ky\$,rec\$,ec) if ec=-1 then faxnum\$=rec.fax\$ } </pre>
catsegs(seg1\$[,seg2\$...])	Returns a pipe-delimited category index string, simplifying code-based library access to category indexes.
cdate(datestr\$ [,fmt\$])	<p>Converts a text date to a date (Julian) number, suitable for date compares, calculations, and use in the dte() function. The datestr\$ can be a string in delimited format, based on the fmt\$ string, or can be a simple string if digits in yyyyymmdd[hmmss] format. The format can be mdy, dmy, or ymd and a delimited date is parsed according to that order.</p> <p>The default date format is defined in uf101d.ini (datefmt=mdy).</p> <p>If a time is included, the value returned will include a fraction. Note that the dte() function requires an integer for the day portion, and the time portion, if provided, is an hours value:</p> <pre> x=cdate("12/31/2009 3:00 pm","mdy") x\$=dte(int(x),fpt(x)*24:"%Mz/%Dz/%Yz %Hz:%Mz") </pre>
clientenv(name\$)	Returns a client-side environment variable value.
cmtocols(centimeters) cmtorows(centimeters)	Returns columns or rows, given a centimeter measure.
cnum(expression)	Returns a number from a text string, after stripping formatting characters such as commas and dollar signs. Parentheses and minus signs indicate negative numbers. Use this function, rather than the intrinsic num() function, to convert text to numbers if the text can contain punctuation.
count(str\$,dlm\$)	Counts elements of a string, parsed by a delimiter.

countq(str\$,dlm\$)	Counts elements of a string, parsed by a delimiter, honoring quoted strings.
cstrans(text\$,fromcs\$,tocs\$)	Returns text\$ after translating it from one character set to another. Character set names include "uc", "utf16", or "utf-16" for unicode, or "utf8" or "utf-8" for UTF-8 encoding, or any of the character sets or symbol sets available in the UnForm unicode directory, such as 9j or 8859-1. The default character set is 9j, a pcl symbol set similar to ISO 8859-1.
cut(col,row,cols,value\$)	Returns the value text at position <i>col</i> , <i>row</i> , for <i>cols</i> columns, after setting the specified position to <i>value\$</i> . If <i>value\$</i> is null ("") or spaces, cut() effectively erases the text. This is useful for moving data in text commands, such as text 10,60, {cut(10,59,10,"")} , which would cut text from 10,59 and move it to 10,60. PostScript input not supported.
dataurlencode(fi\$) dataurldecode(str\$) dataurltype(str\$) dataurlext\$(str\$)	Return a file's contents as a data url, or decode a data url into a binary string value, the MIME type of a data url, or the file extension of a data url (based on the MIME type).
dbconnect(name\$ [,timeout[,errmsg\$]])	Connects to the database source identified by name\$. The support server configuration is used to define the names and associate them with data source connection strings. Typically done in a prejob code block. Requires the Windows Support Server. If executed as a function, returns 1 on success, or 0 on failure.
dbexecute(name\$, command\$, timeout, fdelim\$, rdelim\$, response\$[,errmsg\$])	Executes the SQL command cmd\$ and sets zero or more result rows in response\$. Columns are delimited by fdelim\$ (tab - chr(9) - by default). Rows are delimited by rdelim\$ (CR-LF - chr(13)+chr(10) - by default). Requires the Windows Support Server, and a previously successful dbconnect() function execution in the current job. If executed as a function, returns 1 on success, or 0 on failure.
deliver(filename\$,to\$,tags\$ [,response\$[,errmsg\$]]) □	This function delivers the filename to a destination by fax or email, depending on the to\$ value. The rule is simple: if to\$ contains an "@" character, it is emailed. Otherwise, it is faxed. Tags are in the format name=value[,name=value ...]. Delimit each name=value pair with either commas

	<p>or semicolons. The value can be quoted if it contains commas or semicolons. Tags are substituted with values in the fax/email configuration lines found in deliver.ini.</p> <p>The deliver.ini file contains configuration information for delivery methods. This file is documented in the Deliver Configuration chapter. See also the Deliver command, which simplifies subjob management for delivery while printing batches of documents.</p>
delpage(<i>n</i>)	<p>Command removes page <i>n</i>, pages <i>n</i>+1 to end are shifted down.</p> <p>PostScript input not supported.</p>
dirname(<i>file</i>%)	Returns the directory portion of the file name provided. The value returned uses forward slashes on all platforms, and does not include a trailing slash. See also <code>basename()</code> .
docidexists(<i>lib</i>%,<i>doctype</i>%,<i>docid</i>%)	Returns 1 if the document type and ID exists in the library, or 0 if not.
dtidel(<i>filename</i>%, <i>title</i>%, <i>userid</i>%, <i>ip</i>%, [,<i>style</i> [,<i>errmsg</i>%]])	<p>Deliver <i>filename</i>% to the <i>userid</i>% and/or IP address specified (the <i>ip</i> address can include a session suffix). The <i>title</i>% will be presented to the user. A <i>style</i> of 0 indicates deliver for optional viewing, and 1 indicates immediate popup. If an error occurs, a message will be returned in <i>errmsg</i>%. See the Desktop Delivery and Forms chapter for details.</p> <p>If the <i>filename</i>% parameter is a message starting with "msg:", it is treated as a popup message rather than a file. For example: "msg:Check the printer for invoices" would display a message window with the indicated text as the content.</p>
dtiform(<i>formname</i>%, <i>title</i>%, <i>userid</i>%, <i>ip</i>%, <i>datastr</i>%, <i>response</i> [,<i>timeout</i> [,<i>errmsg</i>%]])	Deliver the HTML form <i>formname</i> % to the <i>userid</i> % and/or IP address specified. The <i>ip</i> address can include a session suffix. The <i>title</i> % value is presented to the user for approval. The <i>datastr</i> % is a URL-encoded string that can be filled before the function is executed to provide default values for the form, and will contain URL-encoded data returned when the user submits the form. The <i>response</i> code will be 0 if the form was submitted, 1 if the user cancelled the form, 2 if the form timed out, or 3 if the user refused the form. The <i>timeout</i> can be specified to limit the amount of time the user has to accept the form (the default is 30 seconds). If an unexpected error occurs, a message is returned in <i>errmsg</i> %. See the Desktop Delivery and Forms chapter for details.

	<p>If timeout is set to -1, then the user will not be prompted to accept the form. Instead, the form will be displayed immediately. However, if the user is not present or the monitor is not running, the form will not be responded to and no timeout notification will occur. Instead, the job will wait until terminated.</p> <p>This option should only be used if the user is guaranteed to be present, such as a follow up form to one that was previously accepted and submitted immediately before.</p> <p>The <code>urlsetval</code> and <code>urlgetval</code> functions can be used to create and parse the <code>datastr\$</code> values.</p>
<code>email(to\$, from\$, subject\$, body\$, attach\$, cc\$, bcc\$, otherheaders\$, login\$, password\$,logfile\$)</code>	<p>Sends an email, assuming emailing is properly configured in the <code>mailcall.ini</code> file, using the information supplied. The arguments are positional but need not all be supplied. For example, <code>email(trim(get(81,1,40)), "info@acme.com", "Please review", messagebody\$)</code> will send a plain message to the address stored at column 81, row 1, for 40 characters in the current page. No attachment, carbon copy, etc. information will be used. A logfile option may be used to capture SMTP communication information, but should be unique to ensure two jobs do not try to use the same file at the same time, which would cause an error.</p> <p>As the arguments are positional, if you need to supply a login and password for the mail server to perform authentication, then all the arguments must be supplied, even if simply null (<code>""</code>). Note that this email function is different from the email command, in that the job itself is not sent, and multiple emails can be sent during the job stream within code blocks. This is useful, particularly in combination with the <code>jobstore</code> and <code>jobexec</code> functions, to develop batch email jobs.</p> <p>The logfile argument may contain substitution tags as described in the deliver command.</p>
<code>entityencode(str\$), entitydecode(str\$)</code>	<p>Two functions that return <code>str\$</code> with HTML or XML entities encoded or decoded. For example, in HTML, the "<" and ">" characters are meaningful for character markup, so to reference those characters literally, rather than as markup, they are encoded to "&lt;" and "&gt;", respectively. When reading or writing data from a HTML page or XML document, it may be necessary to use these functions.</p>
<code>env(name\$)</code>	<p>Returns the value of the operating system environment variable in <code>name\$</code>, or in a literal quoted string. Returns null (<code>""</code>) if the variable does not exist.</p>

err=next	May be used for any <i>err=label</i> option in any function or statement. Forces UnForm's error trapping to ignore an error. You may, of course, name your own <i>err=label</i> if desired.
exec(expression)	<p>Executes a barcode, bold, box, erase, font, image, italic, light, micr, move, shade, or text command from within the code block. <i>Expression</i> 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 precoppy{} blocks. Remember that some commands need quoted parameters to work properly. For example, if you exec() a text command, be sure to add quote characters around the text to be printed, using one of three methods: double any internal quotes, use an expression that uses \$22\$, chr(34), or QUO for quotes</p> <p>For example, exec("text 10,10," + chr(34) + message\$ + chr(34) + ",cgtimes,10")</p> <p>Any linefeed characters in the string are treated as command delimiters. Passing multiple commands in one exec() can improve performance, perhaps noticeably if there are many exec's being performed in a job. Be sure if a text value contains linefeeds that are intended to print multiple lines that you substitute linefeed characters with the mnemonic "\n" sequence. Note that the mget() function can return linefeed-delimited data.</p>
exists(file\$)	Returns 1 (true) if the file path specified exists, 0 (false) otherwise.
fileext(file\$)	Returns the extension of file name provided, without the leading "." . The extension is the segment of the name after the last "." .
finddata(text\$[all], search\$, coloffset, rowoffset, columns, result\$[all])	<p>Searches the text array for a search string, and returns each item found in the result\$ array. The data returned is based on the locations found, offset by the coloffset and rowoffset values, and the length specified. Each result is trimmed of leading and trailing spaces.</p> <p>The function returns the number of positions found, and the result\$ array is indexed from 0 to the number of positions minus 1. Result\$[0] is the first item found, result\$[1] is the second, and so on.</p> <p>This function is useful for report mining, where specific types of rows contain the data desired. See the findpos() function for a description of how the search\$ string is interpreted.</p>

findpos(text\$[all],search\$,result[all])	<p>Searches the text array for a search string, and returns the number of locations found. Each location found is returned in the response[all] array. The result array is structured as follows:</p> <p>result[0,0]=first column result[0,1]=first row result[1,0]=second column result[1,1]=second row</p> <p>The first dimension contains each position found, from 0 through the number of positions minus 1. The second dimension contains column numbers in element 0, and row numbers in element 1.</p> <p>The search string can be a simple string, or a tilde (~) followed by a regular expression. If the string contains a "@" character, then two, three, or four comma-separated digits should follow, indicating the starting column,row, and ending column,row to search in the text array. If the ending row is not supplied, search ends at the last row in the array. If the ending column is not supplied, all columns from the first column are searched.</p> <p>"Total@50,45,54,66" will search for the word "Total" in columns 50 through 54, rows 45 through 66 in the array.</p>
fitpage(pagenum [,"size" [,"charset"]])	<p>Re-scales the specified page to the size specified, or the current paper size if not supplied. The page overlay and text stream are both updated to reflect the new size. If supplied, the character set is used when parsing the text extraction data. It defaults to ISO-8859-1.</p> <p>AFO jobs only.</p>
formatnumbers(lines\$,format\$[,dlim\$])	<p>Formats number values in the lines, using a format mask. The default delimiter is a linefeed (\$0a\$ or chr(10)). A literal \n is also interpreted as a linefeed, in both lines\$ and dlim\$, and if present in lines\$ will result in a response that contains \n rather than linefeeds. The mask uses \$, #, 0, -, . characters. Example: formatnumbers(lines\$,"\$#,###,###.00-").</p>
fromuc(text\$,charset\$)	<p>Converts unicode text to single-byte text in the character set specified. Characters that are not present in the character set are replaced with a "?". Returns the single-byte text. Known character set tables are specified in the UnForm unicode directory.</p> <p>Also, "utf8" or "utf-8" can be used to convert UTF-16 (Unicode) format to UTF-8, a common encoding for XML or HTML data.</p>
get(col,row,cols)	<p>Returns text from the text\$[all] array, without substring or array out-of-bounds errors.</p>

get(col,row,cols,trim\$)	Same as get(), but with a trim "Y" or "N" option.
get(col,row,cols,trim\$,page)	Same as get(), but with a trim "Y" or "N" option, and a page number option to retrieve information from any page of the job.
get(col,row,cols,trim)	Same as get(), but with a Boolean trim (0 or non-0) option.
get(col,row,cols,trim,page)	Same as get(), but with a Boolean trim (0 or non-0) option and a page number option.
getaddress(book\$,entityid\$,doctype\$,address\$)	<p>Opens the address book specified, and fills the address\$ template with data from the address book entry for entityid\$ and doctype\$. The address book template can be referenced as address.entityid\$, address.doctype\$, address.entityname\$, address.contactname\$, address.sendto\$, and address.combine.</p> <p>The function returns 1 if successful, or 0 if not.</p>
getarc(lib\$,doctype\$,docid\$,subid\$,filename\$ [,errmsg\$])	Retrieve an archive image to a user-specified or temporary file.
getcolumn(rows\$,column[,first[,count[,fdelim\$[,rdelim\$]]]]) <input type="checkbox"/>	<p>Slices a column from a block of delimited rows, returns fields delimited by \$0A\$. fdelim\$ defaults to \$09\$, rdelim\$ to \$0A\$. If first and count are supplied, slice begins at row "first" and continues for "count" rows. This provides easy pagination capabilities.</p> <p>If count=0, slice continues to last row.</p>
getdocidprop(lib\$,doctype\$,docid\$,prop\$)	<p>Sets prop\$ to a composite string containing properties about the document specified. These properties include:</p> <p>Prop.date\$ - date in yyyyymmdd format Prop.time\$ - time in hhmmss format (24-hour clock) Prop.title\$ - title string Prop.entityid\$ - entity id string Prop.notes\$ - notes, which can have CRLF line breaks Prop.keywords\$ - semi-colon delimited keywords Prop.categories\$ - semi-colon delimited categories with pipe-delimited segments Prop.links\$ - semi-colon delimited list of links</p> <p>If the document type and ID does not exist in the library, each of the fields in the composite string will</p>

	be empty. Use the docidexists() function to determine if a document exists.
getfile(filename\$)	Returns contents of filename\$ as a string. Can be used to load a file into a string.
getfilefield(filename\$,key\$,field) getfilefield(filename\$,key\$,field,dlm\$,quoted) getfilerec(filename\$,key\$) getfilerec(filename\$,key\$,dlm\$,quoted)	<p>Returns a record or field from a text file, given a key that matches the first field in each record. The dlm\$ field is a field delimiter, such as "," or chr(9) for comma or tab delimiters, and the quoted field is a Boolean (0=false, non-0=true) that indicates fields may be quoted, as would be the case in a csv file. If no matching key is provided, the functions return an empty string. If no dlm\$ and quoted parameter is supplied, then a comma-separated-value format is presumed (dlm\$=",", quoted=1).</p> <p>These functions provide an efficient way of providing data to UnForm from applications. For example, an application could export customer IDs and email addresses, and UnForm could lookup addresses by customer ID.</p> <p>Files are parsed once and cached until they change, so subsequent retrievals are very fast. Caching is permanent (across jobs).</p> <p>Keys are limited to 127 bytes, so the first column must be limited accordingly.</p> <p>In a quoted file, fields that contain a quote character must escape that character with a backslash, like "Board - 1' 2\" length".</p>
getinival(filename\$,section\$[,name\$])	Returns the section or, if name\$ is supplied, the value of the name in the section specified, of the .ini formatted file specified. .ini files are organized in to sections via [name] headers, and lines within the section contain name=value pairs. When a full section is returned, each line is delimited by a linefeed character (chr(10) or \$0a\$). This can be useful in cases where data is stored in .ini file format and UnForm needs to access it. If filename\$ doesn't exist, it is treated as ini file content.
getpage(n,arr\$[all])	<p>Fills text array arr\$[all] with page n data lines.</p> <p>PostScript input not supported.</p>
getpaircount(values\$ [,delim\$])	Returns the number of delimited pairs in value\$. For the string "id=00100,name=Acme Corp", the count would be two. The default delimiter is a comma.
getpairvalue(values\$,number[,delim\$])	Returns the value of the specific name=value pair in the values\$ string. Pairs are delimited by commas

	<p>unless delim\$ is specified. The first pair is number 1, the second is number 2.</p> <p>getpairvalue("id=00100,name=Acme Corp",2) returns Acme Corp.</p>
getpairvalue(values\$,name\$ [,delim\$] [,casesensitive])	<p>Returns the value associated with the name from the delimited list of pairs in values\$. The default delimiter is a comma. If casesensitive is true (1), then the name must match exactly. If the name is not found in values\$, then null is returned.</p> <p>getpairvalue("id=00100,name=Acme Corp","name") returns Acme Corp.</p>
getpatternvalue(pattern1\$,pattern2\$,array\$[][,erasepat,[includepat]])	<p>Searches each element of the one-dimensional array for text that starts with pattern1\$ and ends with pattern2\$. If either is null, returns from the beginning or end of the line. If either starts with ~, balance is a regular expression (use \~ to enable ~ as a search character).</p> <p>The text data found is returned, each element separated by a linefeed (\$OA\$), optionally with the pattern(s) included if includepat is true (non-zero).</p> <p>If erasepat is true (non-zero) then the pattern and text found is removed from the line on which it is found.</p> <p>Only the first match in each array element is returned.</p>
getpatternvalue(pattern1\$,pattern2\$,searchtext\$ [,erasepat,[includepat]])	<p>Searches searchtext\$ for text that starts with pattern1\$ and ends with pattern2\$. If either is null, returns from the beginning or end of the line. If either starts with ~, balance is a regular expression (use \~ to enable ~ as a search character).</p> <p>The text data found is returned, optionally with the pattern(s) included if includepat is true (non-zero).</p> <p>If erasepat is true (non-zero) then the pattern and text found is removed from the line on which it is found. Note searchtext\$ must be passed as a variable, rather than a literal or expression, for this to work.</p> <p>Only the first match in searchtext\$ is returned.</p>
getppdval(name\$,option\$)	<p>Returns a value from the PPD file associated with the job, either a default file selected by the -p driver command line option, or one explicitly named with a -m command line option. PPD files are generally used by PostScript printers to define command</p>

	sequences for settings like duplex, bin, and tray selection. The laser driver can also use a custom PPD file for defining PCL sequences for various printer options. This function can be used to retrieve control sequences for use in boj, eoj, bop, or eop values.
getsubids(<i>lib\$,doctype\$,docid\$[,dlim\$]</i>)	Returns a list of document archive sub ID's, delimited by linefeeds or by the specified delimiter.
gettrans()	Returns the active translation file.
getuserprop(<i>userid\$,prop\$</i>)	<p>This function fills the template variable prop\$ with attributes for the specified user. It returns 1 on success, or 0 if there is an error. If there is an error, the template contains empty fields. The template fields can be referenced as:</p> <p>Prop.username\$ Prop.email\$ Prop.entityid\$ Prop.companyname\$ Prop.telephone\$ Prop.groups\$ (semicolon-delimited groups the user is a member of)</p>
gproperty(<i>name\$</i>)	Returns the DSC structured comment value of the given name from a AFO print stream. The most common use for this is to obtain a document title from the PostScript data, such as title\$=gproperty("Title"). Comments can be found in the header and trailer of the PostScript data.
gtextcount(<i>page</i>)	<p>Returns the number of text elements found on the specified page.</p> <p>This function is valid only for AFO jobs.</p>
gtextitem(<i>page,item,text\$[,col [,row [,cols [,rows]]]]</i>)	<p>Fills text\$ with the specified text element identified by page and item, where item can range from 1 to the number of text elements on the page. If supplied, will fill col, row, cols, and rows variables with the position and size of the text item.</p> <p>This function is valid only for AFO jobs.</p>
gtextfind(<i>page, pattern\$, txt\$[all], rects[all]</i>)	Scans the specified page for text elements matching the specified pattern. For each element selected, the text is added to txt\$[n], and the position and size of the text is added to rects[n,0-3], where 0 is the column, 1 is the row, 2 is the columns, and 3 is the rows. n is 1-based, so the first array element is txt\$[1], and rects[1,0]. In addition, the number of elements is supplied in rects[0,0].

	<p>Pattern\$ can be a simple text phrase, which must be contained in a text element to be selected, or it may be in the format "<i>~regex</i>" to search for a regular expression. In addition, it may have a <i>@col1,row1,col2,row2</i> suffix to limit the search to text elements enclosed by the specified rectangle. To search for a value that includes a literal "@", use "\@".</p> <p>The function returns the number of elements selected, which can be used to as the range of valid elements placed in txt\$[all] and rects[all]. If the function returns 2, then txt\$[1] and txt\$[2] contain the first and second text elements selected.</p> <p>This function is valid only for AFO jobs.</p>
gtextword(<i>page,item</i>)	<p>Returns the item specified, generally a word or phrase, from an AFO job print stream. The item can be any number from 1 to the number of elements on the page (which can be determined with the gtextcount() function, above. If the page/item does not exist (out of range, for example), the function returns null ("").</p> <p>This function is valid only for AFO jobs.</p>
gtextwords(<i>page, pattern\$, direction\$, col, row, cols, rows ,[filter\$ [,instance [,rects[all]]]]</i>)	<p>Returns a list of words found in a rectangular region relative to a search pattern. Words from the same line are space-separated, with linefeeds (\$OA\$) delimiting rows.</p> <p>Pattern\$ can be a simple text phrase, which must be contained in a text element to be selected, or it may be in the format "<i>~regex</i>" to search for a regular expression. In addition, it may have a <i>@col1,row1,col2,row2</i> suffix to limit the search to text elements enclosed by the specified rectangle. To search for a value that includes a literal "@", use "\@".</p> <p>The direction value must be u, up, d, down, l, left, r, or right. This determines the direction from a found pattern to scan for words.</p> <p>The col and row values are offsets from the found pattern, and cols and rows values are dimensions from which to gather words.</p> <p>Filter\$ can be a text string or regular expression prefixed with "~". Only words that include the text or match the regular expression are returned.</p> <p>Instance is the occurrence of the search pattern on the page, if there is more than one. It defaults to 1.</p>

	<p>The <code>rects[all]</code> array will receive positions of the words in the same format as the <code>rects[all]</code> array returned by <code>gtextfind()</code>.</p> <p>This function is valid only for AEO jobs.</p>
<code>gtextwords(page, pattern\$, wordoffset, wordcount, [filter\$ [instance [rects[all]]]])</code>	<p>Returns a space-separated list of words found adjacent or near a search pattern.</p> <p>Pattern\$ can be a simple text phrase, which must be contained in a text element to be selected, or it may be in the format "<code>~regex</code>" to search for a regular expression. In addition, it may have a <code>@col1,row1,col2,row2</code> suffix to limit the search to text elements enclosed by the specified rectangle. To search for a value that includes a literal "@", use "<code>\@</code>".</p> <p>The wordoffset and wordcount values indicate a relative word offset (positive or negative), and the number of words to return</p> <p>Filter\$ can be a text string or regular expression prefixed with "~". Only words that include the text or match the regular expression are returned.</p> <p>Instance is the occurrence of the search pattern on the page, if there is more than one. It defaults to 1.</p> <p>The <code>rects[all]</code> array will receive positions of the words in the same format as the <code>rects[all]</code> array returned by <code>gtextfind()</code>.</p> <p>This function is valid only for AEO jobs.</p>
<code>gtextwords(page, pattern\$, [instance [cutdef\$ [rects[all]]]])</code>	<p>Returns a list of words matching a search pattern, optionally trimmed of specific data. Words from the same line are space-separated, with linefeeds (<code>\$OA\$</code>) delimiting rows.</p> <p>Pattern\$ can be a simple text phrase, which must be contained in a text element to be selected, or it may be in the format "<code>~regex</code>" to search for a regular expression. In addition, it may have a <code>@col1,row1,col2,row2</code> suffix to limit the search to text elements enclosed by the specified rectangle. To search for a value that includes a literal "@", use "<code>\@</code>".</p> <p>Instance is the occurrence of the search pattern on the page, if there is more than one. It defaults to 1.</p>

	<p>Cutdef\$ can contain the word "cut" plus optional letters indicating what to cut from the words. "Cut" by itself removes the matched pattern from the words. "Cut L" removes characters to the left of the match. "Cut R" removes characters to the right of the match. "Cut LM" or "Cut RM" remove to the left or right, plus the match itself.</p> <p>The rects[all] array will receive positions of the words in the same format as the rects[all] array returned by gtextfind().</p> <p>This function is valid only for AEQ jobs.</p>
gtextwordsexec()	<p>This is a utility function related to the various gtextwords(...) functions. Whenever a gtextwords(...) function executes, it builds an exec() able string that highlights words found with a series of box commands. Immediately after gtextwords(), capture this value and exec() it to highlight words. Note that the page number specified in the gtextwords() function should match the page on which the exec() function executes.</p>
ifn(boolean,true,false) ifs(boolean,true\$,false\$)	<p>Return the true or false number, or true\$ or false\$ string, based on the boolean value or expression, where 0 is false and non-0 is true. The ifn() function is used when returning a number, and ifs() when returning a string.</p> <p>ifs(cust\$>"", cust\$, "n/a") would return the cust\$ value if it is non-null, or "n/a" if it is.</p>
imgx(imagefile\$,units) imgy(imagefile\$,units)	<p>These functions to return image x and y dimensions, helpful when trying to scale an image to actual size. Units indicate what is returned: 0=pixels, 1=inches, 2=cols/rows.</p> <p>The function supports jpg, bmp, tif, and png formats. If the file can't be opened or parsed, 0 is returned.</p>
inctocols(inches) inchtorows(inches)	<p>Return columns or rows, given a measurement in inches.</p>
inspage(n,arr\$[all])	<p>Inserts text array arr\$[all] as page n, shifting existing pages as necessary. If n is any number greater than the highest page number, or -1, a page is appended (i.e. inspage(999,x\$[all]) will add page 3 to a 2-page job.</p> <p>PostScript input not supported.</p>

iszonepagehdr(str\$)	Returns true (1) if the string is a page header from a grid zone or column, false (0) otherwise. Normally used when parsing linefeed delimited ocr column data, to skip page header rows.
jobclose(id\$...)	Closes and erases the temporary storage file associated with id\$. Open jobs are all automatically closed at the end of the primary job.
jobids(dlm\$)	Returns a list of job ID values that are active, separated by the delimiter character specified. Any ID supplied in a previous jobstore() command and not closed with a jobclose() command will be returned.
jobexec(id\$,output\$,driver\$,argstring\$ [,async])	<p>Executes a sub-UnForm job using the parameters given. The id\$ identifies a job with one or more pages previously stored with the jobstore() function. The output\$ value defines where the sub-job's output should go. This can be a file name, like "/archive/"+invoice\$+".pdf", a device name, like "/printsrv/hp4000", or a pipe/redirect, like ">lp -dhp4000 -oraw". The driver\$ argument can be set to one of the -p drivers supported by UnForm, such as laser or PDF. The argstring\$ contains any additional command line parameters you wish to add to the sub-job command line. You can use any parameter supported by the uf101c client, though the -i, -o, and -p options are specified using the other three function arguments.</p> <p>A rule set can check uf.subjob, as "if uf.subjob" or "if uf.subjob=1", to test if an instance is running from a jobexec() function.</p> <p>The optional <i>async</i> flag can be set to a non-zero value to force the job to be executed asynchronously, so the jobexec() command returns as soon as the subjob is queued. This operates via the rpq directory, which means the subjob will execute within a few seconds, as long as there is a job license available. If not, the job will remain queued until a license is free. Note that if you use this flag, the main job must be able to operate without the subjob output. For example, if the main job is designed to email or fax a result of a subjob, it will fail. Such processing should be moved into the subjob's execution context.</p>
jobfile(id\$)	Returns the temporary text file associated with id\$.
jobstore(id\$ [,array\$[all]])	Stores the content of the current page in a temporary file, identified by id\$. The value in id\$ is user-defined, and each unique value stores content

	<p>in a different temporary file. The other job-related functions use the <code>id\$</code> value to select which file to use. For example, you could store a whole job with an <code>id\$</code> of "job", and individual documents in jobs identified by their document number. Each would be stored separately and could be jobexec'd separately.</p> <p>If the optional array is supplied, then it, rather than the current page content, is written to the work file. If supplied, the array must be a one-dimensional string array (i.e. <code>dim myText\$[66]</code>).</p>
<code>lbound(arr\$[all][,dimension])</code>	Returns the lower-bound of the array <code>arr\$[all]</code> . If <code>arr\$</code> contains multiple dimensions, you can specify which dimension. For example, if <code>arr\$</code> is dimmed as <code>x\$[100,1:2]</code> , <code>lbound(x\$[all])=0</code> , <code>lbound(x\$[all],2)=1</code> .
<code>left(str\$,length)</code>	Returns the leftmost <i>length</i> characters from <code>str\$</code> , padding with spaces on the right to enforce <i>length</i> . Note also the <code>mid()</code> and <code>right()</code> functions.
<code>libexists(lib\$)</code>	Returns 0 if library <code>lib\$</code> doesn't exist, or 1 if it does.
<code>linecalc(Ins1\$,Ins2\$,operator\$ [,dlm\$ [,allownull]])</code>	<p>Returns a delimited string based on performing an operation on each sequential pair of delimited elements from strings <code>Ins1\$</code> and <code>Ins2\$</code>, such as adding, subtracting, or multiplying each pair, returning the result in a new string with the same delimiter and number of elements as exist in <code>Ins1\$</code>. The default delimiter is a linefeed, which is recognized as either <code>\$0A\$</code> or <code>"\n"</code> in <code>Ins1\$</code> and <code>Ins2\$</code>.</p> <p>Operator\$ can be "+", "-", "*", or "/" to perform the respective arithmetic operation, or "\$" to perform string concatenation. When concatenating, add a suffix to the "\$" to add the suffix between the concatenated values. For example "\$ " would add a space.</p> <p>If both column values are null, the resulting line value will also be null unless the <i>allownull</i> value is true (non-0), which can result in adding two null values and printing "0".</p>
<code>log(msg\$)</code>	Writes a log entry to the server log file, <code>logs/server.date.csv</code> .
<code>log(msg\$,logfile\$,format\$)</code>	<p>Logs a message (time stamped) to the specified file. The file is created if necessary. If <code>logfile\$</code> is null, the server log is used. If <code>format\$</code> is supplied, it is used as the mask for the time stamp. The following character sequences are substituted in the format:</p> <ul style="list-style-type: none"> • %YI – 4 digit year

	<ul style="list-style-type: none"> • %Yz – 2-digit year • %Mz – 2 digit month • %Ms – short month name • %Dz – 2-digit day • %Ds – short day name • %Hz – 2-digit hour (24 hour clock) • %hz – 2-digit hour (12 hour clock) • %mz – 2-digit minute • %sz – 2-digit minute <p>If no format is supplied, this format "%YI-%Mz-%Dz %Hz:%mz:%sz". If the format is supplied but is null (""), then no time stamp is written.</p>
logdeliver(to\$, tags\$ [,tags\$ [,attachfile\$ [,status\$ [,errmsg\$ [,respmsg\$]]]])	Writes a line to the delivery log using the values specified. The tags\$ argument can be used to log subject and body information in name=value pairs. The status\$ value can be null (""), in which case it is set to OK if errmsg\$ is null, or ERR if not.
logjobmsg(msg\$)	<p>This function logs messages during Image Manager job runs. It internally executes a jobdefs'logmsg(jobid\$,msg\$) method.</p> <p>It is only valid in Image Manager job scripting.</p>
logwarn(msg\$)	Adds a message to the job's .err file, which is also presented when the design tool runs a preview that results in warning errors.
lower(expression)	Returns text in lowercase.
ltrim(str\$)	Returns the value of str\$, trimmed of leading spaces.
mailaddr(email\$) mailname(email\$)	Given a full email designation, which includes a name in quotes and an address, often bracketed, these two functions return the address and name portions, respectively.
mcut(col,row,cols,rows,value\$,lf\$,trim\$)	Returns multiple lines of text, optionally with line-feed delimiters and/or trimmed of spaces. The lf\$ argument can be set to "Y" or "y" to add a line-feed character between each line; likewise, the trim\$ argument can be set to "Y" or "y" to cause each line to be trimmed before returned. In addition, mcut() assigns each line in the cut region to value\$. Use null ("") or spaces to erase the source text. PostScript input not supported.
mget(col,row,cols,rows,lf\$ lf,trim\$ trim)	Returns multiple lines of text into a single string, optionally with a line-feed delimiter and/or trimmed of spaces. This function is useful in conjunction with multi-line functionality of the text command. The lf\$

	<p>argument can be set to "Y" or "y" to add a line-feed character between each line; likewise, the trim\$ argument can be set to "Y" or "y" to cause each line to be trimmed before returned. Optionally use the boolean syntax, using 0 or 1 for linefeed off/on, and 0 or 1 for trim off/on.</p> <p>Beware that if mget() results with linefeeds are used in an exec() function, you should substitute the linefeeds with the "\n" mnemonic sequence: sub(mget(1,20,10,40,1,1),\$0a\$,"\n").</p>
mid(arg1\$,arg2,arg3)	Safely returns a substring without generating an error 47 if the value in arg1\$ isn't long enough to accommodate position arg2 and length arg3. Note also the left() and right() functions.
mset(col,row,cols,rows,value\$)	<p>Multi-line set function. Will work with multi-line value\$, delimited with mnemonic \n character sequences or chr(10) values.</p> <p>PostScript input not supported.</p>
msfax(filename\$, faxnum\$, tags\$ [, errmsg\$])	Faxes filename\$, normally an UnForm-generated PDF file, to the fax number specified in faxnum\$. Numerous supported tags can be specified in tags\$, in the format tag1=value,tag2=value,... Requires the Windows Support Server. For more details, see the Windows Support Server chapter.
parse(str\$,n,delimiter\$)	Returns the <i>n</i> th element of the string str\$, when parsed by the delimiter specified. For example, parse("one,two",2,",") would return "two". If the delimiter is null, then any white space delimiter is used.
parseq(str\$,n,delimiter\$)	This is the same as parse(), except that honors quoted values in the string str\$, ignoring delimiters contained in them.
pdfpages(pdf\$)	Returns the number of pages in a PDF file. The file must be in non-optimized format.
pdftoimage(fromfile\$,tofile\$,format\$ [,resolution[,errmsg\$]])	<p>Uses Ghostscript, local to the server or via the Windows Support Server, to convert from PDF file fromfile\$ to an image file tofile\$, using the format format\$. Valid formats match those of the Ghostscript drivers defined in uf101d.ini.</p> <p>If fromfile\$ contains multiple file names separated by semicolons or linefeeds (\$0a\$), all the files are sent to Ghostscript for conversion. Use this to create a pdf file composed of other pdf files:</p> <p>files\$="path1.pdf;path2.pdf;path3.pdf"</p>

	target\$="allfiles.pdf" pdftoimage(files\$,target\$,"pdf")
prm("name")	The prm() function has been added as a synonym to the gbl() and stbl() functions, which return global string table values typically associated with the -prm command line option.
proper(expression)	Returns text in Proper Case.
putaddress(book\$,entityid\$,doctype\$,address\$)	<p>Opens the address book specified, and adds or updates the address entry identified by entityid\$ and doctype\$. The address\$ template can be created by execution of a getaddress() function, even from an invalid entityid\$, then filled with appropriate values. The entityid\$ and doctype\$ arguments are automatically copied to the address\$ template.</p> <p>The address book template fields can be referenced as address.entityid\$, address.doctype\$, address.entityname\$, address.contactname\$, address.to\$, and address.combine.</p>
putdocidprop(lib\$,doctype\$,docid\$,prop\$)	<p>Updates the document properties of the document type and ID in the library specified. The document properties are replaced with the values found in the composite string prop\$. These string properties are:</p> <p>Prop.date\$ - date in yyyyymmdd format Prop.time\$ - time in hhmmss format (24-hour clock) Prop.title\$ - title string Prop.entityid\$ - entity id string Prop.notes\$ - notes, which can have CRLF line breaks Prop.keywords\$ - semi-colon delimited keywords Prop.categories\$ - semi-colon delimited categories with pipe-delimited segments Prop.links\$ - semi-colon delimited list of links</p> <p>All properties found in the string are updated, so you must first read existing properties using the getdocidprop() function, then modify those properties desired, then update them with this function. This function will not add new documents to a library. It only updates existing ones.</p>
putpage(n,arr\$[all])	<p>Replaces page n with text array arr\$[all].</p> <p>PostScript input not supported.</p>
range(str\$,first,count,dlm\$)	<p>Returns a range of fields from the string str\$, starting with the first field, for count fields, delimited by dlm\$. srange("1,2,3,4,5",2,3,"") would return "2,3,4". Delimiters inside quotes are ignored.</p>

removezonepagehdrs(str\$)	Remove zone page header rows from linefeed delimited str\$. Typically used in filters that process ocr or column zones, so the data can be processed without regard to pages. Also see the iszonepagehdr() function.
resolveexpr(str\$)	<p>Scans for expressions in braces, attempts to resolve them, and replaces the braced expressions with resulting values. Provides a more convenient syntax than expression building in code. Example: if a\$="123.01", resolveexpr("a={a\$}") returns "a=123.01". When resolving code block variables, this is only valid inside code blocks.</p> <p>If you need to escape braces in the string, such as in a regular expression that uses a length specifier such as {10}, use \{ and \}.</p>
right(str\$,length)	Returns the rightmost <i>length</i> characters from str\$, padding with spaces on the left to enforce <i>length</i> . Note also the left() and mid() functions.
rtrim(str\$)	Returns the value of str\$, trimmed of trailing spaces.
runjobon(server\$ [,args\$, [errmsg\$]])	Runs a copy of the current job on another UnForm server, using the local command line client. The server\$ value should be set to the name (and optional settings) of the server to run the job on. The format is identical to the command line option -server. Set additional or overriding arguments in args\$, if necessary. If an error occurs, it is logged and also passed back in errmsg\$, if provided. Since the job is run by a client command line, failure history is maintained per the client configuration, to allow recovery of failed jobs if necessary.
set(col,row,cols,value\$)	<p>Returns value\$, after it places value\$ in the text\$[all] array at the position indicated.</p> <p>PostScript input not supported.</p>
setlogin(userid\$,password\$)	<p>Sets the login and password to enable the library object to access a library.</p> <p>An administrator can define secure passwords in the browser interface, and a password ID can be used in place of a plain text password. The syntax is simply the ID with a "store:" prefix in the password\$ field, such as "store:SQLUser1".</p>
settrans(filename\$)	<p>Sets the translation file dynamically as the job runs. This overrides what might be set via a -trans command line option.</p>

sqlconnect(datasource\$[,user\$,pswd\$ [,otheroptions\$ [,errmsg\$]]])	<p>Connects to the data source specified, using the specified user, password and optional parameters. Returns a channel on success, or 0 on failure, in which case the errmsg\$ argument contains an error message.</p> <p>An administrator can define secure passwords in the browser interface, and a password ID can be used in place of a plain text password. The syntax is simply the ID with a "store:" prefix in the pswd\$ field, such as "store:SQLUser1".</p> <p>The channel returned is used for subsequent access to the database using sqlexecute() and sqlfetch() functions.</p> <p>See the Database Access chapter for more information about database support.</p>
sqlexecute(chan,command\$[,errmsg\$ [,result\$ [,fdelim\$ [,rdelim\$]]]])	<p>Executes the SQL command specified, typically a SELECT statement, on the channel specified. The channel must have been previously returned by a sqlconnect() function. If result\$ is supplied as an argument, then a sqlfetch() method is executed to fill result\$ with all rows, and the number of rows fetched is returned.</p>
sqlfetch(chan,result\$[,count [,errmsg\$ [,fdelim\$ [,rdelim\$]]]])	<p>Fills result\$ with some number of rows from the most recent SQL command executed on the channel with the sqlexecute() function. Returns the number of rows filled.</p> <p>The number of rows returned is determined by the count argument. If not supplied, 1 row is returned, allowing a loop to be processed one row at a time. If count is -1, then all available rows are returned.</p> <p>The default field delimiter is a tab (\$09\$), but this can be specified with the fdelim\$ argument. The default row delimiter is a linefeed (\$0A\$), but this can be specified with the rdelim\$ argument.</p> <p>If an error occurs, errmsg\$ will hold a message.</p> <p>If no more rows are available, the function returns 0 and result\$ is empty.</p>
sshost(server\$,port)	<p>Sets the Windows Support Server hostname and port. Default values are defined in the uf101d.ini file in the sshost and ssport settings. This command allows for dynamic changing to a different server.</p>
striplines(text\$)	<p>Returns text\$, stripped blank lines from multi-line text, such as addresses. As a byproduct, all CR</p>

	characters are also removed, leaving simple LF line delimiters.
strtoarr(str\$,arr\$[all],dlm\$)	Converts string str\$ to an array arr\$[all], by splitting str\$ on delimiter dlm\$
sub(str\$,old\$,new\$)	Returns a string where all occurrences of old\$ in str\$ are replaced with new\$.
subidexists(lib\$,doctype\$,docid\$,subid\$)	Returns 1 if the document type, document ID, and sub ID exists in the library, or 0 if not.
tempfile([ext\$])	Creates and returns the name of a temporary file that is removed automatically at the end of the job. The file will have a ".tmp" extension, unless an extension argument is provided.
textimage(text\$, font\$, size, cols, rows, color\$, charset\$, errmsg\$)	<p>Creates a bmp image using Image Magick based on the TrueType font, point size, and image size. For limited Unicode output, you can use this function instead of font embedding to avoid large output caused by embedding a full ttf font. If a server-based version of Magick is unavailable, a Windows Support Server can be configured to support this function.</p> <p>The font\$ value is either a name found in the [fonts] section of the ufparam.txt file, or an actual TrueType font name. On Unix this would be a font file, on Windows (or the support server) it would be the font name recognized by the operating system, such as Arial-Unicode-MS. The size specifies the point size for the text. The cols and rows define the image size. If cols is 0, then the image will be sized to exactly fit the text. Otherwise, the text is centered within the image.</p> <p>The color setting can be "rgb rrggbb" (rr, gg, and bb are hexadecimal values 00-FF), or any color that Image Magick recognizes.</p> <p>The charset defaults to Windows ANSI (9J or iso8859-1). Text is converted to utf-8 for Magick. The function returns the name of the temporary .bmp file, which can be used in an image command expression.</p> <p>Direct unicode can be supplied in text\$ if charset\$ is "uc". To ensure the image isn't re-scaled incorrectly, use imgx/imgy functions to obtain actual cols/rows sizes for use in the image command.</p>

textfile(<i>path</i>\$)	This function creates a file and returns a path name to that file. The value of <i>path</i> \$ is interpreted in three ways. If null (""), a new temporary file is created, and will be erased automatically when the job is complete. Optionally, the value may start with a period to force the extension of the temporary file to match the value, such as ".pdf". Otherwise, the value should be a full path, and that file will be created and returned. Such custom paths are not erased automatically at the end of the job.
textheight(<i>text</i>\$, <i>fontnum</i> <i>fontname</i>\$, <i>size</i>, <i>attr</i>, <i>cols</i> [,<i>linespacing</i>])	Returns the text height, in rows, of <i>text</i> \$, given the font number or name, size in points (or pitch if the font is mono-spaced), style attribute (0=normal, 1=bold, 2=italic, 3=bold italic), columns (for wrapping calculations), and optional line spacing. This will be the equivalent height that would be used by the text command given the same attributes. If <i>linespacing</i> is not supplied, the default is based on the current lpi.
texttokeywords(<i>str</i>\$[,<i>maxkeywords</i>])	Returns a semicolon-delimited list of unique keywords from a block of text in <i>str</i> \$, which may include any white space, including line breaks. The keyword construction honors the site's non-word and non-character settings, and the list is limited to <i>maxkeywords</i> words, if supplied.
textwidth(<i>text</i>\$, <i>fontnum</i> <i>fontname</i>\$, <i>size</i>, <i>attr</i>)	Returns the text width in columns of <i>text</i> \$ given the font number or name, size in points, and style attribute (0=normal, 1=bold, 2=italic, 3=bold italic). The function honors the same font mapping as is used in regular UnForm processing for pdf, and understands the same fonts that are understood for internal calculations for justification, where laser fonts are loaded from the standard fonts.txt file, pdf fonts are mapped from these, and postscript fonts are loaded from .afm files in the psfonts directory. For Postscript, the width is based on the Windows ANSI symbol set. If the font is a TrueType font, and the text value doesn't appear to be unicode text, UnForm will implicitly convert it to unicode. If the text is preencoded to unicode but does not contain any null (\$00\$) characters, add \$0000\$ to the string to prevent this implicit conversion.
touc(<i>text</i>\$,<i>charset</i>\$)	Converts single-byte text in the character set specified to unicode text. Returns the unicode text. Known character set tables are specified in the UnForm unicode directory. <i>Charset</i> \$ can also be "utf8" or "utf-8" to convert from UTF-8 to UTF-16 (or Unicode).

translate(name\$ [,context\$ [,forcecontext]])	Returns the value associated with the specified name, based on the translation file and current rule set. The context value can be "text", "barcode", or "anchor", and if <i>forcecontext</i> is true (non-zero), only context-based names are searched.
trim(expression)	Returns <i>expression</i> after trimming spaces from the left and right side.
ttfchars(fontnum)	Returns a string made up of Unicode characters, two bytes per character, found in the True Type font mapped from the font number provided.
ubound(arr\$[all],[dimension])	Returns the upper-bound of the array arr\$[all]. If x\$ contains multiple dimensions, you can specify which dimension. For example, if arr\$ is dimmed as x\$[100,1:2], ubound(x\$[all])=100, ubound(x\$[all],2)=2.
upper(expression)	Returns text in UPPERCASE.
urlencode(str\$) urldecode(encodedstr\$)	Return URL-encoded and decoded string values. URL encoding substitutes % <i>hex</i> coding for non-printable values and also a number of characters that may be meaningful to URL parsing, such as "&" and "?".
urlgetfld(datastr\$,name\$)	Returns the value of the name\$ field. The value is returned without URL encoding. mailto\$=urlgetfld(datastring\$,"to")
urlsetfld(datastr\$,name\$,value\$)	Returns a URL-encoded string with the field name\$ set to value\$. The field is added if necessary. datastring\$=urlsetfld(datastring\$,"to",someone@somewhere.com)
urldelflds(datastr\$,names\$)	Returns the a URL-encoded string after removing the fields specified in name\$ from the URL-encoded string datastr\$. Multiple fields can be separated by commas. datastring\$=urldelflds(datastring\$,"to,from,subject,body")
urlgetnames\$(datastr\$)	Returns a list of field names in the data string. fldlist\$=urlgetnames\$(datastring\$) count=parsec(fldlist\$,";")

When using variables and line labels, you should avoid using any values that begin with "UF". UnForm reserves all such variables and labels for its use. You may use a backslash (\) at the end of a line to continue the statement on the next line. Lines prefixed with "#" are not added to the code.

Two data elements from the command line can be referenced in code blocks using the stbl() function (use gbl() in ProvideX environments). The `-s sub-file` option will generate stbl values as "*@name*". For example, if the substitution file contains the line 'company=Smith Produce', then stbl("@company") will return "Smith Produce". Further, the `-prm` command line option will directly create stbl values.

15.6 Runtime verbs and functions

The following list is a summary of verbs and functions present in the UnForm runtime engine and are commonly used in UnForm applications. Note that all functions accept an "*err=linelabel*" or "*err=next*" argument, and all verbs accept the same after any parameters, to branch if an error occurs. Optional arguments are shown inside braces {}.

ASC(string)	Returns the ASCII numeric value (0-255) of the first character of <i>string</i> .
ATH(string)	Returns a binary equivalent of a human readable hex string. ATH("1B") returns an escape character.
BIN(integer,length)	Returns a binary integer representation of the specified length. The inverse function of this is the DEC() function.
BREAK	Breaks out of a loop structure. Equivalent to EXITTO <i>linelabel</i> if <i>linelabel</i> is the line after the closing WEND or NEXT.
CHR(integer)	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(string,arg)	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(julian {,time} {:mask}) DTE(julian {,time} {:mask})	Returns a human readable date and/or time, based on the Julian date (see the JUL() function), a decimal time (hour and fraction of hour – 12.5=12:30PM), and a format mask. The mask can contain combinations of placeholder characters and modifiers. The placeholders are %M=month, %D=day, %Y=year, %H=hour (24 hour clock), %h=hour (12 hour clock), %m=minute, %s=second, %p=AM/PM. Modifiers include z=zero fill, s=short text, l=long text. Examples on June 30, 1999 at 1:30 in the afternoon: date(0) returns "06/30/99", date(0,"%M %D, %Y") returns "June 30, 1999", date(0,tim:"%hz:mz %p") returns "01:30 PM".
DEC(string)	Returns the decimal conversion of the binary integer in <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\$+string).
DIM string(length {,char})	Returns a string of <i>length</i> size, of spaces or the specified <i>char</i> character.
DIM name[dim1{,dim2{,dim3}}]	Creates a 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.
DIM() array access	<ul style="list-style-type: none"> • dim(read min(arr\$[,subscript])) returns the lower bound of the array, optionally of the subscript level if more than one dimension • dim(read max(arr\$[,subscript])) returns the upper bound of the array, optionally of the subscript level if more than one dimension. This also indicates the number of items in an associative array • dim(key arr\$[index]) returns the key of the associative array item at the index • dim(index arr\$[key\$]) returns the index of the associative array key specified • dim(drop arr\$[key\$]) removes the key from the associative array
DIR("")	Returns the current disk directory. On Windows, DIR(<i>driveletter</i>) will return the current directory for the specified disk drive.
EPT(number)	Returns the 10's exponent value of <i>number</i> . EPT(100)=3, EPT(12)=2.
ERASE filename	Erases a file. Obviously, care should be taken to only erase temporary work files.
EXITTO linelabel	Exits a loop structure (current level only, in nested structures) and jumps to the specified <i>linelabel</i> .
FBIN(number) I3E(number)	Returns a 64-bit IEEE number in natural left to right ordering.
FDEC(string) I3E(string)	Returns the decimal value of a 64-bit IEEE number.
FID(channel)	Returns a file identification string for the file opened on <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(integer{,string}) DIM(integer{,string})	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 "abcbca".
FIN(channel)	Returns additional file information not found in the FID() function. A common use of this function is to determine file size, which is stored as a binary integer in the first four bytes. To get the length of a file: x\$=fid(filechannel); length=dec(\$00\$+x\$(1,4)). Additional potentially useful information can be found as well. See the language reference manual for more details.
FOR numvar=start TO end {STEP increment}	Initiates a loop, using a numeric variable initialized to <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. A loop can be iterated with CONTINUE, though take care to avoid endless loops.
FOR K\$ INDEX ARRAY\$[ALL] FOR IDX INDEX ARRAY\$[ALL]	Loops over all keys of an associative array, or all indexes of a standard array, setting the FOR key or index for each element in turn.
FOR STR\$ FROM LIST\$	Iterates over the list as a delimited string, using the last character as a delimiter.
FPT(number)	Returns the fractional portion of a number. FPT(100.66) returns .66.
GOSUB linelabel	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 linelabel	Jumps to the specified <i>linelabel</i> .
HTA(hexstring)	Returns a human readable hex string of <i>hexstring</i> . HTA(CHR(2)) returns "02". HTA("0") returns "30".
IF test THEN statement(s) {ELSE statement(s)} {END_IF or FI}	Conditionally executes <i>statements</i> . <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 END_IF are executed without regard to the condition of the last IF test. Nested IF statements are accepted without practical limit.
IF test {THEN{:}}	A block (multiline) IF statement ends with an optional THEN and colon character, followed by the lines that execute if the <i>test</i> is true, and ends with END IF.

statements	
END IF	
INT(number)	Returns the integer portion of a number. INT(99.645)=99.
JUL(year,month,day)	Returns the Julian integer of the specified date elements. The year should be specified, if possible, as a 4-digit year. Otherwise the function will assume a century of 1900. The complement of this function is the DATE() function.
LEN(string)	Returns the length of the string.
LET var=value{,var=value...}	Assigns variables to values. The variables can be numeric, string, or array variables. The values can be any compatible numeric or string expression. LET is implied when an assignment is performed in context. "LET a=1" and "a=1" are equivalent.
MASK(string{,regexpr}) MSK(string{,regexpr})	Returns the position where a regular expression pattern was found in the <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(num{,num...})	Returns the largest number found in the list of <i>nums</i> .
MIN(num{,num...})	Returns the smallest number found in the list of <i>nums</i> .
MOD(num1,num2)	Returns the remainder of dividing <i>num1</i> by <i>num2</i> . MOD(4,3)=1, MOD(6,3)=0.
NUM(string)	Returns the decimal value of a string, assuming the string is a well-formatted value containing digits, a single optional period (decimal point), and a single optional leading hyphen (minus sign). Other punctuation or characters will return an error. NUM("-12.5") returns 12.5. NUM("1,456") results in an error.
ON integer GOTO GOSUB linelabel{,linelabel...}	Branches to one of the indicated line labels based on the value of <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(integer{,err=linelabel next}{,isz=integer}) string	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, specify a block size with the ",isz=-1" option.

PAD(string\$,len[,pad_code][,char\$])	Returns the string truncated or padded to the length specified. The pad_code can be 0 for left padding, or 1 for right padding, and the specified char\$ character is used for padding. The default is right padding and a space pad character.
POS(string1 relation string2 {,increment {,occurrence}})	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(channel) value {,value...} {,}	Prints a series of values, numeric and/or string, to the file channel specified. A line-feed character is added to the channel unless the last character of the statement is a comma.
READ{ RECORD}(channel {,options}) variable {,variable...}	Reads data from the specified channel into the specified variables, looking for field terminator characters to delimit variables. Field terminators include line-feeds, carriage returns, and nulls. Valid <i>options</i> include "err=linelabel", "end=linelabel", "siz=block size". "key=keystring", "ind=index", and "dom=linelabel". For intrinsic keyed files, use the key= or ind= options to read specific records. For text files, use READ to process line-feed delimited files, but be aware that carriage return characters act as field separators. To read text files as binary files, use READ RECORD with a "siz=" option.
REM	Places a non-executing remark line in the code. In UnForm, you can also use a # character.
RETRY	Retries the statement that caused the last error branch to be taken.
RETURN	Returns from a GOSUB branch.
RND(integer)	Returns a pseudo-random number. The random number sequence can be re-seeded by providing a negative integer, so it is common at startup (like in a prejob code block) to seed the RND function with a variable number, such as MOD(JUL(0,0,0)+INT(TIM*10000),32000). The <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(number,precision)	Returns <i>number</i> , rounded to <i>precision</i> . ROUND(1.566,2) =1.57. ROUND(100.83,0) returns 101.

SCALL(string) SYS(string)	<p>Executes the operating system command in <i>string</i>. Returns the result code provided by the operating system. Use this function to interface with the operating system or external commands. This is an alternative to opening a pipe to a command.</p>
SETERR linelabel	<p>Provides a generic error handler to catch errors not trapped by <code>err=linelabel</code> branches in functions and verbs. UnForm also adds error handling code to code blocks, and reports errors in a job error file (<code>temp/jobno.err</code> in the server directory).</p>
SGN(number)	<p>Returns a 1, 0, or -1, depending on the sign of <i>number</i>.</p>
STBL(string1{,string2}) GBL(string1{,string2})	<p>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, <code>STBL(string1)</code> will result in an error (trappable with <code>err=linelabel</code>, of course).</p>
STR(number{:mask}) STR(string{:mask})	<p>Converts 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. <code>STR(99.91:"0000.00")</code> returns "0099.91". <code>STR(19093.255:"###,##0.00")</code> returns "19,093.26".</p>
STRING filename{,err=label} SERIAL filename{,err=label}	<p>Creates a text file of the name specified. Use either a string variable or expression, or a quoted literal string.</p> <p>Examples: <code>STRING "/tmp/test.txt"</code> or <code>STRING "/tmp/"+str(dec(info(3,0)))+".txt",err=next.</code></p>
TCB(integer)	<p>Returns task control information. Commonly used <i>integer</i> values include: 10=last operating system error code, 16=length of <code>MSK()</code> function match, 20 for number of arguments passed to an ENTER command.</p>
TIM	<p>Numeric variable that returns the decimal time of day, from 0.0 to 23.99.</p>
UNT	<p>Numeric variable that returns the next available file channel number.</p>
WHILE condition...WEND	<p>Looping construct that performs statements between WHILE and WEND statements as long as <i>condition</i> is true or non-zero.</p>
WRITE {RECORD} (chan,options)data	<p>Writes data to a file. Numerous options are available, some depending on the type of file. See the full programming documentation available on www.pvxplus.com for more details.</p>

Lexical Substitutions

With the change in Version 6 to the ProvideX run-time engine, it is possible that some BBx syntax in code blocks will be incompatible. For the most part, the lexical substitutions automatically performed by UnForm will handle any differences, with the exception of direct I/O to BBx data files, which can often be handled with the `bbxread()` function. However, if any additional substitutions are required, they can be entered into a user-defined text file called `uflexsub.usr`.

The format for the lines in this file is simply `bbxsyntax=pvxsyntax`. An example is provided in `uflexsub.txt`, which is a file that provides some standard syntax substitutions that the internal lex capabilities do not support. You can add your own by simply creating `uflexsub.usr` and adding lines.

15.7 Error Codes

Error Codes

When code is executed, any errors that are not handled by `err=label` branches are reported as warnings on a job trailer page. High error code numbers are used to report errors in client-server communication. Common error codes are shown in the following table.

Error Number	Description
1	End of record error, which may occur on a buffered disk write operation if the data is too long for the record buffer. This error is rare in UnForm jobs, but could occur if output is being printed to a printer alias defined in the <code>config.unf</code> file.
2	End of file, which may indicate a disk full message, or a file that is too large for the operating system to handle.
10	An invalid file name was given.
11	A missing key on a keyed read operation, or a duplicate key on a keyed write operation with a <code>DOM=</code> option.
12	A missing file error on a file open operation, or a duplicate file error on a file creation operation.
13	Normally a file permission error.
14	A file channel conflict or locking conflict error.
16	Out of resources, such as file handles. If this error occurs, it is often due to opening too many files. This can easily occur if files are opened but not closed in a loop or call construct.
18	Normally a file or directory permission error.
20	Syntax error. Common causes include mismatched parentheses, incorrect spelling of verbs or functions, or missing or incorrect function arguments.
21 or 25	Missing statement, as referenced in an <code>ERR=label</code> , or a <code>goto</code> or <code>gosub</code> branch.
23	Missing GBL/STBL variable name, or missing string template variable.
26	String/Number mismatch, where a string variable or literal is used where a number is expected, or visa versa.
27	Stack error, such as a return without a <code>gosub</code> , or a <code>wend</code> without a <code>while</code> .

Error Number	Description
28	For/Next error, such as executing a next without an associated for.
29	Mnemonic error. Mnemonics are pre-defined codes inside single quotes, such as 'FF' or 'LF'. Therefore, single quotes are not valid as string literal indicators; only double quotes are.
30	Corrupt program, which indicates that UnForm itself is probably corrupted, unless this error occurs on a call statement referencing an external program.
31	Out of memory.
33	Out of memory.
36	Mismatched arguments on a call statement.
40	Numeric overflow, normally caused by a divide by zero.
41	An integer overflow or range error. Some functions require integer arguments, so a floating point number will cause this error. Also, some functions require integer arguments to fall in a certain range, and this error will occur if the function is given a value outside of the valid range.
42	Array subscript error.
43	Masking error.
46	String length error.
47	Substring error, such as a starting position of 0 or a length greater than the length of the string.
119	When running in evaluation mode, there is a limit of 5,000 records in data files. The error occurs when trying to write too many records to a file. An evaluation version can still read records from larger files, but cannot modify them. To prevent the error, production versions of UnForm must run under a non-evaluation license.
996	The client's authkey value does not match the server's.
997	The client's IP address is not in the server's list of valid addresses. To correct this problem, the allow= line in the server's uf101d.ini file must be modified to match the network addresses in use, and the uf101d server restarted.
998	The maximum number of concurrent jobs licensed was exceeded.
999	The server was unable to start the secondary process to handle the job within the allotted time of 30 seconds. Possible causes include a sluggish server and network problems, such as a DNS server timeout.
1024	The Windows uf101c.exe client can report this error if the network connection to the server is too slow.
1057	The Windows uf101c.exe client can report this error if the server is not running or a firewall is blocking the primary listening port.
1058	If a handler process fails to start at the server upon a job connection from a client, this error is reported to the client.

16 LEGACY HTML RULE FILES

HTML OUTPUT

UnForm provides an optional capability to produce HTML files from reports, using a processing engine that is similar to that used for laser printer output. Using this capability, users can convert their standard text-based reports into HTML documents, which are suitable for viewing with Web browsers such as Netscape Navigator and Communicator, and Microsoft Internet Explorer.

Reports can be converted in real-time, as part of a CGI or ASP procedure that responds to a browser request to generate a report, then format it as HTML. Alternatively, reports can be converted with a periodic batch process, such as a nightly procedure that produces various reports, then converts them all to HTML for viewing the next day.

Even without a rule set, UnForm can streamline text reports by producing plain text pages with horizontal rules at the end of each page. These are constructed using HTML templates, so standard company headers and footers can be applied even to reports that are not enhanced via a rule set.

16.1 Creating HTML

Creating HTML

UnForm will create HTML output if you specify "-p html" on the command line. Given this parameter, and with no "-f *rulefile*" parameter, UnForm will look for the "html.rul" file rather than the default "unform.rul" file used for printer output.

By default, the HTML output is generated to standard output (on UNIX only), but it is normally preferable to specify an output file, such as "-o /usr/internet/docs/reports/aging". UnForm can then build the reports with varying styles in stages, and a browser can view interim results as soon as the first page is generated. UnForm will add a ".htm" extension automatically to the output file. UnForm will also create additional files depending on the style of the report. For example, if a table of contents is generated as a separate document, then the base file (aging.htm in the above example) will be the table of contents, and additional files will be generated for the pages of the report (aging.*page*.htm).

A sample command, therefore, might look like this:

```
unform -i aging.txt -o /usr/internet/docs/reports/aging -p html -f ourhtml.rul
```

As HTML structure is very different from that of laser printers PCL, HTML rule sets are very different from printer rule sets. UnForm uses HTML table structures to format pages. These structures have a defined hierarchy of rows, cells, and data, with attributes applied to either cells or data. HTML rule sets follow this structure in that you define rows, then within rows you define cells, and then within cells you define the attributes of the cell and text.

The HTML output that UnForm produces can be in one of several styles. The rule set options used to trigger the style are shown in parentheses:

- * The simplest form is that of one document with all the pages sequentially created as tables. If no output file is specified (-o *filename*), this is what UnForm will produce regardless of any style options you specify.
- * The output can be produced in one file, with a table of contents at the top of the file (toc=y or toc=l, multipage=n). As each page is generated and appended to the file, the table of contents is updated and inserted at the top. The table of contents consists of descriptions linked to the individual pages. The descriptions default to "Page number *n*", but can be created in page code blocks. Additionally, the table of contents can be created as a vertical column (toc=y), or as a bullet list (toc=l).
- * The output can be produced in multiple files (multipage=y), with the table of contents being the primary one, with links to each page as a separate HTML document.
- * The output can be produced as frames (frame=y), with the table of contents in one frame, and pages in the other. The target pages can be stored in a single file, multi-page document, or with each page in an individual file.

Note that all these options but the first require that a table of contents be maintained as each page is generated. In order to construct an updated document as each page is generated, UnForm must generate temporary files with which to build the HTML required. The *filename* specified by the "-o" option is re-created as each page is completed. Therefore, if standard output is generated rather than output files, only the first style can be produced.

This interim generation of files means that the HTML output can be viewed as soon as the first page is generated. This can be very helpful when large reports are being formatted in real-time.

16.2 HTML Configuration

When generating HTML documents, UnForm uses several configuration elements to structure the output. Most of these are created in UnForm's parameter file, which is named "ufparam.txt". Note that you can create a custom parameter file for your site that will not be overwritten during an update of UnForm by copying "ufparam.txt" to "ufparam.txc". Then make any changes to the custom version.

A section in the configuration file headed by "[html]" controls HTML configuration. It will look like this:

```
[html]
page=page.htm
toc=toc.htm
both=both.htm
frame=frame.htm
pagenum=Page number
imagelib=
imageurl=
complete=Report Complete
incomplete=Report not complete (reload page to view again)
```

The following table describes each parameter:

Element	Description
page= <i>filename</i> toc= <i>filename</i> both= <i>filename</i> frame= <i>filename</i>	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.

Element	Description
	To create custom templates for your site, you should copy each file to some other name, modify the file names identified in these four elements, and edit the templates for your needs. See "HTML OUTPUT TEMPLATES", below, for more information.
<code>colwidth=</code> <i>text</i>	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 <code><td width=</code> <i>value</i> <code>></code> 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 colwidth keyword in a rule set.
<code>pagenum=</code> <i>text</i>	This text is used to generate the default table of contents' values. A space and the page number follow the text.
<code>imagelib=</code> <i>directory</i>	This points to a directory where image files are physically stored on disk. If any column definition has an option indicating it contains image file names, then the files in the column are searched for first as named, and then in this directory. If the image can be found, then the image tag can be generated with width and height parameters, which normally speeds up the page rendering speed by the browser.
<code>imageurl=</code> <i>url-prefix</i>	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=</code> <i>text</i> <code>incomplete=</code> <i>text</i>	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.

16.3 HTML Output Templates

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>
```

16.4 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.

16.5 BORDER

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.

16.6 COLDEF

COLDEF

Syntax

1. [coldef | ccoldef] *col*, *cols*, *options*
 { *code block* }
2. coldef "*text* | ~*regexpr*", *coloffset*, *cols*, *options*
 { *code block* }
3. coldef "*text* | ~*regexpr*", *coloffset*, "*to-text* | ~*to-regexpr*", *to-coloffset* *options*
 { *code block* }

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

Syntax 2 defines a region based on a search for a starting point. For each *text* value or *regexpr* (regular expression) found, the region will begin at the column *coloffset* from the point found, and extend for *cols*

columns. For example, **coldef "Customer total",-1,52** will create the region from 1 column before the occurrence of "Customer total", and extend the region for 52 columns.

Syntax 3 defines the region based on two searches, one to find the starting column, one to find the ending column to the right of the starting point. In both cases, the column position is adjusted for the offset.

coldef "Current",-1,"30-Days",-1 would define a region starting one column before the word "Current", extending to one column before the word "30-Days". If just the first string is found, then all columns from there to the last are specified. If just the last string is found, then all columns from the first through there are specified. For this reason, be sure that any absolute column regions are specified first.

Description

Column definitions are used to define columns within a row definition. Each column definition becomes a table cell (<td>...</td>), with each row in the column being separated by a line break (
). There can be up to 255 column definitions within any given row definition. Any given column will be formatted based on the first **coldef** keyword that applies to it. Columns not so defined will be displayed as mono-spaced text, using the HTML <pre> and </pre> tags.

Each column definition can define attributes that will apply to the text and cell formatting, and optionally can have a code block associated with it to add custom Business Basic coding to the data in the column.

Options are comma-separated lists of words and parameters. The options available in the column definition include:

Option	How it gets applied
bgcolor=#rgb, bgcolor=color	Cell gets a bgcolor=value attribute to control the background color. The color can be expressed as an #rrgbb 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 attribute.
bottom, top, middle	Cell gets "valign=value" attribute to control vertical justification. The default is "top".
center, left, right	Cell gets "align=value" attribute to control horizontal justification. The default is "left".
color=#rgb, color=color	Text gets attribute. The color can be expressed as a #rrgbb hexadecimal value or as a color name supported by the target browser, such as red, blue, white, etc..
font=font	Text gets attribute. Several modern browsers support this, though the font typeface selected may not be available on all clients.
hdr=html text	The top of the column gets the <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 with hdr, gets these values in its <td hdrtd>hdron hdr value hdroff</td> structure. Be sure to turn off any <i>hdron text</i> HTML tags in <i>hdroff text</i> .
italic	Text gets <i> attribute.

Option	How it gets applied
image	Text is assumed to be file names that are image files, and gets treated as an tag. The ufpam.txc\l 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 specified by the other options.
size=n	Text gets attribute. Size ranges from 1 to 7, with 3 being considered a "normal" size.
suppress	If this word is present, then column data gets set to null.
underline	Text gets <u> attribute.

Code blocks are optional definitions associated with any given column definition. With a code block, it is possible to manipulate the text of each row in the column. A typical use of this capability might be to convert the plain text to hyperlinks, so that a column of part numbers could be linked to pages in a catalog, for example. Code blocks begin just after the opening brace "{", can extend as many lines as required, and end with a closing brace "}".

The code block is executed for each row of the column. As the code starts, the following variables can be used:

Variable	Description
attr.align\$ attr.bgcolor\$ attr.blink attr.bold attr.color\$ attr.font\$ attr.italic attr.otheropt\$ attr.size\$ attr.underline attr.valign\$	The attr\$ variable is a string template that defines the attributes to apply to the text or cell. These values match those defined above in the Options. Numeric values can be set to 0 (false) or 1 (true). String values can be set to any valid value for that attribute.
colofs	The column offset from the left edge of the text. If the column region is from column 21 through 40, then colofs will be 21. This should be treated as a read-only value.

Variable	Description
cols	The number of columns in the region. Read only.
row	The row number within the current region, from 1 through the last row in the region. With each execution of the subroutine, the row will increase by 1. Read only.
row\$	The text of the current row within the region. This can be manipulated by the code.
rowofs	The position of the current row, relative to the whole page. If you need to refer to data in some other column of the current row, use rowofs. Read-only.

Functions available for your use, in addition to any intrinsic Business Basic functions, include:

Function	Description
get(<i>col,row,cols</i>)	Returns text from the page, given the column, row, and cols parameters.
htmlencode(<i>text\$</i>)	Returns <i>text\$</i> after converting HTML entities into displayable versions.
set(<i>col,row,cols,text\$</i>)	Sets <i>text\$</i> into the page at the given column, row, and columns.
urlencode(<i>text\$</i>)	Returns <i>text\$</i> after URL encoding to make it suitable for inclusion in a hyperlink.

16.7 COLWIDTH

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.

16.8 FRAME

FRAME

Syntax

frame=y | yes | n | no

Description

The **frame** keyword can be used in conjunction with the **multipage** keyword to control the presentation of the report. Without these options, UnForm will produce a single file (named with the **output** keyword or – o command line option, or to stdout), containing an HTML table for each page of output from the source file. With the **multipage** keyword, UnForm will produce unique files for each page of output, plus a table of contents page (whose format is controlled by the **toc** keyword). If frame is set to "y" or "yes", then an additional frame file is created for the browser to view the table of contents constantly while viewing the report pages.

The output filename generated is for the frame file if frame is set to "y" or "yes", and the table of contents file if frame is not present or is set to any other value.

This keyword is ignored if there is no *filename* specified for the output.

16.9 HDRON, HDROFF, HDRTD

HDRON, HDROFF, HDRTD

Syntax

hdron=*value*
 hdroff=*value*
 hdrtd=*value*

Description

When a coldef **hdr=***text* option is present, UnForm will add *text* to the top of the column, in a separate cell. In order to make a column-heading stand out, it may be desirable to give it attributes that are distinct from the column text. These keywords define HTML text attributes to add before and after any column header. **hdrtd** applies `<td value>` to the cell tag, while **hdron** and **hdroff** apply to the heading text. Values for individual row groups can be specified in the **rowdef** or **coldef** keywords.

For example, **hdron=<small>**

 and **hdroff=</small>** would make column headings small and bold.

Be sure to close any tags in the **hdron** value with corresponding tags in the **hdroff** variable.

16.10 LOAD

LOAD

Syntax

load *filename*

Description

The **load** keyword is used to load a secondary text file into the rule file at parsing time, at the position of the **load** keyword. This provides the ability to maintain separate text files for the definitions, grouped in any manner desired. For example, a common set of options for all reports could be defined in a second file, and each report could reference that file.

UnForm will try to open the file first as named, then in the UnForm directory if it is not found. Note that the prefix setting, if present, in UnForm's config.unf file can be used to affect file searching.

Example:

```
[Report1]
load "stdoptions.txt"
```

16.11 MULTIPAGE

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.

16.12 NULLROW

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.

16.13 OUTPUT

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*.

16.14 OTHEROPT

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.

16.15 PAGESEP

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.

16.16 PREJOB, PREPAGE, POSTJOB, POSTPAGE

PREJOB, PREPAGE, POSTJOB, POSTPAGE

Syntax

```
prejob | postjob | prepage | postpage {
  code block
}
```

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

Description

These keywords are used to add Business Basic processing code to the document generation process. They represent four different subroutines that UnForm executes at specific points during processing. The *code block* can be an arbitrary number of Business Basic statements; the total number of statements in all code blocks can be about 6,000 (or less, depending on program size limits imposed by the run-time environment).

- * **prejob** executes after the rule set has been read, and after the first page is read, but before any printing takes place. Use this code to open files or databases, prepare SQL statements or string templates, create user-defined functions, and initialize job variables.
- * **postjob** executes after the last page has been printed. Use this to close out your logic, such as adding totals to log reports. There is no need to close files, since UnForm will RELEASE Business Basic.
- * **prepage** executes after each page is read, but before any printing takes place. Use this to gather data associated with any page, or to modify the content of the text if you need such modifications to apply to all copies.
- * **postpage** executes after the last copy of each page has printed.

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

You may use the following variables and functions in your *code block*:

- * **text\$[all]** is a one-dimensional array of the text for the page. For example, text\$[2] is the second line of the page.
- * **mid(arg1\$,arg2,arg3)** (or fmid\$(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 fget\$(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 fset\$(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.

16.17 ROWDEF

ROWDEF

Syntax

1. [rowdef | crowdef] row, rows, options
 { code block }
2. rowdef "text | ~regexpr", rowoffset, rows, options
 { code block }
3. rowdef "text | ~regexpr", rowoffset, "to-text | ~to-regexpr", to-rowoffset options
 { code block }

Syntax 1 defines an absolute row region. **rowdef 5,3** for example, would define a row region starting with row 5, and extending 3 rows down (5-7). If the "crowdef" format is used, then *row* is the starting row, and *rows* is the ending row. **crowdef 5,7** would define the same region as **rowdef 5,3**.

Syntax 2 defines a region based on a search for a starting row that contains the text or matches the regular expression. For each *text* value or *regexpr* found, the region will begin at the row *rowoffset* from the point found, and extend for *rows* rows. For example, **rowdef "Customer total",0,1** will create a region from each row containing "Customer total" (0 offset is that row), and extending for 1 row (just that row).

Syntax 3 defines the region based on two searches, one to find the first row, one to find the ending row below the starting row. In both cases, the row used for the region is adjusted for the offset. **rowdef "Customer:",1,"Customer:",-1** would define a region between each occurrence of the text "Customer:". If just the first string is found, then all rows from there to the last are specified. If just the last string is found, then all rows from the first through there are specified. For this reason, be sure that any absolute regions are specified first.

Under format 3, if the last string is not found, UnForm will continue that row definition on the page following the first unallocated row at the time this row definition is evaluated on that page.

Description

Row definitions are used to define sets of rows for which a given group of column definitions would apply. Each row definition defines a group of rows that will be presented within a single table row (<tr> ... </tr>). Under any given row definition, place the column definitions (**coldef** keywords) that will be used to format the rows.

For example, an A/R Aging Report might contain a report heading, column headings, one or more customer headings, and, under each customer heading, one or more detail lines. At the end of the detail lines would be customer totals. This report would have five row definitions, for each type of row: report heading, column heading, customer headings, detail lines, and totals. Each of these types of rows will have its own set of column groups (or in some cases, no column groups at all, allowing simple mono-spaced presentation.)

There can be up to 255 row definitions within any rule set.

Each row definition can define attributes that will become defaults for the text and cell formatting of all the column definitions. Additionally, row definitions can define an option called "suppress", which causes UnForm to suppress the display of the row region. A comma separates each option.

Option	How it gets applied
<code>bgcolor=#rgb</code> , <code>bgcolor=color</code>	Cell gets a <code>bgcolor=value</code> attribute to control the background color. The color can be expressed as an <code>#rrggbb</code> hexadecimal value or as a color name supported by the target browser, such as red, blue, white, etc..
<code>blink</code>	Text gets <code><blink></code> attribute.
<code>bold</code>	Text gets <code></code> attribute.
<code>bottom</code> , <code>top</code> , <code>middle</code>	Cell gets <code>"align=value"</code> attribute to control vertical justification. The default is "top".
<code>center</code> , <code>left</code> , <code>right</code>	Cell gets <code>"align=value"</code> attribute to control horizontal justification. The default is "left"
<code>color=#rgb</code> , <code>color=color</code>	Text gets <code></code> attribute. The color can be expressed as an <code>#rrggbb</code> hexadecimal value or as a color name supported by the target browser, such as red, blue, white, etc..
<code>font=font</code>	Text gets <code></code> attribute. This is supported by several modern browsers, though the <i>font</i> typeface selected may not be available on all browser clients.
<code>hdr=html text</code>	The top of the column gets the <i>html text</i> , followed by a line break <code>
</code> tag. Use this option to replace top of page column headers with "in cell" column headers.
<code>hdron=hdron text</code> <code>hdroff=hdroff text</code> <code>hdrtd=hdrtd text</code>	The column header, if defined, gets placed in a cell with <code><td></code> 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> .
<code>italic</code>	Text gets <code><i></code> attribute.
<code>noencode</code>	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.
<code>otheropt=options</code>	The table cell gets additional attributes not otherwise specified by the other options.
<code>size=n</code>	Text gets <code></code> attribute. Sizes range from 1 to 7, with 3 being considered a "normal" size.
<code>suppress</code>	The rows are not displayed.

Option	How it gets applied
tr	Each row in the row group gets a <tr> tag, ensuring that 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.

16.18 TITLE

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.

16.19 TOC

TOC

Syntax

toc=y | yes | li | list | sh | short

Description

If this keyword is set to "y" or "yes", UnForm will generate a simple table of contents by constructing hyperlinks to each page generated. The hyperlinks are placed either at the top of the document, in a separate main document, or in a document referred to as the table of contents in a frame.

The following templates use a table of contents. Templates refer to files in the UnForm directory, and are referenced in the parameter file under the "[html]" section: "both=" and "toc=". In each case, the placement of the table of contents is based on the placement of the tag "[toc]" within the template file.

The text displayed for each hyperlink is generated from the "pagenum=" item of the "[html]" section of the parameter file (ufparam.txc or ufparam.txt.) This text can also be generated by Business Basic code in the prepage{} or postpage{} code blocks, by setting the string variable "toc\$" to the value desired.

If the keyword is set to "li" or "list", then the hyperlinks are created within an HTML unordered list (...), 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.

16.20 WIDTH

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.

16.21 Sample HTML Rule Set

Sample HTML Rule Set

Below are sample rule sets defined in the sample rule file, `samhtml.rul`. The sample text input files used by UnForm for the PCL output examples are redefined here for HTML. Comments are interspersed in the rule sets to help clarify which keywords perform which tasks.

16.22 Aging Report Sample

Aging Report Sample

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

```
uf101c -i sample3.txt -o aging.htm -p html -f samhtml.rul
```

You can substitute a different path/file name for "aging" to produce the HTML file elsewhere, such as in the HTML document tree of your Web server.

The form is called "aging" to distinguish it from other rule sets. If the "-r aging" option is used on the command line, then this set will be used.

[aging]

A detect statement identifies a report as the one defined by this rule set. If no "-r ruleset" option is used on the command line, then this detect statement will be evaluated. If the text "Detail Aging" appears in any column on row 2, this rule set is used.

```
det ect 0, 2, "Detail Aging"
```

The HTML output will produce 132 columns and 66 rows per page.
cols 132rows 66

Any text consisting of 3 or more dashes will be erased. This removes all the dashed underlines at customer totals. There are other ways to accomplish this, including defining a row set and using the suppress option, or using a prepage{} code block to erase such text from the text\$[] array.

hline "---"

The title used in HTML output for this report will be "Aging Report".

title "Aging Report"

If this line were not commented out (with the #), then anytime this rule set was used and no "-o filename" was present on the command line, the output would go to "/tmp/aging.htm."

#output "/tmp/aging"

This report will be generated in multiple files (one per page), with a table of contents page, and with an HTML frame construct.

multipage=ytoc=yframe=y

Between each page will be an HTML <p> tag (a paragraph separator). Any HTML text could be supplied, including references to global strings inside square brackets ([variablename]). The hdroff/hdron 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>hdroff=<i>hdroff=</i>

This rowdef keyword defines a row set from row 1 for 5 rows. All column definitions within this row will default to a background color RGB hex value of FFE0E0 (lots of red, high green and blue content).

rowdef 1,5,bgcolor=#ffe0e0

For the above row set, there are three column sets: 1 through 10, 11 through 110, and 111 through 132.

The columns are left, center, and right justified, respectively. Otherwise, except for the background color, the browser will use its default values for displaying the data.

coldef 1,10,leftcoldef 11,100,centercoldef 111,22,right

This row definition causes UnForm to suppress display of rows 6, 7, and 8 (the column heading information). The rule set will define the column headers as necessary in other row sets.

rowdef 6,3,suppress

Each customer has a heading line, distinguished by the occurrence of a phone number in those rows.

The initial quoted value "~\(...-...-...)" instructs UnForm to search for a regular expression match that looks like a U.S. phone number in parentheses. From any and all such rows, it will start at 0 rows up or down, and continue for 1 row. This defines those and only those rows that contain the phone numbers.

Columns defined for those rows will be bold, with blue text on a white background. As no columns are defined under this row definition, UnForm allocates one column set the full 132 columns wide, and applies the row defaults to the text.

Customer headerrowdef "~\(...-...-...)"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 linesconst LEFT="bgcolor=#e8e8e8,color=black"const
RIGHT="bgcolor=#e8e8e8,color=black,right"
rowdef "~./../.",0,"----",-1
```

Each invoice line is made up of 13 columns of information. Each has been defined by the ccoldef keyword by starting and ending column values. Additionally, each is given a header value that will appear at the top of the column, and a constant that references other attributes defined earlier in the rule set.

```
ccoldef 1,10,hdr="Invoice",LEFTccoldef 11,20,hdr="Due Date",LEFTccoldef 21,31,hdr="PO
Number",LEFTccoldef 32,39,hdr="Ord Number",LEFTccoldef 40,45,hdr="Terms",LEFTccoldef
46,52,hdr="Type",LEFTccoldef 53,64,hdr="Future",RIGHTccoldef 65,75,hdr="Current",RIGHTccoldef
76,86,hdr="30 Days",RIGHTccoldef 87,97,hdr="60 Days",RIGHTccoldef 98,108,hdr="90
Days",RIGHT,color=redccoldef 109,119,hdr="120 Days",color=red,RIGHTccoldef
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 totalsrowdef "-----",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,rightccoldef 65,75,rightccoldef 76,86,rightccoldef
87,97,rightccoldef 98,108,rightccoldef 109,119,rightccoldef
120,132,bold,right
```