



DocOrigin Reference Manual

Version: 3.3.006.01

Release Date: January 21, 2025

[Online Documentation](#)

© 2007-2025 Eclipse Corporation WSL Inc. - All Rights Reserved

eclipsecorp.us

Table of Contents

Using DocOrigin	16
Getting Started	17
Proof-of-Life.....	17
Your First DocOrigin Form Design	17
Preview Your Design.....	17
Terminology: Panes & Containers	17
Scraping Data Off of Reports.....	18
License Keys	19
Evaluation Spoiler Stripe.....	19
License Key File	19
Time-Limited Evaluation License	19
Permanent License	19
Maintenance and Support	19
Product Release Issue Date	19
No Need for New Keys Unless Upgrading	19
Temporary Emergency Keys.....	20
Keep M&S Up-to-Date	20
Multiple Outputs	21
One Output Per Run.....	21
One Output Per Document.....	21
Different Output Formats	21
Multiple Outputs Per Document.....	23
Multiple Outputs Per Document 2.....	23
Usage.....	23
Output Configuration Names.....	23
Sending Mail	25
The Email From Address	25
The Email Subject.....	25
The Email Attachment(s).....	26
The Email's Text-Based Body.....	26
The Email's HTML-Based Body	26
Images in the Email HTML body.....	26
Summary.....	27
Debugging.....	27
Email Sending Verification	27
Email Configuration	27
RTF Support	28
Limitations	28
Importing Rich Text Files.....	29
Barcodes	30
Overview	31
Platforms.....	31
Minimum Configuration Recommendation.....	31
Merge Architecture	32
A Classic Merge Command Line.....	32
FolderMonitor Architecture	33
Copyright Notices	34

Internationalization	35
Definitions.....	35
Considerations	35
Configuration Editor	36
Configuration files	37
More Dynamic Configuration	37
User Interface.....	38
The ConfigEditor Main Window	38
Typeface Substitutions.....	41
Advanced Properties	42
.prt Override	44
Design.....	45
Design Tutorial	45
Form Files.....	45
File-Print	46
Form File Import/Conversion	47
Import Prerequisites	47
Batch Conversion.....	47
Conversion Effort Estimates.....	48
Fragments	49
Libraries	49
Adding Fragments.....	50
Fragment Resolution	50
Fragments in Merge	50
Default Object Templates	51
Design Images	52
PDF Page As Image	52
Image as Template	52
PDF Page Images	53
Object Tags	54
AutoEmail Message Text.....	54
Barcode Overrides.....	54
Ellipsis.....	54
HTML Images as URL References	54
PDF/UA Tags	54
RadioButton and CheckBox Options	55
Direct Data Binding.....	55
RetainWhenBlank Tag	56
Design Keyboard Shortcuts	58
Alt+Ctrl+N - Looking at the XML for an Individual Object	58
Ctrl+Shift+E - Save Image	58
Alt+X - Convert HEX Value to Unicode Symbol	58
Ctrl+G - Manage Object Tags.....	58
ALT - Reuse Previous Object Tool	58
ESC - Toggle Between Text and Object Edit.....	58
Ctrl+Shift+A - Select All Objects in Pane	58
F5, F7, F8 - Preview.....	59
Ctrl - Line on a Diagonal.....	59
Data Explorer Options.....	60

Open Data File.....	60
Change Document Separator Tag.....	60
Select in Form.....	61
Allow for Dragging Tables.....	61
Copy Data Path To Clipboard.....	62
Form Explorer Options.....	63
Table.....	63
Link Objects into Pane.....	64
Expand All Panes.....	64
Collapse All Panes.....	64
Expand Children.....	65
Collapse Children.....	65
Copy Form Path To Clipboard.....	66
TagAs.....	66

FilterEditor.....68

Subject Matter Expertise.....	68
Learning FE.....	69
Beginner's Tip.....	69
Sample XML File.....	69
Show Those Menu Items.....	70
Filter Editor - 101.....	70
How Does It Work?.....	71
Defining a Rule.....	71
Defining an Extraction.....	71
Review.....	72
Rule order.....	72
States.....	72
The *Start* State.....	72
The *AnyState* State.....	72
The Current Line.....	72
Extractions.....	72
Variable Length Extractions.....	73
Moving On.....	73
Filling In the XML.....	73
XML Output Data Structure with Multi-Instance.....	73
Non-Extraction Actions.....	75
Start New Document.....	75
Extraction Steps.....	76
The Lexicon.....	76
Sentinel Detection.....	76
Ways Into a State.....	76
FilterEditor JavaScript.....	77
Custom Functions.....	77
Now that you have defined your custom function, how do you use it?.....	77
What if you want to edit your script?.....	77
FilterEditor Functions.....	78
xFilter Properties.....	80
Filter file.....	80
Overlay Data file.....	80
Overlay Symbolset.....	80
Overlay file contains carriage control characters.....	80
Max # of lines to transform in FE.....	80

XML Prototype.....	80
Document Separator tag	81
Remove empty nodes from the output XML.....	81
XML Output Data Structure with Multi-Instance.....	81
FolderMonitor	82
FM on Windows.....	82
FM on Unix.....	82
Command Options	84
-cluster	85
-concurrent	86
-errorFolder	87
-exitNow	88
-exitWhen.....	89
-fileOperationsRetryCount.....	90
-fileOperationsSleepTimer	91
-FMScript	92
-JPSName	93
-logfileFormat	94
-pause.....	95
-processedFolder	96
-quarantineTime	97
-queue	98
-script.....	99
-scriptFolder (FM).....	100
-sortingOrder.....	101
-UpdateCounts	102
-useDeferred	103
Functional Overview	104
Multiple Monitors	105
On Unix...	105
On Windows...	105
FM.ini	106
FMTransaction.....	106
Installation	107
Testing a new install	107
Queues	108
_job Script Object	109
Command Line Option Access	109
Processing Instruction Access	109
Job Name Discovery	110
The DocOrigin PI	110
Direct-to-Merge PI.....	111
Processing the Job	112
Sample Processing Script:.....	112
Error Handling.....	114
Quarantined Files.....	114
Merge	115
Command Options	116
-{prt}option.....	117

-appendPages.....	119
-attachment.....	120
-blend.....	121
-combineDocuments.....	122
-config.....	123
-convertAttributes.....	124
-createData.....	125
-data.....	126
-dataFileResults.....	127
-documentTag.....	128
-dumpLevel.....	129
-dumpPath.....	130
-dumpTimes.....	131
-duplex.....	132
-ellipsis.....	133
-ellipsisCropLastWord.....	134
-embeddedDefault.....	135
-embedjpg.....	136
-embedPlainTextAsRtf.....	137
-emptyNodeInstantiatePane.....	138
-filter.....	139
-filterOutput.....	141
-filterParm.....	142
-FnfDataExts.....	143
-form.....	144
-fragmentAutoResolve.....	145
-HighlightURL.....	146
-htmTemplate.....	147
-imagePath.....	148
-inputDataType.....	151
-inputDateFormat.....	152
-inputTray.....	153
-jsonDataExts.....	154
-language.....	155
-lineBreak.....	156
-lj4BoundFont.....	157
-mergeCaseSensitive.....	158
-minimizeTextObjectHeightOnSplit.....	159
-mode.....	160
-noStretch.....	161
-numCopies.....	162
-output.....	163
-outputBin.....	164
-outputRef.....	165
-paginate.....	166
-paperType.....	167
-pdfCompress.....	168
-pdfScript.....	169
-preferImages.....	170
-printInDocumentOrder.....	171
-prtOutputRollbackToFile.....	172
-rtfParserVersion.....	173
-scriptFile.....	174
-scriptFolder (Merge).....	175

-scriptTools	176
-shiftX	177
-shiftY.....	178
-showNames.....	179
-showXatwNames	181
-splitSegs.....	182
-spoolerDocName.....	183
-spoolerJobOwner.....	184
-stitchingFastCount.....	185
-stretchFields	186
-symbolSet	187
-testMail	188
-textpresentation	189
-textVerticalAlignmentMethodOnSplit	190
-tracelevel	191
-translate.....	192
-trimData.....	193
-useBuiltinSS.....	194
-useFilterFormList	195
-wininfo	197
-XfdfDataExts.....	198
The Merge Algorithm.....	199
XML Data Files	201
XML Includes	203
XML Attributes.....	204
Image data	205
JSON Data Files.....	206
Combine the Form and Data.....	208
Field Name Matching.....	209
Barcode Options	210
2D Barcodes	210
Linear Barcodes	210
Barcode Bar Widths.....	210
AusPost Barcode	212
Code 11	213
Code 25	214
Code 39	215
Code 128.....	216
DataMatrix.....	217
EAN-13.....	218
MaxiCode	219
PDF417	221
Pharmacode	222
Pharmacode 2Track.....	223
PostNet.....	224
QR Code	225
UPC	226
USPS-IM.....	227
Zebra Internal Barcodes	228
Code 11	228
Code 39 (3 of 9)	228
Code 128.....	229

Standard 2 of 5.....	230
Interleaved 2 of 5.....	230
Industrial 2 of 5.....	231
UPC-A.....	231
PDF417.....	231
Data Matrix.....	232
QR Code.....	233
RFID.....	233
Auto Translate.....	234
Translate Key.....	234
A Translate Example.....	235
Setting Field Values.....	237
Auto Email.....	238
How it Works.....	238
Adding Auto Email to a Form.....	240
Designing DocOrigin Emails.....	240
Images in Emails.....	241
Links.....	241
Testing Your HTML Email.....	242
Beacons.....	242
Just the HTML Please.....	243
Global Fields.....	244
Auto/Embedded Fields.....	245
Merge Licensing Info.....	247
Merge Version.....	247
License Key Renewal.....	247
Merge Output.....	249
The Basics.....	249
Downloaded / Resident Fonts.....	250
Output Filenames / Options.....	253
Scripted Output Destinations.....	254
Multiple Output Streams.....	255
Output to Another Program.....	256
Merge Filters.....	257
Filter options.....	257
Standalone usage.....	257
Executable (non-script) filters.....	257
ConvertDatToXml Filter.....	258
ConvertTxtToXml Filter.....	262
Dat2XML Filter.....	263
DocumentSort.....	264
JavaScript Filters.....	265
DatToXml Filter (Script-based).....	267
CsvToXml Filter (Script-based).....	268
XMLAttrs Filter (script-based).....	269
Codepage Handling.....	271
Multiple Filters Per Run.....	273
Merge External PDFs.....	274
What do you do to achieve external PDF integration?.....	274
What Merge does.....	275
The DocOriginOverlay.wjs script.....	276
Multiple documents per Merge run.....	276

Merge External PCLs.....	277
Achieving External PCL File Integration.....	277
Preparing External Files	278
Step 0: Get Organized!	278
Printing Files in PCL5 Format.....	279
Preparing your PCL file for integration	279
Shortcut for PDF-based External Files.....	279
Limitations	280
Merge Scripting	281
Suggested Scripting Guidelines.....	281
Merge Events	282
Hyperlinks	284
Fillable Forms	285
Single Output File.....	285
Templates	285
HTML Script	285
Input Fields	286
Input Field Type.....	287
Fillable HTML Forms	292
Fillable PDF Forms	298
Pagination	302
Pagination Checkboxes	303
Mandatory	303
Allow Multiple.....	303
Allow Split	303
Allow Break Before	304
Allow Break After	304
Allow Break Between.....	304
Force Page	304
Layout in multiple columns	304
Minimum Height	305
Pagination Generalities	305
Footers/Headers	306
Processing Instructions	309
Profile Files	310
Profile File Format.....	310
Opening a Profile File	310
[@Variant]	313
Transparent Color Image Support.....	314
MultiMerge	315
Html-Adaptive Processing.....	316
Html-Adaptive Template Files	317
MultiPart Output.....	318
Command Options	319
-action.....	320
-adaptiveTemplate.....	321
-mergeargs.....	322
MultiPage	323
Usage.....	323

MultiPage Command Options	324
Notify.....	325
Purpose/Uses	326
Infrastructure	327
Configuring	328
Command Options	329
-context	330
-fileMask.....	331
-themeColor.....	332
-waitTime	333
PCLExtract.....	334
Extracting Documents from a PCL.....	334
Other options	334
PDFExtract	335
Combined PDFs.....	336
Combining PDFs using PDFExtract	337
Extracting Documents from a PDF	338
Extracting Pages from a PDF	339
Selection by Bookmark	340
View Bookmarks	341
Command Options	342
-attach	343
-docs	344
-dumpIndex.....	345
-dumpObjects.....	346
-in	347
-multiSelect.....	348
-onto	349
-out	350
-overlay.....	351
-pageCount	352
-pages	353
-quiet	354
RunScript	355
Syntax	355
Parameters	355
Predefined Vars.....	356
Command Options	357
-allowDJScript	358
Runscript -script.....	359
FormConvert	360
Command Options	361
-adjustLabelWidth.....	362
-config (FormConvert)	363
-containerShift.....	364

-fieldCorrection	365
-ics.....	366
-importFilter.....	367
-in (FormConvert).....	368
-out (FormConvert).....	369
-retainSpaces.....	370
-saveXdp.....	371
-setConfig.....	372
-target.....	373

SendMail 374

SendMail Configuration	375
Immediate or Queued.....	375
Creating but not Sending and Email (-hold)	376
Post-Send Housekeeping.....	376
Parallel Connections.....	376
Option List	377
Sendmail Verification	380
Signing and Encrypting mail	381
Email on Linux	382
Updating cURL via yum	383

Session 385

DocOrigin Inter-process Communication	385
Session File Format.....	385
Script access to Session data.....	385
Merge Command Line Options.....	385

Command Line Processing..... 387

Indirect Parameter Files (.prm files)	388
Conditional PRM Files.....	388
Text Substitution.....	388
Default Parameter files	389
Design.prm	390
Order of Precedence	391
Invoking Command Line Apps	392
.prm Files	392
Convenience Invocation.....	392
... On Windows	392
... On Linux.....	393
Invoking RunScript.....	394
Common Command Line Options.....	396
-alert	397
-alertLevel	398
-alertMsg.....	399
-alertSubject	400
-alertTo	401
-cache	402
-cd	403
-clearLog.....	404
-debug	405

-define	406
-fileMask.....	407
-include.....	408
-locale	409
-logfile.....	410
-logSeverity	411
-message.....	412
-monitor	413
-phase.....	414
-setSession.....	415
-themeColor.....	416
-trace	417
-verbose	418
-\$X.....	419

Scripting..... 420

Script Files.....	421
The Merge DOM.....	422
\$Variables	423
Defaults:.....	423
As of 3.0.001.11.....	423
Script Functions.....	424
Naming Conventions	424
Functions and JavaScript Objects.....	424
Predefined Variables	425
_auto (Automatic Field Values).....	426
_cache (Scripting Object).....	427
_chart (Draw Bar Charts and Pie Charts).....	429
_data (The Data DOM)	449
_document.....	451
_file (Read/Write Files)	459
_fromBase64.....	476
_fromDOUnits	477
_fromXmlString	478
_inlineToRtf (Central "\x" to RTF).....	479
_job (Current Job and Command Line Parameters)	480
_locale (Format Currency, Number, Date/Time)	481
_logf	487
_logfEx	488
_merge (Call DocOrigin Merge)	489
_mergeEmbedded.....	490
_message	491
_metadata.....	492
_odbc (Database Access).....	494
_os (Operating System Functions)	499
_page (The Page DOM)	500
_parser	501
_printer (Output Configuration)	512
_printf	515
_profile (Access Profile Files).....	517
_prompt.....	520
_resolve	522
_run (Execute Another Program)	523

_runNoWait.....	527
_runScript.....	528
_sendmail.....	529
_session (Accessing Session Data).....	532
_sprintf.....	533
_summary.....	534
_system.....	535
_toBase64.....	537
_toDOUnits.....	538
_toXmlString.....	539
_tracef.....	540
_xml.....	541
XmlFile Class (Write XML Files).....	542
XmlInput Class (Read XML Files).....	546
XMLHttpRequest (Send/Receive HTML Pages).....	551
DOM Functions.....	556
domObj.ancestor.....	557
domObj.appendInstance.....	558
domObj.childNamed.....	560
domObj.bookmark.....	561
domObj.clone.....	562
domObj.deleteObject.....	563
domObj.descendant.....	564
domObj.DOM.....	565
domObj.DOMValue.....	566
domObj.fillFromNextPage.....	567
domObj.fillObject.....	568
domObj.getAllTags.....	569
domObj.getObjectsByName.....	570
domObj.getObjectsByTag.....	572
domObj.getObjectsByTagValue.....	573
domObj.getObjectsByType.....	574
domObj.getPickedObjects.....	576
domObj.insertPage.....	577
domObj.relayout.....	578
domObj.reparent.....	579
domObj.setTag.....	580
domObj.getTag.....	582
domObj.sum.....	583
domObj.xhtmlToRtf.....	584
DOM Properties Accessing the Document Structures.....	585
Examples.....	592
Debugging Script.....	593
Techniques.....	593
Colors.....	596
Setting Colors.....	596
Pre-defined Colors.....	596
Script Units.....	598
Custom Design Script Menu.....	599
File Naming Conventions.....	600
Characters Restricted from File Names.....	600

% Placeholders in File Names.....	601
Folder Monitor Instance Name Substitution.....	601
\$X String Substitutions.....	603
Paths.prm	603
Test.prm.....	603
External file include	605
Startup \$'s.....	606
Special \$'s.....	607
\$\$F.....	607
\$\$D	607
\$\$X.....	607
Glossary.....	608
Terms	608
Acronyms	609
Return Codes	611
FilterEditor	611
SendMail	611
Merge	612
PDFExtract	613
FolderMonitor	613
Process Spawning.....	614
Scripting	614
Xfilter	614
Reserved for Unix	615
User	615
Latest ReadMe.....	616
Web Services.....	617
Processing Details.....	618
Jobs	618
REST API	618
SOAP API	618
Job Internals	619
Returned Results.....	620
Limitations	621
The Default Script	621
Direct Merge call	622
Calling The WS.....	622
Prerequisites	624
Deployment.....	625
REST	625
SOAP	625
Configuration.....	626
Logging.....	627
Script security.....	628

DEMO Application.....	629
REST.....	629
SOAP.....	629
Test.....	629

Using DocOrigin

This section, *Using DocOrigin*, aims to cover general how-to-use topics that cut across the nitty-gritty technical information. It is in an attempt to describe how DocOrigin's facilities can be used together to accomplish commonly encountered goals in document generation.

The rest of the Reference Manual aims to cover all technical information about DocOrigin. Please contact support@eclipsecorp.us if there is something we have missed.

For samples, tutorials, and step-by-step instructions on how to use our software and web apps, see the [User Guide](#) (login required).

You can also read our technical blog, [Pane Talk](#) (login required), which covers advanced topics in technical detail.

Style Conventions:

Convention	Meaning
bold	GUI buttons, menu items, and dialog tabs keyboard shortcuts function returns
<i>italics</i>	(As of <version>) (Mandatory) (<Program> Only)
monospace	options, code, commands, functions, script file names, paths and extensions
<u>underline</u>	emphasis
UPPER CASE	abbreviations
Capitlziation	program names Design object names
"quotes"	jargon, introducing a new term
<i>green</i>	values and dynamic parts of options
	between possible values
[]	optional parts of values
{ }	option modifier
+	keyboard shortcuts
>	menu navigation
.../	Used before "DO" or "User" folder in file paths to indicate the user's chose install directory

Getting Started

Ideally you have been directed to this page via the post-install notes that pop up once you've done the two minute install of DocOrigin. So you've just completed an install – now what? To make life easier on yourself, per the post install instructions, you should copy the DO.bat (or DO shell script on Unix), which is installed in the DO/Bin folder of your chosen installation location, to somewhere in your path.

In the following, ... / means your chosen installation folder.

Proof-of-Life

Browse to the .../DO/Samples folder and double-click Sample_Invoice.bat . That will run the DocOrigin Merge tool on a supplied sample form with some sample data and produce a PDF. If you wish, try clicking on some of the other samples in the folder.

Your First DocOrigin Form Design

To run the DocOrigin Design tool, browse to the .../DO/Bin folder and double-click Design.exe or just use the DocOrigin Design icon on your desktop. Hit the **New** button at the far left of the toolbar. Now you will see a series of tool icons at the right of that toolbar. Text Label, Field, Box, Line, Barcode, Image, Table, Group, Pane, and Container. They have tooltips so they won't be hard to identify. Click on one of them, let go. Now, having selected the tool, drag out an area on the "canvas" – the big center portion of the Design screen. See the [Design](#) section for more information. When giving fields names, note that they must match the exact case as used in the data files. XML is case sensitive. Therefore, so is DocOrigin.

Preview Your Design

After you've put a few things down on the Design canvas, use the **Tools>PDF Preview...** menu item (or hit **F5**). Let Design supply data for you – just hit **Preview**. There you go, you've created your first document. If you explore the menus you will see how to add Pages, align objects, and set the grid size. There are also many settings in the **Object Properties** and **Common Properties** dialogs on the right side of the screen to explore.

No "Preamble"

If you are used to JetForm Output Designer, then here's some advice. Don't look for the preamble, there isn't any. It just works.

Terminology: Panes & Containers

DocOrigin believes in dynamic forms. Most things are not rigidly placed on the page, rather, the contents of a document flow down the page and possibly overflow onto more pages as necessary. You can put objects directly on the page but usually you put them in what we call "[Panes](#)", and what other software has called "subforms" or "fragments". Panes look like panels that run from the left side to right side of the "[Container](#)". Generally, a Container covers pretty much the whole page, but leaves some room around the edges for aesthetic purposes and/or to cover off the area that can't be printed on by some printers. You may have noticed that when you clicked "New Form" you start with a Container and a Pane. There can be only one Container per Page, but you can have multiple Panes within a Container. When data is merged with the form, Panes will be instantiated as necessary and they will "flow" down the Container until it is full. Then, a duplicate Page and Container are created and Panes continue flowing down that Container. Notice there are **Overflow Footers...** and **Overflow Headers...** buttons in the **Object Properties** dialog.

The order in which you layout the panes is the order they will appear in a document. The area on the left of the Design application is what we call the **Form Explorer**. It shows you the structure of your form. You can move things around in there if you want to change the order of your Panes, for example.

Scraping Data Off of Reports

If you have a spool/overlay file or a print image as produced by some other application and you want to "scrape" data off of that for use with DocOrigin Merge, use [FilterEditor](#).

License Keys

DocOrigin is a licensed product. In order to have full use of the product you require a license.

Evaluation Spoiler Stripe

The primary usage of DocOrigin is to do document generation. If you have a copy of DocOrigin, but do not have a valid license then the generated documents will all have a rather large "**Evaluation**" stripe across them. Once you have a license, that spoiler is not generated.

License Key File

The license is embodied in a key file (.xkey or .key) that you receive from the seller. It is placed next to the Merge executable, i.e. in the .../DO/Bin folder of wherever you chose to install DocOrigin. Starting with version 3.0.003.29 (early 2015) the key file formats changed somewhat and newer .key files began to be used. The older .xkey files continue to work, but will gradually be phased out. There is no requirement to switch to the newer format.

Time-Limited Evaluation License

It may be that you have requested and received an "Evaluation License" which means that it is time-limited and that the Evaluation spoiler stripe will return after a certain amount of time.

Permanent License

If you have purchased a license, then your key file is 'forever'. You can run the version of the software that you have installed forever. The spoiler stripe will never come back.

Maintenance and Support

However, Maintenance and Support (M&S) privileges are not forever. They must be paid for according to your negotiated purchase agreement. M&S means that you can not only get support but that you can get product version updates with fixes and new features. If you let your M&S expire then you no longer have any entitlement to new product releases. The one you have will continue working but new releases would show a spoiler stripe.

Product Release Issue Date

Product releases each have an 'issue date' embedded within them. You can download and install new product versions whenever you wish. As long as the issue date of that product version precedes the expiry date of your M&S agreement, you will be able to use that new software version and not encounter a spoiler stripe. However, if you let your M&S agreement lapse but install a product release with a later issue date, then you will get the spoiler stripe again, since you would not be entitled to product updates at that point — having let your M&S agreement lapse.

A product release that is running in your shop today will continue to run without need of a new key file. That is regardless of whether you let your M&S agreement lapse. If you do let it lapse then you won't be able to use any new product versions (versions issued after the date that your M&S lapsed.)

No Need for New Keys Unless Upgrading

There is no requirement for you to get a new license key file every time (annually) you renew your M&S agreement. You *may* need a new key file when you want to upgrade to a new product version. Whether you need one or not depends on whether the issue date of the new product release is later than the M&S expiry date encrypted in your current license key file. Assuming that your M&S is paid up-to-date there is no further charge for getting an updated key file. In fact with versions since 3.0.003.22, if you have internet connectivity, then you can get a key that reflects your latest M&S agreement expiry date simply by running: `DO Merge -renew`

Of course, the M&S has had to have been paid, received, and recorded.

Temporary Emergency Keys

Facilities exist for you / your distributor to get temporary keys to tide one over across some emergency situation. If your system does not have (is not permitted) internet access then contact your distributor for keys (temporary and permanent).

Keep M&S Up-to-Date

DocOrigin is under constant development. New features come into being frequently. Fixes, as required, are quite prompt. However, it is your choice as to when you wish to upgrade to a new release. That could be frequently or very infrequently, it's your choice. Note that recovering from a lapsed M&S state, so as to gain access to new features, is something you need to negotiate with your distributor. It could easily require that a whole new license be purchased.

Multiple Outputs

In a single run of DocOrigin Merge, one can produce multiple outputs. There is no restriction to creating only a single file. The variety that is possible here seems quite endless.

One Output Per Run

Let's take a simple case of producing PDF output. The `-config` option you give to Merge will reference a PDF configuration file. Quite often, so often in fact that it is the default if no `-config` option is provided, that configuration file will be `Default-PDF.prt`. Of course, you may have customizations, typically with additional fonts, and you might say `-config MyCustom-PDF.prt`. Whatever the case, you do nominate, explicitly or implicitly, an output configuration file (a `.prt` file).

If your data stream contains only one document's worth of data, then yes you will get only one PDF file out. If your data stream contains the data for many documents, typically each housed between `<Document>`, `</Document>` bookends, then, for the `Default-PDF.prt` output configuration, you will still get only one output PDF!

Why is that? Well, well, ... *we agonized over this long ago decision* - the `Default-PDF.prt` stipulates

```
<CombineDocuments>Yes</CombineDocuments>
```

That means that the output for all of the documents will be combined into a single, very archive-like, PDF. That's great, if that's what you want, and you very often do. You might ship that whole archive off to your print shop, or indeed use it as an archive. It is very easy to use the PDFExtract tool to extract any documents of particular interest.

One Output Per Document

But what if you want an individual PDF for each document's worth of data? Well, specify an overriding:

```
-PDFCombineDocuments No
```

in your invocation of Merge, or if you like, specify `<CombineDocuments>No</CombineDocuments>` in your custom PDF `.prt` file.

Now we've got a bunch of files, one per document, coming out in a single run of Merge. That can be, and often is, quite desirable. It's particularly desirable if you want to distribute those outputs in different ways! You might use `_sendmail()` to email them, or you might direct them to individual web folders based on client account information in the data stream. Good stuff.

Different Output Formats

Now that we're producing a different output (file?) for each document, who says they should all be the same format, PDF in our example? Surely, I don't have to split up my data streams into one for PDF, one for PCL, one for HTML, one for PS. Of course not. That would be crazy. We expect that you will want to cater to your end users' communication preferences. Some will want paper, some email, and some web postings, either in PDF or HTML.

A simple case is that of PDF + paper (let's say PCL, as an example). To accomplish that in one run of DocOrigin Merge, you simply specify a semi-colon-separated list of output configuration files as your `-config` option. For example:

```
-config Default-PDF.prt;Default-LJ4Color.prt
```

Now Merge is fully prepared to emit some outputs in PDF and some in PCL. You might look at the document's data for the account's communication preference and choose to create PDF for one document, but PCL for another. How do you do that?

The secret sauce is in the *Ready-To-Print* event. Here's something I coded recently in my SE Support role.

```

_logf("Print? %s", _cache.Print);
_logf(" Pdf? %s", _cache.Pdf);
_logf("Email? %s", _cache.Email);
_printer.setActive(false, "LJ4");
if (_cache.Print == "Yes") {
    _printer.setActive(true, "LJ4");
    _printer.setOutputFile("$U/Output/[NCLI]/EXT_[NCLI].pcl", "LJ4");
}
_printer.setActive(false, "PDF");
if (_cache.Pdf == "Yes") {
    _printer.setActive(true, "PDF");
    _printer.setOutputFile("$U/Output/[NCLI]/EXT_[NCLI].pdf", "PDF");
}
if (_cache.Email == "Yes") {
    _printer.setActive(true, "PDF");
    _printer.setOutputFile("$U/Output/[NCLI]/EXT_[NCLI].pdf", "PDF");
}

```

Naturally, I had included both an LJ4 and a PDF output configuration in my semi-colon separated `-config` option. We needn't worry about how the `_cache` variables were set. There could be lots of ways, and one might well refer directly to a DOM element like:

```
_document.header.commPref._value
```

or whatever your data stream holds. Naturally, you don't have to log out the values either, at least not after you have finished debugging!

i Oh, <sigh>, we have always called these output configurations "*drivers*". I hate that. They are not drivers in the sense of device drivers – you know those other things that give you so much grief, and need constant updating! These are not them. These you set and forget. But the "drivers" word is in my brain.

Output drivers can be active or inactive. If they are active, output will be produced. If they are not active, output will not be produced. So the point of this code is to set one of these "drivers" active and the other "inactive".

Line 4, initially says that we do not want PCL output for this document, but lines 5 and 6 say that if "print to paper" is desired, we will make it active. Pretty simple.

Line 7 defines where I want the output to go. Naturally, as we are producing multiple outputs per run, we can hardly specify a single hard-coded name! So really, what line 7 does is to set an output name *template*. It references field names in square brackets causing dynamic text substitution to happen. It can reference a whole pile of `%x` type date/time elements too. Look inside any `.prt` file for a list or see: [File Naming Conventions](#). And then there's script. You are in a scripting language. If for some astonishing reason the above flexibility isn't enough, you can certainly script your way to the perfect output file name.

Note that whether PDF is wanted or email is wanted, I make the PDF driver active. After all, I intend to email a PDF. It might be that if email is desired that I would direct the output to some staging area, then email it, then delete it. Whereas if email wasn't desired, I might direct the output to some client web folder.

I think you can see that this bit of script is not rocket science. You are empowered to create outputs of different formats in a single run.

i It is critically important to know that as Merge produces each output it is **NOT redoing** the merge of the data and the form design. That would take processing resources and it entails all sorts of side effect risk. Even something as simple, but legally important, as the recorded time-of-printing could differ. DocOrigin Merge does the data merge only once and does each output generation from its internal Document DOM. All documents are created equal.

Multiple Outputs Per Document

The above shows that you can create a different format output for each document's data. But more than that you can create multiple outputs (of the same or different formats) for each single document.

You'll note in my code sample above that making PDF active does not mean that LJ4 must be inactive. Not at all. It's quite fine to have multiple drivers active and have Merge emit multiple outputs for whichever documents deserve to have them.

Multiple Outputs Per Document 2

So far, the notion of having multiple outputs per document has been for the purpose of creating different format outputs. But what if I want to produce two PDFs say? Well, Merge can do that, or three, or ...

Try a `-config` statement like:

```
-config Default-PDF.prt;Default-PDF.prt
```

Yes, that's right, I used the same `.prt` file twice. What happens then is that Merge let's you refer to the first one as, in this case, PDF, and the second one as PDF2, etcetera if there were more than two of the same type.

Now you could produce a PDF to be sent to the account's folder, and another to be sent to the accountant's folder. Of course, you could do file copies too! A much more common usage is to have one `.prt` have `<CombineDocuments>Yes</CombineDocuments>` and the other to have `<CombineDocuments>No</CombineDocuments>`. That way you can produce the combined archive PDF and individual mailable, postable PDFs at the same time. Now that's handy!

Of course this technique expands to multiple format outputs as well. Endless possibilities.

Usage

Most of the usage for multiple outputs is pretty obvious but perhaps a couple others are worth mentioning explicitly.

You may very well want to switch between output destinations **based on page count**. That information is definitely available to you in the *Ready-to-Print* event and so it is quite easy to direct small number of pages documents to one paper handling destination and larger number of pages documents to a different paper handling destination. Different paper folding devices and envelope stuffing machines can then be applied. Savings are very possible in terms of postage and envelope costs.

The **DocumentSort** filter can be run seamlessly on the batch of data, sorting on any number of keys. This operates very quickly and then... you can easily **break up your outputs based on a change of "region" or "branch"** or some such thing. The "western region" can get all of its documents in one combined PDF while the "southeast" and "mid-west" get their own set of documents. Of course you could be creating an omnibus, all regions, output at the same time. It may make sense to sort on zip-code and achieve postage rate savings that way.

It's so easy to overlook. A form design has a *Main* Section but it can also have a *Summary* Section. You can design panes for that Summary Section as you see fit. Data that you collect throughout a batch can then be presented as a Summary Section report. And of course, that **Summary Section output** might have a different destination than what applied for the Main Section.

It might be that certain documents contain data values that are outside the norm or have larger-than-usual dollar amounts. Perhaps you would direct those to a **'special attention' output** location, and/or to a watched folder that participated in a review workflow operation.

Output Configuration Names

Also known as "Printer Names".

In any `.prt` file there is a `<Printer>` section. They look like:

```
<Printer Name="XXX" Type="XXX">
```

Typically, those XXXs are the same, though they needn't be. When I coded

```
_printer.setActive(false, "LJ4");
```

the LJ4 was referring to a Name="LJ4" setting in one of the referenced .prt files. Naturally, if you reference the same output configuration file multiple times, then that same name will occur multiple times. As said before, when that occurs, Merge will automatically, name the second occurrence as XXX2, the third XXX3, and so on.

Sending Mail

Conceptually, it is pretty easy to understand the notion of using document generation to produce a document and then using an email tool to send that output somewhere. Nothing magical there, also totally unintegrated, requiring two tools to understand and peer into the data to figure out what to do.

Far more exciting is the ability to **send mail from within a form**. As the [Multiple Outputs](#) topic has shown, DocOrigin Merge can produce all sorts of outputs and can specify where those outputs are to go. One of those distribution mechanisms is "email".

The email addresses are surely in the same data stream as Merge has just processed, and therefore they are in the Document DOM, easy pickings for use with the `_sendmail()` function. How does this happen?

The place for orchestrating the sending of mail is the *End-of-Document* event. You could use any object, but the form object itself is likely the one whose *End-of-Document* event you should put the script into. Sometimes folks use the email address field as the object to hang the script off of. Consider...


```
if (_document.customer.email._value != "") {
  var args = {};
  args.to = _document.customer.email._value;
  args.from = "CustomerRelations@mycompany.com";
  args.subject = "Your statement: " + NCLI._value + NATR._value + NSEQ._value;
  args.attach = _printer.getOutputFile("PDF");
  args.text = "@$$F/PlainTextEmail.txt";
  args.html = "@$$F/HtmlEmail.txt";
  // args.cidfolder = "$$F";
  // args.hold = "Y";
  _sendmail(args);
}
```

What's happening in that code? Well, really it's just setting up the arguments to a `_sendmail()` call, which it then makes on line 11. See [Sendmail](#) and `_sendmail`.

Line 1 predicates the whole thing on whether we have an email address to send to. You may have other criteria. You may even dynamically look up the email address based on some other customer account id, or some such thing. The intent is pretty obvious though.

The Email From Address

A from address is required but is often set in the [Sendmail Configuration](#) .prm file. The address can be in the format: "friendly address <actual address>" as you will often see on emails these decades. If you provide only the *actual address* without it being enclosed in angle brackets, then the address will automatically be suffixed with <actual address>, essentially a repeat.

 Some versions of Microsoft Exchange do not have the wit to handle current style email addresses that include the friendly name. If you have issues with your from address and are using Exchange, then specify your from address as "<actual address>". When DocOrigin sees the '<' it will presume that a friendly name has already been provided and so will not add any suffix. That lets Exchange be happy.

The Email Subject

In line 5, you can come up with any subject line you like. You can be bland, or you can personalize it based on the data at hand.

The Email Attachment(s)

Line 6 is pretty simple; you want to attach the PDF file that was just produced for this document. Dead easy. Want more than one attachment? Perhaps, the information in the data "suggests" that you attach a specific brochure. The `args.attach` value can be a semi-colon separated list of attachments. So, attach away.

The Email's Text-Based Body

Line 7 could be as simple as:

```
args.text = "This is your statement";
```

but that wouldn't be very communicative. You can formulate the text-based body of your email any way you like. Clearly, you could build up a string that contained all sorts of information taken from the data stream. This particular example says to get the text-based body of the email from a file called `PlainTextEmail.txt` which resides in the same folder as the form design (that's what `$$F` means).

One way to come up with the text-based body for an email is to define a text label in the form and have it include as many [square bracketed] field references as applicable. Then the code can be something like: `args.text = _document.emailstuff.bodytext._value;` That postulates a pane named `emailstuff` that is marked invisible in the form design. It makes it easy to design a text-based body and see what it looks like on the Design canvas. But scripting works too.

There's a clever combination that we owe to an SE. The label text field sets its value to `[@$$F/PlainTextTemplate.txt]`. That template may include the usual [square bracketed] field references. The cool thing here is that those contents get read into the text label and then Merge automatically does [field] substitution. This makes it pretty easy for a non-form-designer to contribute to the overall effort. **The relevant department can not only contribute but take control of what the email message text will say. No form design skills are required.** When that department posts the latest version of the referenced template file, all will flow nicely, with no forms department involvement required.

As of 3.1.001.01 much of this is more easily provided by the `_email_` fragment introduced in that version to facilitate sending email without using any script. See [Auto Email](#).

The Email's HTML-Based Body

Line 8 is really just more of the same. Just that here we are supplying the value for the HTML-based portion of the email body. It is not a requirement to provide both textual body, and HTML-based body, but perhaps it will offer a competitive advantage to do so. Again an external department can take on the effort, and gain the control of defining what those vital contact points with the customer look like.

Images in the Email HTML body

Line 9 is commented out but it brings to mind a powerful feature related to the HTML-based body. **The html body can include embedded image references.** These take the form of

```

```

Perhaps it would include a height and width specification too, your choice, normal HTML.

The DocOrigin `_sendmail()` function will detect such constructs and embed the nominated image, or images into the mail package that is sent. And `_sendmail()` will inject the necessary plumbing to have that image show up where it is supposed to. Likely the recipient will get an "Images are not shown below, click here to show images" message in his email -- depending on his email client software. Normal operations; the images are visible.

Where the `cidFolder` specification comes in is to tell `_sendmail()` where to look for images. Mind you, most often the `cidFolder` specification would be housed inside the **DocOriginSendMailServer.prm** file. But that choice is yours.

Summary

In sum, you can produce a very complete, and tailored e-mail message, and you can have external departments take ownership of many of the details. Do recall that this is happening as the Merge run proceeds, no separate after-the-fact process is involved.

If you do want to do after-the-fact-processing, then feel free to use **RunScript** and its ability to invoke `_sendmail()` in the same way -- though it wouldn't have access to the Merge-created Document DOM since it would be running as a separate process.

i If you wish to send an email attaching a combined PDF, then that needs to be done at the *End-of-Job* event (not *End-of-Document*) since the PDF will be open throughout the run, not being complete until end-of-job.

Debugging

Line 10, `args.hold = "Y";` is strictly for debugging. With it you can force the DocOriginSendMailServer to not send the message but to keep it in the "outbox". That way, you can examine the mail packet during your development phase and see if all is well. The line would certainly be commented out in production. This could be an across-the-board debug setting in DocOriginSendMailServer.prm or, as is shown here, it may be that only certain messages are to be held up.

Email Sending Verification

See [Sendmail Verification](#).

Email Configuration

See [Sendmail Configuration](#).

See Also

[Auto Email](#)

RTF Support

We believe that support of RTF (Rich Text Format) is very important. But let's be clear that DocOrigin support of RTF is limited. What is there is amazingly useful, covering of most things that a typical business form needs. For the most part, it is used for Terms & Conditions text, which is usually fairly bland, requiring only bolding, possibly italicization, and font size and typeface changes. The selected typeface must be included in the current output configuration (.prt). DocOrigin supports all of that, but not a lot more.

We do support superscript, subscript, and basic underlining as well as color changes within the text paragraph.

We do preserve tab characters and support a basic set of tab stops, which happen to default to 1/2", so an embedded tab will *sort of* work. If we encounter the start of a paragraph in the RTF, we interpret that as leaving a blank line and then starting at the left margin. If already at the start-of-line, the same except no blank line is added.


That's all. **DocOrigin is not a word processor and doesn't profess to be one.** We recognize only that RTF that fits nicely into our existing business form text capabilities. See the chapter below for a more detailed list of limitations.

The limited tab support often tricks people into believing we have support for RTF-defined lists. We do not. Some system engineers have worked out ways to accomplish this feat but it is not through direct usage of an rtf-defined list.

As of 3.1.002.06, it's possible to enable parsing of paragraph formatting commands. See [-rtfParserVersion](#). Supported are per paragraph indentation, tab positions, alignment, and space before/after.

DocOrigin Merge strives to and *seems to* succeed in ignoring all of the RTF syntaxes that it doesn't understand. But you would be well advised to keep your RTF to a much more reasonable subset. For example, at worst, use Wordpad, not Microsoft Word to edit your RTF. The bloat that is in MS Word is unbelievable. This generally makes it impossible to find your actual text or to do any debugging since the RTF string lengths will be huge and unmanageable. **KEEP IT SIMPLE.**

DocOrigin supports rich text format directly in text objects through the use of the Common Properties Tab. In addition, rich text formatting is retained from your clipboard when copied from other files such as PDFs, Word documents, and other DocOrigin files. Lastly, external rich text files (RTF) can be directly imported into a DocOrigin file (see chapter below).

 Note that it's a good idea to resave your rtf fragments in Wordpad before using it in DocOrigin. This will significantly simplify the RTF structure and make it possible to parse with not sophisticated RTF parsers.

Like all textual data fields, RTF text fields support square-bracketed references to field names, in the style [fieldName]. But such a string cannot have any RTF syntax within it. Don't bold the field name in your RTF editor, or make any such rich text changes. It must be literal. Your very best course of action is to open your RTF in a plain text editor and verify that your [fieldName] reference is intact. Looking at your RTF in a plain text editor will be instructive and will surely lead you away from RTF editors that inject so much extra syntax.

Limitations

Supported Rich Text Features:

The following features of rich text formatting are supported:

1. Color
2. Font | The available fonts are defined by the Printer Configuration File selected for the document as well as the fonts installed and the DocOrigin Design Machine and Server.
3. Bold
4. Italic
5. Single Underline

6. Point Size
7. Strikethrough
8. Subscript
9. Superscript
10. Paragraph Formatting:
 - a. Line Spacing
 - b. Paragraph Space Before and After
 - c. Hanging Indent
 - d. First Line Indent
 - e. Tabs

Note: Paragraph formatting applies to the entire text object. Meaning, you cannot have a regular paragraph, followed by an indented paragraph with hanging indents.

Unsupported Rich Text Features:

1. No Tables
2. No Images
3. Bullets
4. Outlines
5. Numbering

Importing Rich Text Files

When importing external rich text files into a text or field object, the paragraph formatting is defined in the DocOrigin object, such as line spacing and tab stops. Therefore, if you want .25" tabs, define the Tab setting in the DocOrigin object that the RTF is being imported into and simply insert a literal tab in the RTF file. The syntax for importing the external RTF file is `[@path/filename.rtf]`.

The auto-bullets and the auto-numbering feature is coming shortly. In the meantime, these features can be accomplished using the desired character and hanging indents. Be sure they define the same value for the Hanging Indent and Tab stop. Using the Character Map application is useful for capturing the desired character(s).

See Also

[-rtfParserVersion](#)
[Merge External PDFs](#)

Barcodes

DocOrigin handles various barcodes. As for all things that are "printed", referring to your favourite .prt configuration file is a good course of action. Each of them contain quite a vast quantity of information in comments.

That is where you will find which barcodes are supported for the selected configuration.

If you are using barcode type X then it is presumed that you know all about barcode type X. That is, that you know what data is required by that barcode type. DocOrigin is not the source for specifications on various barcode types. Often those specs are veritable scientific journals, very hard to follow, so you are required to understand the barcode type that you are trying to use, most especially in terms of the data that such a barcode type requires. Given the right data, DocOrigin will render it.

Knowledge of how to render the various supported barcode types is embedded in the DocOrigin software. You need supply only the data. However, certain variations in barcode rendering are available, (it varies by barcode type). You can see the default configured settings for those variations in the relevant .prt file. Do check there first.

Whether it is barcodes, fonts, or layout style, most organizations desire, if not require, some standardization across all of their documents. So generally, barcode configurations are defined once at a site and never looked at again. These configurations are defined in the site's .prt files and used by DocOrigin to do the applicable rendering. Those configuration options are meant to, and do, apply across the board for all instances of the use of that barcode type, with that .prt, in all forms.

In some cases, organizations choose to vary a barcode's options. For example, they might choose to include the text in one rendering, but choose to exclude it in another rendering. While this (not following a site standard) is somewhat unusual, DocOrigin accommodates it through the designer's use of the setTag function. See [domObj.setTag](#) (previously named setParm).

See the Merge [Barcode Options](#) for details on using various barcodes and their options.

Overview

DocOrigin is a set of applications that format customer application data into professional-quality documents and forms. DocOrigin can create a wide range of documents, such as:

- general business forms like invoices, shipping documents, packing slips and purchase orders
- government and institutional forms
- customized banking and insurance statements and documents
- Service Bureau document production

DocOrigin can generate printed output for a variety of printers, or create multi-page Adobe® PDF documents with indexing and security features. Printer Drivers can be customized to meet the needs of new models to adjust paper tray selection etc.

All of the DocOrigin applications run on the Microsoft® Windows® platform. Essentially all of the non-GUI applications run on Linux-like OSs and some, most notably DocOrigin Merge, also run on AIX (and IBM i/PASE).

Platforms

DocOrigin is available for the following platforms

- Windows 64-bit (this platform alone, includes the GUI tools; e.g. Design, FilterEditor, ...)
- Linux 64-bit, e.g. Red Hat, Ubuntu
- AIX 64-bit
- AIX/PASE for the IBM i

Minimum Configuration Recommendation

DocOrigin requires very little as a minimum if you simply wish to output documents occasionally, but we presume you are part of an enterprise that produces millions of documents on a production basis. Still, DocOrigin's recommended minimum requirements are quite modest. In the context of an Intel, 64-bit processing system:

- Memory: 4GB (Ok); 8GB recommended
- CPU: i5 or i7 class
- Processors: quad-core (for running multiple instances of FolderMonitor)
- Disk: 7.2K rpm, (heavy duty usage – 15K rpm or SSD), keep 20GB free
- Microsoft C-Runtime library for Visual C++ 2017

Linux-like systems generally require somewhat less than Windows systems.

DocOrigin will run on much less, e.g. any modern \$500 laptop, but the above is meant for production purposes and can be readily scaled up as necessary for enterprises with demanding and/or high-volume printing/output needs.

You will need Java 8 if you are going to use Merge External PDFs or DocOrigin WebService functionality.

(As of 3.3.002.01) Windows GUI applications are dependent on .NET 4.5 framework.

Merge Architecture

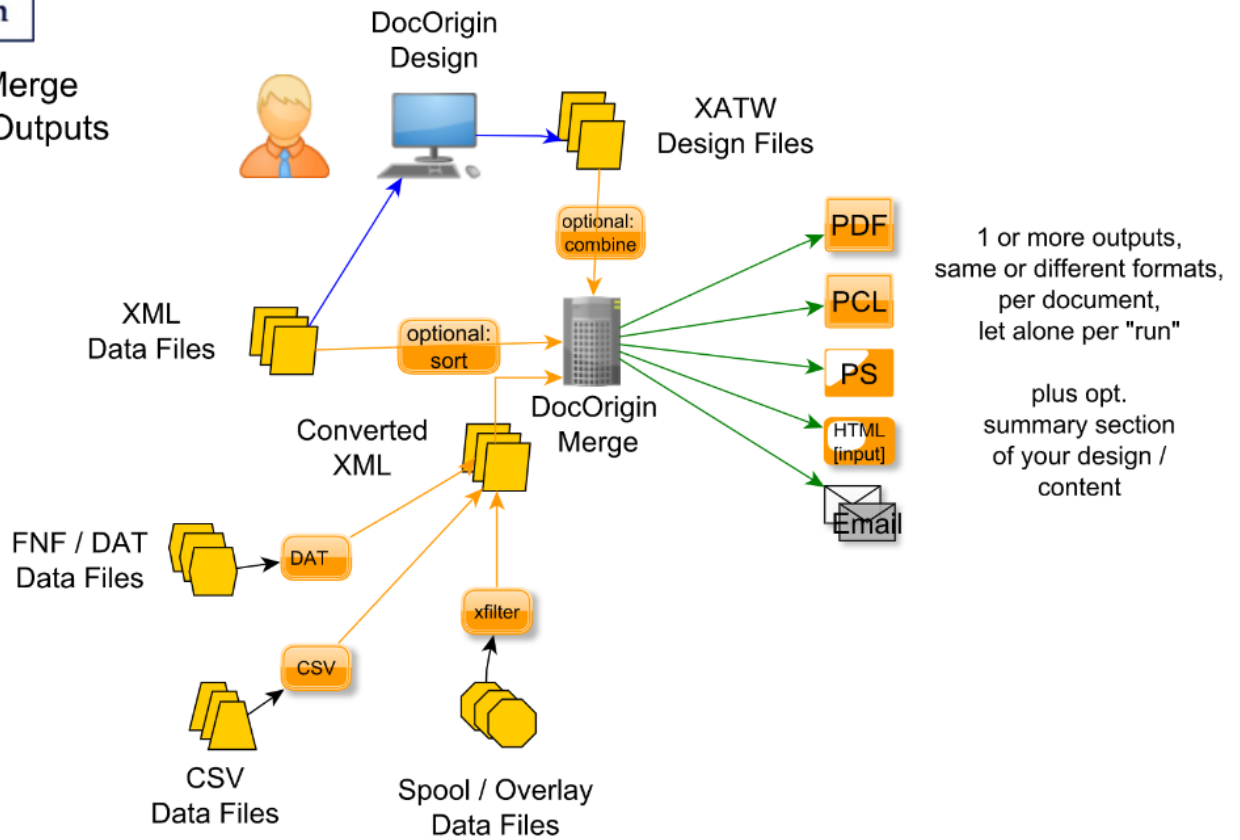


DocOrigin Merge Inputs and Outputs

Template Design

Merge Input

Merge Output

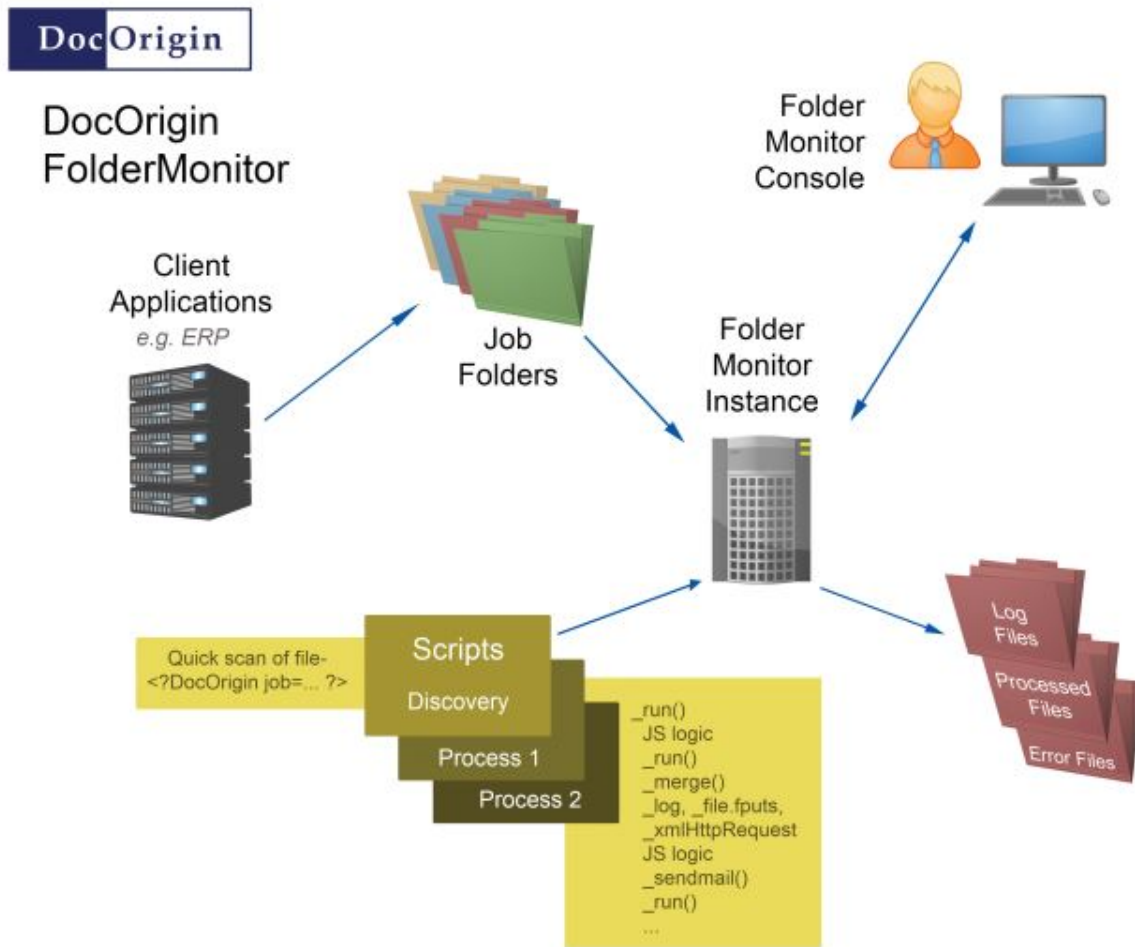


A Classic Merge Command Line

```
DO Merge -form $F/Sample_Invoice.xatw -data $F/Sample_Invoice.xml -output a.pdf
```

We're not suggesting you run Merge by madly typing such command lines in over and over, but that line alone tells most Dev folks volumes about getting things done via Merge.

FolderMonitor Architecture



This is just **one** instance of FolderMonitor, you can have several of them operating completely independently of each other.

The `_merge()` line in the abstracted Job Processing script is putting all of the power of Merge (see previous architecture picture) at your fingertips.

One could easily view segments of the Job Processing script as steps in a straightforward workflow. Of course one of those steps might be to communicate with your existing corporate workflow, your OMS, or your source version control, or document repositories. You decide how much you wish to do where, and integrate not duplicate, or just as bad, live as separate islands.

Copyright Notices

See COPYRIGHTS_and_DISCLAIMERS.txt file within your installation under DocOrigin\DO\Bin folder.

Internationalization

DocOrigin is designed to allow a wide range of language and country (locale) support. All text is handled as multi-byte Unicode text. This allows use of virtually all standard left-to-right languages including Chinese, Thai, and other Southeast Asian fonts. Design allows locale (language/country combinations) to be specified at the form and individual text string level. Locale-specific word and character analysis routines provide correct word-wrap and editing capability.

The following describes internationalization issues, including Unicode, and highlights those areas where Merge meets the boundaries of its operating environment, printers, and so on.


Definitions

Term	Definition
Internationalization	The engineering of software to support multiple languages and multiple countries (locales), with consideration for areas such as language support (some sort of Unicode characters), time, date and number formatting, and the ability to easily localize (translate) maintain all components of the software.
Unicode	An international standard for the assigning of character glyphs (symbols) to numeric codes.
UTF-32	A comprehensive enumeration of fonts, where each character has a unique 32-bit numeric code. Although very complete, UTF-32 is rarely used.
UTF-16	A 16-bit-per-character storage of text. In its simplest form, this format can store just over 63,000 characters. (216 – 2048). This is more-or-less what is known as UCS 2. The full UTF-16 format also includes a provision for more that 63,000 characters by using a scheme of "surrogate pairs". Two 16-bit characters are used to encode a single UTF-32 Unicode character. (The values in the range xD800-xDFFF are used, which accounts for the missing 2048 codes mentioned above). The use of surrogate-pairs (or just surrogates) is necessary to encode some east Asian languages (Chinese).

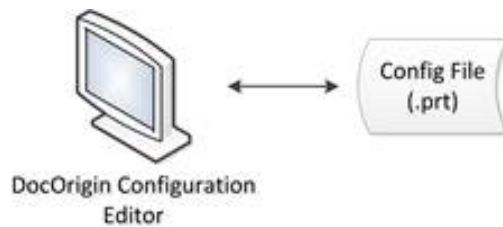
Considerations

- DocOrigin Design allows to enter any 16-bit Unicode character by typing in character HEX code and pressing Alt+X. For example in text label type in "99aa" and then hit Alt+X.
- DocOrigin applications use 16-bit UTF-16 to store all form text. Most code, including the text word-break logic, will recognize surrogate pairs of characters and handle them correctly.
- Merge uses TrueType fonts as a source of font metrics (character widths). This format is restricted to 65,535 characters, and cannot handle surrogates.
- All file I/O in Merge reads and writes 16-bit characters. (The file I/O routines determine if a single-byte file has been opened, and convert it on-the-fly.)
- The error messages for Merge are stored in an XML file (16-bit) that can be translated for different locales.
- PCL Printing — Merge supports only 8-bit PCL printers. Any Unicode character above 127 is mapped using a PCL symbol set mapping which is sent to the printer. Any character that is available in the printer can be printed by using its corresponding Unicode character in the data or form. Merge uses a combination of downloaded TrueType fonts and symbol sets to display Unicode fonts (for example, Chinese).
- PDF and Postscript Printing — the Merge implementation of PDF and Postscript relies on the Adobe list of standard glyph names. PDF output will use glyphID encoding to display all other Unicode characters.
- Script — The JavaScript language is able to handle 16-bit Unicode characters.
- Merge includes scripting extensions for date, time, and currency conversions using the IBM ICU development system.

Configuration Editor

 DocOrigin Configuration Editor, like all DocOrigin GUI apps, is supported on Windows 64-bit only.

DocOrigin Configuration Editor (ConfigEditor) is an interactive Windows application that assists in editing and customizing printer configuration (.prt) files. Also called "config" files or "PRTs", they define fonts, printer options, and various default features that are shared across forms. Each form has an affiliated PRT, and Design uses this information to provide the appropriate features that may be incorporated into the form, for example, the available fonts, as well as check box and radio button shape and size. Merge uses this information to enable and guide certain output generation features.



Configuration files

The following are some of the settings defined in Config files:

- Fonts. Fonts must be explicitly defined in a form's associated config file to be used in Design. Merge must be able to extract font metrics (character widths) from the font files, which are listed in the PRT. It also contains specifics for embedding fonts in PDF files. As well, it is possible to specify which fonts will be treated as always available on the particular output device or in PDF, and which must be downloaded or embedded into the document.
- Page sizes. Specifics of the available page sizes (Letter, Legal, A4, custom, etc.) and the margins for the unprintable area, which are specific to the printer, and cannot be changed. This information enables Merge to accurately position the output on a printed page.
- Printer paper handling options. These include duplex printing, input trays, output trays, and paper types. For direct PCL and Postscript printer drivers, these options allow wider control over paper handling than normal Windows printing.
- Checkbox and radio button options. Specifics for checkbox and radio button size and appearance.
- Individual or combined output files. Specifics that determine whether multiple data sets create individual output files or a single combined file.

Several default configuration files are shipped with DocOrigin and can be found in the DO/Config folder. These files are distinguished by a file name that includes the prefix `Default-`, such as `Default-PDF.prt` and `Default-PCL5Color.prt`. These files are locked (read-only) when installed. They may be used as a template for creating custom configs for an organization's requirements. Any changes must be saved under a different file name in the User/Config folder. This ensures that the original, unaltered configuration files can always be used for testing or troubleshooting. Do not give your custom files a `Default-` prefix.

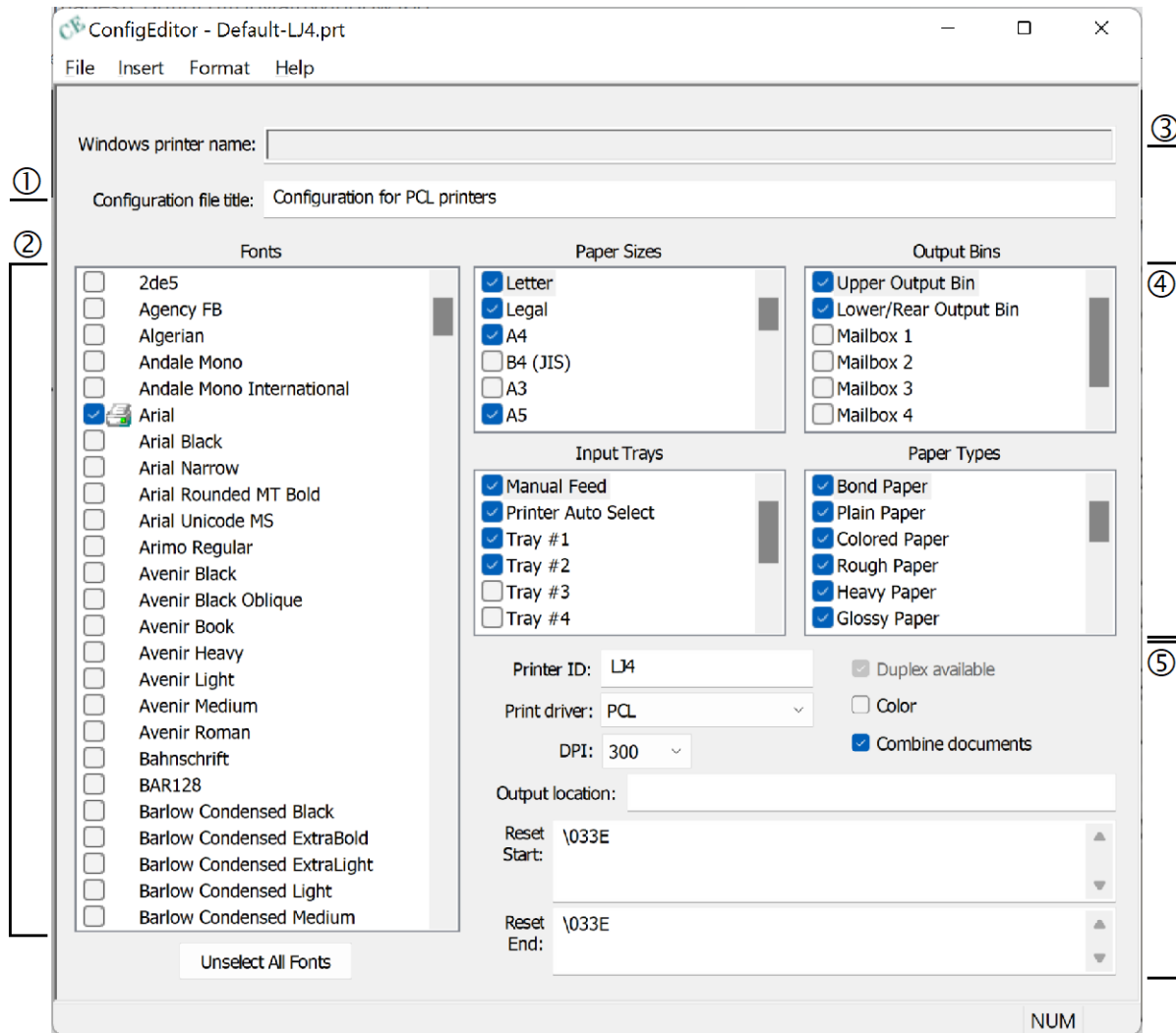
More Dynamic Configuration


Settings put in these output configuration files are expected to apply across the board for all forms that reference them as their output configuration. That's great about 99% of the time, but there are some times when the value(s) supplied in the output configuration file should be overridden. The primary example of this is barcode settings. While you can specify one standard setting in the output configuration file, such as showing the text associated with the barcode or not, perhaps some forms, or even just some instances of that barcode type on a form, need to go against the standard and have a different setting. This can be done dynamically in the script inside the form. See [domObj.setTag](#) for how to do that.

User Interface

DocOrigin ConfigEditor provides a Microsoft Windows user interface that lets you create or modify the configuration files for printers or PDF.

The ConfigEditor Main Window



1. For a new configuration, defaults to the selected printer name, which may be changed.
2. To select a font for availability in Design and Merge, check the box.
To indicate whether a font's resident in the printer (), right-click on the font name and click Printer Resident Font.
3. For a configuration, defaults to the selected printer name.
4. To select an entry, check the box.
To review or modify an entry, double-click the entry, OR click the entry, and click the appropriate Format menu command.
To add a new entry, click the appropriate Insert menu command.
5. Enter the appropriate values, and check the applicable options.

Setting	Description
Windows printer name	<p>Name of a Windows printer driver resident on the computer, populated when a new configuration file is created.</p> <p>To create a configuration file for a Windows printer driver, click File - New. In the Print dialog, select the appropriate printer, and click OK to populate the Windows printer name. ConfigEditor queries the driver for many of the available values, such as available and/or resident True Type fonts. These values will be used to populate the ConfigEditor user interface.</p>
Configuration file title	Descriptive name for the configuration file.
Fonts	<p>List of all Windows fonts currently installed on the computer. Fonts for use in Design and Merge must be explicitly selected.</p> <p>Fonts can be flagged as "printer resident", which indicates to Merge that the font is available from the printer; therefore, it does not have to be downloaded to the printer or embedded in a PDF file. To select or clear the setting, right-click on the font name and click "Printer Resident Font". When the setting is selected, a printer symbol appears between the check box and font name.</p>
Paper Sizes	<p>Available paper sizes.</p> <p>Settings:</p> <p>ID is an internal code that tells Merge how to request this page size in the printer.</p> <p>Width and Height are the full page paper size; typically, these are not changed. In most cases, the settings are fixed for each unique ID value.</p> <p>Margins specify the non-printable area of the page. Modify the top-left position to move the printed image on the printed page. The top and bottom margins will affect how the Merge pagination logic will split overflowing pages.</p> <p>Printer request sequence, when specified, will override the default paper/page select sequence that is sent to the printer. The default PCL escape sequence is: <code>\033&1%dA</code> in which %d is replaced by the assigned paper size ID. For standard paper sizes, the automatically supplied IDs correspond to the paper size needed in this PCL request sequence. For custom paper sizes, in PCL, the escape sequence should be <code>\033&101A</code>, since 101 is PCL for "Custom Page Size". Your printer may support other built-in page sizes with assigned IDs. If so, you should use that ID for its paper size definition.</p>
Output Bins	<p>Used for printers with multiple output bin selections. This setting is not used by the PDF driver.</p> <p>Settings:</p> <p>ID is a unique integer, less than 1000.</p> <p>Display name is the output bin name that appears in Design.</p> <p>Printer request sequence, when specified, will override the default select sequence that is sent to the printer. For PCL, the standard output bin selection sequence is <code>\033&1%dG</code> where %d is replaced by the associated output bin ID.</p>

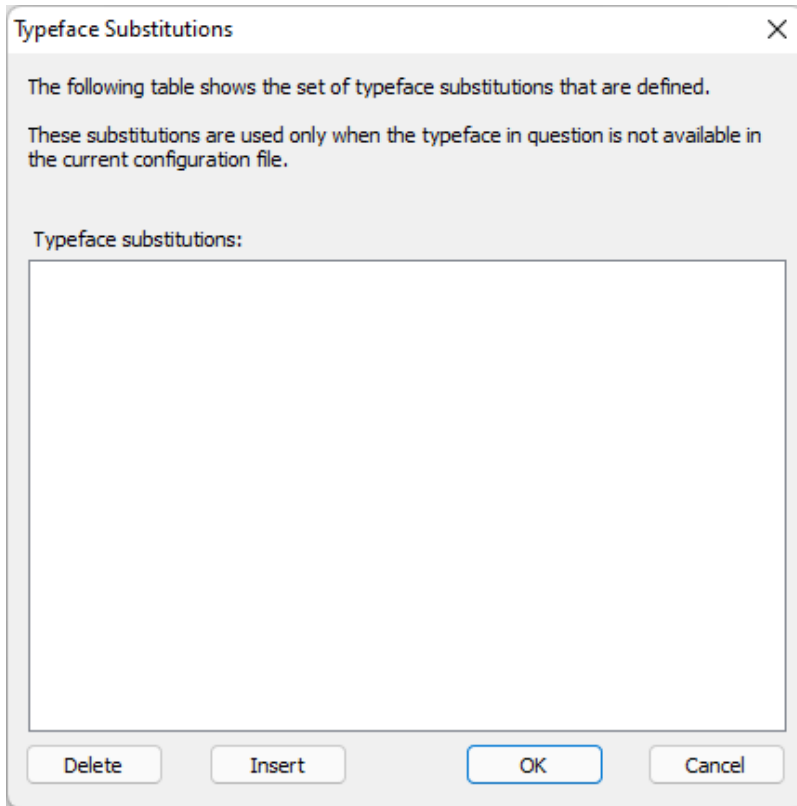
Setting	Description
Input Trays	<p>Printer paper trays from which the paper will feed the printer.</p> <p>Settings:</p> <p>ID must be a unique integer, less than 1000.</p> <p>Display name is the input tray name that appears in Design.</p> <p>Printer request sequence, when specified, will override the default select sequence that is sent to the printer. For PCL, the standard input tray selection sequence is <code>\033&L%dH</code> where %d is replaced by the associated tray ID.</p>
Paper Types	<p>Weights and types of paper that the printer will handle.</p> <p>For PCL, the standard printer request sequence is <code>\033&n%dWd%s</code> where %s is the name of the paper type, e.g. Bond, Plain, ..., and the %d is the length of that name plus one.</p>
PrinterID	A unique name for this configuration file. This is the printer name that must be specified in the <code>_printer</code> script functions of Merge.
Print Driver	Type of driver; select the appropriate value from the drop-list.
DPI	Applies to printer output only. Specifies the dots-per-inch resolution of the printer. Note that increasing the resolution will increase the amount of data for raster images that is sent to the printer.
Duplex	Indicates whether the device can print on both sides of a page. When checked, enables the form designer to specify duplex printing in Design.
Color	Applies to printer output only. Indicates whether the printer can handle color printing. When not checked, Merge will convert full color images to monochrome prior for printing.
Combine Documents	<p>Merge is capable of processing multiple sets of data and forms in a single run. When checked, the resulting output will be combined into one large multi-document file or print job; when not checked, the output will be broken into individual documents.</p> <p>When this setting is not checked and Merge will produce multiple documents to disk files, set the <code>OverwriteFile</code> property to "N" (No). This ensures that each document is written to a separate disk file (an integer sequence number is appended until a unique name is constructed). Click <code>Format - Advanced Properties</code> to set the <code>OverwriteFile</code> property.</p>
Output Location	Default location where Merge will write the document to be output. This name is a "template". It can have many dynamic substitution elements in it. See File Naming Conventions .

PCL Printer request sequences: It's really best to have **printer-specific** information if you are doing anything out of the ordinary. Hopefully, such manuals will advise you of the needed request sequences. As a good start, Google "pcl technical reference manual", and that should suffice for *most* things. Custom Page size setting seems particularly problematic. Check with your printer! A good try would be to compute your page size in decipoints (inches x 72 x 10). Then use `\033&fwwwrihh hJ\033&l101A`. The `&f` sets the page size (at least on some printers), and the `&l101` says it is a custom page size. So even if you had chosen a page size ID of **427**, you would still use `&l101`. The standard page size printer request sequences are handled automatically.

Typeface Substitutions

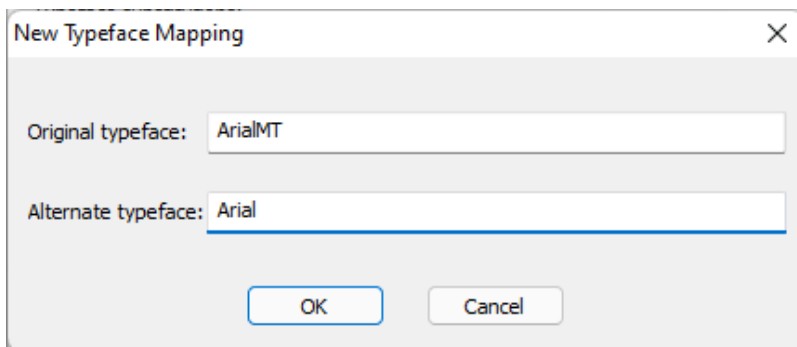
There may be circumstances where a shared form may require one or more fonts that are not available either on a user's computer or on the target printer. The `Typeface Substitutions` command in the `Format` menu enables you to create a list of substitutions that Merge will use in these situations.

Initially, no substitutions are defined.

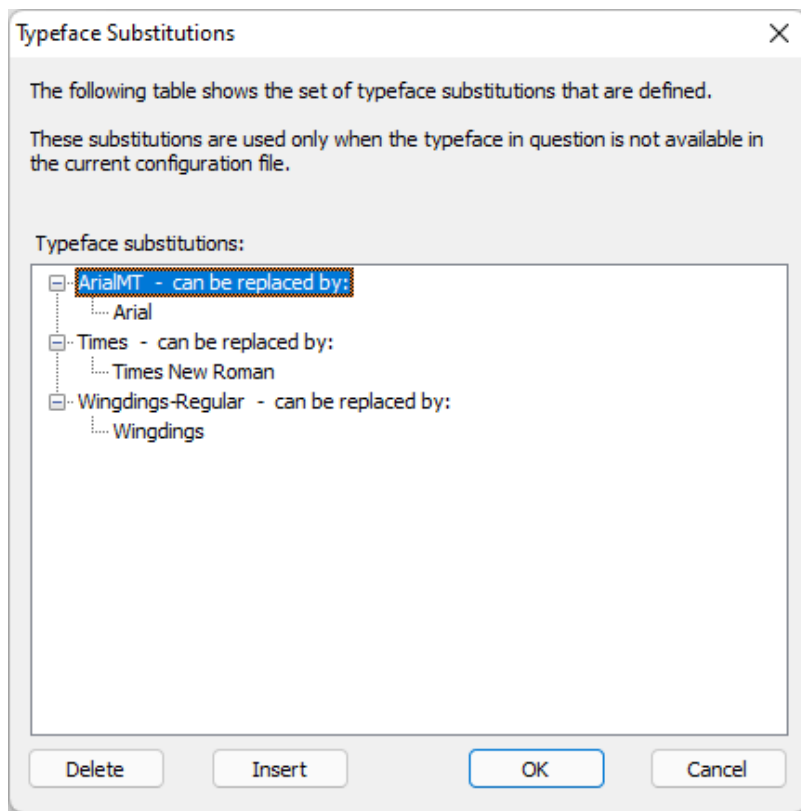


To add an entry, click **Insert** or right-mouse click in the dialog and click **Insert new Mapping...**

Enter the Original typeface and substitution typeface name.



The added entries are added to the list, three typeface substitutions are shown in the sample below.



To delete a substitution, click on the substitution typeface and click **Delete**.

To delete a typeface and all of its substitutions, click on the substitution typeface and click **Delete**.

Any number of substitutions may be associated with a typeface, by clicking on the appropriate typeface and adding an entry. Each substitution is placed at the bottom of the list for the typeface. In the event that a substitution typeface is required, the order of precedence is the list order.

Advanced Properties

The Advanced Properties command on the **Format** menu allows changes to some additional configuration file settings. These settings will rarely require modification.

Note that not all of the following properties are available for every printer ConfigEditor displays only those that are relevant for the current device.

Category	Property and Description
2 of 5 Barcode	<p>Barcode25AddCheckSum indicates whether to automatically add a checksum digit at the end of the barcode. Value: Y or N</p> <p>Barcode25Ratio is the ratio of wide to narrow bars. Value: An integer</p> <p>Barcode25Text indicates whether to display human-readable text below the barcode. Value: Y or N</p>
3 of 9 Barcode	<p>Barcode39Text indicates whether to display human-readable text below the barcode. Value: Y or N</p> <p>Barcode39Ratio is the ratio of wide to narrow bars. Default is 3. Value: An integer</p>

Category	Property and Description
Bolding	SynthBold is the degree of artificial bolding used when bold versions of fonts are defined but are not available in an actual bold font. Merge can bold a regular font by "thickening" the letter in its print drivers.
Check Box	<p>CheckboxShape is the shape of all check boxes on the form. Value: Circle, Square or None</p> <p>CheckboxSize is the size of the check box, in inches. When set to zero or left blank, the size of the check box is the font size specified for the field in Design. Value: A number</p> <p>CheckboxStyle is the appearance of the check box for each state (checked or unchecked). Value: Fill - sets a solid color when checked, and empty when unchecked Check - sets a checkmark when checked, and empty when unchecked xy - a string of 2 characters sets the check box to the first character when checked, and to the second character when unchecked. When only 1 character is specified, set the check box to the character when checked, empty when unchecked.</p>
Font Embedding	EmbedFonts indicates whether a non-printer resident font encountered in the document should be downloaded to a printer or embedded in a PDF. Value: Y or N
Overwrite File	<p>OverwriteFile indicates whether an output file created by Merge (PDF or printing to a file) will be overwritten when a file of the specified name already exists. Value: Y or N</p> <p>When set to "N", the file will be made unique by appending a number to the end of the file name. For example, when "N" is specified, and a file named "myfile.pdf" already exists, Merge will try "myfile_1.pdf", "myfile_2.pdf" and so on, until it finds a file name that doesn't already exist.</p>
PDF Bookmark	Bookmark indicates whether to generate bookmarks in PDF files. Applies only to documents output to PDF files. Value: Y or N
PDF Viewer Preferences	ViewPreferences specifies PDF viewer settings as per the "Interactive Features" section of the PDF Reference manual, version 1.7 which is available online from Adobe (www.adobe.com) or AIIM (www.aiim.org).
QR Code Barcode	<p>QRCodeCharacterSet is the default character set encoding for QR Code barcodes. Value: Alphanumeric, Numeric, 8Bit or Kanji</p> <p>QRCodeLevel is the default QR Code error correction level. See the QR Code documentation for details. Value: L, M, H or Q</p> <p>QRCodeMode is the default QR Code character mode. See the QR Code documentation for details.</p>

Category	Property and Description
Radio Button	<p>RadioButtonShape is the shape of all radio buttons on the form. Value: Circle or Square</p> <p>RadioButtonSize is the size of the radio button box, in inches. If set to zero or left blank, the size of the radio button is set to the font size specified for the field in Design. Value: A number</p> <p>RadioButtonStyle is the appearance of the radio button for each state (selected or not selected). Value: Fill - sets a solid color when selected, and empty when not selected Check - sets a checkmark when selected, and empty when not selected xy - a string of 2 characters sets the radio button to the first character when selected, and to the second character when not selected. When only 1 character is specified, set the radio button to the character when selected, empty when not selected.</p>
Word-wrap Text	<p>SplitWord is the action to take when a text string (from a label or field) with word-wrapping cannot fit on the line even when split at normal word-break positions. Value: Y - specifies the word will be split, or N - specifies the word will be allowed to overflow to the right</p>

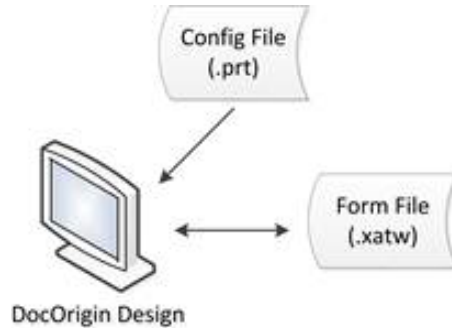
.prt Override

Merge uses a series of command-line options to customize and control its operation. Several of these command-line options will override the equivalent setting in the printer configuration file. For a list of these particular command-line options, see [Command Options - Merge](#), Printer Options.

Design

DocOrigin Design is an interactive Windows application with specialized tools for drawing forms. The forms created by Design are used by Merge to create output documents.

i DocOrigin Design, like all DocOrigin GUI apps, is supported on Windows 64-bit only.



Design Tutorial

The original detailed Design tutorial can be downloaded here:

[Original Fundamentals of DODesign.pdf](#)

⚠ DocOrigin Design User Guide Available with a Login for All DocOrigin Customers

A more extensive [Design User Guide](#) is available to all current DocOrigin Users. The material contains step-by-step instructions, feature samples, troubleshooting tips, accompanying collateral, and more. You require a User Name and Password to access the material, which can be obtained by emailing our support team at support@eclipsecorp.us.

Form Files

Using Design, a form designer can create forms that range from simple one-page static (fixed) forms to dynamic forms that can span multiple pages with many sections, or repeating detail lines that can span multiple pages depending on the data.

Forms define:

- the fixed or static objects of a document, such as titles, logos, lines, boxes and so on
- the location and formatting characteristics of variable data fields, that correspond with the values in the user-supplied data
- JavaScript (Mozilla® ECMA262) scripts that are executed when Merge processes a job. The scripts can be used to modify the data content or the form appearance when Merge is run.
- the checkmark-based logic by which multi-page, dynamic, documents are created, such as where page breaks can occur, repeated header or footer information, and so on.

Form files use a file extension of `.xatw`.

File-Print

Like most Windows applications, Design sports a File-Print operation and has the standard Print icon on its toolbar. But it's not really a normal print operation.

Virtually always, when you want to see how your form design is coming along you will do a Tools-PDF Preview..., Tools-HTML Preview..., or Tools-Merge Test... operation. File-Print is almost never used.

What File-Print does is to use "Windows drivers", as in Default-WIN.prt , to print out a blank document. It does not matter which printer configuration is defined for the form design, the Default-WIN.prt configuration file will be used for File-Print.

So it is not emitting PCL or PS2, or PDF, but is rather making Windows GDI calls suitable for devices known to the Windows system for which there are device drivers from the device's vendor. Adobe PDF is often another output choice that can be made. Just as Microsoft Word, or any app, can output to the Adobe PDF pseudo device, DocOrigin's GDI calls can be directed to the Adobe PDF driver to create a PDF.

But this is not really a data merge operation. File-Print prints a blank form.

The other interesting fact is that for File-Print, Design makes it as if every pane and every table row were mandatory, occurring once. Hence the File-Print operation produces a blank form with every pane and table row in it. This could be useful for form design documentation. For simple static forms, a File-Print result could be used for simply printing a form, e.g. for "print-then-fill" operations.

It's likely that you want to use Tools-Preview PDF and specify an applicable test data file, but File-Print can be used for a blank form using Windows drivers technology.

Form File Import/Conversion

(Windows Only)

DocOrigin is often used as a replacement for Adobe® LiveCycle™ Output, which is also referred to, correctly or incorrectly, as Adobe® or JetForm Central. While DocOrigin uses an open architecture with easily readable files, those other products use a proprietary binary file format.

DocOrigin Design's **File > Open** menu item allows you to select *.XATW files (of course), but also *.IFD (Adobe® Output Designer), *.XDP (Adobe® LiveCycle Designer), and *.PDF files. In the non-XATW cases, the "open" is actually an "import" in that the nominated form design file is instantly converted to an XATW as part of the file-open process.

Import Prerequisites

✔ Alternative Form Convert Option

If you do not have the prerequisites, you can use [Eclipse Form Conversion Service](#) instead.

IFD and PDF conversion require, to the best of our knowledge, that the user have a licensed copy of one of:

- Adobe LiveCycle Designer version ES2, possibly ES4
- Adobe Acrobat Professional versions 8.1, 9, and X
- Adobe Experience Manager, possibly pre-version 11

The Adobe LiveCycle Designer contains utilities that convert IFD files to XDP format, and PDF files to XDP format. As the XDP file format is textual XML, such files are read directly by DocOrigin during its conversion process. You don't have to learn/use the LiveCycle Designer program. DocOrigin uses the needed utility silently, behind the scenes.

If you have installed Adobe LiveCycle Designer under C:\Program Files (x86)* (or **D:**\Program Files (x86) -- wherever Windows tells us the "special" Program Files (x86) folder is,) DocOrigin will find the needed utilities automatically. If you have installed it elsewhere you will need to change the following registry settings to point to the actual location of the utilities.

```
HKCU\Software\OF Software\DocOrigin\Design\FileLocations\ConvertIFDShell
HKCU\Software\OF Software\DocOrigin\Design\FileLocations\FileLocations\ConvertPDF
```

(Prior to 3.2.001.02)

```
HKCU\Software\OF Software\DocOrigin Design\3.1\Design\FileLocations\ConvertIFDShell
HKCU\Software\OF Software\DocOrigin Design\3.1\Design\FileLocations\ConvertPDF)
```

The above registry locations are used by Design. For the FormConvert non-GUI application (see Batch Conversion below) you can define the locations of the from-Adobe tools via the following environment variables: DO_IFD_CONVERT and DO_PDF_CONVERT.

Batch Conversion

DocOrigin supplies a non-GUI program called "[FormConvert](#)" which can be used to convert any of the above format files to XATW format. The same prerequisites apply for batch conversion as for conversion using DocOrigin Design.

Bumps in the Road

If you encounter errors during the conversion process, e.g. #5907, #5903, #1084, or 50463490, these are not DocOrigin errors but rather errors from the underlying Adobe tooling that we referred to above. If such a thing should occur, your installation of the Adobe products is probably not correct. We suggest that you:

1. Prove to yourself that you can open the IFD directly in Adobe Output Designer, and most preferably prove that you can compile it for a PDF presentment target. If you can't open it in Output Designer there seems little chance that Adobe LiveCycle Designer will be able to do its import operation. If you really can't open it in Adobe Output Designer, the best idea is to reinstall that product. Failing that please contact Adobe.
2. Prove to yourself that you can open the IFD with your installed Adobe LiveCycle Designer, whether that is from a standalone purchase, or from an install of Acrobat Professional 8.1, 9, or X. If you cannot open the IFD in Adobe LiveCycle Designer, then it is possibly because you have Adobe Output Designer installed under other than `x:\Program Files (x86)`. Correct that. If that is not the case then, < sigh > re-install Adobe LiveCycle Designer. If the reinstall fails to help, please contact Adobe. If Adobe LiveCycle Designer cannot open the IFD then there is little hope for DocOrigin to open it since it relies on that Adobe tooling.
3. You *could* locate/verify that `ConvertIFDShell.exe` exists somewhere on your system, likely under your `Program Files (x86)` folder. If you can't find that you really must find and reinstall the Adobe software. If you do find it, you could *attempt* to do the conversion after you manually set the registry entries mentioned above to the location of where you found the .exe. This is somewhat grasping at straws since
 - a. above should have worked. Please don't leap to the idea that only that one executable is needed. It must be surrounded by all the usual Adobe Output Designer files, configuration and whatnot. Those configuration files must certainly match those that you have been using in production.
 - b. above applies to opening a PDF as well – except that there is no connection to Adobe Output Designer. And
 - c. (desperation) applies except look for `ConvertPDF.exe`.
4. No, you cannot get an IFD out of an MDF. You must have an IFD for any form worth converting.

DO_SAVE_XDP

IFD and PDF files are not human-readable. They are "binary". The Adobe tooling does the much-appreciated task of producing an XML, human-readable (well, it's no literary thriller!), XDP file. It is that XDP file that DocOrigin reads and interprets. It can sometimes be handy, at least for SEs, to look at that intermediate XDP output that is used behind the scenes. By setting the environment variable `DO_SAVE_XDP` to some known file name, e.g. `c:\Temp\LastConversion.xdp`, you can always look at the latest intermediate XDP file as created by the Adobe tooling for importing a PDF or IFD.

Stepwise Conversion (optional)

DocOrigin uses background tooling from Adobe to get an XDP file and then reads that XDP file. If you wish, you could use Adobe LiveCycle Designer to import the IFD or PDF and save the XDP explicitly, completely separate from any DocOrigin involvement. Then you could have DocOrigin open (import) that XDP as a separate independent step.

Why mention that? Well, the Adobe tooling is not perfectly in sync. You may get slightly different results from using Adobe LiveCycle Designer itself versus from using the Adobe `ConvertIFDShell`, or `ConvertPDF` tooling. DocOrigin has no control over that. Converting your form designs from a binary, proprietary, format to an open, textual, format is precious, so having any alternative avenues is worth knowing about.

Conversion Effort Estimates

Ok, how long is a piece of string?

This is impossible to say. The actual import operation is mere seconds. But the results may need modification. Mostly it depends on whether the designer used only standard preamble or developed 'interesting' custom preamble. Also, the structure (or lack thereof) of the data can play a significant role in the conversion effort. If there is no natural correlation between the data structure (and even field names) with the structure and names used in the form design, then the effort will be greater. If the subforms are in a random jumbled order rather than a logical top-down sequence, another small effort will be required to shuffle the order of the panes into sensibility.

Most often, the bigger effort is simply to round up all of the needed collateral: the form designs, the relevant production configuration files, the images, sufficiently base-covering sample data (single page, multi-page, multi-documents), and corresponding expected outputs.

For conversion directly from PDF, the utility used will break up text strings any time there is a change in font (typeface, point size, plain/bold/italic). This then is what we have to work with. You may well wish to apply Design's `Format-Combine Multiple Text Labels` menu item.

Do also consider data conversion – of course, [ConvertDatToXml Filter](#) is handy in that regard.

Fragments


(As of 3.1.001.01)

DocOrigin Design and Merge provide a means to create a library of pre-defined form objects that can be placed on a form template. These library objects are called **Fragments**. A fragment is any DocOrigin template object - a Field, Text Label, Table, Group, Pane, Page - that can be drawn in Design. Fragments are typically created with a preset set of attributes and settings.

In the simplest use of fragments, you select a fragment from one of the libraries and place it on the page. This is an **embedded** fragment. It just becomes a normal part of your design. It can be edited, moved, modified just as if you had drawn it all yourself. This is more-or-less the equivalent of doing a copy/paste from one form to another.

A second option called **Linked Fragments** also exists. A Linked fragment is a "read-only" instance of a library fragment. You can position a linked fragment, and give it a new name, but you cannot modify any of its other attributes. Those other attributes are *linked* to the original library, and only change when the library is changed.

Whenever a template with a linked object is re-loaded in Design, it is automatically updated to the latest version of that fragment in the library. So an update to the original library will automatically be reflected in any template that **links** to it. A similar mechanism is also available (as an option) in the Merge program (see below). This allows library changes to be reflected automatically in the runtime Merge output.

 Note that although the above automatic updates to existing templates by Merge may be a powerful way to make system-wide changes to forms, it should always be used with caution. Proper testing and QA of the changes to production forms is very important. Changes to library fragments can cause unexpected effects. Text labels that originally display correctly might now overflow or overlap other design elements. Font changes may require changes to Merge configurations. Use of automatic updating of linked fragments in Merge should always be carefully tested before going "Live".


Libraries

Fragment Libraries are essentially normal DocOrigin form files stored in a designated Library folder, but with a file extension of **.xatwlib**. You can create them and modify them with Design in exactly the same way you work with other designs.

The DocOrigin install comes with some standard fragment libraries which are located in the DocOrigin/DO/Library subfolder. These libraries typically have "Default-" at the front of their name - such as "Default-Zebra.xatwlib" and "Default-Fragments.xatwlib". The Default-Zebra library has a number of barcode formats for Zebra label printer. The Default-Fragments library includes a template for the Auto Email feature of Merge - see [Auto Email](#).

You can of course copy these files and modify them, but you should never save the new files back to this /DO/Library folder as that folder will be replaced whenever you install a newer version of DocOrigin software.

Libraries you create or modify should be stored in the DocOrigin/User/Library folder. (or more correctly \$U/Library). These User libraries and will not be affected by any upgrade of the DocOrigin software.

 It is recommended that when creating a Fragment Library, you use the Design Notes feature to provide a brief description of each fragment. The Tools>EditObjectNotes or Tools>EditPageNotes commands will enable you to add the notes. When the Fragment Selection dialog is displayed (see below) the Notes for the fragment are displayed for the current selection. This gives allows you to provide additional document to the designer.

Adding Fragments

Fragments are added using the + button on the Design toolbar. This allows you to select a fragment library and either Link or Embed a library object into your design (including full pages). When a fragment is **Linked** a small 'infinity' symbol will be displayed in the Design Form Explorer next to the object as an indication that it has been linked. Design will not let you change the attributes of a linked fragment except for its position and name.

You can review the list of fragments used on a form using the Tools>ViewAllFragments command.

- i** When adding fragments to a completely filled Container, it is recommended that you turn on the option "Display a small gap between top-level Panes". This allows you to insert Pane fragments into the Container by targeting them between existing Panes. To set this option "On":
- Select the menu option Tools/Options
 - Check on the option "Display a small gap between top-level Panes"
 - Click "OK"

Fragment Resolution

When a form containing Linked Fragments is opened in Design, each associated fragment library is checked to see if the library contains a different (newer) version of the fragment. If so, that new library version is loaded into the form and replaces the original one.

If a linked fragment cannot be found in its original library, or if the library cannot be found, Design displays a list of all the fragments and highlights those that have not been found. If you know that the fragment has moved to a different library, you can make that change immediately.

Linked fragments whose library entry have not been found are said to be **unresolved fragments**. You can continue to use the form with these unresolved fragments and they will work as they always did. However, Design will continue to report on any unresolved fragments whenever the form is opened or saved.

Fragments in Merge

By default, Merge does NOT attempt to resolve link fragments. The existing definition in the form file will be used. You can however direct Merge to check all linked fragments and load in new definitions just as is done in Design by using the **-FragmentAutoResolve** option. Set

```
Merge ... -FragmentAutoResolve Yes ...
```

to enable this processing.

Linked fragments whose library cannot be found, or that cannot be found in the library are simply ignored and the original fragment definition in the form continues to be used.

A detailed summary of which fragments were resolved can be found in the trace file by using the -Trace option of Merge.

Default Object Templates

There is a set of default object templates in the DO\Bin folder ("M_" is for metric):

- Barcode.xml
- Container.xml
- Group.xml
- Image.xml
- LabelField.xml
- Line.xml
- Pane.xml
- Rectangle.xml
- Table.xml
- TextField.xml
- TextLabel.xml

- M_Barcode.xml
- M_Container.xml
- M_Group.xml
- M_Image.xml
- M_LabelField.xml
- M_Line.xml
- M_Pane.xml
- M_Rectangle.xml
- M_Table.xml
- M_TextField.xml
- M_TextLabel.xml

To customize the properties of each of these objects, you can copy these templates into the User\Overrides folder and modify for your needs.

Some templates don't have the full set of attributes defined in the template, e.g. for TextField.xml it specifies `<border visible="no"/>` but does not define border thickness. If you want to adjust the thickness of the border you can copy the border definition from another template or from a form you design. To see an object's XML in Design, select the object and type [Alt+Ctrl+N](#).

Design Images

One of the many object types that can be included in a form design is a graphic image. These can take the form of static images or of image fields.

By selecting the image tool icon (looks like a maple leaf) from the toolbar the designer can drag out an area to be filled by a static image. Then, using the **Image File** browse button in the **Object Properties** dialog, the designer can select the desired graphic (logo, etc.). The designer can choose various options to fit the chosen image into the nominated area.

The designer can choose image files of the following formats/file extensions:

- *.bmp, *.gif, *.jpeg, *.jpg, *.png, *.tif, *.tiff
- *.pdf - see below; needs a 3rd party tool

When defining an image field the designer uses the field tool to drag out an applicable area and then selects the **Image** option from the **Display as** dropdown in the **Field** tab of the **Object Properties** dialog. The data file is expected to supply a file name of one of the acceptable image types. By default, the data-nominated image will be scaled to fit inside the field's extent without any scaling distortion. Other 'fitting' options can be selected via script using the `this._imageMode` property. See [DOM Properties](#)

PDF Page As Image

(As of 3.1.002.01)

If you select a PDF file as the image then you will be presented with a follow-on dialog to choose which page of the PDF to pick and also which DPI to use (default 150). This functionality is **predicated upon the existence of a 3rd party tool**. See [PDF Page Images](#).

Image as Template

A designer may choose to bring in a, typically full page, image as a means to, well, have a background image, but also to be able to precisely place fields in their proper locations relative to that background image. To bring in the, possibly temporary, background image, the designer is likely to use the `Insert > Image as Template...` menu item. The designer then gets to pick an image just as with the usual image tool. By default, this Image-as-Template image is marked invisible but of course, you can change that if you do want it to appear in the generated output.

A popular choice may be to select a page of a PDF file to use as that image. See the previous subtopic.

Once that image has been placed (it will automatically be placed at the top-left (0,0) and will be in the background, i.e. behind the dynamic area occupied by panes), the designer can select the "Auto-Field" tool and simply point at an area of the screen that has a discernible rectangle around it. Design will find that rectangular area and create a field that exactly matches it. Likely the designer would name the field and then hold down the Alt key so as to bring back the "Auto-Field" tool. In that way, the designer can very rapidly place many fields in exact size and registration with the background image. If the rectangle is a small square, Design will deem it to be a checkbox field and set the field's attributes accordingly. Note that before starting this process it is recommended that you set your grid unit to be very small, at least .01", if not .001". In that way, you will get much better registration with the background rectangles.


PDF Page Images

(As of 3.1.002.01)

You are now able to reference a page of a PDF file as an image. The syntax is as follows:

```
path/fileName.pdf?page=3&dpi=150
```

The default value for page is 1, and the default dpi is 150.

 This functionality relies on the existence of **3rd party software** which you must download and install on your own. This functionality will not work until you do so. Considerable flexibility is built-in to allow you to choose your desired 3rd party software. See Default-PDF2PNG.wjs below.

Default-PDF2PNG.wjs

To support the PDF-Page-as-an-Image feature, Merge (and Design) run a supplied script, named: "Default-PDF2PNG.wjs". Naturally, that is in the DO/Bin (\$E) folder. You may override that script by copying and revising it in file **User/Overrides/PDF2PNG.wjs**. Merge will choose that override file if it exists.

If you look in that script file you will see that by default we have chosen to use the pdftopng.exe tool that is available from Xpdf (<https://www.xpdfreader.com/> -- you would want the **xpdf** tools). If you override the script you may choose other such software tools. There appears to be a myriad set available.

To run that script Merge uses the RunScript program as follows:

```
RunScript -script [...]PDF2PNG.wjs -in="pdf spec provided" -out="a temp file for the png"
```

The script is pretty small. It parses the supplied pdf spec to determine the pdf file name, the desired page number, and the desired DPI (with defaults). It then runs **\$P/pdftopng.exe** with its needed parameters. Of course, your override script might run something else and indeed might look for other options on the pdf spec (what I would call a 'query string'). The end result of whatever you run must produce an image file in the supplied -out file. pdftopng.exe does that quite nicely. (As it happens, **pdftopng** produces a file name of its own choosing but our script renames that to the demanded -out file name.)

\$P?? What's that? \$P was also introduced in 3.1.002.01 as a means to specify a semicolon-separated list (path) of folders to search to find the base file name. Since we do not distribute these 3rd party files we do not know where you will install them. By defining \$P, typically in your **\$O/Paths.prm** file, you can specify a path for DO software to use to look for files.

Alternatively, you could put **pdftopng** directly in the \$E (DO/Bin) folder. We recommend that you do not put additional software in \$E as future updates may cause it to be deleted.

Since the script is totally open, you can do whatever you want. Use whatever tools you want, and handle whatever file types you want. Just pay attention to the -in and the -out parameters supplied by Merge and you're good to go.

See Also

[\\$X String Substitutions](#)
[JSON Data Files](#)

Object Tags

✓ New Tags Dialog

(As of 3.3.002.01) The new tags dialog includes an extensive list of possible tags, their values, and a short help text.

Object Tags are additional, seldom-used, attributes that can be added to DOM objects in Design. This is done using the **Format > Object Tags...** command in Design. Object Tags can also be set in Merge using the DOM Script function [domObj.setTag](#).

AutoEmail Message Text

(Applies to the older "_email_" fragment, not the newer "_email_5_" fragment)

The AutoEmailConvert option is used on the Message object (usually a Label) within an AutoEmail page. This flag controls whether the message text is actually already in HTML, or is in normal DocOrigin RTF and must be converted to HTML.

AutoEmailConvert Y (the default) will convert normal DocOrigin formatted text from RTF to HTML.

AutoEmailConvert N assumes the text message is already in HTML so RTF formatting is ignored.

Barcode Overrides

There are a number of Barcode options specified in the configuration files (.prt). These include default display options such as Barcode39Text, Barcode39DerivedText, etc.; default bar widths or ratios between wide and narrow bars, special options like QRCodeCharacterSet, etc. These options can also be set on an individual barcode label or field within Design by setting the corresponding object tag. So, for example, setting tag Barcode39Text to value 'N' would suppress the default display of human-readable text below the barcode for a specific Label or Field.

Ellipsis

Set the value to the hex Unicode code for the character to be used on the particular Field when a text-overflow condition occurs. Normally when field text overflows its defined size in Merge, the text is truncated and the current ellipsis character (defaulted to ...) is added at the end. This Object Tag allows you to override the default ellipsis character for a specific field. Note that you can set ellipsis to 00 to indicate that the text is truncated but no character is appended to the text. See the Merge command-line option **-ellipsis** for globally resetting the ellipsis character.

HTML Images as URL References

Normally the Merge HTML driver will embed any document images directly into the output html file. If you already have the image available somewhere on the internet, you can specify that the driver simply references that URL rather than embedding the image. To do this, set the Image object (static or Field) tag ImageURL to the web URL for the image. Note: The Tag Name URL is also supported.

PDF/UA Tags

A number of object tags are used to enable the designer to specify PDF/UA accessibility tags. Generally, these tag names all start with "Tag" such as TagAlt, TagAs, TagTitle, etc. Note that TagAuthor, TagKeywords, TagSubject, and

TagTitle can be set for any PDF document - they are written to the PDF /Info dictionary and will display as document properties in PDF viewers such as Acrobat. Others are only used when explicitly generating PDF/UA.

RadioButton and CheckBox Options


DocOrigin RadioButtons and CheckBoxes have certain Size and Shape settings defined in the configuration file (.prt). These can be overridden on a specific object by setting the corresponding Object tag. The setting for CheckboxShape, CheckboxStyle, RadioButtonShape, and RadioButtonStyle can be overridden this way.

Direct Data Binding

(As of version 3.2.001.01) Direct data binding allows you to specify that a Field value or Text Label value can be fetched directly from the data file by name, rather than using the usual data merging mechanism of DocOrigin Merge. To do this, you set the Field or Label's **BindData** tag to the name of the field in the data DOM. The name can be either a dotted data DOM expression or just the leaf name if the data name is unique in the data file. You can also set **BindData** to an asterisk to indicate that the data node's name is the same as the Field or Label's name.

BindData is Case-Sensitive

The BindData option must match the case of the input data. XML, by nature, is case-sensitive. The DocOrigin merge option `-mergeCaseSensitive Y/N`, does not apply to BindData.

 It is rather important to know that DataBinding happens on Form DOM (not Document DOM) at the beginning of each document, so before any data-merging even starts.

The use case is intended for directly accessing information in the data DOM without using implicit data mapping in the form DOM. Common use cases are:

- when a field occurs in the data DOM but appears in another area of the form DOM
- a shared field name is unique only by its parent node. For example:
 - you want to pull Customer Bill To Name in the Remit Tear-off of an Invoice
 - the data contains two CustomerName fields
 - fields are unique only by their parent nodes - `<ShipTo>` and `<BillTo>`
 - the Ship To Company Name can be directly bound with the following:
 - Tag name: `bindData`
 - Tag Value: `CustomerInfo.ShipTo.CompanyName`
 - the Bill To Company Name can be directly bound with the following:
 - Tag name: `bindData`
 - Tag Value: `CustomerInfo.BillTo.CompanyName`

Global Field Tip

If you want to use the Customer Bill To Name throughout the form using the Global Field setting, create a new unique form field Name such as `BillToCompanyName` and set it to Global.

C:\DocOrigin\DO\Samples\Sample_Invoice.xml

```
<CustomerInfo>
  <ShipTo>
    <CompanyName>Perfect Printers</CompanyName>
    <Address1>425 Lansing Drive</Address1>
    <Address2>Moline, Illinois</Address2>
    <Address3>USA 61265</Address3>
```

```

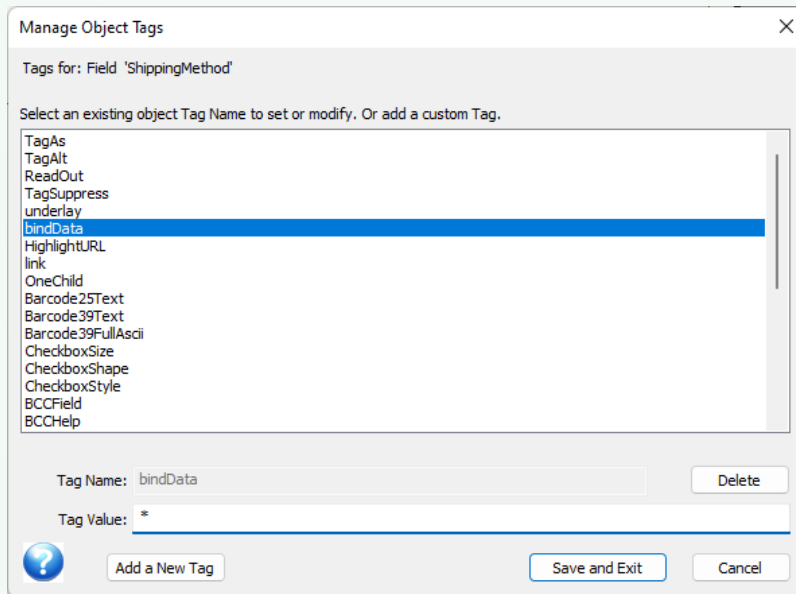
<Address4>Tel:800.555.2323 Fax:309.762.4411</Address4>
</ShipTo>
<BillTo>
  <CompanyName>Head Office, Perfect Printers</CompanyName>
  <Address1>800 Industrial Avenue</Address1>
  <Address2>Chicago, Illinois</Address2>
  <Address3>USA 60611</Address3>
  <Address4>Tel:800.555.3363 Fax:312.222.2345</Address4>
</BillTo> ...

```

✔ Use an Asterisk for Unique Fields

If you want to quickly map a single-occurring field to the form, you can match the form field name to the data field name and enter an asterisk in the value. Try this

- Start a new form
- Open the **Sample_Invoice.xml** in the Data Explorer
- Drag the field ShippingMethod onto a new form
- Add the following tag:
Tag name: bindData
Tag Value: *



- Run a PDF Preview with the **Sample_Invoice.xml**
- You should see UPS on your test PDF

❗ Direct Data Binding Limitations

The bindData tag is limited to a single occurring field (not a field that is part of an array such as Description in a repeating node called Detail).

See Also

[Copy Data Path To Clipboard](#)

RetainWhenBlank Tag

(As of 3.2.001.01) Typically, when you place a new label object onto your form design, if there is no value assigned to the label, it will be automatically removed. The RetainWhenBlank tag avoids this automatic removal of empty labels.

See also the [Auto/Embedded Fields](#) description of [!Data name]

Design Keyboard Shortcuts

Alt+Ctrl+N - Looking at the XML for an Individual Object

In Design, if you select an object on the canvas or in the FE, then you can hit Alt+Ctrl+N, and you will be rewarded with the XATW XML that corresponds to the selected object.

It could be useful to look at the rich text encoding of a label, especially, simply to see if it is pure plain text or has some rich text. It can resolve some puzzles about hidden-in-rtf typeface usage. It's also a nice source of simple, well-formatted rtf for test purposes.

Anyway, you might want to get a quick look at the to-the-metal encoding of your design object(s). You can select multiple objects, or a group or pane or something and see the XML for all that you have selected. Of course, there is always "Tools">"View Form as XML" to really quench your thirst, but that can be an overwhelming firehose.

Ctrl+Shift+E - Save Image

One of the great things about DocOrigin is that images used in the Design are kept in the Design file. After years and years with a previous product where you always had to go on endless searches for the graphics files that went into a form design, what DocOrigin does is a true blessing. This is so handy for sharing and indeed for providing collateral to your support organization. They have the images, they don't have to come back to bug you for them. You can find them in base64-encoded format in the XATW file. With effort you could copy that text out of there, convert, ..., but hey, just select the image and hit Ctrl+Shift+E and Design will offer to save that graphic in an appropriately typed (.jpg, .png, ...) file. Now you have it, you can use it in a new version of the form. You can avoid Form Verification Report errors that say the images aren't where they are expected to be.

Alt+X - Convert HEX Value to Unicode Symbol

Know any good Unicode characters? Sometimes it can be a head-scratcher to figure out how to enter them into a label's text string. If you know that special character's Unicode hex value it is a breeze. You type in its exactly 4 hex characters and then hit Alt+X. For example, suppose you want the division symbol, no not that one, this one: ÷. Its Unicode value in hex is 00F7. If you type 00F7 followed by Alt+X, you will get that ÷ symbol as sweet as anything.

Ctrl+G - Manage Object Tags

This is just a normal accelerator key that you can see on the menu. It is used for editing object tags. It's mentioned here because people don't always get into using object tags right away, but then when they do they may want to use them a lot for PDF/UA or barcode options or whatever has been dreamt up by the user. So knowing Ctrl+G can be a real timesaver. See PDF/UA tagging.

ALT - Reuse Previous Object Tool

In Design you select an object tool from the toolbar, e.g. field or text, ..., and then you place that object on the canvas. If you want to reuse that same tool to place another one on the canvas, you don't have to explicitly reselect it from the toolbar. Instead, you can just hold down the ALT key and you will see that your cursor changes to the previously selected tool's icon. This might well speed up your work.

ESC - Toggle Between Text and Object Edit

If you have a text label selected (i.e. with pick handles around it) you can hit ESC and thus go into text-editing mode. Conversely, if you are in text editing mode in a text label, you can hit ESC and you will leave text editing mode and have the text object be selected with pick handles around it. This is an essential skill.

Ctrl+Shift+A - Select All Objects in Pane

I've often wanted to move all the objects in a pane. If you select just one object in a pane, you can then do Ctrl+Shift+A and it will extend the selection to include all the objects in the pane. Have at! You can then use the

arrow keys to shift them around a bit. Or you might apply a gang update using the modeless property dialogs. Handy.

F5, F7, F8 - Preview

You can click on the PDF or HTM toolbar icons, or use F5 or F7 instead. Also F8 for Merge Test. When you use these it applies whatever options were used most recently. If you want to supply new options, use Ctrl+F5, Ctrl+F7, or Ctrl+F8. You would then get the usual dialogs to set the options before doing the 'preview deed'.

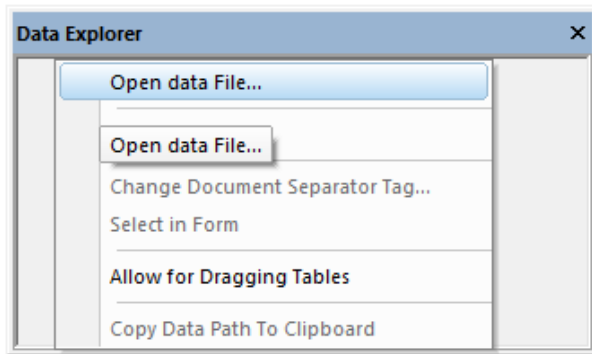
Ctrl - Line on a Diagonal

When you want to draw a line that is not horizontal or vertical you can: select the line tool; hold down Ctrl; and draw a diagonal line.

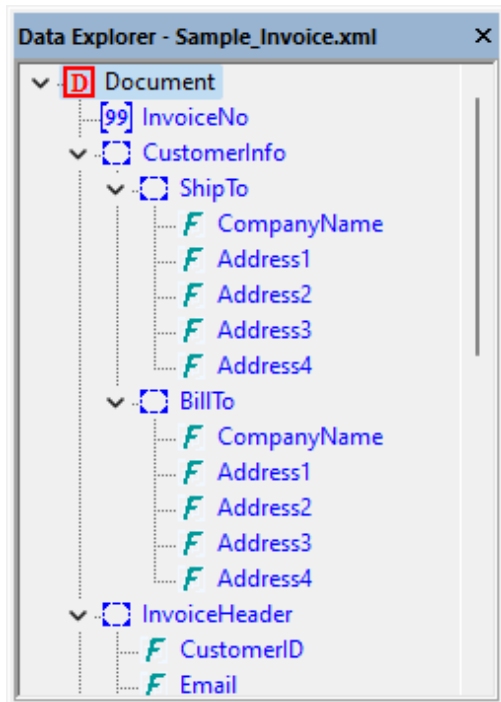
Data Explorer Options

Open Data File...

Right-mouse-click in the Data Explorer, navigate to the data file, and select Open.

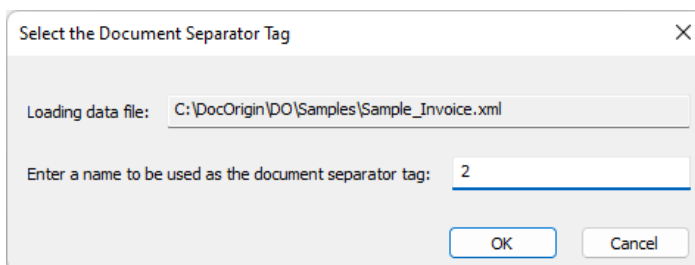


Once the data file is open, your Data Explorer will display the data structure.



Change Document Separator Tag...

You can define your Document Separator Tag by name or by data DOM level. Simply enter the name or number in the dialog shown below.



By default, the Document Separator Tag is Document. If your Document Separator Tag changes by name but is always the second level of your data DOM, use 2.

Open the sample file **C:\DocOrigin\DO\Samples\Sample_Invoice.xml** in the Data Explorer and change the Document Separator Tag to each level to see the results.

```
<?xml version="1.0" encoding="UTF-8"?>
<Sample_Invoice>      <!-- Level 1 | Rare, only used for single document data files -
only 1 top level is allows in an XML document -->
  <Document>          <!-- Level 2 | Most common level -->
    <InvoiceNo>2009-1704-3</InvoiceNo>
    <CustomerInfo>
      <ShipTo>        <!-- Level 3 | Rare -->
        <CompanyName>Perfect Printers</CompanyName>
        <Address1>425 Lansing Drive</Address1>
```

See Also

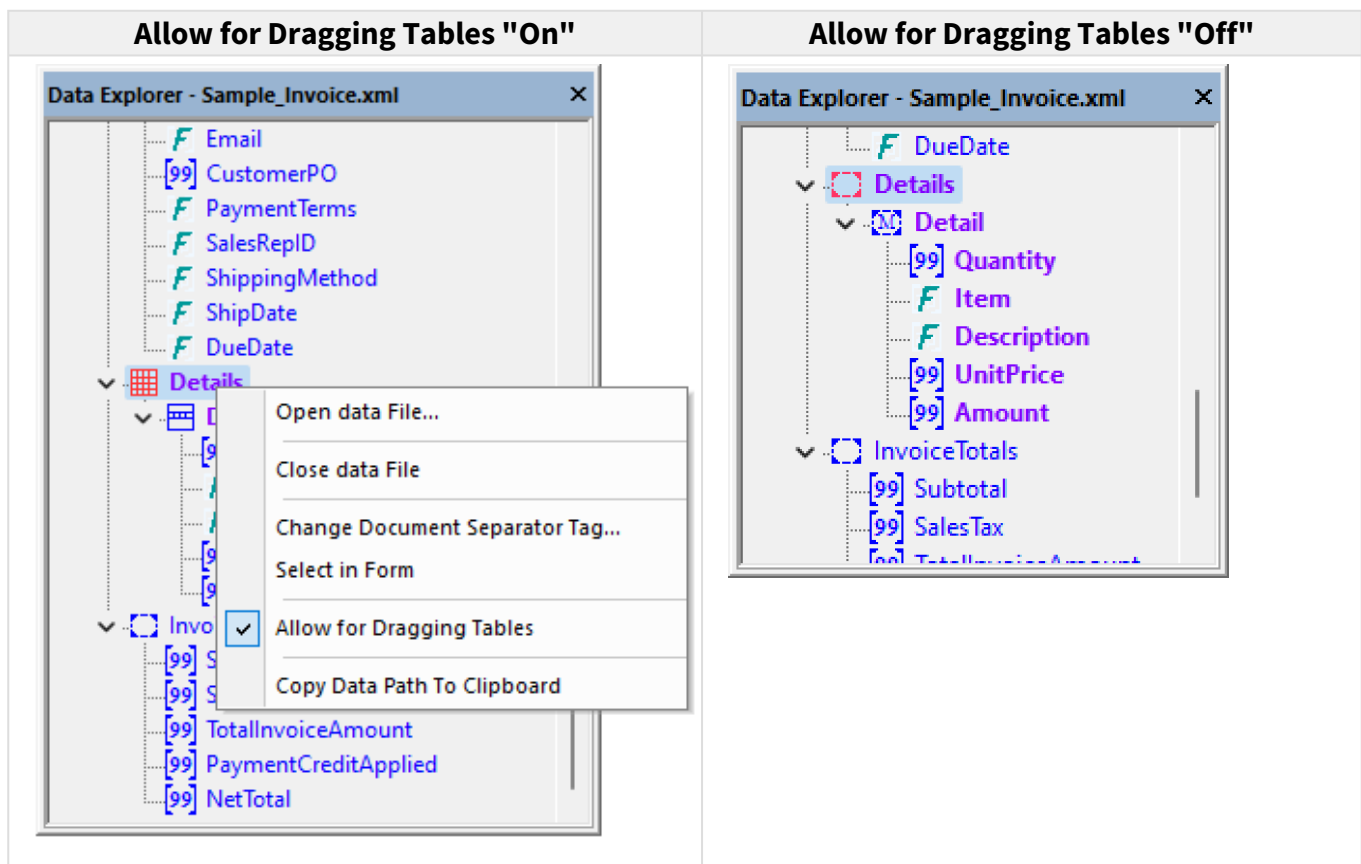
[-documentTag](#)




Select in Form


The Select in Form option allows you to right-click on the Data Explorer field and highlight the field in the Design Workspace.

Allow for Dragging Tables

You can check the option **Allow for Dragging Tables** "On" or "Off". Right-click on the Data Explorer to choose your desired option.



Allow for Dragging Tables "On"	Allow for Dragging Tables "Off"
<p>When the Allow for Dragging Tables is "On", the table icon  will show <i>if</i> the data element can be dragged as a table. The repeating child node will appear as a row icon .</p>	<p>When the Allow for Dragging Tables is "Off", the multiple occurring icon  will show if the data element is repeating in the sample data.</p>

 **Allow for Dragging Tables** is only available for a single XML parent node that has a repeating child node, as shown in the sample above. If the data contains additional repeating nodes, such as specifications, panes with fields will be created when dragging data from the Data Explorer.

By default, panes are used when dragging from the Data Explorer.

Allow for Dragging Tables "Off" (Default)	Allow for Dragging Tables "On"										
	<table border="1" data-bbox="870 789 1484 821"> <thead> <tr> <th>Quantity</th> <th>Item</th> <th>Description</th> <th>UnitPrice</th> <th>Amount</th> </tr> </thead> <tbody> <tr> <td>Quantity</td> <td>Item</td> <td>Description</td> <td>UnitPrice</td> <td>Amount</td> </tr> </tbody> </table>	Quantity	Item	Description	UnitPrice	Amount	Quantity	Item	Description	UnitPrice	Amount
Quantity	Item	Description	UnitPrice	Amount							
Quantity	Item	Description	UnitPrice	Amount							

Copy Data Path To Clipboard

As of 3.2.001.12, the Copy Data Path To Clipboard allows you to copy the data DOM for Direct Data Binding Tag.

See Also

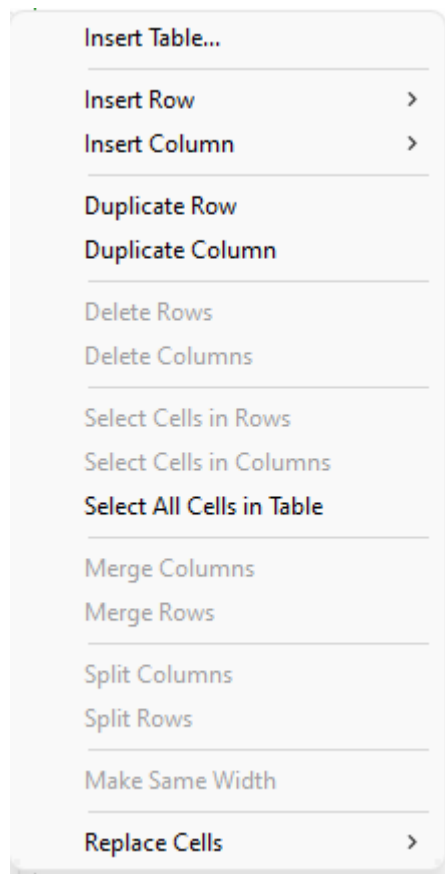
[Direct Data Binding](#)

Form Explorer Options


Form Explorer provides direct, quick access to some options for the form object properties. In addition, it provides some features to expand and collapse the Form Structure. To access these options, right-mouse click on any object in the Form Explorer.

Table

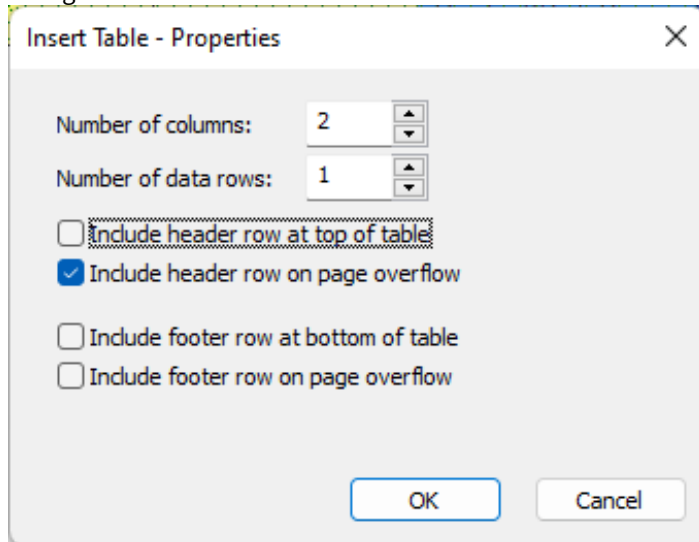
The Table sub-list allows you to insert a new table or modify the existing table. Most options are self-explanatory.



One option that could use further explanation is **Replace Cell**. This allows you to change the object type from one to another. To best explain this:

- open the sample form **C:\DocOrigin\DO\Samples\Sample_NestedTableWithRowShading.xatw**
- see the **SalesByProductTable** table
 - You can try this yourself by selecting the field called **FullName**
 - Right-mouse click and select **Table > Replace Cell > Table**
 - You will see that a table with 1 header, 1 row, and 4 columns was created. This is because that is the current default for the Table objects.
 - If you want a different table design, click on the down arrow to the right of the Table Icon 

- Change the number of columns to 2 and turn off the Include header row



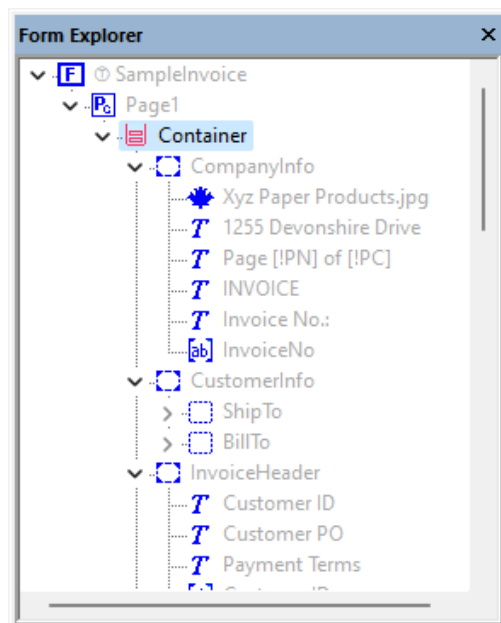
- Selecting the field called **EmployeeID**
- Right-mouse click and select **Table > Replace Cell > Table**
You should see a different table inserted

Link Objects into Pane

This feature lets you select one or more objects and link them into a Pane. Unlink removes that pane from the tree and its children become the child of that unlinked pane's parent. Note, you cannot unlink a pane that is a direct child of the Container.

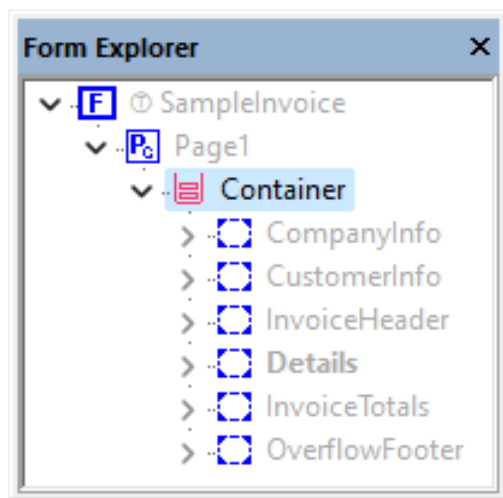
Expand All Panes

This feature expands all panes in the Form Explorer, regardless of which object you right-mouse click.



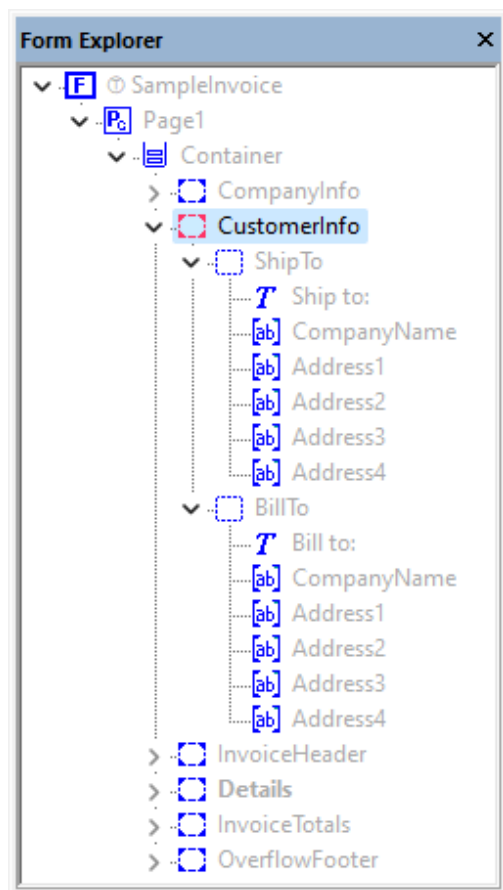
Collapse All Panes

This feature collapses all panes in the Form Explorer, regardless of which object you right-mouse click.



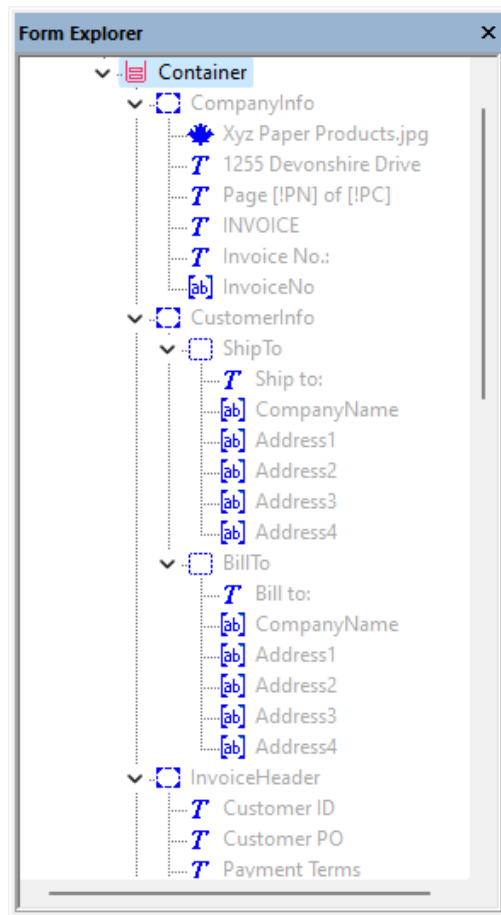
Expand Children

This feature expands all panes, tables, and groups in the Form Explorer *from the point of the right-mouse click*. This means it will only expand *that* branch. As you can see in the image below, the CustomerInfo branch is expanded and the other branches on the same level remain collapsed.



Collapse Children

This feature collapses all panes, tables, and groups in the Form Explorer *from the point of the right-mouse click*.



✓ Navigating Complex Form Structures

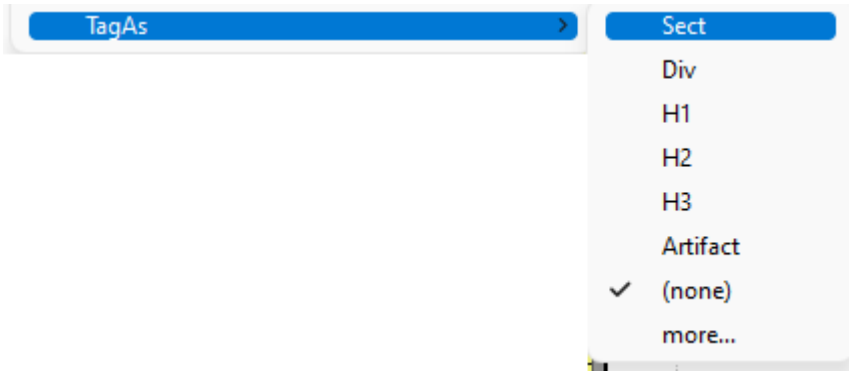
Sometimes the form DOM is expansive, matching complex data structures such as Insurance Policies or TelCo Bills. So it is useful when navigating the Form Explorer to **Collapse All Panes**, then select the branch you want to work in and select **Expand Children** for that branch only.

Copy Form Path To Clipboard

(As of 3.2.001.12) The Copy Form Path To Clipboard allows you to copy the form DOM for scripting. This is similar to the result of the Script Editor when you select **Insert Object Reference**. However, that form object structure returns the relative structure from the object you are scripting in. Meaning, if you are creating a new field called **City_State_Zip** in a Group called **ShipTo** and you reference City, State, and Zip within that same group, the script will insert only the field name. If you use the **Copy Form Path To Clipboard** feature, you will capture the entire form structure from the document level. For example, if you select the field Address4 in the ShipTo pane of the Sample_Invoice.xatw form, the result is `_document.CustomerInfo.ShipTo.Address4`.

TagAs

The TagAs option lets you select the PDF/UA TagAs options from a list of options.



FilterEditor

⚠ Caution: This has been updated for 3.0.002.02 and later

DocOrigin FilterEditor (aka FE) is a GUI tool that provides for an interactive way to design rules for extracting data from textual overlay / spool / print /report files (we call them **Source Files** in this document) that are produced by existing systems. FilterEditor extracts the data based on the defined rules and produces an XML data file that can be consumed by DocOrigin Merge and possibly many of your own or your partners' XML data requirements.

The result of a FilterEditor design session is an .xfilter file which is then used at Merge runtime (or standalone) to convert the source file into XML. See [Merge Filters](#).

ℹ DocOrigin FilterEditor, like all DocOrigin GUI apps, is supported on Windows 64-bit only.

Subject Matter Expertise

FilterEditor cannot automatically detect where fields are, how wide they are, or what their target field names are. You must understand the source file and be able to identify sentinel locations and values that you and the script you are producing can count on.

Once you have defined a discernable sentinel you can define any number of data extractions relative to that sentinel's location. And you can have the script continue to look for other sentinels which when found can be used to do more sentinel-relative data extractions.

Find sentinel -- extract, extract, extract...

Learning FE

- ✓ Opinion: Do not use the supplied .xfilter examples for learning. Those bombard you with too much. Perhaps they are fine after the fact, but NOT when you are learning. Come armed with only a sample source file and a sample target XML file.

Beginner's Tip

In my opinion, you need a sample XML data file target to start – **even if it is an empty one.**

Make sure that your sample XML data file has the usual DocOrigin structure. That is...

```
<?xml version="1.0" encoding="utf8"?>
<XmlData>
  <Document>
    ... more if you have it
  </Document>
</XmlData>
```

FE normally looks for the <Document> tag, but you can specify a different tag name when you create a new filter.

Here is a sample collateral of an empty XML to start with [empty.xml](#)

If you start with an empty XML data file, you should first add as many data field element names as you can, including any desired XML structure. This then gives you target fields to place your extracted data from the flat textual input you are dealing with. You can add more fields at any time, but it is surely best to have the field added before defining the extraction that fills it.

Sample XML File

You NEED a sample XML target file.

We are about to answer the age-old question of which comes first, the egg or the chicken.

In the absence of the provision of sample target XML collateral, **the form design comes first.**

The form design is your ticket to getting a sample target XML file. With that XML file, **and subject matter expertise** (no getting around that), you are on your way to a productive FilterEditor session. Without it, you are on your way to hair loss.

You are extracting the data so as to present it in a professional document format, i.e. to present it as merged into a form (document) template. So there is no loss in designing the form first, and a lot of gain. The first iteration of the form design need not be the ultimate in layout perfection, but it will set the framework of which data items are needed and whether they are in panes that are allowed to (expected to) occur multiple times. With even just that much of the form design done, you have taken a giant stride. I aver that you now know more about your ultimate document presentation objective; you know exactly which data fields are needed, and by virtue of Design's ability to create and save a data file with automatic contents that reflect the expected data structure, you have the grail of the **mandatory** target XML data file for your FE "ride".

In fact, using the sample data produced for a form design is such a common operation that FE now allows you to specify an XATW file as the sample XML file. FE doesn't interpret that literally. If the user has chosen an XATW file, FE will cause sample data for that form file to be produced and will then use that generated sample XML as sample XML to assist in creating your filter.

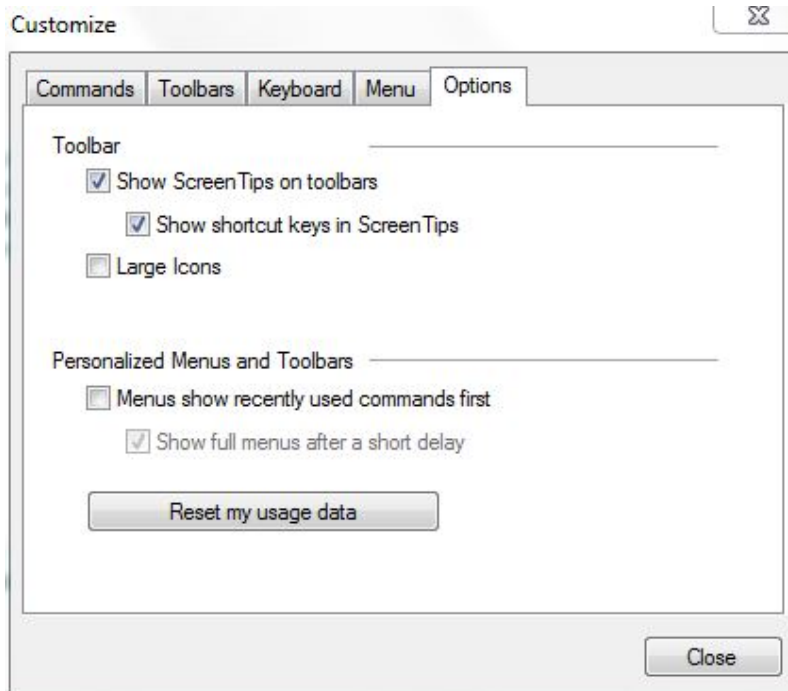
- Acquire subject matter expertise
- **Design the target form**
- Use auto-data generation to get a sample data file
- NOW, start FE, and load up that XML file, and load up the print/overlay/spool source file

You are on your way to a rewarding FE session.

Show Those Menu Items

There is a Microsoft standard, user-configurable option for determining whether the menus in an app automatically show all their entries right away or instead show only recently used menu entries. You are **learning**. You **need** to see all possible menu items right away. You *will* see things that intrigue and inform you. You want to see all menu items when you are learning any app.

Use **View > Toolbars and Docking Windows > Customize...** – choose the **Options** tab.

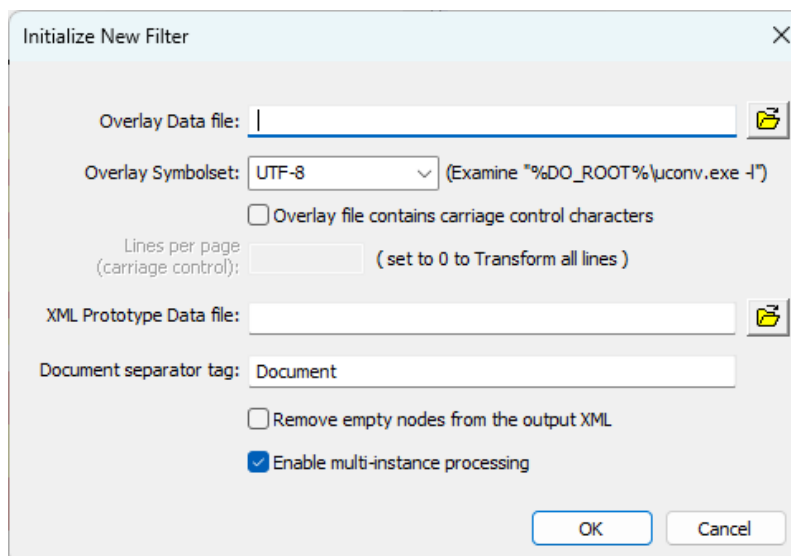


FE Show full menus

Ensure that *Menus show recently used commands first* is unchecked. Now you have a hope of learning.

Filter Editor - 101

For your first real FE session, after you wasted your time looking at a goulash of samples. (*I told you not to*), you launch FE. You do a File – New You will see:




FE Designing a new filter

Supply the name of your overlay/print/spool text "source" file. One that you understand and have the needed subject matter expertise for.

Supply the name of the XML file that you saved from Design using your initial form design, the one that is the eventual target of the data you are about to scrape off of the above source file.

Now you're cooking.

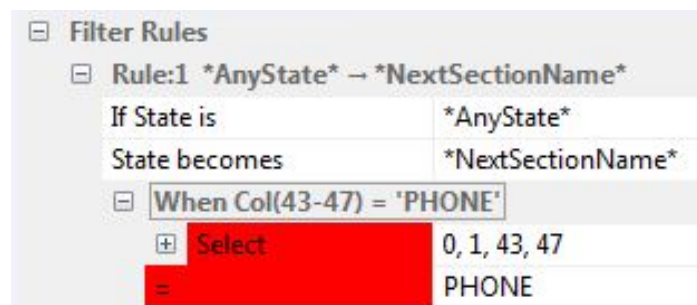
 If you can't supply those two things, you are not ready for FE 101.

How Does It Work?

In FilterEditor, you define *Rules* and *Actions*. Each Rule is really a specification of something to look for in the coming lines of the source file. I call it looking for a *sentinel*. An Action is, at least 99% of the time, an *extraction*, that is to grab some data off of the source file and put it in the target XML file. FilterEditor in a nutshell is "Find sentinel, extract, extract, extract", "Find sentinel, extract, extract, extract", ...

Defining a Rule

Look over the central source file canvas area. Do you see some string of text that indicates the beginning of (or at least anchor point for) a section of the report? Select that text with your mouse or shift-arrow keys. Now right click that selected text and choose: Add Rule. FE will add a Rule for you. You will see it in the right-hand panel which is named "Filter Rules". It will look something like this.



FE Defining a Rule

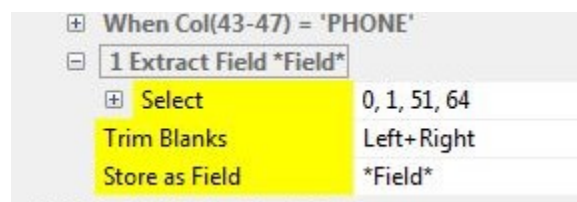
Your job is to type overtop of `*NextSectionName*` to provide the name of the State that you believe the detection of this sentinel puts you in.

You [have a] Rule!

Defining an Extraction

Back to the source file. Do you see a piece of text that should be "scraped off" as a value for a field? Select that text by mouse or shift-arrow keys and right mouse click it. In the context menu choose **Extract Data**.

A data extraction Action will be added below your Rule. It will look something like:



FE Defining an Extraction

Your job is to replace the `*Field*` entry with the name of an actual field in your XML file. That's easy because you have a dropdown of the entire XML structure to pick from.

Review

- You have added a Rule.
- You have added a Data Extraction

Uh, that's all there is! Easy! Add more data extractions. When you have extracted all the data you can based on the current sentinel, select the text for another sentinel, and Add Rule. Repeat, repeat, repeat.

Rule order

The Rules are processed, top to bottom. The first one that finds a sentinel wins, and the extractions related to that found sentinel are done. Don't worry about the order of Rules right away. It is terribly important but there is a Move Up/Down capability so you can work the order out later.

States

When one of your Rules finds a sentinel, it always sets a *State* name. In fact, each Rule is of the form **If State is xxx, then look for this sentinel**. This is very important. You may have hundreds of Rules defined, but only the Rules that apply to the current State of the line being scanned matter, the others are skipped. So it is not a free-for-all of looking for every possible sentinel all the time. It is controlled; look for only those sentinels that apply to the State the current line has, and if you find one, you most likely go into a new State, where different Rules apply. In essence you "walk" down the source file going from State to State.

A tiny update. In fact, you can provide a comma-separated list of **If State is** States, but the effect is still the same. Not all Rules apply, only those pertinent to the State the current line is in.

The *Start* State

FilterEditor always begins in the *Start* state. This can be a convenient state to use when looking for something at the top of the file. Using the *AnyState* state would check such a rule on every line. But the *Start* state is very specific to the top of the file.

The *AnyState* State

There is a special "pseudo" State with the name *AnyState*. The *AnyState* State means 'I don't care what State the current line has'. If my Rule says "If State is *AnyState* ..." then that Rule applies regardless of what the current State of the line being scanned is. Because such rules are so widely applicable, they are generally placed at the bottom of your list of Rules. Rules which are state-specific should precede those that are "any old state" non-specific. Use the Move Up/Down feature available as a right mouse click context menu on Rules to reorder your Rules, highly specific up top, highly general at the bottom.

The Current Line

Unsurprisingly, FilterEditor (FE) has a concept of *the current line*. That current line can move only forward, never back up a few lines, inexorably forward. Sometimes this is also called the **anchor** line. To begin with, FE reads the first line of the source file. It then cycles through all the Rules that apply to see if this line has a sentinel in it. If it doesn't, it moves on to the next line and repeats running through all applicable Rules. Eventually, it finds a sentinel. Once a sentinel is found (FE has matched the Rule's State name to the line's current State and the "Test" has succeeded), the Rule's Actions (if any) are performed. Typically these actions are Extractions of field data. Once the Actions are complete the current State changes to the one specified in the Rule as the "State becomes" name. The "current line" is advanced by 1 to the next line in the source file, and the whole process cycles again.

Extractions

Once a Rule has successfully matched the current line in the source file, you can add Extraction definitions which tell FE how to take data fields from the source file and store them into the XML file.

Extractions are not bound by the endlessly forward movement of the current line, nor are they bound by one line at a time inspection. Extractions specify relative line and absolute column ranges. They can reach back to before

where the sentinel was found, using a negative line number. They can reach forward as many lines as necessary to nab the data of interest. Having found a sentinel, a guidepost, you can pick data from all over the place *relatively* speaking. Think of a "header" section of a report – it's probably multi-lines long. If your Rule has found a sentinel that indicates the presence of that "header", then you will likely want to do extractions of many fields from that header area. Do so – you don't need to find more sentinels within the already identified header area.

Doing an extraction has no effect on FE's notion of the current line. However, one Action that you may choose to include after doing a bunch of extractions from an identified header area is to Advance n lines so as to skip over the rest of the header area before starting to look for sentinels again.

Variable Length Extractions

This is advanced material. Please skip it until you have had a good deal of experience with FE.

Per the above, you can specify the exact columns and lines that you want to extract. But what if the number of lines varies. A classic example is to want all the description lines up to the next detail line header. A Custom JavaScript function could certainly get you there but the **span** action step *may* be sufficient.

1. Select the first line of the data to be extracted. Leave the number of lines at 1.
2. Right-click the Select line of the extraction definition.
3. Choose the **Add Action Step** context menu entry.
4. Select the Span `only if not=` offering.
5. Fill in the column start and end arguments and the text that will terminate the selection, e.g. text that occurs in the next detail line header

You will now be selecting as many lines as exist until the 'sentinel' that you identified occurs. Depending on your circumstances you may choose to use the "Span only if =" offering.

Moving On

When all the extractions defined for this 'sentinel find' are done, it's time for FE to resume looking at the next line to see if it can find any applicable sentinels in it. Just as it was doing when it started out. It may seem that with all the extractions that just went on, the current line is all "picked over". Irrelevant. Extractions from a line have no bearing on a line's still being subject to sentinel searches. Line-by-line, that *current line* will move forward, and each line will be searched for sentinels that apply for the then current State. (An explicit Advance n Action can jump the current line ahead by more than 1.)

Filling In the XML

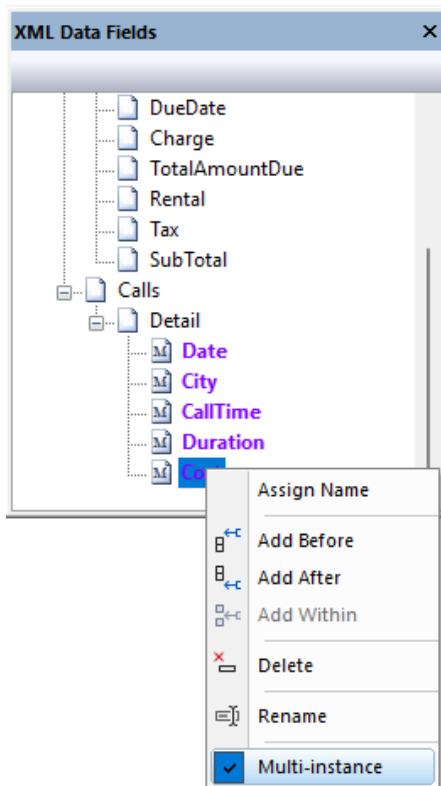
When you define an extraction, aside from selecting the area on the source file to be scraped off, you also nominate a target element in the XML file. Initially that's dead easy. But eventually you are going to do an extraction that wants to go into a data element that you already put stuff into. Now what? When this happens, FE will automatically make a duplicate empty copy of the parent XML structure including all it's child nodes. This in effect creates a duplicate "record" for more data. The element that you just scraped off the source file now has a home to go to. And the upcoming sibling scrapings also have fresh new homes to go to. It's really quite a simple process, but you do need that target XML to make it feasible.

XML Output Data Structure with Multi-Instance

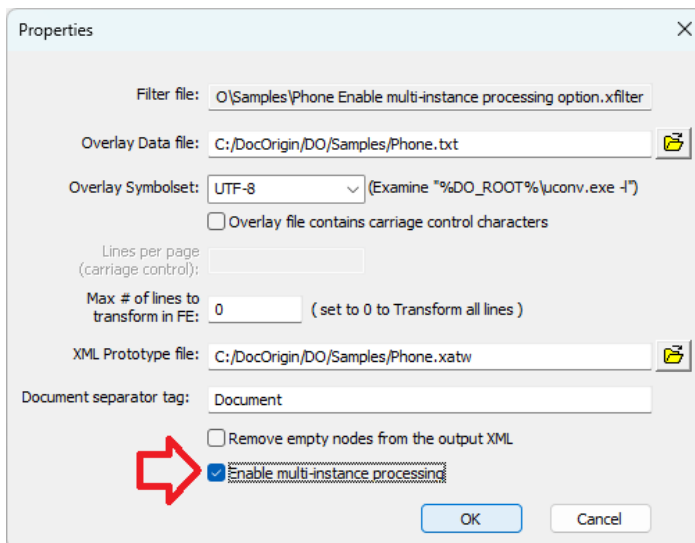
(As of 3.2.001.07)

In cases where a target form has multiple fields with the same name that are not global, it may be desirable to adjust the XML Output Data structure by marking select nodes in the XML Data Fields as "multi-instance".

This can be done in the XML Data Fields by right-clicking on a node and selecting "Multi-instance". Instead of Filter Editor automatically creating a duplicate empty copy of the parent XML structure, including all its child nodes, FE will duplicate the child nodes as many times as necessary within a single set of parent node tags.



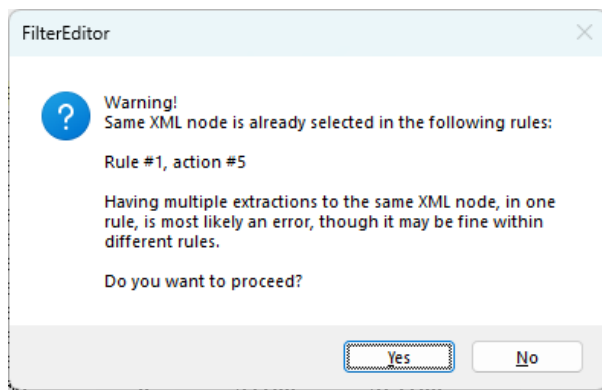
You can define the “Enable multi-instance processing” when you Initialize the xFilter for the first time or set it in the **File > xFilter Properties** menu.



❗ xFilter Properties > Enable multi-instance processing option

Note that for backward compatibility, “Enable multi-instance processing” must be selected in the **File > xFilter Properties** menu to activate the XML restructuring. The option is shown “On” in the previous screenshot.

When a user assigns an extraction to a field that is already mapped, DocOrigin will prompt a Warning to confirm the user wants to proceed.



The results will be very different based on the option you choose. The table below shows the same overlay file with the same XML structure, but one shows the results when the Enable multi-instance processing option is "On," and the other shows the results when it is "Off".

Enable multi-instance processing option "On"	Enable multi-instance processing option "Off"
<pre> <Detail> <Date>03-11-2001</Date> <City>Santa Barbara</City> <CallTime>15:24</CallTime> <Duration>50</Duration> <Cost>5.50</Cost> <Date>03-23-2001</Date> <City>Billerica</City> <CallTime>10:15</CallTime> <Duration>7</Duration> <Cost>.50</Cost> <Date>03-27-2001</Date> <City>Reno</City> <CallTime>10:15</CallTime> <Duration>50</Duration> <Cost>5.50</Cost> </pre>	<pre> <Detail> <Date>03-11-2001</Date> <City>Santa Barbara</City> <CallTime>15:24</CallTime> <Duration>50</Duration> <Cost>5.50</Cost> </Detail> <Detail> <Date>03-23-2001</Date> <City>Billerica</City> <CallTime>10:15</CallTime> <Duration>7</Duration> <Cost>.50</Cost> </Detail> </pre>

Non-Extraction Actions

Not quite every Action is an extraction. Some Actions may have nothing to do with text selected on the source file. They are just things that you want to have happen whenever this Rule fires. In the Filter Rules panel, right mouse-click on a Rule, and choose Add Action. You will get a dialog that identifies a set of Actions that are available. Choose one and it will be added to the list of actions (including extractions) that are performed when the Rule is triggered.

Start New Document

The xfilter processing automatically gives you a start-new-document tag, i.e. <Document> at the start of the XML file. However, if the report you are scraping can contain multiple documents' worth of data, then you should (must!) include a **Start New Document** Action in some Rule. A classic would be the detection of "Page 1" somewhere in the source file. There must be some means by which you can identify when a new document's information begins. Use that to define a Rule and add the **Start New Document** Action to that Rule.

Extraction Steps

You will immediately notice that an Extraction (really a special kind of Action) is not a "one-liner". In fact it is a macro that revolves around the text that has been selected on the extraction. It could be that you will want to "fiddle with" that extracted text before you stuff it into an XML element. As a strong default, FE automatically does a "Trim blanks" *Step*. It then does a "Store as Field" *Step*. This is what you want, and all you want, most of the time. But you can get adventurous by right-clicking on the Extraction line, and choosing "Add Step". It brings up a dialog of available steps. It's much *like*, and even overlaps the dialog for "Actions", but it is different. The steps are generally ones that involve the text that was selected for the extraction. You can add as many steps as you like. You can delete them too. It is remotely possible that you might even delete the "Store as Field" step.

The Lexicon

1. **Rules** are Rules; order matters; find a sentinel
2. **Extractions** – a common, specific type of Action.
3. **Actions** – especially non-extraction Actions. One-liners: 'take this action'
4. **Steps** – apply to only Extractions; used to play with the nominated text in some way
5. **State** – the data transformation process always maintains a current State name. this State name is matched against the State defined in each Rule to determine which Rules to execute. After execution of a Rule's Actions, a new current State is set.

Sentinel Detection

A Rule automatically gets a when clause that refers to the sentinel text that you highlighted and specifies an equal (=) condition. That's almost always what you want. However, all those arguments of the Rule are editable by you. You don't have to go precisely by the text that you had highlighted; you can freelance it. In particular, one sometimes wants to change that equals to 'not equals'. You can do that by right mouse clicking the = and choosing `Change Test`. You'll find an interesting selection of tests to choose from. They are all meant to provide a true/false result. You might want to study them sometime, but mostly only for keeping in your back pocket.

Ways Into a State

Sometimes there is more than one way to get into a State. But you have a strong desire to express the Extractions for that State in only one place. The classic example that we hit is coming from State "A" to a Detail line, then being on a Detail line, wanting to get to the next Detail line. You could open it up completely and express your rule as.. If State is **AnyState** etc, but that might be a bit too open, a little unnerving even. Instead, you can claw back control by keying in a comma-separated list of States so that you end up with something like: If State is A,Detail etc. That way your sentinel selection is predicated on a chosen list of current States, not the open-ended **AnyState**. It works quite well.

FilterEditor JavaScript

One of the *Steps* you can perform on an *Extraction* or an *Action* is the **JavaScript** step. This allows you to insert a line or a whole block of JavaScript to be executed. Using this script you can access the currently selected line or lines of text, change them, access the current output XML tree, and other text lines in the source file.

A set of access functions are pre-defined to allow you to access various settings and values within the transformation processor. And you also have the option to define your own custom functions.

Custom Functions

You can use the FilterEditor **Filter>Edit Script...** menu item to define additional "helper" JavaScript functions that can be called from a **JavaScript** action.

As an example, you could add the following JavaScript function to remove commas from a numeric string:

```
function removeCommas() { // remove commas
  var text = $$GetText(); // get current selection
  var newtext = text.replace(/,/g, ''); // remove commas
  $$SetText(newtext); // update selection
}
```

In the above...

\$\$GetText() is a built-in xfilter function to let you access the selected text for this Extraction.

\$\$SetText(theText) is a built-in xfilter function to update that selected text – not on the source file canvas, but in memory such that subsequent Steps, such as **Store as Field** would be using that updated text.

Now that you have defined your custom function, how do you use it?

Besides the **Filter>Edit Script...** menu item, you can provide some JavaScript in two places.

1. If you were to right-click on a Rule or on an existing Action under a Rule, you could use the **Add Action...** context menu entry, and thence select "JavaScript Execute some JavaScript code". You will get an edit window that allows you to add the script.
2. If you were to right-click on a step of an Extraction Action, you could use the **Add Action Step** context menu entry, and thence select "JavaScript Execute some JavaScript code". You will get an edit window that allows you to add the script. A typical place to do this is by right-clicking on the Trim Blanks Extraction Action step.

What if you want to edit your script?

In either of the above two cases, you will see an Action or an Action Step that is led off by the word **JavaScript**. Click on the area to the right of the word **JavaScript** and you'll get an edit window that allows you to view or edit the script. As you exit that window FilterEditor will automatically "compile" the script and check for common JavaScript syntax errors.

It could look like this:

[-] 3 Extract Data Statement.Details.Detail.Over90Amount	
[+] Select	0, 1, 113, 124
Trim Blanks	Left+ Right
Javascript	removeCommas();
Store as Field	Statement.Details.Detail.Over90A...
[+] 4 Extract Data Statement.Details.Detail.Over60Amount	
[+] 5 Extract Data Statement.Details.Detail.Over30Amount	

FE Custom JavaScript Call

Notice how easy it was to invoke your pre-defined custom function.

All the standard DocOrigin add-on functions are available - including `_message()`, `_logf`, `_file`, `xmlClass`, etc. By default, FE produces its own log file named `FilterEditor.log`.

✔ Custom Functions are IMPORTANT

PLEASE Do not skip by the above topic of defining your own reusable custom JavaScript functions. You may put them in a file and simply `#include` that file, e.g. `#include "$S/feUtils.wjs"` in every xfilter that you create. If something is difficult using the bare FE rules and actions, define a custom function once and add it to your repertoire. Make your life easy.

FilterEditor Functions

A number of additional functions are available within the xfilter scripting world. These routines are used to access or modify the processing of the transaction processor.

\$\$Advance(*n*) - causes the transformation processing to skip forward in the source file "*n*" lines.

ⓘ The xfilter Transformation process always advances one line in the source file after every successful Rule and Actions are processed. So calling `$$Advance(0)` still means that the new State will be looking at the next line in the source file. If you want to skip the next "*n*" lines in the source file, you must call `$$Advance(n-1)`. `$$Advance()` cannot move backward in the source file. Darn it!

\$\$GetActionNumber() - returns the current Action's number. Useful for debugging messages.

\$\$GetCurrentState() - returns the State name that the current Rule matches - that is, the State name(s) from the "If State is" section of the Rule.

\$\$GetField(*name*) - get the text currently assigned to field *name* in the output XML tree. This will retrieve the current or most recent value assigned to *name*. *name* can be either the "leaf" or field name, or a full dotted syntax such as "Header.Address".

\$\$GetLineCount() - returns the number of lines in the entire source file.

\$\$GetLineNumber() - returns the current line number within the source file. Useful if you're logging or displaying an error message. NOTE - lines are numbered starting with 0 as the first line in the file.

\$\$GetNextState() - returns the name of the next state the current rule will transition to. Useful for debugging messages.

\$\$GetRuleAnnotation() - returns the annotation text associated with the current rule (or null). Useful for debugging messages.

\$\$GetRuleNumber() - returns the Rule# of the current rule being processed. Useful for debugging messages.

\$\$GetText() - returns the current Selection text. If the selection is a single line (typical) that string is returned. If the selection spans multiple lines, an array of strings is returned.

\$\$GetTextLine(*n*) - fetch the *n*th overlay line relative to the current rule's trigger line. This function returns the entire "raw" input line but without the trailing newline (`\n`) character. *n* can be negative.

`$$GetTextLine(-$$GetLineCount())` will return the very first line in the file.

`$$GetTextLine(0)` will return the line that caused the state assignment rule to trigger.

`$$GetTextLine(1)` will return the line that follows the line that caused the state assignment rule to trigger.

\$\$Return(*true/false*) - sets the Test logical state when a JavaScript statement appears within the Test part of a Rule. By default, all scripts return "true" unless this routine is called.

\$\$SetField(*name, value*) - updates the current XML tree field with a new value. Note that it is somewhat different from how the "Store a Field" operation works. "Store as Field" will not **replace** an output value, but instead

automatically creates a new instance of the field's parent structure or pane.
 \$\$SetField will replace a current value.

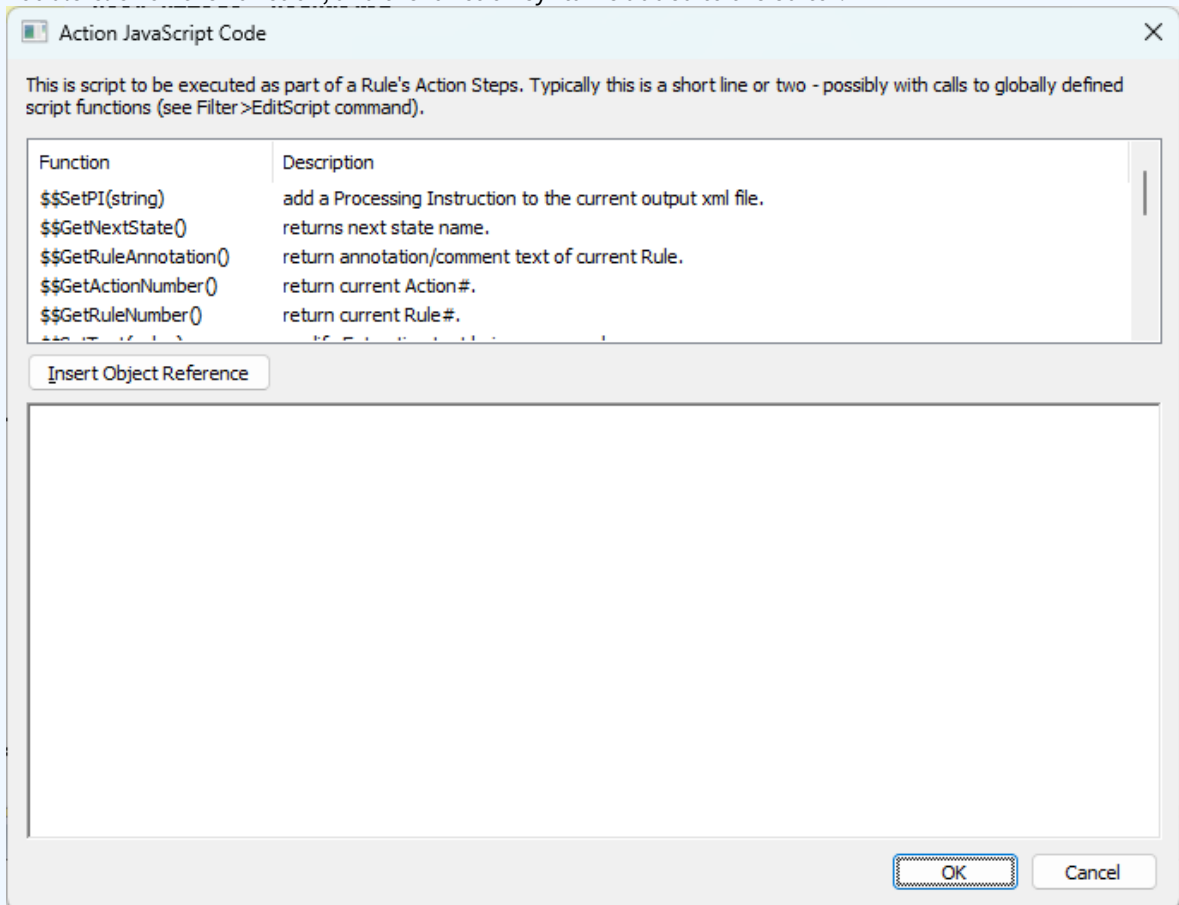
\$\$SetNewState(*name*) - overrides the name of the State when this Rule is complete. This overrides whatever is currently set in the "State becomes" section of the Rule.

\$\$SetPI(*string*) - adds a DocOrigin Process Instruction to the start of the generated xml file. *string* is the text that will appear in the PI.

\$\$SetText(*newtext*) - update the current Selection to be this new text. *newtext* can be either a single string or an array of strings.

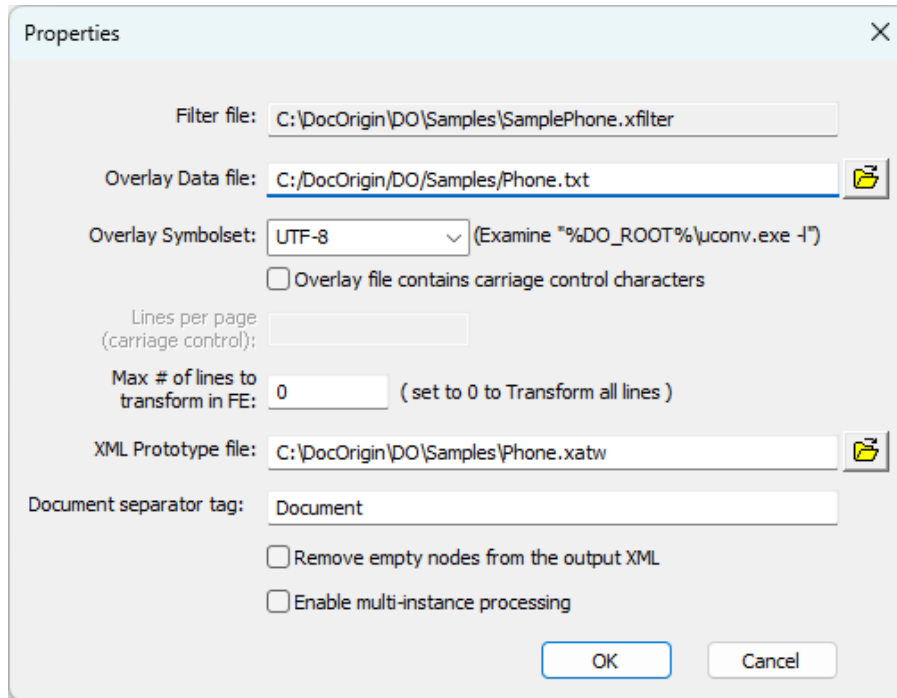
JavaScript Dialog Support for version 3.3.003.01 and above, you can:

- Click Insert Object Reference, and the object `path.name` is added to the editor.
- Double-click on the Function, and the function syntax is added to the editor.



xFilter Properties

Let's review the options of the xFilter Properties in depth.



Filter file

The name of the current xFilter file. If this is blank, please save your file.

Overlay Data file

The name of the sample overlay file to use for developing your xFilter.

Overlay Symbolset

What Symbolset is used for your data file? UTF-8 is the default and most common.

Overlay file contains carriage control characters

(Checkbox) Self-explanatory.

Max # of lines to transform in FE

Limit the number of lines processed. Zero (0) is the default. Set to 0 to transform all lines.

Setting this value to a set number of lines is rarely used. It can be defined if you only want to process the first 10 lines of the input data file.

XML Prototype

Defines what XATW or XML used to define that data mapping.

Document Separator tag

Define the tag used to define each document. The default is Document. Any word can be used. However, merge assumes the document tag is Document. And if it is different, the document tag can be defined by the number in the XML node or tag name with the merge option `-documentTag`.

Remove empty nodes from the output XML

For Filter processing, we can create a data tree with panes with no descendant fields set to an extracted value. So the whole pane is really unnecessary in the output stream and could cause optional panes to be instantiated unnecessarily. If the option is checked "On" and DocOrigin finds a node where all those descendant fields (nodes with no descendants) are empty, we eliminate the node. This is recommended to check on.

XML Output Data Structure with Multi-Instance

(As of 3.2.001.07)

In cases where a target form has multiple fields with the same name that are not global, it may be desirable to adjust the XML Output Data structure by marking select nodes in the XML Data Fields as "multi-instance".

FolderMonitor

FolderMonitor is an application for processing "submitted" data files. FolderMonitor runs as a background task on a server. The system administrator will have had to make decisions on where collateral, e.g. form files will be deployed on the server, where scripts are kept, and where data files are to be submitted for processing. User applications write data files into a specific file folder(s) that are monitored by FolderMonitor. When files appear in these folders, FolderMonitor will process one file at a time; determine what processing is required on it, and execute a nominated-by-you JavaScript script to do the processing.

FM on Windows...

There is a FolderMonitor Service (DocOriginFolderMonitorSvc) that runs as a Windows Service. Typically it is set to start automatically on reboot.

There is also a GUI program — DocOriginFolderMonitorConsole — that allows a system administrator to start, stop, and look at the log of a FolderMonitor process.

When the system administrator uses the FolderMonitorConsole app to start FolderMonitor processing, the service launches the actual DocOriginFolderMonitor application as a background app. It is that app that watches the folders it is configured to watch, for any submitted data files to process.

On Windows, one can configure FolderMonitor to use multiple monitor instances. See [Multiple Monitors](#) and [-monitor](#). When that is done, the FolderMonitorConsole app allows you to start an individual monitor instance (or all at once, if desired).

FM on Unix...

There is no DocOriginFolderMonitor Service, nor any DocOriginFolderMonitorConsole.

The system administrator runs the DocOriginFolderMonitor process directly via the command line. Undoubtedly they will end that command line with an ampersand (&) so that the app will continue to run in the background as they continue to use the system console for other duties. Something like:

```
{path to DocOrigin folder}/DocOrigin/DO/Bin/DO DocOriginFolderMonitor -monitor "{instance or monitor name}" &
```

The instance or monitor name can refer to a section of an INI file, see FM.ini on the [Multiple Monitors](#) page. Of course, the command would also specify any needed options, almost certainly via an @xxxxx.prm command line reference.

A convenience starter shell script named 'fm', is provided in the .../DO/Bin directory. It is just a script, copy and edit it as you see fit. It enables one to say: fm start, fm stop, and several other actions that may be of interest. Do examine the script and tailor it to your needs. It is not intended to be a see-all/do-all interface for any site, but an example of what you can do. You will likely want to adjust it to reflect the queues/folders that you use. On re-install, we will overwrite 'fm', so copy it and make it your own.


Note that multiple instances of the FolderMonitor process can be run in the background. They operate independently. To get FolderMonitor's pid:

```
ps -ef | grep "DocOriginFolderMonitor"
```

You stop a FolderMonitor process on Linux the same way as you would stop any background job on Linux, by using the kill command:

```
kill -s TERM {pid}
```

`kill` does not abort the job that is being processed. The default signal sent by the `kill` command is `SIGTERM`. FolderMonitor catches `SIGTERM`. The signal handler just sets a flag and continues. Because we catch the signal, it is not propagated to `FMTransaction`, which is the child process that actually processes a job. If the flag is set when we finish processing a job, or when we have finished a scan for new jobs, we do the usual shutdown, reporting quarantined files etc. But any job being processed when the `kill` signal was received is safely done.

 We don't believe that the DocOrigin product should take over your site. Rather it should fit into what you already do. That is why we provide `fm` as only an example for you to revise, or even discard. We do not want to impose constraints on how you do your work. Nor do we want to provide a system with so many configurable options that you need another system to configure those options! We believe that we should keep it open, and keep it simple. For any individual site, the specific operations needed should be easy to accomplish -- and understand.

Command Options

FolderMonitor has several command-line options that control its operation. Those options are listed in the menu on the left of this page.

FolderMonitor also has the standard DocOrigin command-line options as described in the [Common Command Line Options](#) section of this document. In particular, it requires that `-logfile` and `-message` be defined either explicitly or in one of the default `.prm` files.

-cluster

(FMControl.prm only)

Define a cluster name.

Syntax

-cluster *name*

Description

If a server is to support multiple FM services, each for multiple FM Monitors, then each group of Monitors needs a cluster name, and the FM service also needs a unique service name. This is rarely used, likely in OEM situations.

-concurrent

(DocOriginFolderMonitor.prm only) (As of 3.0.004.13)

Define how many jobs this FolderMonitor instance can run concurrently.

Syntax

`-concurrent n`

Description

A FolderMonitor (FM) instance processes jobs from the `-queue1` through `-queue10` definitions that specify folder names and file masks to monitor. By default, each instance processes one job (i.e. data file) at a time. With the `-concurrent` option, FM can be instructed to process multiple jobs in parallel. The default value for *n* is 1 -- meaning 'process one job at a time'. The maximum value for *n* is 62. A sensible value would be in the vicinity of the number of logical processors that your server has. (On Windows: `echo %NUMBER_OF_PROCESSORS%`).

FM processes a data file by handing it off to the FMTransaction process. Before *3.0.004.13*, FM waited for FMTransaction to complete before moving on to the next file. Now FM will launch additional copies of FMTransaction up to the `-concurrent` setting value. If it can launch no more copies of FMTransaction, it will wait for one of them to end, before launching a new one. Naturally, the order in which the jobs are processed will not necessarily be first in first out. Also, the log file will have interleaved messages from all of the jobs being run concurrently. Of course, each message entry carries its JobId tag.

This setting is most advantageous when many independent, often single document, jobs are being placed in the folder(s) that an FM instance is watching. It would also be of benefit when there is an occasional large (many, many documents) job mixed in with large numbers of small jobs. In such a scenario, the large job would not hold up all of the smaller jobs.

-errorFolder

(Mandatory)

A FolderMonitor folder into which failed queued data files are stored.

Syntax

-errorFolder *foldername*

Description

This specifies the folder into which queued data files are copied if they are not successfully processed. In addition, a short excerpt from the logfile is also stored at the same time. It contains the last few logfile entries - typically highlighting the error condition.

See Also

[Command Line Processing](#)

[Command Options - FolderMonitor](#)

-exitNow

A file name that will have FM quit processing any more jobs and end as soon as the currently in-process jobs are done. I.e. a nice shutdown as opposed to a "kill".

Syntax

```
-exitNow [EXIT.NOW|filename]
```

Description

The default value is *EXIT.NOW*. So by default, if a file named *EXIT.NOW* is found in the *queue1* folder of a FolderMonitor instance, that instance will stop as soon as the current job is completed. And the *EXIT.NOW* file will be deleted. If there is an *EXIT.NOW* file in the folder when FolderMonitor is started then FolderMonitor will do one job (if there is one) before it stops. You can choose a different name for the file to be looked for by setting the `-exitNow` option. If you supply a blank value (`-exitNow=""`), then this file checking will not occur.

See Also

[-exitWhen](#)

-exitWhen


A file name that will have FM quit processing any more jobs and end as soon as the specified file comes up for processing.

Syntax

```
-exitWhen [EXIT.WHEN | filename]
```

Description

The default value is *EXIT.WHEN*. The `-exitWhen` option will cause the same thing to happen as `-exitNow` but only when the *EXIT.WHEN* file comes up for processing. By default, if a file named *EXIT.WHEN* is the next job to be processed, then the FolderMonitor instance will stop and that *EXIT.WHEN* file will be deleted. You can choose a different filename by setting the `-exitWhen` option. If you set it to nothing (`-ExitWhen=""`) then this file checking will not occur.

 This file name must fall into the job file name mask, otherwise it will never be encountered. Given a mask of `*.xml`, you might wish a name like: `ExitWhen.xml`.

See Also

[-exitNow](#)

-fileOperationsRetryCount

(As of 3.3.005.02)

Syntax

-fileOperationsRetryCount *n*

Description

Specifies the number of retry attempts for file operations (default = 1) to handle failed I/O operations.

See Also

[-fileOperationsSleepTimer](#)

-fileOperationsSleepTimer

(As of 3.3.005.02)

Syntax

-fileOperationsSleepTimer *n*

Description

Defines the sleep duration, in milliseconds, between retry attempts (default = 100ms) to handle failed I/O operations.

See Also

[-fileOperationsRetryCount](#)

-FMScript

(As of 3.1.001.10)

Identifies an override script to be run when the FolderMonitor instance itself starts and stops. That is, this is not on a per job basis, but for an entire run of a FolderMonitor instance.

Syntax

-FMScript *scriptName*

Description

FolderMonitor, by default, runs script *\$E/Default-FMScript.wjs* when it starts and when it ends. This -FMScript option allows you to specify an alternative script to be run at those times. The *scriptName* must include path information. Typically it would be *\$O/FMScript.wjs*.

When the script is invoked it is provided with an `_fm` object which contains the command line options and a set of statistics that may be of interest to the script writer. Please read the comments in *\$E/Default-FMScript.wjs*.

The statistics that are supplied in the `_fm` object are:

Property	Description
StartTime	When the FM instance started, in seconds since Jan 1, 1970.
EndTime	When the FM instance ended, in seconds since Jan 1, 1970. This is 0 when the script is called at the start of an FM instance.
PassJobs	The number of jobs that were processed successfully.
FailJobs	The number of jobs that failed, and hence were copied to the error folder.
Quarantined	If FM encounters a job but cannot get read access to it, it quarantines that file. It will retry the file from time to time. This statistic gives the number of files that were still in quarantine when FM was about to end.
Pauses	FM will continue processing jobs until all its queues are empty. At that time it will pause for the time specified by the -pause option. This statistic identifies the number of times that FM paused because it had no jobs to do. You might use this statistic to judiciously adjust your -pause time.
RC	This is the return code that FM is about to exit with.

The above properties can be accessed case-insensitively. (Unusual for JavaScript.)

See Also

[Command Line Processing](#)
[Command Options - FolderMonitor](#)

-JPSName

Skip job name discovery and use the supplied name for the job processing script.

Syntax

-JPSName *filename*

Description

The -JPSName option specifies the JavaScript file to be executed to process the job. If this option exists the job name discovery script will not be executed, rather the script named herein will be used as the job processing script.

If this option is not specified, and there is no DocOrigin PI then the usual job name discovery script will be invoked.

If the option does specify a name, then the Folder Monitor (FM) processing will look for that job processing script (JPS) in the designated "script folder" . The JPS file name is the provided job name concatenated with ".wjs". For example, if the option specified "Invoice" then the FM processing would look for a file named "Invoice.wjs" in the folder identified by the -scriptFolder option. If that file is found, that script will be launched and its eventual return code will determine whether FM considers the job to have succeeded or failed. The JPS can perform many, many steps. It's completely open-ended. Typically it does include a DocOrigin Merge step.

If the job processing script (JPS) file is not found, FM will revert to using \$E/Default-JobProcessing.wjs. Note that that default script has a ##include \$0/JobProcessing.wjs and so if you have created such an override file, that override script will be used.

If the data file is XML and has a DocOrigin processing instruction that includes a job=xxxxx attribute, then this -JPSName option will be entirely ignored. The job attribute tells FM which JPS to run.

Usage

Oftentimes a given FolderMonitor instance is used for only a single job type, e.g. only packing slip jobs are placed in the instance's watched folder. That is an ideal time to use -JPSName. Depending on your personal taste, your -JPSName script could still handle multiple job types. A JPS can do everything a JND can do. There would be nothing stopping your JPS' logic from briefly examining the data file and then having logic, within the one JPS, to process any of the expected possible job types. Some people prefer to factor logic out into individual files, some prefer to keep it all together, to have fewer files to maintain. The choice is yours.

See Also

[Job Name Discovery](#)

[Command Line Processing](#)

[Command Options - FolderMonitor](#)

[_job Script Object](#)

-logfileFormat

(Windows DocOriginFolderMonitorSVC.prm only)

Specify the file name template for a Folder Monitor instance's log file.

Syntax

`-logfileFormat filename`

Description

Specify the name template of the log file where FolderMonitor will write job-related information, such as start and end times, and error messages. This is related to FolderMonitor only and should not be used for Merge or any other application. This file name template is used in conjunction with the FolderMonitor instance name to arrive at the actual log file name to be used by that instance of FolderMonitor. This parameter is expected to be used in your `DocOriginFolderMonitorSVC.prm` parameter file (Windows-only). As the Windows-only FolderMonitorConsole application is used to start various Folder Monitor instances, this logfile name template plus the FolderMonitor instance name is used to provide the launched instance with its proper `-logfile` command line option.

In the Folder Monitor context the special placeholder, `%i`, can be used to indicate where in the log file format the instance name is to be placed. See the [File Naming Conventions](#) section for additional restrictions and options. This includes the ability to inject various date-time-related placeholders in the log file name, as well as referencing file values.

If the specified log file already exists, DocOrigin programs append to the end of the file, otherwise, the file will be created.

See Also

[File Naming Conventions](#)

-pause

Amount of time FolderMonitor will pause when it finds there is no data file to process.

Syntax

-pause *seconds*

Description

This is the period of time, in seconds, that FolderMonitor should pause when it finds that there are no data files to process. The default setting is 0.5 seconds. This pause time ensures that FolderMonitor will not use an inordinate percentage of the CPU time in futile times to keep checking for jobs to do.

See Also

[Command Line Processing](#)

[Command Options - FolderMonitor](#)

-processedFolder

Optional FolderMonitor folder into which successfully processed queued data files are stored.

Syntax

-processedFolder *foldername*

Description

The *processedFolder* is an optional folder into which queued data files that are successfully processed will be copied. If this option is not specified, no copy of these files is retained. In that case, the data files are just deleted after they have been successfully processed.

See Also

[Command Line Processing](#)
[FolderMonitor Command Options](#)

-quarantineTime

How long to wait for a data file to be closed before assuming it never will be.

Syntax

-quarantineTime *minutes*

Description

The quarantine time setting tells FolderMonitor how long to wait for a data file to be properly closed by whatever application is creating it. After this number of minutes has elapsed, and the file can still not be opened, FolderMonitor will trigger an Error to the logfile. If you have configured email Alerts on Error conditions, an email will be sent to notify someone of the condition. See the [-alert](#) section for more details.

See Also

[Command Line Processing](#)
[Command Options - FolderMonitor](#)
[-useDeferred](#)

-queue

Specify one or more file folders to serve as input queues to FolderMonitor.

Syntax

-queue *foldername/file mask*

Description

The -queue options list one or more file folders and file masks that will serve as Queues for FolderMonitor. At least one (-queue1) must be specified. The queue specification must include a file mask; e.g. *.xml , or *.*

These queue parameters are named -queue1, -queue2, ... -queue10 .

Your application must write files into one of these Queues. FolderMonitor will retrieve and process them.

See Also

[Command Line Processing](#)

[Command Options - FolderMonitor](#)

-script

(Mandatory)

A JavaScript file to be used for Job Name Discovery (JND).

Syntax

```
-script filename
```

Description

The `-script` option specifies the JavaScript file to be executed to *discover* the job name. Commonly called the "JND" script. This job name is subsequently used to identify a second script that is used to process the data file.

The JND script is intended to examine the supplied data file (identified in `_job.datafile`) and compute which job name applies. E.g, is it an "Invoice" job or a "Statement" job, etc.? Typically this is done by looking at the first few lines of the data file for some identifying string. Often, for XML files, it is the root tag name. For field-nominated format files, it is often the job name in the `^job` card, but you can use whatever technique you like to identify which job name to apply to the given data file.

The JND script's task is to return a string that identifies the job name. E.g.

```
return "Invoice";
```

If the script cannot identify an applicable job name it should: `return ""`; That will cause the job to fail with error code 84 (and the data file to be saved in the designated ErrorFolder).

If the script does return a name then the Folder Monitor (FM) processing will look for a job processing script (JPS) in the designated `scriptFolder`. The JPS file name is the provided job name concatenated with ".wjs". For example, if the JND returned "Invoice" then the FM processing would look for a file named "Invoice.wjs" in the folder identified by the `-scriptFolder` option. If that file is found, that script will be launched and its eventual return code will determine whether FM considers the job to have succeeded or failed. The JPS can perform many, many steps. It's completely open-ended. Typically it does include a DocOrigin Merge step.

If the job processing script (JPS) file is not found, FM will revert to using `$/Default-JobProcessing.wjs`. Note that that default script has a `##include $0/JobProcessing.wjs` and so if you have created such an override file, that override script will be used.

Besides identifying the job name to apply, the JND can also set arbitrarily named properties in the `_job.options` object. The `_job.options` object is available to the JPS when it is run. As it happens, the JND can also set user-chosen global JavaScript variables. Those too are available to the JPS. Note that neither the global variables nor the `_job.options` object are available to the scripts defined in a form design. Merge is a different process than FM and so does not have access to that information. Typical information that you might store in the `_job.options` object are items gleaned off of the `^job` card when using a field-nominated format data file.

If the data file is XML and has a DocOrigin processing instruction that includes a `job=xxx` attribute, then this `-script` option will be entirely ignored. No JND script will be run as the job attribute tells FM which JPS to run.

See Also

[Command Line Processing](#)
[Command Options - FolderMonitor](#)
[Job Name Discovery](#)
[_job Script Object](#)

-scriptFolder (FM)

(Mandatory)

A FolderMonitor folder containing the scripts for processing jobs.

Syntax

```
-scriptFolder foldername
```

Description

This folder contains the scripts used for processing each job. FolderMonitor looks for a script called *jobname.wjs* in this folder (where *jobname* is whatever name is returned by the *discovery* script from the `-script` command option).

The default job processing script (as opposed to job name discovery script) assumes that *jobname.xatw* is the name of the form design file to be used and executes DocOrigin Merge to merge the data file with the form file. Frequently you will want to override the `$/Default-JobProcessing.wjs` file with an override file in `$/JobProcessing.wjs`.

See Also

[Command Line Processing](#)

[Command Options - FolderMonitor](#)

[-script](#)

-sortingOrder

(As of 3.3.001.01)

Specify files sorting order for FolderMonitor scanned folders (input queues).

Syntax

-sortingOrder *order*

Description

Option *order* is one of:

- none - unsorted;
- name - lexicographically by name;
- name_reversed - ... reversed;
- natural - on Windows natural sorting by name (like File Explorer), on Unix same as "name";
- natural_reversed - ... reversed;
- size - by file size;
- size_reversed - ... reversed;
- creation - on Windows by creation date, on Unix by file properties change;
- creation_reversed - ... reversed;
- access - by access time;
- access_reversed - ... reversed;
- written - by modification time (Default);
- written_reversed - ... reversed.

To get "predictable" order, sorting orders other than "none", "name"/"name_reversed" and "natural"/"natural_reversed" apply "name" as a secondary sorting order in case of collision of the primary criteria.

-UpdateCounts

(As of 3.1.002.10)

An option to control sending of jobs counts updates to Folder Monitor Console.

Syntax

-UpdateCounts <*true*|*false*>

Description

When enabled, FolderMonitor maintains the number of processed files and files waiting in the queues and sends this information to Folder Monitor Console. Enabling job count updates can affect folder monitor performance. The default value is *false*.

-useDeferred

(As of 3.0.004.11)

Select between "deferred mode" and "quarantine mode".

Syntax

-useDeferred **Y/N**

Description

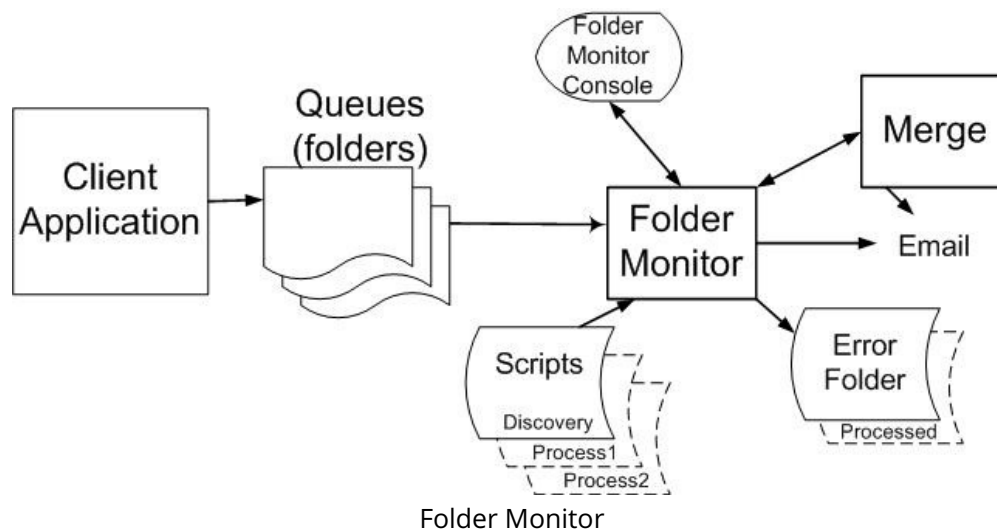
A new method of handling data file access where the inability to access a file is detected earlier, while still in FolderMonitor, rather than leaving it up to FMTransaction. In "deferred mode", a file that can't be accessed exclusively (likely it is still being written by the data-providing app), is put on a deferred list. All files in that list are retried before any new job is considered and so are likely to be processed in a much more timely fashion compared to the old "quarantine mode" of trying quarantined files only when no other jobs, in any queue, were available to be processed. Default value is **Y**, use **N** to switch back to "quarantine mode".

See Also

[-quarantineTime](#)

Functional Overview

FolderMonitor is configured using a series of command-line parameters. These include the standard command line parameters (see [DocOrigin Command Line Processing](#)) and command-line conventions. The specific FolderMonitor command-line options are listed in the Command Line Options section.



See also

[FolderMonitor Architecture](#)

FolderMonitor is fed data files from Client Application(s). These files must be written into Queues (folders) identified in FolderMonitor's configuration. Each of those files is considered a 'job', in DocOrigin parlance. When the file appears in the Queue (folder) FolderMonitor will read the file and process it using a two-step process:

1. A job "discovery" script is run. This script assigns a *job name* to the file. For example, by looking at the data stream the script may determine that this is an Invoice job, or a "Purchase Order" job, or a
2. A "processing" script is run. The script file name is the job name returned by the first step. (In other words, if the job name is determined/discovered to be abc , FolderMonitor processes the job with a script called abc .wjs .)

This scheme allows a fairly simple script to determine what kind of file it is (and hence what form or processing to apply). This then leads to running job-specific scripts against each job type. These scripts are typically tiny, cut-and-paste items, but, if the need ever arises, the power is there to orchestrate all manner of processing flows.

Through these scripts, FolderMonitor can access DocOrigin Merge, send emails, convert the data, etc. If errors are encountered the data file will be retained in an Error Folder for later examination and remedial action.

On Windows, the **FolderMonitorConsole** program can be used to start and stop the FolderMonitor service.

Multiple Monitors

As of 2.0.157.37, FolderMonitor has had the ability to run multiple instances which thereby allows one to achieve greater throughput, as jobs may be run simultaneously and make use of however many CPUs your server is configured with.

Each monitor instance watches its own set of folders for incoming jobs. As usual, the first folder, in the list of folders you configure a monitor to watch, gets the highest priority. All jobs in that folder must be completed before the monitor will look at the next folder in the list of folders that you configured it to watch. Note that if a job shows up in any higher priority folder than the one currently being processed then as soon as the current job is done, processing will revert back to doing the jobs now found in higher priority folders.


With multiple monitors, each monitor instance runs at the same priority, competing for system resources. Each monitor instance must be configured to watch its own set of folders, not overlapping with any other monitor's set of folders.

How does one get multiple monitors? Well ...

On Unix...

It's easy, just start up more FolderMonitor programs to run independently of each other in the background. Of course, each one you start must be given a parameter file (.prm) that is specific to the monitor's business intent. It would define which folders were of interest to that monitor, what job name discovery script to use, error file folder, processed file folder, ... - i.e. the usual.

On Windows...

 As of 3.1.002.03 multiple monitors can be configured with a single file: \$0/DocOriginFolderMonitor.ini See FM.ini below. Of course, you can continue to use multiple .prm files as described immediately below, if you like.

One provides a DocOriginFolderMonitorSvc.prm file (in the usual .../User/Overrides folder) which simply lists the names you wish to give to the monitor instances that you want to have. E.g.

```
-monitor Payroll:Manual
-monitor AdHoc:Automatic
-monitor LongJobs:Manual
```

Pick any names you like (even with spaces, but do you really want them?). Use as many as you like.

The Automatic or Manual attribute determines whether or not these monitor instances will be started up automatically when the DocOriginFolderMonitorSVC service itself is started.

The DocOriginFolderMonitorConsole can be used to start/stop any monitor (as chosen from a dropdown list). Or to stop them all with one convenient GUI selection.

When directed to start a FolderMonitor instance, the service will launch the FolderMonitor app passing it a parameter file name such as DocOriginFolderMonitorAdHoc.prm -- for the AdHoc named monitor that you listed. It is your job to provide those .prm files, configured in the usual way for a FolderMonitor, i.e. defining which folders it is to watch, the job name discovery script to use, etc.

That's it. As easy as that. Pick some names for the monitor instances you want and provide their configuration details. Individual monitor start/stop control is easy with the DocOriginFolderMonitorConsole.

FM.ini

(As of 3.1.002.03)

Instead of maintaining several PRM files for many instances of FM, you can instead configure all of the options for all of your monitors in a single ".ini" file. That file must be deployed at `$0/DocOriginFolderMonitor.ini`. The existence of that file will mean that none of your FolderMonitor-related PRM files, if they exist, will have any effect. FM INI files use the `_profile` ([Access Profile Files](#)) functionality.

The INI file is required to have sections. The section names would be the names of the monitor instances that you would like to use. E.g. `[AdHoc]`, `[HiPri]`, `[Payroll]`. Those names could have blanks in them but that is probably a bad idea, especially if you intend to use a third-party INI editor to maintain your INI file. Within each of those sections, you specify the options that you want to apply to that FolderMonitor instance. E.g. `scriptFolder=$0`. Notice the lack of a leading dash and the need to use an = sign. Otherwise, it is the same set of option specifications that you would have made in a PRM file.

You can start out your INI file with a section named `[COMMON]`. In that section, you can specify default options that you wish to apply to all the FolderMonitor instances. You can, if you like, repeat an option under an individual FolderMonitor instance section, so as to override the option setting you placed under `[COMMON]`. The INI file also supports a `[DEFINE]` section, see `_profile` ([Access Profile Files](#)).

In options `queuen`, `ErrorFolder`, and `ProcessedFolder` you can use the `%i` placeholder, which gets substituted with the applicable FolderMonitor instance name. For example, entries such as:

```
queue1=$U/FolderMonitor/%i/Jobs/*. *
ProcessedFolder=$U/FolderMonitor/%i/ProcessedJobs
ErrorFolder=$U/FolderMonitor/%i/FailedJobs
```

can be used in the `[COMMON]` section if you have standardized on your folder naming conventions across your various FolderMonitor instances.

You may include other custom sections with your own option settings. These will be ignored by built-in FolderMonitor processing but may be advantageous to you if, as part of your processing, you used the various `_profile` functions to load the INI file and grab your user options from it.

How does FolderMonitor know which sections are FolderMonitor instance sections versus custom user sections? Every FolderMonitor instance section, to be recognized as such, must have a `MonitorStart=Automatic/Manual` entry. Yes, that's right, the INI file replaces the `DocOriginFolderMonitorSvc.prm` as well. The service will detect the INI file and determine which instances exist and whether to start them automatically or not.

Even FolderMonitor instance sections can have unknown-to-FolderMonitor user options in them. They will be ignored. Of course, in doing that you run a risk that DocOrigin might invent an option name that matches one that you have used. Beware.

Both a leading `;` or `*` comment out a line. A third party 'INI Editor' may have stricter rules.

FMTransaction

 This is purely for technical interest. It is not need-to-know information.

A `DocOriginFolderMonitor` instance doesn't actually process a job itself. It invokes an `FMTransaction` helper app to process **one** job, then exit. It is that helper app that actually invokes the nominated job name discovery script and deduced job processing script, which in turn *probably* invokes `DocOrigin Merge` and may invoke filters as well.

The reason for using a spawned helper app is that it keeps the long-running `DocOriginFolderMonitor` instance immune from the effects of [user] code that leaves file handles open or that leaks memory. Such things are cleaned up by the OS when the helper app exits after doing one job.

Installation

On Unix, FolderMonitor is installed automatically. On Windows, DocOrigin FolderMonitor is installed as a separate step from the main DocOrigin installation. It is installed using the Microsoft installer by clicking on the `.../D0/Bin/FolderMonitorSetup.exe` file provided as part of the install set.

Testing a new install

When FolderMonitor is installed it adds a default set of test forms (.xatw) and scripts in order to assist in understanding how the system works. This setup also allows you to confirm that all components are correctly installed.

The installation of FolderMonitor adds a Windows service called: **DocOriginFolderMonitor**.

IF YOU HAVE MADE NO (REPEAT NO) CHANGES IN THE FOLDERMONITOR CONFIGURATION FolderMonitor and the Windows service can be tested by doing the following:

- Start FolderMonitor Console from the desktop icon or the Start menu.
- Once the FolderMonitor Console window is displayed, simply click on the **Start FolderMonitor** button. The Windows service will be started and FolderMonitor will begin its scan of its queue folder(s). Several lines of text will appear in the Log Messages window as the service starts the FolderMonitor instance(s) and they do their initial processing.
- Click on the **Copy Test Files to Scan Folders** button. This will copy two XML data files into the first queue folder. *This button has been dropped as of 3.0.004.05 since it was nearly certain to fail once you made adjustments to your configuration. If you really have made no changes to the initial FolderMonitor installation and wish to do a "proof of life" test, you can run the FolderMonitor-CopyDataFiles.bat command file that is supplied under .../D0/Samples. It simply copies two shipped sample XML files to the standard jobs folder. Of course, you could copy in any test files you like, but they should correlate to the Job Name Discovery and Job Processing scripts that you have defined.*
- You should now see several additional lines appear in the **Log file messages box** showing that Merge has been run and has processed those two data files.
- Check the `.../User/Output/PDF` folder and you should see two new PDF files that were created by Merge using the two data files copied to the queue folder.
- If you now check the `.../User/FolderMonitor/ProcessedFiles` folder you should see the two data files there, indicating that they have been processed by FolderMonitor.

The site-specific job name discovery script is to be placed, by you, in `.../User/Overrides/JobNameDiscovery.wjs`. That file is referenced in the main discovery script file `.../D0/Bin/Default-JobNameDiscovery.wjs`. You never edit files under the `.../D0` folder. Note that you can specify a different script to be used as your Job Name Discovery script. See [Job Name Discovery](#).

Similarly, you should copy the `.../D0/Bin/Default-JobProcessing.wjs` script to your `.../User/Overrides/Default-JobProcessing.wjs` file, and edit it there according to any changes you wish to have in the default processing of a job. Again, options are available to use alternate Job Processing scripts. See [Processing the Job](#).

Queues

FolderMonitor operates by monitoring one or more file folders - also known as Queues in this system. Queues are examined in sequence - all jobs (data files) in Queue1 are processed first, then all in Queue2, etc. After every job is processed the system returns to scan Queue1, then Queue2 etc. This allows you to configure the system to have a High-Priority Queue (Queue1) and a Normal-Priority Queue (Queue2). Jobs placed in Queue2 would be processed sequentially unless a job appeared in the higher priority Queue1.

For simple installations, only one Queue need be configured. You can configure up to 10 separate Queues if you wish.

Jobs within a Queue are always processed in the order they were added to the Queue (i.e. date & time order).

In some situations, files may be placed in a Queue that cannot be processed or opened by FolderMonitor. In this case there is a Quarantine procedure that attempts to run the job at a later time. See the Error Handling section below for details.

job Script Object

To facilitate the passing of various job settings to the *Job Name Discovery* step and the *Job Processing* step, DocOriginFolderMonitor creates and maintains a JavaScript object called `_job`. You can access various job settings using this `_job` object:

- `_job.name` - is the current job name as determined by the *Job discovery* script. (readonly)
- `_job.datafile` - is the full filename of the data file being processed. This can be used if you need to open or read the file. (readonly)
- `_job.command.xx` - provides access to the complete list of all DocOriginFolderMonitor command line settings. (readonly)
- `_job.options.xx` - provides access to the complete list of all the attributes that were provided on the `<? DocOrigin ... ?>` processing instruction, if it was provided.

Command Line Option Access

For example, `_job.command.logfile` will have the value of the current logfile name. `job.command.scriptfolder` will have the value of the `-scriptFolder` setting. etc.

As the `-parm` command line option can be used multiple times, the syntax for picking up the value of a specific `-parm` option is as per the following example:

- ✓ In the command line: `-parm=foo:bar`
In the script: `_job.command.Parmfoo` (the P in Parm must be upper case).
It is highly recommended to use -cache rather than -parm.

Processing Instruction Access

If your data file contains a DocOrigin processing instruction such as:

```
<?DocOrigin job="jobName" target="Printer23"?>
```

Then you could reference the `target` attribute or whatever attribute name was used in the processing instruction via `_job.options.target`.

FYI: Though `_job.options` properties are transferred from a JND script or from the PI to your JPS, they are NOT transferred into Merge when your JPS executes an `_merge()` call. In the above example, your form script could not successfully reference `_job.options.target` ... or any other `_job.options` property.

Job Name Discovery

Once a file has been found in one of the FolderMonitor folders (Queues), FolderMonitor first must determine what processing must be done on this particular file.

There are a few ways that FolderMonitor can do that. In priority order

1. Detect a DocOrigin PI containing only Merge-acceptable attributes (as of 3.1.002.03)
2. Detect a DocOrigin PI with a job="JobName" attribute
3. Skip job name discovery completely and use the Job Processing script identified by a -JPSName option (as of 3.1.002.03)
4. Use the Job Name Discovery script identified by the -script option
5. Default to using the \$E/Default-JobNameDiscovery.wjs script (which may bring to bear an override \$O/JobNameDiscovery.wjs)

A common option is to run the JavaScript script defined in the -script command option to "discover" the type of processing that should be done on the file. Like all script files, any job name discovery script (JND) must be saved in UTF-8. This script must return a "job name" - a unique text name that will be used to identify the job processing script to run on the file.

 Note:
JND === Job Name Discovery
JPS === Job Processing Script

Sample discovery script:

```
var sData = _file.readfile(_job.datafile);
i = sData.indexOf("SalesTable");
if (i > -1) return "Table Sample - Nested Table";
i = sData.indexOf("CustomerInfo");
if (i > -1) return "Sample Invoice";
return ""; // No job name found. Send to Error Folder
```

The above script reads the contents of the data file, then searches for the keyword "SalesTable" in the file. If this string appears in the file, the job name returned is "Table Sample - Nested Table". Similarly if the keyword "CustomerInfo" is found, the job name becomes "Sample Invoice".

If no job name is found in the file (see The DocOrigin PI below) or by the job name discovery script, an error is triggered and the data file is moved to the Error Folder (see [Error Handling](#)).

If a job name is returned, but no such Job Processing script exists, then the default Job Processing script (\$E/Default-JobProcessing.wjs) will be run. If you wish to explicitly have the default Job Processing script applied, have your JND return "_default_". As of 3.1.001.22 if the file **jobname.wjs** is not found in your nominated script folder it will try the file **jobname.jps**. That is, the .jps extension can be used for Job Processing scripts.

The DocOrigin PI

One good practice, if you have influence in data stream creation, is to include a DocOrigin processing instruction (PI) in the XML data. E.g.

```
<?DocOrigin job="aName" for="aGuy" printerTarget="aPrinter" outputFormat="aFormat"?>
```

If such a PI exists, then the JobNameDiscovery script won't even be invoked. The job attribute in the PI will define the job name, i.e. will define which job processing script to run.

The other attributes, if any, on the PI will be conveniently placed in `_job.options.name` properties that then become available to the job processing script. The attributes that you choose to put on the PI (aside from job) are totally up to you.

Given the above example PI, the job processing script (aName.wjs) could refer to `_job.options.printerTarget`, using its value in any way the script desires to affect the processing of the job.

Quite convenient really.

Caveat: While the `_job.options` properties flow through from the JND (or DocOrigin PI) to the JPS, they do not flow through to Merge (i.e. to script in your form design). Your JPS could transfer such information by creating suitable `-cache` command line options for Merge to see.

Direct-to-Merge PI

As of *3.1.002.03*, if the DocOrigin PI does not have a `-job` attribute and in fact all the attributes in the PI are valid Merge command line options, then both Job Name Discovery and Job Processing scripts will not be considered. Instead Merge, (and only Merge, not multiple steps), will be run directly using the options specified as attributes in the DocOrigin PI. Surely you would specify the `-form` option at a minimum. This is very direct with no chance for a script to peek at the job context. Perhaps it will suit your needs.

See Also

[XmlFile Class \(Write XML Files\)](#) -- the `xfile.pi()` function.

Processing the Job

Once the job name has been established, a job processing script (JPS) is called to process the job. A file called *jobName.wjs* is expected to be available in the "scriptFolder" (the `-scriptFolder` command line option). Like all script files, any job processing script must be saved in UTF-8. This script is run to process the data file. If the script is not found, an attempt is made to run a script called `Default-JobProcessing.wjs` as found in the `DO/Bin` folder of the installation location. If this does not exist an error is triggered.

The processing script is JavaScript that can use all the standard DocOrigin extensions. See [Scripting](#) for details. In addition, the script has available to it the `_job (Current Job and Command Line Parameters)` object with the job name and data file name set. Your script can use these to read the file.

Also, the JND script (or a DocOrigin PI) can set properties in `_job.options`. These properties are available to your JPS to use as it sees fit. Note though that these `_job.options` properties are NOT passed through to Merge. Your form's script does not have access to `_job.options` properties. If you wish to transfer info to Merge you should use the `-cache` command line option.

Sample Processing Script:

```

Default-JobProcessing.wjs

var args =
{}; // place to house Merge args
args.form = "$R/DO/Samples/" + _job.name + ".xatw"; // our default design file name
args.data = _job.datafile; // the data file we were given
args.output = "$U/Output/" + _job.name + ".pdf"; // chosen default output location

##include "$0/JobProcessing.wjs"

var rc =
_merge(args); // run Merge

return rc;

```

The above example uses, by site convention, whatever job name is passed to it, as the name of the form file (.xatw). The form file is expected to be in the samples folder such that the delivered samples work out-of-the-box, but surely you will override that.

By defining your own override `JobProcessing.wjs` file you can override the parameters that are setup by default to be passed to `_merge()`. In fact, you could ignore these parameters altogether, not even call `_merge()`, and return before the default call to `_merge()` is executed. You have complete control; this is just a stub.

It runs DocOrigin Merge to create a PDF file of the same name. Merge's return code is returned to FolderMonitor. If a non-zero return code came from Merge, FolderMonitor will log this as an error in the **ErrorFolder**.

Note that all of this is for default processing. If you have defined a `jobName.wjs` file, then these default processing steps won't be used at all. Rather it will execute the `jobName.wjs` file that you defined.

Once processing has completed, the file is removed from the active Queue. If the processing resulted in an error return from the processing script, (a JavaScript `return(n)` where *n* is non-zero), or an error occurs in the opening or compiling of the script, the job data file is moved to the Error folder specified by the command line `-errorFolder` setting. Along with the original data file, there will be a `.log` file created with the same base filename. It contains the last few lines of FolderMonitor's logfile. This file can be examined to quickly determine the source of the failure.

We recommend that a client's JavaScript program uses a positive integer between 200 and 250. That way you will always be able to tell whether the error was detected by your own JavaScript or by a DocOrigin program.

If the processing runs successfully, the script should return a value of 0 (zero). The job is then deleted from the Queue. If the optional `-processedFolder` option has been specified on the command line, or more likely, in the `DocOriginFolderMonitor.prm` file, a copy of the processed data file will be written to that folder.

Error Handling

Quarantined Files

When writing a file into one of FolderMonitor's Queues, it's possible that some error occurs which leaves the file incorrectly closed. Or perhaps FolderMonitor attempts to open the file while it is still being written into the Queue. In these situations, FolderMonitor makes multiple attempts to open the file. If after numerous attempts to open the file over about a 20-second period of time, the file is still not opening, the file gets put on a temporary Quarantine list. After any other queued jobs are processed, FolderMonitor comes back to the Quarantine list and re-attempts to open the file. As long as it cannot be opened it remains on the "Quarantine" list and is periodically checked again - up to a limit set by the `QuarantineTime` command option (which defaults to 30 minutes).

If the file still cannot be opened after this `QuarantineTime` period has elapsed, an error message is logged to the FolderMonitor logfile. Typically one would configure FolderMonitor to trigger an email alert to some email address whenever errors are logged (see [-alert](#)). This allows someone to be alerted that a problem exists. FolderMonitor will continue to attempt to open and run the file and re-issue the error every `QuarantineTime` minutes.

To remedy the situation it is necessary to either cause the file to be completed and closed properly, or to delete the file manually from the Queue.

Whenever FolderMonitor is shut down, a list of all Quarantined files is listed in the logfile.

To summarize - if a file cannot be read, FolderMonitor will repeatedly attempt to open the file. At regular intervals, (after possible email alerts) errors will be logged. Other files in the Queues will also continue to be processed normally.

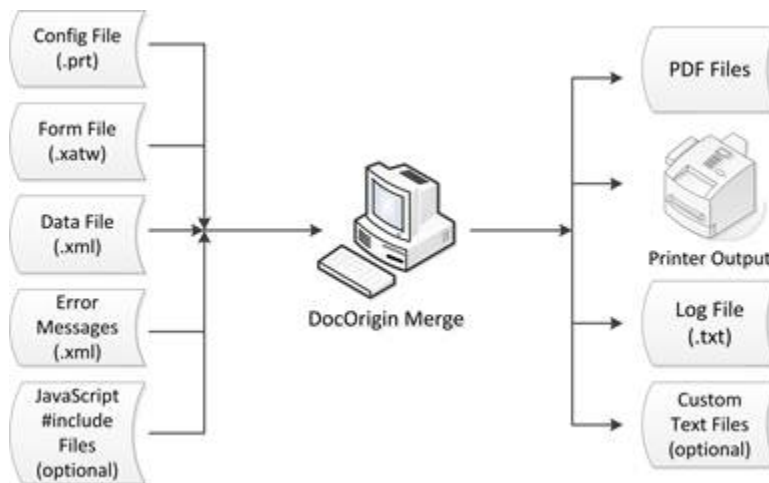
Merge

At the heart of DocOrigin is the DocOrigin Merge application. It combines user forms and data to create and output multi-page documents to PCL or PostScript® printers, a Windows printer, or PDF. Merge is a Windows console application; it is a non-interactive process that can be called from a user application. Merge can also run on some Unix platforms.

Merge provides a wide range of features:

- Support for all TrueType fonts
- Barcode support for UPC, EAN13, PDF417, 2 of 5, 3 of 9, QR, and DataMatrix
- Full word wrap and data justification
- Dynamic resizing of fields and tables
- Output in color or monochrome
- Scripting in JavaScript.
- Embedding of field values in text labels (Dear [FirstName]), and in dynamic output file names.
- RTF (Rich Text Format) support for bolding, italicization, underlining, typeface selection, point size selection, color selection, hyperlinks, plus embedding of field values. The RTF may be pulled in from an external file.
- Pagination with headers and footers plus "widows and orphans" control, also, as of 3.0.003.20, multi-column output with column headers and footers, and dynamic balancing of column length.
- Embedding of fonts in PDF and PCL. Font embedding is not supported for PostScript, of course fonts can be downloaded to a PostScript printer outside of DocOrigin, and then be used as "resident" fonts.

When Merge is called, a number of options are passed to the application either on the command line or via a command file. These options direct the input, output and processing activities of Merge for the current job. From the command line options, Merge determines the configuration, the form and data files to use, and when appropriate, it may be directed to use a translated set of its error messages. Merge combines the data and form to create output documents, and records its activity for the job including any errors to a log file.



Merge

Command Options

Merge is invoked with a command line with a number of options that control its operation. As with other DocOrigin commands, it also uses a common set of command line features described in the [Common Command Line Options](#) section.

See Also

[Common Command Line Options](#)
[Command Line Processing](#)
[Indirect Parameter Files](#)

`-{prt}option`

Sets printer/driver-specific parameters.

Syntax

`-{prt}option value`

Description

`prt` specifies a printer name in the printer configuration file (`.prt`). For example, in the `Default-LJ4.prt` configuration file the LaserJet 4 is specified as `<Printer Name="LJ4" Type="PCL" >`. The `prt` value for this configuration would be `LJ4`. Use this to override settings in the printer configuration file. For example:

<code>-LJ4CombineDocuments yes</code>	causes all LJ4 output to print to a single spool file
<code>-PDFOutputfile myfile.pdf</code>	prints the PDF output to myfile.pdf

- i** Note that for the `-outputfile` setting, Merge will do standard date/time substitutions as described in [File Naming Conventions](#). In addition, if the file name contains a field name between square brackets as in `[Field1]`, Merge does field substitutions in the same way it does for embedded fields within a text label. In the example below, Merge looks for a field with the name "myFileName" in the data. If it finds a field by that name it substitutes in its value. If no field is found, the square brackets and everything between them are removed from the resulting file name.

```
-LJ4Outputfile $U/Output/[myFileName].pcl
```

Note that `$U` is automatically defined as the User folder of the installation location.

Example Printer Options

Most settings in the `.prt` (configuration files) can be set on the command line. The command line setting will override the default settings in the configuration file. Here are a few that may be more common:

`-PDFBookmark Yes|No`

Turn on/off bookmark generation in pdf files.

`-PCLColor Yes|No`

Control whether a PCL printer prints in color or black-and-white.

`-prtCombineDocuments Yes|No`

Control if all documents in the job are in one PDF file / printer stream or not.

`-prtDuplex form|left|top|right|simplex`

(As of 3.0.003.13) specify the duplexing option to use for the `prt` output configuration.

`-PDFEmbedData Yes|No`

(As of 3.0.003.10) if the `-PDFEmbedData` specification is given for PDF-based output, the data for the document will be embedded in the PDF. Technically, it adds:

```
/PieceInfo <</DocOrigin <</LastModified (xxxx) /Private nn 0 R>> >>
```

to the document catalog and creates object `nn` as an xml representation of the data. This applies to single document PDFs only, not combined documents. That is, PDFExtract cannot extract the data for a single document from a combined PDF. Instead, you always get the data for the full set of documents in the PDF file regardless of your possible document selection options. (As of 3.0.004.06) PDFExtract extracts only the data for those documents that were selected. (As of 3.0.003.13) use the script function `_file.readPdfData()` to extract that XML.

`-prtEmbedFonts` Yes|No

Embed fonts into PDF files or download fonts to printers.

`-PDFMode` PDF/A

Have PDF files be generated in PDF/A (archive) format.

`-PDFPageMode` UseNone

Add a PDF PageMode option to the generated file. Options include *UseOutlines*, *UseThumbs*, *FullScreen*. See your PDF reference manual for details. The *UseThumbs*, for instance, defaults to showing Thumbnail images of each page in the document in the PDF viewer.

`-prtOutputfile` *filename*

Specify individual filenames for each driver when using multiple print drivers. For EML, if no value is set then **%T/%u.html** is used.

`-prtOverwritefile` Yes|No

If the output file already exists should it be overwritten? If no, a suffix is added to make it unique.

`-prtSpoolerDocName` "*any name*"

Specify a custom document name for the spooler to show. See `-spoolerDocName`.

`-PDFTitle` "*any title*"

(As of 3.1.002.07) set the PDF property Title explicitly. The Title in the Acrobat **File > Properties** list is normally set to the PDF file name. This can be overridden by using the TagTitle object tag on the Form object, or by using the `-Title` or `-PDFTitle` option on the command line. Some versions of Acrobat will use this title as the text in the Acrobat tab when the pdf is displayed.

See Also

[File Naming Conventions](#)

[_printer](#)

[_file](#)

[Multiple Outputs](#)

-appendPages

Change where pages are added when a container overflows.

Syntax

-appendPages

Description

When the process of merging the data with the form design is finished, many containers can be overflowing with instantiated panes. The pagination algorithm has the task of splitting such overflowed containers into pages. Normally those pages are inserted in the output right after the overflowing page. When the -appendPages option is specified, the overflow pages will be added to the end of all existing pages.

By the time the pagination algorithm begins, all mandatory pages will have been instantiated, regardless of whether their containers are over-full or not. One instance per mandatory page. Now each of the pages (or layouts), per the order of definition in Design, will be examined, overflow conditions will be detected and additional pages will be added as necessary. This -appendPages option simply defines where those pages will be added.

Suppose you had three-page layouts brilliantly named as Layout1, Layout2 and Layout3. Suppose they were all mandatory, and also that there was enough data that flowed into Layout1" that three instances of it (or of it and its designated overflow layout) were required. The result...

Normal	-appendPages
Layout1-a	Layout1-a
Layout1-b	Layout2
Layout1-c	Layout3
Layout2	Layout1-b
Layout3	Layout1-c

See Also

[Pagination](#)

-attachment

(Merge only) (As of 3.1.001.25)

Attach a file to the generated PDF output.

Syntax

```
-attachment filename [; description]
```

The textual description is optional. If present, it will be displayed in pdf viewers such as Acrobat.

Description

For every document in the job PDF attachments list is reinitialized from -attachment options if any is set. Then this PDF attachments list is shared through all output configurations (drivers) defined.

Multiple -attachment commands may be used to attach several files. Example:

```
-attachment "C:/DocOrigin/User/Forms/myfile.txt;This is a sample description"
```

```
-attachment "$$F/anotherfile.txt;This is another file."
```

Scripting

(Merge only) (Added in 3.1.002.01)

There are such script functions to manage PDF attachments from script:

- **_printer.addPDFAttachment(*name* [, *description*])** - add an attachment to PDF attachments list for current document
- **_printer.clearPDFAttachments()** - clear the list of PDF attachments for current document
- **_printer.resetPDFAttachments()** - init PDF attachments list from command line -attachment options for current document

 Note that some of these functions make more sense on start next print driver event.

See Also

[_printer \(Output Configuration\)](#)

-blend

Set the blend mode for PDF.

Syntax

-blend *value*

Values

- **Normal** - Selects the source color, ignoring the backdrop.
- **Multiply** - Multiplies the backdrop and source color values.
- **Screen** - Multiplies the complements of the backdrop and source color values, then complements the result.
- **Overlay** - Multiplies or screens the colors, depending on the backdrop color value.
- **Darken** - Selects the darker of the backdrop and source colors.
- **Lighten** - Selects the lighter of the backdrop and source colors.
- **ColorDodge** - Brightens the backdrop color to reflect the source color.
- **ColorBurn** - Darkens the backdrop color to reflect the source color.
- **HardLight** - Multiplies or screens the colors, depending on the source color value.
- **SoftLight** - Darkens or lightens the colors, depending on the source color value.
- **Difference** - Subtracts the darker of the two constituent colors from the lighter color.
- **Exclusion** - Produces an effect similar to that of the Difference mode but lower in contrast.

The default value is **Multiply**.

Normal is recommended for transparent images.

Printer Configuration File

You can define the blend mode in the Printer Configuration File (PRT). For example:

```
<!-- <Blend>Normal</Blend> -->
```

See Also

[PDF Blend Modes: Addendum](#)

For sample collateral, see [Transparent Color Image Support](#) in the User Guide (Login Required).

-combineDocuments

Output multi-doc data as a single file or as individual files.

Syntax

-{prt}combineDocuments **Y|N**


Examples

-PDFCombineDocument=Y

-LJ4CombineDocuments=N

Description

Choose if data files containing multiple documents should be output as individual files (**N**) or a single, combined file (**Y**). This option is typically set in configuration (PRT) files.

 If your output filename structure is such that a unique name is supplied for each document, it will override the **Y** setting and produce individual documents.

See Also

[-config](#)

[-output](#)

[-{prt}option](#)

[Multiple Outputs](#)

[Merge Output](#)

-config

This Merge command line option identifies the output configuration file to use.

Syntax

-config *filename*

Description

Identify the output configuration file (.prt) used by Merge, which specifies information such as available fonts, page sizes, and printer driver(s). It is quite likely to have been defined for you in the Default-Merge.prm file. If not defined, the default is Default-PDF.prt.

See Also

[Command Line Processing](#)

-convertAttributes

Control the automatic transformation of attributes to leaf data elements.

Syntax

-convertAttributes **Y|N**

Description

As of 3.0.005.05, attributes on structure tags in XML data are automatically transformed into additional leaf tags for the given structural element (default is **Y**). Use **N** to prevent this automatic transformation. Note that putting attributes on leaf nodes will not work.

See Also

[XML Attributes](#)

-createData

Ask Merge to generate sample data for the nominated form, and then exit.

Syntax

`-createData` *outputFilename*

Description

Request that Merge generate sample data for a form design.

The output would be the same as one would get in Design with PDF Preview when using the Auto-generate test data option and hitting Save Data.

After producing the data, Merge exits. It does not immediately merge that data with the form to produce a document.

This data might be used as sample XML for a FilterEditor definition.

-data

(Mandatory)

Identify the data file to be merged with the form.

Syntax

-data *filename*

Description

Identify the data file to be merged with the form. This is a mandatory command line option.

You can also supply a file mask to process multiple data files. In this case, all data files must use the same Form and configuration file (PRT).

See Also

[Command Line Processing](#)
[-dataFileResults](#)

-dataFileResults

(Merge only) (As of 3.3.001.01)

Specify a path to the file to store individual data files processing results.

Syntax

`-dataFileResults` *filePath*

An optional Merge parameter that specifies the path to the file, where processing exit codes of individual data files are stored.

This may be useful when passing multiple data files for one Merge run using multiple `-data` command line options.

Output example:

```
data1.xml=0  
data2.xml=0  
data3.xml=53
```

This parameter is optional. The resulting file is UTF8 INI file which can be parsed using DO tools.

See Also

[-data](#)

-documentTag

Specify an XML tag used to delimit the data for a single document. The default value is "Document".

Syntax

```
-documentTag xmltag
```

Description

We expect that data files will often contain data for multiple documents. There must be an opening and closing pair of tags around each document in the data file. For example, for an *xmltag* of "Employee", the data file should look like:

```

<Data>
  <Employee>
    ...
  </Employee>

  <Employee>
    ...
  </Employee>
  ...
</Data>

```

Note that the root tag name Data is just an example. That tag may have any name. Often we use <XmlData>. If your data file looked like the above you would specify the Merge command line option:

```
-documentTag Employee
```

Our typical, and default, document separator tag is <Document>...</Document>.

The *xmltag* can also be a number, usually the value 1 or the value 2. The value 1 tells Merge to use whatever the root tag is as the document tag. Clearly, there could be only one document in such an XML file. A value of 2, tells Merge to use the second tag in the XML file, the one after the root tag, as the document separator tag. For example, the XML file above could be processed by using a command line option of:

```
-documentTag 2
```

As Employee is the second tag in, that is equivalent to -documentTag Employee.

See Also

[Command Line Processing](#)

-dumpLevel

Dump internal data values at various stages of the Merge process.

Syntax

`-dumpLevel` *short* | *full*

Description

Dump internal data values at various stages of the Merge process.

- Specifying *short* displays only a couple of files.
- Specifying *full* causes a dump of all intermediate files.

The files are written to the folder name specified by `-dumppath`.

See Also

[Command Line Processing](#)

-dumpPath

Identify to the dump routine the folder into which the debug or dump information is written.

Syntax

-dumpPath *foldername*

Description

Identify to the dump routine the folder into which the debug or dump information is written. *foldername* must be an existing folder name.

See Also

[Command Line Processing](#)

-dumpTimes

Write out the internal timing information captured during program execution.

Syntax

-dumpTimes

Description

Write a file called `times.txt` that captures the internal timing information at various stages of program execution.

The file is written to the *foldername* specified by `-dumppath`.

See Also

[Command Line Processing](#)

-duplex

(Merge only) (As of 3.0.003.13)

Specify a duplexing option

Syntax

-{*prt*}duplex *value*

Description

Possible values are:

- **form** - indicates to use the duplex settings that were set in the form. This is the default.
- **top** - indicates to print the form duplexed with top-binding no matter what is set in the form.
- **left** - indicates to print the form duplexed with left-binding no matter what is set in the form.
- **right** - indicates to print the form duplexed with right-binding no matter what is set in the form.
- **simplex** - indicates to print the form as simplex (one-sided) no matter what is set in the form.

In an output configuration file (.prt), the option is specified in the format:

```
<Duplex>form</Duplex>
```

When multiple configuration files are in use, or anytime, it is wise to prefix the option name with the printer type.
E.g. **-PDFduplex simplex**

On the command line, options are case-insensitive.

See Also

[Command Line Processing](#)

-ellipsis

(Merge only)

Specify the character to use for the ellipsis mark.

Syntax

```
-ellipsis character-in-hex|Y|N
```

Description

When a single line field overflows, Merge automatically supplies an ellipsis (...) character at the end of the amount of text that does fit into the space available. The character that is used by default is Unicode 0x2026. You can change that character by using this `-ellipsis` option. For example, if you wanted to see a plus sign (+), add the hex 2B.

```
-ellipsis 2B
```

(As of 3.2.001.12), the option for **Y/N** is available. **Y** uses the default symbol of an ellipsis mark. And the **N** simply truncates the content with no symbol.

```
-ellipsis N
```

(Prior to 3.2.001.12) If you wanted to not see the ellipsis, you could specify it as the blank character codepoint --hex 20.

```
-ellipsis 20
```

-ellipsisCropLastWord

(Merge only)

Specify if the last word should be cropped before text is truncated with an ellipsis.

Syntax

-ellipsisCropLastWord **Y|N**

Description

When a field overflows, Merge automatically supplies an ellipsis (...) character at the end of the amount of text that does fit into the space available. Choose **Y** to have as many characters as possible showing (potentially using a partial word) or choose **N** to always have a full word as the last word before an ellipsis.

See Also

[-ellipsis](#)

-embeddedDefault

Specify a replacement string for embedded fields that exist on the form but not found in the data.

Syntax

`-embeddedDefault` *replacementString*

Description

If the data field is not found but is used as an embedded field then it doesn't get replaced. With this option, you may specify the default replacement string. If *replacementString* is missing then an empty string is used for replacement.

First use-case: for example such label "EMAIL: \[EMAIL\]" remain unchanged if no EMAIL found in the data. If you want to get something like "EMAIL: " instead, you may supply Merge with "-embeddedDefault " option.

Second use-case: you may want to check if all required fields are present in the data, so you may specify something like "embeddedDefault """, run Merge and then search output for "" string.

-embedjpg

(Merge only) (PDF output only)

Embed the JPG graphic file as is.

Syntax

-embedjpg **Y|N**

Description

Control whether JPG files are embedded "as is" or pre-rendered to a raster applicable to the rendering area size. The default setting is **N**.

Typically, passing the merge option **-embedjpg Y** results in a much smaller PDF. The results depend on the image, its JPG encoding flavor, JPG compression, and resolution.

Users should strive to pre-convert their images to the size and resolution that is required and, more likely, choose the PNG format.

Rotated Image Limitation

Embedding doesn't happen if the image doesn't fit the area. For example, a rectangular image rotated to 0 or 180 degrees may be embedded, but for 90 or 270 degrees, embedding will be ignored. However, even for 180 degrees image is not going to be rotated.

-embedPlainTextAsRtf

(As of 3.2.001.01)

Syntax

-embedPlainTextAsRtf **Y|N**

Description

Controls whether <EOL> should be automatically converted to \line (new line) when embedding plain text into RTF. The default value is **N** (no conversion).

-emptyNodeInstantiatePane

(As of 3.2.001.06)

An option to not instantiate a pane if there are no child fields.

Syntax

-emptyNodeInstantiatePane **Y|N**

Description

Set this option to **N** to not instantiate a pane if the corresponding data node has no data and no children or no data and all children are empty. The default setting is **Y**, i.e. do instantiate.

-filter

Convert incoming data to Merge-compatible XML format.

Syntax

```
-filter "filterName filterOption1 Option1Value filterOption2 Option2Value ..."
```

or

```
-filter "filterName filterOption1=Option1Value filterOption2=Option2Value ..."
```

A filter can, itself, have many parameters. These parameters need to be shown as separate from the rest of the Merge parameters. Think of the entire filter name and its parameters as a single parameter to Merge – that is what it is. To accomplish that, the entire filter name and its parameters must be enclosed in quotes ("). Within that set of double quotes, there may be filter option values that have special characters. Such filter option values need to be surrounded by single quotes (').

As usual, one can choose to put equal signs (=) between name and value operands. Note that the equal sign itself is a special character.

Much of the quoting difficulty can be eliminated by using parameter (.prm) files which are not subject to the whims of various command line interpreters.

Description

Typically used to convert incoming data to Merge-compatible XML format. Filters are also used to sort incoming XML into a new order. Another filter could be used to combine several forms into one form for use in the Merge process.

filterName is the name of a program or JavaScript file that is invoked by Merge to convert the data.

- If *filterName* has a .exe extension or has no extension at all, an external (custom) converter is called.
- If *filterName* has an extension of xfilter then the built-in DocOriginFilterProcessor.wjsinc conversion facility is used to process the data stream according to the rules you defined via [Filter Editor](#) to create the xfilter file.
- Otherwise *filterName* is assumed to be a file containing JavaScript that will convert the data. (Typically these have a .wjs extension.)

When *filterName* is not a fully qualified file name, it is assumed to be relative to the folder containing Merge.

filterOption# is an option name that pertains to the *filterName* being used.

Option#Value is the value for the preceding *filterOption#*. The value will have to be surrounded in single quotes if it contains blanks or other special characters.

Examples

```
-filter "ConvertDatToXml -split '^field AccountNo'"
```

```
-filter "uconv -f cp1252 -t utf-8 -o %out %in"
```

```
-filter "DocumentSort -keyname Document.InvoiceHeader.CustomerID -keyname CustomerPO  
-sortOrder=D"
```

Multiple Filters

It is possible to specify multiple -filter options for a single Merge command line invocation. These filters will be run one after the other in the order in which they are specified on the command line, or more likely, in a .prm file.

Third-Party Filters

It is possible that an existing executable can be used as a filter. However, it is unlikely that it will expect parameters in a style such as `-in filename` and `-out fileName`. When defining the command line parameters for these third-party filters you can supply `%in` and `%out` as placeholders for the file names to be used for the filter. Merge will supply those names in a way that is applicable and suitable for chaining multiple filters in a row. See the `uconv` example above. If you supply neither `%in` nor `%out` then Merge will automatically affix `-in fileNameIn` and `-out fileNameOut` to the end of the provided command line.

Normally Merge expects executable filters to be in `$(E)` (i.e. `DO/Bin`). For third-party filters, you will need to specify the applicable path and file name.

See Also

[Merge Filters](#)

-filterOutput

Save a copy of the xml data after all filters have been run.

Syntax

`-filterOutput filename`

Description

Used when the `-filter` option has been specified. This command will save a copy of the output xml code created by the filter conversion to the *filename* specified. This can be useful for testing and debugging filter logic.

When multiple filters are in use, it is the output from the last one in the chain that will be left in the file specified here.

See Also

[Merge Filters](#)

-filterParm

⚠️ Deprecated - do not use

This was used to specify additional parameters to a single filter. Now that multiple filters are supported, the use of `-filterParm` is meaningless.

Syntax

`-filterParm` *parameters*

Description

This is used when the `-filter` setting has specified an external executable (non-script) filter program to be run. *parameters* is any additional command-line parameters you wish to pass to the filter program.

See Also

[-filter](#)
[Merge Filters](#)

-FnfDataExts

Define the default file extension(s) for FNF input data.

Syntax

```
-FnfDataExts ext1 [;ext2 ...]
```

Description

Default extensions for FNF are .jet, .dat, and .fnf. This Merge option would rarely (or never) be used in the command line. It is defined in `...D0/Bin/Default-Merge.prm` and can be modified by creating the override `.../User/Overrides/Merge.prm`.

See Also

[-JsonDataExts](#)
[-XfdfDataExts](#)

-form

(Mandatory)

Specify the form or forms to be used in the Merge.

Syntax

-form *filename*

Description

Specify the form design file (.xatw) to be merged with the data. This is a mandatory Merge command line option.

Although it is possible to dynamically set the form file when using the `-filter` option, the `-form` parameter must still be present (but may be overridden).

Multiple forms may be specified by separating them with semicolons. They will be automatically combined into a single temporary form into which the data is then merged. All pages of all forms specified will be appended in the order the files are listed on the `-form` list. When using multiple form files care should be taken to ensure that page names and pane names are unique across the entire set of forms.

If the second and subsequent files are specified without a file path, the path of the first file will be assumed. E.g.

```
-form C:/DocOrigin/User/Forms/one.xatw;two;three.xatw;C:/four.xatw;five
```

will combine the 5 forms.

```
C:/DocOrigin/User/Forms/one.xatw  
C:/DocOrigin/User/Forms/two.xatw  
C:/DocOrigin/User/Forms/three.xatw  
C:/four.xatw  
C:/five.xatw
```


-fragmentAutoResolve

(Merge only)

Resolve linked fragments.

Syntax

-fragmentAutoResolve **Y|N**

See Also

[Fragments](#)

-HighlightURL

This option provides control over the display of hyperlinks.

Syntax

-HighlightURL **Y|N**

Description

This option provides control over the display of hyperlinks that are created in a PDF when an email or URL format is detected. The default for this option is **Y**.

Also available as Printer Configuration Option: **N**

A URL such as a website www.EclipseCorp.US or email info@EclipseCorp.US with the default option of -HighlightURL=**Y** will show as follows:

Website: www.EclipseCorp.US

Email: info@EclipseCorp.US

and the -HighlightURL=**N** will show as follows:

Website: www.EclipseCorp.US

Email: info@EclipseCorp.US

Note: The hyperlink is still active in both settings.

Script Option

If you wish to suppress the hyperlink, try adding the following script to the field or text label that contains a URL link or email address, the string will be shown without the link formatting.

```
this._value = this._value.replace('.',String.fromCharCode(0x0008) + '.' );
```

Note: This option is not functional for all fonts.

-htmlTemplate

Specify an alternative HTMLTemplate.htm file

Syntax

```
-htmlTemplate yourHtmlTemplate.htm
```

Description

This lets you provide your own HTML Template file instead of using the supplied Default-DOhtmlTemplate.htm that is installed in `.../DO/Bin`.

Obviously, this applies only when you are generating HTML output.

The big bonus here is that you can include JavaScript or CSS or whatever you like as befits your organization's standards. You might include the JQuery tool set. Or perhaps some generic JavaScript functions for use in your browser JavaScript scripts.

No doubt you will copy the standard `.../DO/Bin/Default-DOhtmlTemplate.htm`, likely to your overrides folder, say as `.../User/Overrides/htmlTemplate.htm`. Edit it. Then in your Merge command line, or in your Merge.prm file you could code the option as follows:

```
-htmlTemplate $O/HTMLTemplate.htm
```

See Also

[Fillable Forms](#)

-imagePath

(As of 3.1.002.06)

Specify the locations where Merge should look for image files. It is also used for locating text resources.

Syntax

```
-imagePath foldername [;foldername2...]
```

Description

Specify the folders where Merge is to look for image files identified in the data stream or via script settings. If your Form design defines a Field with a **Display As** type of **Image**, then the data found for such a Field is expected to be a file name. That image file name may or may not be fully qualified.

foldername, *foldername2*, ... identify the locations where the image files may reside.

When Merge is looking for a referenced image file it tries to find it in multiple ways. The algorithm used depends on whether the reference is for a field or for a label. It also depends on the type of image file name specification.

There are 4 types of image file references:

1. Absolute L:\BigImageBin\Logo.jpg
2. Relative: ./subsidiary/A/B/C/Logo.jpg (with or without the ./)
3. Naked: Logo.jpg
4. http[s]://-..Logo.jpg (This quickly turns into a temp file name, and thus a case 1).

An example of a common -imagePath specification is:

```
-imagePath $$F;$U/Images
```

BTW: Using . in the -imagePath is not recommended. It is meaningless. You really don't know what the current directory will be at the time that image selection is called upon.

Similarly, using \$\$D is frowned upon since a filtering operation may be performed independently before Merge is run and thus the data file that Merge processes is actually in the "temp" folder. You can use \$\$D but you are at risk.

As of 3.1.002.06

As of 3.1.002.06, we have attempted to make image file selection more definitive or at least more definitively explained.

- ✔ Note that while the option name is -imagePath (for historical reasons), it is actually resource path. That is, references like [@[path/]message.rtf] follow the algorithm set out below.

Fields

If the image file specification contains an absolute path:

1. Try it as is, if found, use it
2. Strip all pathing off; get down to a leaf file name and extension.
We strip off all pathing since the data stream may have been for an old Central system or for dev/test vs prod. Setting the -imagePath properly will overcome these situations.
Append that leaf file name to each element of the -imagePath, if found, use it.
3. Not found – Issue an error message, identifying the field, and image. Note that only one error message will be issued per failing image file spec, regardless of how many times that same failing spec occurs in the run.


If the image file specification contains a relative path:

1. DO NOT try it as is. The user has no believable concept of what the current directory is.

2. Append that relative path to each element of the `-imagePath`, if found use it (e.g. we want it to find `$U/Images/. / subsidiary/A/B/C/Logo.jpg`)
3. Strip all pathing off; get down to a leaf file name and extension. Append that leaf file name to each element of the `-imagePath`, if found, use it
4. Not found – Issue an error message, identifying the field, and image.

If the image file specification is a naked name:

1. DO NOT try it as is. The user has no believable concept of what the current directory is.
2. Append that leaf file name to each element of the `-imagePath`, if found, use it
3. Not found – Issue an error message, identifying the field, and image.

 Note that we never look in the XATW to resolve a field's image name. If you really wanted to do that you could use a dynamic (scripted) label, setting its `_imageName` to the value of an [invisible] field.

Labels

Typically label images are defined at Design time and remain static. The actual image is stored in the XATW and an `_imageName` property relates all instances of the image label to its stored image data.

However, a user could use script `this._imageName = "[pathing/]xxxx.yy";`. In which case, Merge must effect a correlation between the dynamically assigned `_imageName` and actual image data.

1. If there is an exact (case sensitive) full name match to any of the names in the XATW, we use it - we're done. This is expected to take care of 99.992% of the cases.
2. If scripting has occurred such that the above fails to find an exact match, then:
 - a. Follow the same process as for fields for absolute paths and relative paths. `-imagePath` applies.
 - b. For naked names, look in the XATW for `imageData` names that end in that naked name. Partial paths are not considered. The matching is case-sensitive, even on Windows.
 - c. If not found yet, use the naked name as per the processing of a Field. `-imagePath` applies.
 - d. Not found -- Issue an error message identifying the field and `_imageName`.

Note to Linux users. The Design tool is Windows-only and hence the image file paths that are browsed to will have Windows syntax. If you wish to refer to them via scripting be sure to use that Windows syntax to effect a match. BTW, you are in the .008% (or less) club.


Prior to 3.1.002.06

If the object being processed is a field or `-preferImages` is set to `fromDisk` the search order becomes:

1. Look for a file on disk with the exact name specified in the data file
2. Look for the file in each folder specified in `-imagePath` for the image file
3. Look in the list of embedded images in the form file for the image file (only the file name and extension count)

Otherwise, the search order becomes:

1. Look in the list of embedded images in the form file for the image file (only the file name and extension is considered)
2. Look for a file on disk with the exact name specified in the data file
3. Look for the file in each folder specified in `-imagePath` for the image file

 *(Independent of version)*
Once images are found, their location is remembered such that subsequent references do not have to go through the search again.

 *(Independent of version)*

For HTML Image Embedding inside the HTML body of an email message the [_sendmail](#) option to use is `-cidFolder`. The `-imagePath` option is not applicable to embedding images in an HTML message body.

See Also

[-preferImages](#)

[HTML Image Embedding](#)

-inputDataType

(As of 3.2.001.01)

Instructs Merge what type of input data to expect.

Syntax

-inputDataType *type*

Description

Specify an input data type for Merge. This command option overwrites the assumption of the data type based on the file name extension. If this command option is not set, `.fnf` and `.dat` files are treated as Field Nominated Data and `.xml` files are treated as XML.

type may be one of: XML, FNF, XFDF, or JSON

-inputDateFormat

This Merge command line option identifies the format of Date data to expect from the input xml data stream.

Syntax

```
-inputDateFormat format [;format2...]
```

The default inputDateFormat is yyyyymmdd.

Description

This command option applies only when dates are to be recognized and re-formatted by Merge. The option is applied only to form Fields which are flagged with "Format as Date". If your data stream has already pre-formatted the date and it requires no additional formatting the Field should NOT be marked as a Date field, but rather a normal Text Field.

When the "Format as Date" option is selected the `-inputDateFormat` option will cause the data to be matched to one of the supplied formats and converted to the standard internal Merge date format of `yyymmdd`. The Merge date formatting can then re-format based on whatever date format you have requested in Design and produce the required output.

The parameter to the `-inputDateFormat` command is a list of one or more date formats, separated by a semicolon. Example:

```
-inputDateFormat mmddyyyy;yyyy-mm-dd
```

This would expect data stream dates in either `mmddyyyy` format or `yyyy-mm-dd` format. If dates are found that do not match either format, a warning message is issued into the log file.

This command recognizes only `yy`, `yyyy`, `dd`, or `mm` as valid date designators.

See Also

[Command Line Processing](#)

-inputTray

(Merge only) (As of 3.0.002.05)

Override the default input tray defined in the form properties.

Syntax

-inputTray *trayId*

Description

The Form Properties dialog allows you to set an input tray to be used. Using this command line option allows you to override that setting at Merge run time. Note that the Page Properties dialog allows you to set the input tray for each page layout, however most often that is left to the setting "Leave setting currently in effect". It is those pages that may be affected by setting this default, form level, input tray setting.

Of course, this can also be overridden by script on objects and in fact is identical to saying `this._inputTray = _1001;` in the form object script.

trayId must be one of the numerical identifiers defined in the applicable .prt (printer configuration file), for input trays. Typical values are **1001**, **1007**, ...

-JsonDataExts

Define the default file extension(s) for JSON input data.

Syntax

-JsonDataExts *ext1* [*;ext2* ...]

Description

The default extension for JSON is `.json`. This Merge option would rarely (or never) be used in the command line. It is defined in `...DO/Bin/Default-Merge.prm` and can be modified by creating the override `.../User/Overrides/Merge.prm`.

See Also

[-FmfDataExts](#)
[-XfdfDataExts](#)

-language

(As of 3.2.001.01)

Specify how the language is determined using the [Auto Translate](#) feature.

Syntax

-language *key*

Description

-language *FR*

Indicates that the [FR] section of the translate .ini file should be used for all translations.

-language [*LANG*]

Indicates that the current value of Field LANG on the Form will contain the section name to be used.

-language "[!Data *LANG*]"

Indicates that the current value of the data field LANG (in the actual data file) will contain the section name to be used. This option allows the LANG value to be accessed without adding a separate (and probably invisible) field on the form.

See Also

[Auto Translate](#)

[-translate](#)

-lineBreak

(As of 3.2.001.09)

Specify a line break algorithm.

Syntax

```
-lineBreak 0|1|2|3
```

Description

Most likely only values **1** and **2** will be used. Values **0** and **3** are mostly for tests.

Value **1** is the current default behavior. Value **2** is a Microsoft Word-like line wrap.

- **0** - old original DocOrigin algorithm, no ICU involved
- **1** - current default ICU-driven algorithm - breaks on some dashes and some punctuations
- **2** - MS Word-like line break algorithm which breaks on all dashes and doesn't on punctuation
- **3** - mixed ICU and MS Word-like line break algorithm which breaks on some dashes (ICU doesn't break between minus and digit) and doesn't on punctuation

If the user wants this behavior for both Design and Merge then they can add this option to `DocOrigin.prm`.

See Also

[Command Line Processing](#)

-lj4BoundFont

Indicate whether a target PCL printer supports unbound or only bound fonts.

Syntax

`-lj4BoundFont Y|N`

or often in the relevant `.prt` file as

`<BoundFont>Yes</BoundFont>` OR `No`

Description

The PCL reference manual says:

- ✔ "The terms 'bound' and 'unbound' refer to the symbol set capacity of a font. A bound font identifies a font which is restricted (bound) to a single symbol set. An unbound font (or unbound typeface) indicates the capacity to be bound a set of symbols selected from a complementary symbol index (such as the Master Symbol List, or the Unicode symbol index.)"

In DocOrigin terms, a "bound font" is a PCL font that spans only a single PCL symbol set (up to 256 characters). This is determined at Merge time – essentially hard-coded into the font that is sent to the printer. A further limitation (a DocOrigin limitation) is that we always include the first 128 ASCII characters in this count, so in fact, the DocOrigin limit is 128 "extended characters".

An 'unbound font' allows any number of characters in the font (so, greater than 256).

In practice, this means that most English and European languages can be downloaded as either bound or unbound (because there tends to be less than 128 non-ASCII characters involved). But Asian languages such as Chinese are probably limited to being unbound fonts as most forms are going to use more than 128 Chinese glyphs (characters).

Why does this all matter?

By default, most DocOrigin supplied `.prt` files use unbound fonts. But there are some PCL printers that do NOT support unbound fonts. So when printing to those printers you must tell the DocOrigin PCL driver to use 'bound fonts'. Many printers such as the HP laser printers tend to support either.

Generally, unbound fonts are used by the default DocOrigin `.prt` output configurations. You can change that by setting the `<BoundFont>Yes</BoundFont>` option in a custom DocOrigin `.prt` file, or via the command line override of `-LJ4BoundFont Y` option.

vs -useBuiltinSS

The `-usebuiltinss=Y` option tells us to believe that the target printer has the standard symbolsets that are built into DocOrigin via the `Default-D0PcLSymbolSets.ss` file. By saying yes, DocOrigin doesn't have to download the symbolset mapping tables, so the PCL is a bit smaller and tighter. Our default `.ss` file has the basic symbolsets that are defined in the PCL spec, so they all should be in the printer – but who knows with some of the quasi-clones.

With a Bound font the symbolset is built into the download font info, so the symbolset and its font info come as a "bound" `-lj4BoundFont` set. With unbound fonts, the font itself has no symbolset info, just a bunch of Unicode characters. DocOrigin has to supply a symbolset to map 0-255 into those Unicode characters.

See Also

[-useBuiltinSS](#)

-mergeCaseSensitive

(As of 3.3.001.01)

Allow the Data and Form DOM case to be sensitive or insensitive.

Syntax

-mergeCaseSensitive **Y|N**

Description

This option should only be used if there is a case mismatch between the Data DOM and the Form DOM. This is not uncommon for customers transitioning from Adobe Output since the Field Nominated Data file and form fields were case-insensitive. By default, this option is **Y** since the XML tags are case-sensitive. Case mismatch can be corrected in Design by dragging the Field from the Data Explorer to the targeted Form Field. We recommend correcting the case if your data is static and the number of Forms being converted is manageable.

i Note: objects that are case mismatched will not show as mapped objects (bold purple) in the Data Explorer.

Case Mismatch		Case Match	
Form Explorer	Data Explorer	Form Explorer	Data Explorer
<ul style="list-style-type: none"> ▼ [ab] DETAILS <ul style="list-style-type: none"> > [ab] DETAIL_HEADER ▼ [ab] DETAIL <ul style="list-style-type: none"> [ab] QUANTITY [ab] ITEM [aa] DESCRIPTION [ab] UNITPRICE [ab] AMOUNT 	<ul style="list-style-type: none"> ▼ [ab] Details <ul style="list-style-type: none"> ▼ [ab] Detail <ul style="list-style-type: none"> [99] Quantity F Item F Description [99] UnitPrice [99] Amount 	<ul style="list-style-type: none"> ▼ [ab] Details <ul style="list-style-type: none"> > [ab] Detail_Header ▼ [ab] Detail <ul style="list-style-type: none"> [ab] Quantity [ab] Item [aa] Description [ab] UnitPrice [ab] Amount 	<ul style="list-style-type: none"> ▼ [ab] Details <ul style="list-style-type: none"> ▼ [ab] Detail <ul style="list-style-type: none"> [99] Quantity F Item F Description [99] UnitPrice [99] Amount

w The -mergeCaseSensitive option does not apply to BindData.

-minimizeTextObjectHeightOnSplit

(As of 3.2.001.03)

Reduce the size of text object (parts).

Syntax

-minimizeTextObjectHeightOnSplit **Y|N**

Description

Y is original and default behavior to recalculate size of split object parts. Use **N** to prevent resizing.


See Also

[-textVerticalAlignmentMethodOnSplit](#)

-mode

(Merge only)

For PDF output, generate files in PDF/A (Archive format), or PDF/UA (Accessibility format).

 The PDF/UA accessibility format feature is available under only specific licensing arrangements. Modes PDF/A-1, PDF/A-1b, PDF/A-3, and PDF/A-3b became available as of 3.1.001.25.

Syntax

-mode **PDF/A** | **PDF/A-1** | **PDF/A-1b** | **PDF/UA** | **PDF/A-3** | **PDF/A-3b**

Description

These modes cause all fonts to be embedded in the PDF file, so files will likely be larger.

These options may be necessary for compliance with certain standards. Many of the PDF/A and PDF/UA requirements additionally require that the form be designed or tagged in a specific way. That must be done and verified by the form designer.

- -mode **PDF/A** creates a PDF/A compliant output document (Archive format).
Equivalent to PDF/A-1 and PDF/A-1b.
- -mode **PDF/A-3** or **PDF/A-3b** are base PDF/A compliant and allow embedded/included files as well.
Use the Merge -attachment option to attach files.
- -mode **PDF/UA** (as of 3.0.003.22) creates a PDF/UA compliant document (Accessibility format).
Form objects that do not have an explicit TagAs setting will be automatically tagged.

See Also

[-attachment](#)

-noStretch

(As of 3.3.006.01)

Control whether lines can stretch vertically to the parent pane.

Syntax

-noStretch **Yes** | **No**

The default setting is **No**, i.e., stretch.

Description

Control whether Merge prevents lines that touch a pane edge-to-edge to stretch vertically, as-is the default behavior. Applying -NoStretch **Yes** causes the lines that touch a pane edge-to-edge to remain at their drawn height (not grow to the parent pane). The -NoStretch option is available as a Merge command line option and will be applied to the entire template. Or, the option can be used as a tag on individual line objects.

Note this option overrides the -stretchFields option as well.

See Also

[-stretchFields](#)

-numCopies

(Merge only)

Set the number of copies of the document to output.

Syntax

-numCopies *copies*

Description

Set the number of copies of the document to output.

-output

Specify the location where the document output will be written.

Syntax

-output *filename* | **Select**

Description

Specify the location where the document output will be written, either a path/file name, a physical device such as LPT1:, or a network path. Note that the *filename* may include placeholders for various date-time-related elements, and also for dynamic field values. The output location can also be specified dynamically using the `printer.setOutputFile()` function, typically in the **Start next Print Driver** event script for a Form object. Further, the output location can be defined in the `<Outputfile>` tag of the printer configuration file (`.prt`).

For a WIN configuration file (`Default-WIN.prt`), using **Select** will cause Merge to prompt for a printer.

For more details see `_printer.setOutputFile()` function.

See Also

[File Naming Conventions](#)

[_printer](#)

-outputBin

(Merge only) (As of 3.0.002.05)

Override the default output bin defined in the form properties.

Syntax

-outputBin *binId*

Description

The Form Properties dialog allows you to set an output bin to be used. Using this command line option allows you to override that setting at Merge run time. Note that the Page Properties dialog allows you to set the output bin for each page layout, however most often that is left to the setting "**Leave setting currently in effect**". It is those pages that may be affected by setting this default, form level, output bin setting.

Of course this can also be overridden by script on objects and in fact is identical to saying `this._outputBin = _1001{_}`; in the form object script.

binId must be one of the numerical identifiers defined in the applicable `.prt` (printer configuration file) for output bins. Typical values are **1001**, **1002**, ...

-outputRef

Specify the Windows printer the output would normally go to.

Syntax

-outputRef *printername*

Description

(From [_printer.setOutputFile](#)) If `file::` is used with a WIN-based output configuration, it signifies that Merge is to instruct the Windows printer driver to send its raw output to a file on disk. This works for only some Windows printer drivers but has been known to be of use to capture such things as the raw ZPL destined for a Zebra label printer. When using this option with a WIN output configuration you need to identify that actual Windows printer that the output would normally go to if you weren't choosing to direct it to a file. You do that by using the `-outputRef` command line option. It identifies an actual printer, whose Windows driver will be asked to send its output to your nominated `file::` location instead. Your mileage may vary.

See Also

[_printer.setOutputFile](#)

-paginate

(Merge only) (HTML output only)

Control whether Merge creates multiple pages (shown as tabs) of HTML or produces a (possibly) long single page.

Syntax

-paginate **Y|N**

Description

Control whether Merge creates multiple pages (shown as tabs) of HTML or produces a (possibly) long single page. The default setting is **Y**, i.e. to create multiple pages.

Note that it is -paginate not -HTMppaginate.

If you were to apply it to non-HTML output you wouldn't like the result. The page will simply overflow and be lost. This feature is for HTML only!

-paperType

(Merge only) (As of 3.0.002.05)

Override the default paper type defined in the form properties.

Syntax

-paperType *paperTypeId*

Description

The Form Properties dialog allows you to set a paper type to be used. Using this command line option allows you to override that setting at Merge run time. Note that the Page Properties dialog allows you to set the paper type for each page layout, however most often that is left to the setting "Leave setting currently in effect". It is those pages that may be affected by setting this default, form level, paper type setting.

Of course, this can also be overridden by script on objects and in fact is identical to saying `this._paperType = _1001{_}`; in the form object script.

paperTypeId must be one of the numerical identifiers defined in the applicable .prt (printer configuration file) for paper types. Typical values are **1001**, **1002**, ...

-pdfCompress

Disable compression of PDF object streams

Syntax

-pdfCompress **Y|N**

Description

By default, the PDF objects that are created have their streams compressed. You can turn that off if you like. This is likely of value for only DocOrigin development for them to investigate any issues in object construction.

Note that this does not effect the storage of graphics which do tend to be the largest component of output PDFs. Always use the minimum reasonable resolution for your graphic images if you are at all sensitive to PDF file sizes.

-pdfScript

Specify an alternative script insertion file for fillable PDF.

Syntax

```
-pdfScript yourPDFScript.js
```

Description

This lets you provide your own JavaScript file for insertion into Fillable PDFs. That is as opposed to the default file: `DOPDFScript.js` that is installed in `.../DO/Bin`.

Obviously, this applies only when you are generating fillable PDF output.

The big bonus here is that you can include generic JavaScript functions for use in your PDF Viewer scripts. That is, as referenced by your form object's PDF Input event, or by your input field attributes.

No doubt you will copy the standard `.../DO/Bin/DOPDFScript.js`, likely to your Overrides folder, say as `.../User/Overrides/PDFScript.js`. Edit it. Then in your Merge command line, or in your `Merge.prm` file you could code the option as follows:

```
-pdfScript $O/PDFScript.js
```


That file contains DocOrigin functions as well, so do be careful not to alter them, but feel free to add functions outside of the DocOrigin object. Do note that the file extension is `.js`, not `.wjs`. It is plain JavaScript, without any DocOrigin extensions. You should probably reference Adobe's PDF API manual a lot. A link to it is found on the [Fillable Forms](#) page.

See Also

[Fillable Forms](#)

-preferImages

(As of 3.1.002.04)

 Note: This option was turned into a "do nothing" option in 3.1.002.06. See [-imagePath](#).

Specify the image search location

Syntax

`-preferImages` *fromXatw*|*fromDisk*

Description

When Merge is processing image file names found in the data stream it has an algorithm that it follows, see: [-imagePath](#). One portion of that algorithm is affected by this `-preferImages` option. Depending on the setting of this option it may choose to look for the named image file inside the XATW being used by Merge or choose to look in the file system.

The default setting is *fromXatw*.

Recommended Review

That default, *fromXatw*, is for backward compatibility purposes. We recommend that the setting be *fromDisk*. But of course, that choice is up to you. If you wish to change the default setting please revise your **\$O/Merge.prm** file.

See [-imagePath](#) for the search order algorithm.

See Also

[Command Line Processing](#)

[-imagePath](#)

-printInDocumentOrder

(Merge only)

Control the order in which Merge lays down objects and their backgrounds.

Syntax

-{*prt*}printInDocumentOrder *Y|N*

Description

When `-printInDocumentOrder` is not specified, it defaults to *N* (i.e. off). In that normal mode Merge will do several passes over the set of objects to be printed.

1. Lay down any shaded backgrounds
2. Lay down any images
3. Lay down text
4. Lay down fields

This works admirably well 99% of the time. However, sometimes you want to have greater control over the object drawing sequence. By specifying `-printInDocumentOrder`, (when specified, it defaults to *Y*), Merge will proceed down the document DOM tree, as represented in the Form Explorer view, and draw each object as it goes. For each object in turn it draws the background of the object before drawing its foreground. In this way, you can control whether one object gets drawn above or below another object by ordering those objects in the Form Explorer.

If your Merge run is using multiple output configurations you really should specify the leading *prt* as in:

`-pdfPrintInDocumentOrder` or `-lj4PrintInDocumentOrder`

If you are trying to stamp an image ovetop of a page's content it is common to put that image at the bottom of the Form Explorer list. Then specify `this._transparent = true;` on that image object because you want the lighter areas of the image to allow the underlying page content to show through. But, (and backwards compatibility keeps us from changing this), for PCL you must use `-lj4PrintInDocumentOrder=Y` whereas for PDF, you must use `-pdfPrintInDocumentOrder=N`. The underlying technologies address these requirements differently.

-prtOutputRollbackToFile

(As of 3.1.002.00)

For backwards compatibility, when Merge fails to open a printer, write to a file instead and name it the same name as the printer that failed to open.

Syntax

-prtOutputRollbackToFile **Y|N**

Description

Default is **N**. Merge behavior is changed from older versions so that failing to open a printer returns an error code (ERR_MERGE_PRINTDOCUMENT). Use this option to replicate old Merge behavior where a printer fails to open, Merge will write to a file instead. Unix functionality is unchanged and tries to write to the file.

See Also

[Merge Return Codes](#)

-rtfParserVersion

(As of 3.1.002.06)

Specify the DocOrigin RTF parser version.

Syntax

-rtfParserVersion **1|2**

Description

The default is "1". This is the original RTF parser left for backward compatibility.

(As of 3.1.002.06) it's possible to enable parsing of paragraph formatting commands. DocOrigin RTF parser version "2" supports per paragraph indentation, tab positions, alignment, and space before/after.

See Also

[RTF Support](#)

-scriptFile

Specify a script for Merge to run as soon as the form file is opened

Syntax

-scriptFile *scriptFileName*

Description

Specify the fully-pathed name of a script file for Merge to associate with the top-level Form object. Typically, such a script file has a **Start of Job** event script which is thus executed as soon as the Form is opened.

```
-scriptFile C:/DocOrigin/User/Scripts/InitForm.wjs
```

This option is rarely used.

The script file must include the wrapper for any event it wants to supply. For example, this would require:

```
function OnStartJob() {  
    ...  
}
```

as the wrapper for the **Start of Job** event. Script for multiple events, e.g. OnStartJob, and OnStartDoc can be supplied in the script file, provided the script for each event is surrounded by its applicable wrapper.

One place where this is used is in scripts for Design External Tools. Design's External Tool runs

```
Merge -scriptFile toolName.wjs
```

and effects (most likely) a change in the XATW being edited.

-scriptFolder (Merge)

Specify the folder where the script files reside.

Syntax

```
-scriptFolder foldername
```

Description

Specify the folder where the script files reside. By default, script files are assumed to be in the same folder as the form (the folder specified by the `-form` command line option). Use `-scriptFolder` to change this default.

```
-scriptFolder C:/DocOrigin/User/Scripts
```

When Merge encounters a `#include` in a script embedded in the form, and the include name does not provide a fully qualified path, Merge will look for the name in the `-scriptFolder` folder.

A common alternative is to use a folder mapping variable as in:

```
#include "$S/MyScript.wjs"
```

This same path is used for a referenced script-based filter that has no fully qualified path.

-scriptTools

(Merge only) (HTML output only)

Disable the automatic inclusion of a library of standard DocOrigin script functions into a generated HTML file.

Syntax

-scriptTools **Y|N**

Description

The standard script functions normally are automatically included in all HTML output. These functions are used when creating fillable HTML forms. They also include routines for handling dynamic manipulation of HTML forms and signature pad data input. Using this option you can suppress the inclusion of this script if you are creating static non-interactive HTML output. The default option value is **Y**.

-shiftX

As of 3.3.004.01

Reposition your output horizontally on the page.

Syntax

-{*prt*}shiftX *n*

Description

Specify a numeric amount and unit of measurement to move your output left or right on the page. You can use quotes to employ a negative value and specify a PRT by using its name as a prefix. For example:

```
-LJ4ShiftX="- .25in"
```

This option is recommended for development to fine-tune your design. In production, we recommend modifying margins as required in a PRT to work correctly with your specific printer.

See Also

[-shiftY](#)

-shiftY

As of 3.3.004.01

Reposition your output vertically on the page.

Syntax

-{*prt*}shiftY *n*

Description

Specify a numeric amount and unit of measurement to move your output up or down on the page. You can use quotes to employ a negative value and specify a PRT by using its name as a prefix. For example:

```
-LJ4ShiftY="- .25in"
```

This option is recommended for development to fine-tune your design. In production, we recommend modifying margins as required in a PRT to work correctly with your specific printer.

See Also

[-shiftX](#)

-showNames

Overprint the PDF output with the names of the container, panes, and groups.

Syntax

```
-showNames [color[,offset[,truncatePath]]]
```

(*truncatePath* available as of 3.3.006.01)

Description

This option is strictly for debugging purposes and even more so for support personnel who need to get acquainted with an unfamiliar form design. It causes each container name, pane, and group that is output to be overwritten with the name (PDF only). Even during initial form design, it can provide some insights into "surprising" output results.

By default, *color* is **blue**, *offset* is **0** (from the top left corner of the parent object), and *truncatePath* is **false** (show the object path).

Since a parent pane and its first child pane share the same top left position, there are going to be some overwrite collisions. Depending on what happens to be under the overprinted pane name, you may have to squint a bit to make it out. Nevertheless, this option has proven to be useful in getting oriented with a form design that the viewer has never seen before.

Position Option

Be careful not to enter a position larger than the parent object since it could cause overlap. In the example above, the Bill To and Ship To are 3.5-inch groups so using an offset larger than 4 inches causes overlap or name to be hidden. In addition, the container name is always centered at the top of the container so the container and first pane name overlap in the Sample_Invoice.xatw like this

~~Company~~Info.

Example 1

```
-showNames
```

Quantity	Item	Description	Unit Price	Extended Price
	FB001	Poster Boards, 22" x 28" Assorted Fluorescent	\$7.00	\$7.00

Example 2

```
-showNames red,1,true
```

CompanyInfo DocOrigin®	Container	INVOICE
Phone: +1.678.408.1245 Visit us at EclipseCorp.US		Invoice No. 2024-1704-3 Invoice Date 12/13/2024
Bill To <small>BillTo</small> Head Office, Pretend Printers 100 Any Avenue Chicago, Illinois USA 60611 Tel: 800-555-5555 Fax: 312-222-2222		Ship To <small>ShipTo</small> Pretend Printers 100 Some Street Moline, Illinois USA 61285 Tel: 800-555-9999 Fax: 888-555-9999

See Also

[Colors](#)
[Script Units](#)

-showXatwNames

(As of 3.3.006.01)

Overprint the PDF output with the names of the XATW templates.

Syntax

-showXatwNames [*color*]

Default is the option is off by exclusion.

Description

This option is strictly for debugging purposes and is only used when multiple XATW templates are merged. The XATW file name will be shown in blue at the bottom-left corner of each page. This is especially useful when troubleshooting a large batch of forms such as an Insurance Policy Packet.

Note that this option only shows if 2 or more forms are run.

-splitSegs

Control the splitting of output text strings into shorter strings when generating printed or PDF output.

Syntax

`-splitSegs Y|N`

Description

Due to small scaling problems with true type fonts, it is possible that longer text words can overlap the following word. This tends to be an unpredictable effect of display size and scaling. To mitigate this problem, Merge will split larger words into separate segments and start each at an accurately determined position, effectively reducing the cumulative positioning error of several characters in a row. By default, this option is set to **Y**. To disable the feature set `-splitSegs N`. When `-splitSegs Y` is in force, the number of characters after which a split is done varies by font size: 5 characters for 10pt, 6 for 12pt, etc.

-spoolerDocName

(Windows only)

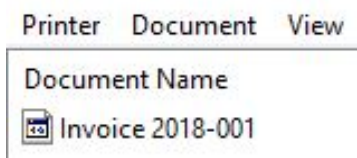
Supplies a custom document name to the spooler.

Syntax

-spoolerDocName *name*

Description

By default, Merge will provide the spooler with a document name of "Merge *prt* Document". Using this option you can supply a custom name such as "*Invoice 2018-001*".



Applies only to the output specified with "prt::" prefix. While you can specify this on the command line or in a .prt (output configuration) file, it is more likely that you would use the `_printer.setOption()` function to set the name dynamically. For example:

```
_printer.setOption("spoolerDocName", "Invoice "+InvoiceNumber._value);
```

It references field names in square brackets causing dynamic text substitution to happen. Also, it can reference a whole pile of "%x" type date/time elements too. See [File Naming Conventions](#).

See Also

-spoolerJobOwner
 _printer
 -{prt}option

-spoolerJobOwner

(Windows only) (As of 3.3.003.01)

Supplies a custom job owner to the spooler.

Syntax

-spoolerJobOwner *jobOwner*

Description

Change the spooler's job owner. Applies only to output specified with the "prt::" prefix. While you can specify this on the command line or in a .prt (output configuration) file, it is more likely that you would use the `_printer.setOption()` function to set the name dynamically. For example:

```
_printer.setOption("spoolerJobOwner", "newOwner", "PDF");
```

See Also

[-spoolerDocName](#)

[_printer](#)

[-{prt}option](#)

-stitchingFastCount

(As of 3.3.002.00)

Switch approach of how to count external PDF pages.

Syntax

-stitchingFastCount **Y** | **N**

Description

Default is **Y**. Use **N** to enable Merge to use this slower but reliable approach. Some non-DocOrigin created PDFs require this method.

See Also

[Merge External PDFs](#)

-stretchFields

(As of 3.2.001.06)

Control whether fields can stretch vertically to the parent.

Syntax


`-stretchFields Y|N`

The default setting is **N**, i.e., do not stretch.

Description

Control whether Merge allows fields to stretch vertically; in the same way, DocOrigin stretches vertical lines that touch a pane edge-to-edge. This is useful when fields are used to define the object borders of a table.

Applying `-stretchFields Y` causes the fields that touch a pane edge-to-edge to extend the height of the pane. Hence, all cells match in the pane.

 *As of 3.3.003.01* `-stretchFields` can be applied as a tag in Design, allowing control at the object level. Note that child objects of the tagged object are affected so you can place the tag on the pane so all child fields will grow to height.

This feature is unavailable for other objects, such as groups or boxes. However, you can replicate the same behavior by adding the following script to the objects you want to dynamically adjust to match the height of the parent pane.

```
this._height = this._parent._height;
```

-symbolSet

(As of 3.0.002.03)

Specify the symbol set (codepage) that the incoming data is in.

Syntax

`-symbolSet` *codepage*

Description

Inform Merge which codepage the xml data stream is in. This is unlikely to be useful since xml files state their encoding explicitly. If the xml "lies" by providing non-UTF-8 data but leaves out the encoding specification, thereby implicitly defining it as being UTF-8, then this option would be useful.

Note that specifying the `-symbolset` option to a [filter](#) (not Merge itself) can be very useful.

codepage is the name of the codepage the data is in. The list of available codepages is large. You can see that list by running:

```
uconv -l           // that's a lowercase 'ell'
```

uconv is in the `.../D0/Bin` folder of your installation.

cp1252 and *cp850* are popular choices.

See Also

[Codepage Handling](#)

[ConvertDatToXml Filter](#)

-testMail

Specify an alternate (an override) address for all calls to `_sendmail()`.

Syntax

`-testMail` *emailAddress*

Description

If you specify this alternate email address all calls to `_sendmail()` via script, or via internal email alerts, will be re-directed to this email address. Use this when testing to avoid sending inadvertent emails to addresses that might otherwise be used in a production situation.

See Also

[_sendmail](#)
[-alertTo](#)

-textpresentation

Control whether the printed text is broken into words or segments when written.


Syntax

`-textpresentation 0|1|2|4`

Description

The HTM and ZPL driver defaults to `-textpresentation 1` (always a line-at-a-time). Most other drivers (PCL included) default to `-textpresentation 0`.

The output of text to printers relies on accurate knowledge of the exact printer handling of font metrics. This can vary, depending on the printer models, resolution, etc. For this reason, Merge normally breaks lines of text into words and sometimes smaller word-segments and positions each independently. This reduces the occurrence of text overlap or right-justification errors. The `-textpresentation` command allows you to override this default behavior.

 Note: When generating PDF output, if you want/have to access the generated PDF, you definitely want to use `-PDFCompress N` (naturally this will generate (much) bigger PDF files).

Text Presentation Options

1 (Whole line) - each line of text is printed as a single unit, rather than being broken into smaller segments. This can be useful in the generated output if you are actually parsing the output file to recover text for other purposes. It will also result in slightly smaller output files.

The downside of `-textpresentation 1` is that you may find that text lines end up slightly too long, or too short which can be an issue if long lines are supposed to be right-justified. This is particularly noticeable when printing text for something like a table of contents, where a long row of dots is followed by a page number. If that text is right-justified with the aim of aligning the page numbers, `-textpresentation 1` may not accurately align that text. Using the default `-textpresentation 0` option should fix that problem.

2 (Shrink font) - used to reduce the size of the font. Works only with HTML and EML configuration.

4 (Rasterized Text) - converts the label/field into an image, allowing any font to be used regardless of printer capabilities. This is particularly useful for forms requiring non-standard fonts like Arial in ZPL labels. If you use rasterized text, the text quality may be slightly degraded.

Can be specified as a command-line option (CLO) or directly as an object tag in the label configuration.

See Also

[-splitSegs](#)

-textVerticalAlignmentMethodOnSplit

(As of 3.2.001.03)

Control the split strategy for text objects.

Syntax

-textVerticalAlignmentMethodOnSplit **0|1|2|3**

Description

When a text object splits across pages, determine the vertical alignment of its object parts.

0 - original behavior to recalculate the size of split object parts

1 - same vertical alignment for all split object parts

2 - last object part is top-aligned

3 - all but first object part are top-aligned

See Also

[-minimizeTextObjectHeightOnSplit](#)

-tracelevel

(As of 3.0.003.19)

Control which aspects of trace output are to be produced

Syntax

```
-tracelevel [F,P,S,D]
```

Description

Control which aspects of trace output are to be produced. Trace output can be very voluminous so selecting specific areas of interest is a good idea. If `-trace` is used without any `-tracelevel` option then all trace output will be generated. The available values are *Font*, *Pagination*, *Script*, and *Detail*. They can be abbreviated using just their first initial. These areas of interest can be combined by including them in a comma-separated list.

Example

```
-tracelevel P,S
```

Which indicates a desire to see tracing about pagination and scripting.

See Also

[-trace](#)

-translate

(As of 3.2.001.01)

Specify an Auto Translate .ini file.

Syntax

-translate *inifilename*

Description

The Auto Translate functionality requires the use of command-line options: `-translate` and `-language`. The Translate .ini file is used in conjunction with the `-language` option to control the automatic conversion of fields or text labels into local languages. For example:

```
-translate $$F/translate.ini
```

The Translate .ini file follows the standard format:

```
[section]  
name=value
```

Each section is used for the language value, with one key-value pair per line following each section. The name of the key is the lookup for an entry called a "TranslateKey", and the value is the text you would like to display. The `-language` option defines the section to be used for the translation.

See Also

[Auto Translate](#)

[-language](#)

[Auto Translate Sample Collateral](#) in User Guide (User Login Required)

-trimData

Causes all trailing blanks on data values to be removed immediately when read from the data file.

Syntax

`-trimData Y|N`

Description

Causes all trailing blanks on data values to be removed immediately when read from the data file. This helps to avoid certain undesired word wrap problems with data which has a large number of trailing blanks. This value defaults to **Y** (data is trimmed). It does not affect leading blanks and is applied to all lines.

-useBuiltinSS

(Merge only)

PCL printing only. Control the symbolset handling done by Merge.

Syntax

`-useBuiltinSS Y|N`


Description

Some old printers (we've encountered it on a Kyocera) are incapable of handling downloaded symbolset definitions. Supplying `-useBuiltinSS Y` will instruct Merge to use the printer's built-in symbolset definitions for its resident fonts. With any modern printer you are unlikely to use this option.

Normally, Merge will keep track of all of the characters that a document uses. Merge then creates a dynamic symbolset definition that is perfect for the given document. In that symbolset definition, the first 127 characters are the standard 7-bit ASCII characters. After that, a new symbolset codepoint is added for each character outside of 1 to 127 range. Merge will download that symbolset definition before and when selecting a font it will specify that downloaded symbolset.

Note that in this dynamic symbolset the 0x80 (128) codepoint does not refer to the Euro symbol but is rather whatever first character, outside of the 1 to 127 range, that was encountered for this font for this document. It might refer to the é character or whichever character valued greater than 127 was first encountered. If you peer into `.pcl` files, do not be misled by the existence of the 0x80 codepoint.

If you specify `-useBuiltinSS Y` then Merge will not use the above dynamic symbolset technique for resident fonts. Instead, it will use symbolsets that are built into the printer. PCL printers from different manufacturers may support a variety of built-in symbolsets for their resident fonts. These printers generally have a control panel, or some means, to cause them to print out the set of fonts they have and in so doing also show the symbolsets that those fonts use. Merge is aware of only certain symbolsets. These are defined in file `.../D0/Bin/Default-D0pclSymbolsets.ss`. The symbolsets that are defined in there are far and away the most common, and probably cover everything you need. If you happen to want to use a symbolset that is not there you can create a custom override file, `$0/D0pclSymbolsets.ss`. Of course, it is a plain text file.

 It's incredibly unlikely that you would ever have to do this. A useful resource, if you do go down this path, is http://www.pclviewer.com/resources/pcl_symbolset.html.

As Merge processes a text string it will detect when it encounters a character that is not in the currently selected built-in symbolset. When that happens, it will work out just what symbolset the offending character is in and switch to that symbolset. Later, possibly even "soon", it may have to switch back or to some other symbolset that contains the next character(s). Notice that symbolset selection is independent of font selection. By using `-useBuiltinSS Y` you are saying that the printer does not understand downloaded dynamic symbolsets but rather understands only built-in symbolsets.

-useFilterFormList

(Merge only)

Identify to Merge the forms found by a data conversion filter

Syntax

```
-useFilterFormList
```

Description

Have Merge switch to using a set of forms as determined by a data filter.

Operation

This option is used only in concert with a `-formList` option that is provided to the `ConvertDatToXml` filter. The `-formList` option identifies the name of a file to which all form names found in the data file are written. The `ConvertDatToXml` filter does that collection of form names and writes them out to the identified file. Then, if Merge is given this `-useFilterFormList` option it will discard any initial `-form` specification it was given and instead use the list of forms that the `ConvertDatToxml` filter found.

Example

Suppose an application produces a data stream that can contain data for each of three possible documents: `PurchaseOrder`, `Invoice`, and `Statement`. Those documents have independently maintained forms. The field nominated format data stream might have something like this in it:

```
^form PurchaseOrder
^field ... ..
^form Invoice
^field ...
...
```

Your run of Merge does not know ahead of time which of the three forms will be used. So you supply Merge with a `-form` option that lists all the potentially encountered forms.

```
-form PurchaseOrder;Invoice;Statement
```

You specify to the filter that you want it to collect up a list of all the forms that the data file of the moment uses.

```
-filter "ConvertDatToXml -formList formsUsed.txt ...."
```

Having specified that, the `ConvertDatToXml` filter would end up putting the text

```
PurchaseOrder;Invoice
```

into the file you nominated for that purpose, in our example `formsUsed.txt`.

If you provide Merge with the command line option `-useFilterFormList`, after the `ConvertDatToXml` filter runs, it will discard the original `-form` specification and instead use the form list that it finds in the file that was named by the `-formList` option that was given to `ConvertDatToXml`. In this example, it would end up with a form list of just `PurchaseOrder;Invoice`. As per usual, those forms would be combined into a temporary single form which would be used in the merging of the xml data that resulted from the `ConvertDatToXml` filter. Naturally, if our data stream had had `^form Statement` in it, then that form name would have been in the list of form names that was created. Only those form names that occurred in the data stream would end up in the created form list. The form name list has no duplicates and is in the order of their first occurrence in the data stream.

No Path Information

When ConvertDatToXml extracts the form names from its data file, it discards all the path information, leaving it with a 'naked' form name such as Invoice. When that name is passed on to Merge it will look for that Invoice.xatw file in the current directory. You will have to take the appropriate steps to ensure that the current directory is where the form design files can be found.

See Also

[ConvertDatToXml Filter](#)

-wininfo

Display "Merge-relevant" driver information for any Windows printer driver.

Syntax

-wininfo *drivername*

Description

When *drivername* is not provided, a printer selection dialog is displayed for selecting a printer driver. The output is written to the console window.

This command can be used to assist in configuring Merge for options such as paper type and duplexing.

If this option is specified, Merge will exit immediately after providing the requested Windows printer info.

An example output excerpt is as follows:

```
Windows printer options for 'HP LaserJet 200 color M251 PCL 6'  
Duplex is Available.  
  
InputTray 15 Automatically Select  
InputTray 257 Printer auto select  
InputTray 258 Manual Feed in Tray 1  
InputTray 259 Tray 1  
  
PaperType 297 Unspecified  
PaperType 296 Plain  
PaperType 295 HP EcoSMART Lite  
PaperType 294 HP LaserJet 90g  
PaperType 293 HP Color Laser Matte 105g  
PaperType 292 HP Premium Choice Matte 120g  
PaperType 291 HP Premium Presentation Matte 120g  
PaperType 290 HP Brochure Matte 150g  
PaperType 289 HP Cover Matte 200g  
PaperType 288 HP Matte Photo 200g  
PaperType 287 HP Premium Presentation Glossy 120g  
PaperType 286 HP Brochure Glossy 150g  
PaperType 285 HP Tri-fold Brochure Glossy 150g  
...
```

See Also

[Configuration files](#)

-XfdfDataExts

Define the default file extension(s) for XFDF input data.

Syntax

-XfdfDataExts *ext1* [*;ext2* ...]

Description

The default extension for XFDF is `.xfdf`. This Merge option would rarely (or never) be used in the command line. It is defined in `...DO/Bin/Default-Merge.prm` and can be modified by creating the override `.../User/Overrides/Merge.prm`.

See Also

[-FnfDataExts](#)
[-jsonDataExts](#)

The Merge Algorithm

To generate the output, Merge performs the following steps:

1. **Load the form file** into an internal Form DOM (Document Object Module).
2. **Process Filter script** or process if specified. This enables translation of the input data prior to merging with the form.
3. Process any **"Start Job" event script**. For information about the Merge events, see [Merge Events](#).
4. **Open the data file** - possibly altered by the Filter process above.
5. **For each document in the data file:**
 - a. **Load the XML data** for one document into the Data DOM.
 - b. Process any **"Start of Document"** event script.
 - c. Combine the Data DOM with the Form DOM to **create the Document DOM**. This includes replicating panes for which there are multiple instances of data elements within this document's Data DOM. See [_data, _document](#).
 - d. **Process global fields** to replicate their values to all instances within the document, see [Global Fields](#).
 - e. Replace each **automatic field** reference (\$fieldname) in the form.
 - f. Determine the actual size for all panes.
 - g. Process any **"End of Merge"** event script.
 - h. Replace each **embedded field reference** (for example, [FieldName] within a text label string) with the corresponding field value. For more information, see [Auto/Embedded Fields](#).
 - i. **Word-wrap all text fields**, handle truncation and resize fields that overflow.
 - j. **Determine the actual size for all panes** given that the fields are filled with data.
 - k. **Paginate the document**. The processing to this point treats pages as though they can have an infinite height. In this step, any page that overflows is split between panes or within panes that are flagged as being allowed to be split across pages. The overflow is moved to a new overflow page. This process is repeated until all the panes fit properly on all pages. See [Pagination](#).
Whenever a new page is inserted, overflow footer panes may be inserted at the bottom of one page and overflow header panes may be inserted at the top of the next page. Also on the newly inserted page, footer panes and header panes are processed to make sure that the values for any global fields are replicated and auto/embedded fields are resolved.
 - l. Process any **"End of Paginate"** event script.
 - m. Process any **"Ready to Print"** event script.
 - n. Perform a **final word-wrap** check to catch any fields that were altered in the previous script.
 - o. **Print the document** - for each print driver do the following:
 - i. Stretch vertical lines to reach the bottom of pane if applicable.
 - ii. Process text labels that contain a **"Page N of M"** type embedded field, as the number of pages in the document is now known.
 - iii. Apply field formatting (date, currency, etc.).
 - iv. Process any **"Start next Print Driver"** event script.
 - v. **Establish the output location**. Merge can create a single output file for all documents or a different output file for each document.
 - vi. Insert "blank" pages where necessary - this is usually to allow for proper printing of a duplexed document. Some pages are always to appear on the front side of the sheet of paper. To cause that to happen an additional page may have to be inserted.
 - vii. **Render all pages** to the output: to a PDF, a printer or a disk file.
 - viii. **Terminate the document**; close the open output file when individual files are required for each document.
 - p. **Update the summary variables** for the document and page counts. These updated page and document counts are available in the script engine. Script can be added to the "End of Document" event to store this information for later use by the Start of Summary script event.
6. Process any **"Start of Summary"** event script. This script can create a summary data file that is used for summary printing.
7. **Process any summary data** supplied in the previous step. This results in a full "document merge" of the summary data with the form using only those Panes whose Job Section has been set to "Summary".
8. Process any **"End of Job"** event script.
9. Process **"Finalize"** event script.
10. End of processing; **exit Merge**.

 **Word-Wrapping**

Word wrapping is reprocessed when changes to text are made.

XML Data Files

Data that is to be combined with a form is specified in XML format. A typical XML data file looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<XmlData>
  <Document>
    <ChargeTo>
      <Name>TONY BLUE</Name>
      <Addr1>1/44 GREEN ST</Addr1>
      <Addr2>YELLOWTOWN 3135</Addr2>
    </ChargeTo>
    <Header>
      <Invoice>10026645</Invoice>
      <Date>19MAR2010</Date>
      <SalesPerson>3</SalesPerson>
      <AccountNum>001327</AccountNum>
    </Header>
    <DetailLines>
      <LineItem>
        <ItemCode>F/GW712</ItemCode>
        <Desc>EXCELLENCE 7.5KG WASHER</Desc>
        <Qty>1</Qty>
        <Price>1089.00</Price>
        <Discount>NETT</Discount>
        <TAX>108.91 **</TAX>
      </LineItem>
      <LineItem>
        <ItemCode>F/ED56</ItemCode>
        <Desc>4.5KG S/S DRYER</Desc>
        <Qty>1</Qty>
        <Price>618.00</Price>
        <Discount>NETT</Discount>
        <TAX>61.24 **</TAX>
      </LineItem>
      <LineItem>
        <ItemCode>E/DS1100</ItemCode>
        <Desc>TURBO HEAD</Desc>
        <Qty>1</Qty>
        <Price>53.90</Price>
        <Discount>NETT</Discount>
        <TAX>0.00 **</TAX>
      </LineItem>
    </DetailLines>
    <Totals>
      <TotalPrice>$1760.90</TotalPrice>
      <TotalTAX>$174.50</TotalTAX>
    </Totals>
  </Document>
  <Document>
    a second set of data
  </Document>
</XmlData>
```

Although the data stream may have many sets of data, Merge processes all the data between a `<Document>` and `</Document>` tag as a single document. Note that Merge ignores any tags and data not included between the `<Document>` and tags. See also: [-documentTag](#) for setting a different document separator tag.

The XML tag names ("ChargeTo", "Name", "Header" and so on) must correspond to field names within the form. Design provides several features that enable a hierarchical structure to be created. For example, the form for this data file might include a group of data fields called "ChargeTo" that contains the fields "Name", "Addr1", and "Addr2". Merge will match the structure of the data to that of the form in order to correctly fill the data fields.

The general rule for merging data into a form is that the form design can have extra structural levels (e.g. groups and panes) but, to match, every structural level in the XML file must exist in the form design's structure.

Form	Data	
a.b.field	<a><field>xxx</field>...	match
a.b.c.field	<a><c><field>xxx</field>...	match
a.c.field	<a><c><field>xxx</field>...	no match

XML Includes

One XML data file can include other XML data files. A fuller write-up of this feature can be found at [PaneTalk XML Includes](#).

In excessive brevity:

1. Use the `-filter IncludeProcessor` on the Merge command line.
2. Define the `xi` namespace on the main XML file's root tag:
`<XmlData xmlns:xi="http://www.w3.org/2001/XInclude">`
3. Reference the XML file to be included like thus:
`<xi:include href="a file name.xml"/>`

XML Attributes

You will notice that there are no attributes in the sample XML. That is the preferred style, in fact, it was the required style until 3.0.005.05 (late December 2015). At that point, attributes could be provided on structure tags (not on leaf tags). Those attributes would automatically be transformed into additional leaf tags for the given structural element. For example:

```
<DetailLine Description="Construction Paper 50/Pack">
<Quantity>3</Quantity>
<Item>CP106</Item>
<UnitPrice>2.00</UnitPrice>
<Amount>6.00</Amount> </DetailLine>
```

would become (behind the scenes)

```
<DetailLine>
<Description>Construction Paper 50/Pack</Description>
<Quantity>3</Quantity>
<Item>CP106</Item>
<UnitPrice>2.00</UnitPrice>
<Amount>6.00</Amount>
</DetailLine>
```

No filter specification is required. That operation is automatic. To avoid the automatic transformation of attributes to leaf data elements you can use the `-ConvertAttributes No` command-line option.

Note that putting attributes on leaf nodes (nodes with data, and no child nodes) will not work. For example, the following is not allowed. It will not work.

```
<Item Desc="Paper">CP106</Item>    !!Wrong
```

- There is an `XmlAttrs.wjs` file delivered into the `DO/Bin` folder. It is essentially obsolete but left around in case someone used it. This is doubly so since with the advent of the `XmlInput` class it is easier to write filters for XML data files. See [XmlInput Class \(Read XML Files\)](#).

Besides writing filters, there is an even easier way to get at attributes, even those on leaf nodes. Contrary to what you would think, this requires that you override the standard-setting and specify `-ConvertAttributes No`. In that way, you've told Merge not to do its standard thing w.r.t. attributes but to let you handle it.

There is an `attribute()` function on each data DOM node. The `_data` DOM node for any given field is most easily referenced via the field's `_dataSource` property. Hence a construct in the pattern `_field.dataSource.attribute("attrName{_)")` will get the value of the named attribute, or "" if the attribute does not exist. Do beware of the possibility that `_dataSource` may return null if there is no `_data` node associated with a given field.

See Also

[_data \(The Data DOM\)](#)

Image data

The most common, and expected, way to specify a data value for an image field is for it to provide the file name of the image. This can also be an http: URL, though that will involve using, behind the scenes, an invocation of curl to fetch the image to a temporary file in local storage and then process it from there.

An alternate way is to use a standard "data URI". In this format, the image data is base-64 encoded and it is provided in the following style:

```
<imageFieldName>data:image/png;base64,iVBORw0KGgoAAA...</imageFieldName>
```

Per the usual base-64 standard, the encoding is split into lines that are 80 characters long.

JSON Data Files

(As of 3.2.001.01 experimental limited support of JSON input data was introduced.)

(As of 3.2.001.09 support was extended and improved.)

You should understand that Merge was designed to consume XML data and there is no one-to-one conversion between XML and JSON but we do our best to make it as close as possible so Merge may consume JSON as well. There may be tricky scenarios and you may have to adjust or restructure your JSON in some cases.

A typical JSON data file looks like this:

```
{
  "Document": [
    {
      "ChargeTo": {
        "Name": "TONY BLUE",
        "Addr1": "1/44 GREEN ST",
        "Addr2": "YELLOWTOWN 3135"
      },
      "Header": {
        "Invoice": "10026645",
        "Date": "19MAR2010",
        "SalesPerson": "3",
        "AccountNum": "001327"
      },
      "DetailLines": [
        {
          "LineItem": {
            "ItemCode": "F/GW712",
            "Desc": "EXCELLENCE 7.5KG WASHER",
            "Qty": "1",
            "Price": "1089.00",
            "Discount": "NETT",
            "TAX": "108.91 **"
          }
        },
        {
          "LineItem": {
            "ItemCode": "F/ED56",
            "Desc": "4.5KG S/S DRYER",
            "Qty": "1",
            "Price": "618.00",
            "Discount": "NETT",
            "TAX": "61.24 **"
          }
        },
        {
          "LineItem": {
            "ItemCode": "E/DS1100",
            "Desc": "TURBO HEAD",
            "Qty": "1",
            "Price": "53.90",
            "Discount": "NETT",
            "TAX": "0.00 **"
          }
        }
      ],
      "Totals": {
```

```
        "TotalPrice": "$1760.90",  
        "TotalTAX": "$174.50"  
    },  
    {  
        "a" : "second set of data"  
    }  
]  
}
```

Note, you may examine the format of JSON input data via opening sample or your form in Design, navigating menu **Tools > PDF Preview > Auto Generate Test Data > Save JSON Data** and checking the resulting JSON file.

Also, if you have JSON data file and are not sure how it is going to be matched with your form you may open Design, at the bottom-left, find "Data Explorer" pane, and using the context menu, "Open data file" on it examine what the resulting data structure will look like.

See Also

[XML Data Files](#)

Combine the Form and Data

When combining data with dynamic forms, the following steps occur:

1. If Merge is processing a Summary document, only Panes marked as Job Section "Summary" or "All" are enabled. Otherwise, Panes marked "Main" or "All" are enabled.
2. Load the Data DOM from the data file (.xml).
3. Create an empty Document DOM to build the "printed" document into. This contains only the root node to start with.
4. Run through the Form DOM identifying objects (pages and panes) that are mandatory. As one is identified, it is added to the Document DOM to ensure that it ends up as a part of the final document. A pane is included if it and all of its ancestor panes are flagged as mandatory.
5. Set the Document DOMs context as its root node.
6. Set the Form DOMs context as its root node.
7. Start processing by pointing to the first data item of the Data DOM.
8. For each data item in the Data DOM do the following:
 - a. Get the next data item from the Data DOM.
 - b. If end of the Data DOM, return to step 5.4 in [The Merge Algorithm](#).
 - c. Establish the item's fully qualified name from the Data DOM. For information [_data \(The Data DOM\)](#).
 - d. Look for the fully qualified name in the Document DOM.
 - e. If NOT found, look through the entire Document DOM for a match on the fully qualified name.
 - f. If found:
 - i. If the pane does not have data yet, then set the Document DOM's context and place data in all fields and return to step 8.1.
 - ii. If the pane may be instantiated multiple times, insert another copy of the pane immediately after the last copy in the Document DOM. Set the Document DOM's context and place data in all fields. Return to step 8.1.
 - iii. Toss this piece of data. Return to step 8.1. Assume there is no available place for this data in this document.
 - g. Look in the Form DOM for a match on the fully qualified name.
 - h. If found:
 - i. Instantiate the pane identified by the fully qualified name along with its ancestors in the Document DOM
 - ii. Set the Document DOM's context and place data in all fields.
 - iii. Return to step 8.1.
 - i. If not found, clear the data node and all its descendants. There is no place for this data.
 - j. Return to step 8.1.

Static Forms

When a form has no containers, DocOrigin considers this form "static" (as opposed to "dynamic") and instantiates all pages regardless of whether the Page property setting "Mandatory" is checked or unchecked.

Field Name Matching

An element node in the fully qualified name of the item in the Document DOM may be omitted from the fully qualified name of the corresponding item in the Data DOM.

Examples:

- A data item `InvoiceDetailLine.Field1` may map to a Document DOM item `InvoiceTable.InvoiceDetailLine.Field1`.
- A data item `InvoiceTable.Field1` may map to a Document DOM item `InvoiceTable.InvoiceDetailLine.Field1`.
- A data item `Field1` may map to a Document DOM item `InvoiceTable.InvoiceDetailLine.Field1`.
- A data item `InvoiceTable.Header.Field1` will NOT map to a Document DOM item `InvoiceTable.InvoiceDetailLine.Field1`.
Because `Header` is not a part of the fully qualified name of the Document DOM item, the mapping is not allowed.
- A data item `InvoiceDetailLine.InvoiceTable.Field1` will NOT map to a Document DOM item `InvoiceTable.InvoiceDetailLine.Field1`.
Because the first two-element nodes are not in the same order in the fully qualified names, the hierarchies are different.

Saying it a slightly different way: The form design can have more structure than the XML data.

The form design can have panes and groups that add structure that is not necessarily in the XML data.

For a match to occur all ancestor nodes in the XML data must match up with ancestor nodes in the form design, though the form design may have extra structure interspersed between those nodes.

Barcode Options

Barcodes in DocOrigin designs can be either static barcodes (the value is fixed when the form is Designed) or dynamic Barcode Fields, filled when Merge is run. The static barcodes are created using the barcode drawing tool in Design. To use dynamic barcodes, draw a Field, then set the Object Properties Field 'Display as' to Barcode. Both options then allow you to select from a dropdown list of built-in barcode formats.

i Note that the Zebra printer has built-in barcode generation that is defined and used somewhat differently - see the [Zebra Internal Barcodes](#).

2D Barcodes

2-dimensional barcodes are drawn as raster images that are scaled to fill the entire barcode object or field. DocOrigin supports the following 2-dimensional barcodes: [DataMatrix](#), [MaxiCode](#), [PDF417](#), and [QR Code](#).

Linear Barcodes

Linear barcodes are those that are typically a series of bars that is scanned left-to-right, as opposed to 2D Barcodes which are 2-dimensional arrays of marks or dots (see the 2D Barcodes section below.) The various supported linear barcodes are listed below, along with their available options. Note that all these options can be set either in the associated PRT, in Design using object tags, or at Merge runtime using script. See the examples below under Barcode Bar Widths.

In DocOrigin, barcodes are defined within a rectangular area - either a Field rectangle or a drawn static Barcode rectangle. By default, most barcodes will automatically scale to fit within the entire rectangle. So if you draw a 3"x3" rectangle, you'll get a barcode that's 3" high and 3" wide. You can override this default behavior and supply an actual bar width - either in the PRT configuration file, via script, or directly as a tag property of the object from within Design. See the Barcode Bar Widths section below, and the detailed list of supported barcodes below. The height of a barcode is scaled to ensure that the entire barcode, including any human-readable text that appears below it - fits within the margins of the Field or Barcode Label area.

The text font of a barcode object or field is used when drawing any associated text under the barcode. This includes typeface, size, bold, italic, etc.

Barcodes are printed with black bars, with the exception of the Pharma barcodes which will use the text color as the bar color as well.

Barcode Bar Widths

In DocOrigin all linear barcodes maintain fixed ratios for the bars. (This is unlike systems like Jetform where you explicitly set widths for individual bars) With DocOrigin the .prt configuration file defines a basic bar width. If not overridden by a setParm() call, or by an Object Tag setting, DocOrigin will use that width and its internal knowledge of the ratios of bar widths in a given barcode type to render the chosen barcode.

If the basic bar width is set to 0, then DocOrigin will use its internal knowledge of the barcode's relative bar widths to come up with a base bar width so as to stretch the barcode across the width that was designed for it in the form design. In general, the default basic bar width is set to 0, so this scaling is in effect by default.

In an override PRT, using setParm(), or using an object tag in Design, you may override the basic bar width. In that case, DocOrigin will use its internal knowledge of the relative bar widths in a given barcode type to produce the fixed-width rendering. For example, with barcode type Code 128, the internally known ratios, per the Code 128 spec are: 1:2:3:4 -- DocOrigin applies those ratios to the overridden basic bar width.

In the Default-LJ4.prt file you will see the following configuration for Code 128 options:

Default-LJ4.prt

```

...
<Barcode128Text>Y</Barcode128Text>           <!-- Draw text below barcode
-->
<Barcode128CharacterSet>8859-1</Barcode128CharacterSet> <!-- The encoding character
set to use -->
<Barcode128BarWidth>0</Barcode128BarWidth>       <!-- Set the width of a bar; 0
- scale to designed width -->
...

```

Other barcode types have similar-but-different option names and settings. Some have more than others. You can override the default barcode width in Design by selecting the barcode object or field, then using the **Format > ObjectTags** command:

The screenshot shows a dialog box with two input fields. The first field is labeled 'Tag Name:' and contains the text 'Barcode128BarWidth'. The second field is labeled 'Tag Value:' and contains the text '0.05in'. Below the fields is a blue circular icon with a white question mark, and to its right is a button labeled 'Add a New Tag'.

You can also override these configured settings at Merge runtime, using `setParm()`.

```

// set this field's barcode to a base unit width of 10 microns.
this.setParm("Barcode128BarWidth", 10);
// set this field's barcode to a base unit width of 1 millimeter.
this.setParm("Barcode128BarWidth", "1mm");

```

AusPost Barcode

This is a barcode used by the Australian Post.

Options

- **BarcodeAUSPOSTText Y** - if Yes adds human-readable text above the barcode.
- **BarcodeAUSPOSTTextBelow N** - if Yes adds human-readable text below the barcode.
- **BarcodeAUSPOSTBarWidth 0** - set width of narrowest bar. See Barcode Widths section above.

Code 11

Code 11, also known as USD-8, can encode any length string consisting of the digits 0–9 and the dash character (-).

Options:

- **Barcode11Text Y** - if Yes adds human-readable text below the barcode. Each character is drawn beneath its bar representation, so spread across the width of the barcode.
- **Barcode11Checksum Y** - if Yes, computes a trailing checksum automatically, 1 digit if the length of the message to encoded is less than 10 characters, and 2 digits if 10 characters or longer.
- **Barcode11NarrowBarWidth 0** - set width of the narrowest bar, 0 means auto calculate to fit the object's dimensions.
- **Barcode11Ratio 0** - set ratio of narrow to wide bars, 0 means auto calculate to fit the object's dimensions.

Code 25

Standard 2 of 5 supports numeric data only (0-9). Interleaved 2 of 5 is a compressed variant that takes the same options below.

Options:

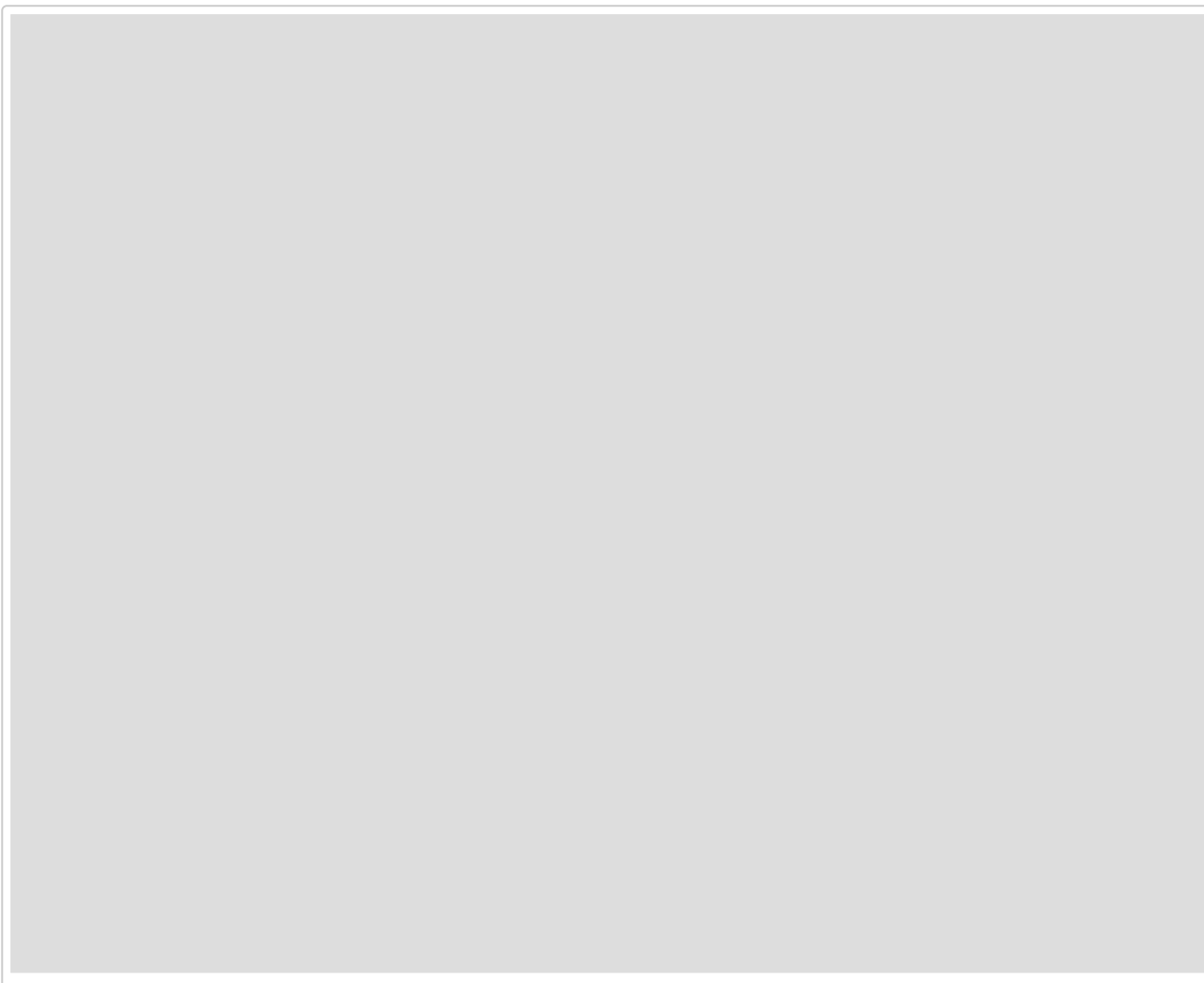
- **Barcode25Text Y** - if Yes adds human-readable text below the barcode.
- **Barcode25AddChecksum N** - if Yes, computes a trailing checksum automatically.
- **Barcode25BarWidth 0** - set width of narrowest bar. See [Barcode Bar Widths](#) section above.
- **Barcode25Ratio 3** - set ratio of narrow to wide bars.

Code 39

Code 39 or 3-of-9 supports a 43-character set of upper case letters, digits, and a few special characters. A variant using the Barcode39FullAscii option will encode a full ascii character using two standard code 39 characters.

Options

- **Barcode39BarWidth 0** - Set the width of the narrowest bar. See [Barcode Widths](#) section above.
- **Barcode39Ratio 3** - Set the ratio of narrow to wide bars.
- **Barcode39Text Y** - If Y, adds human-readable text below the barcode. Each character is drawn beneath its bar representation, so spread across the width of the barcode.
- **Barcode39AddChecksum N** - If Y, computes a trailing checksum automatically. Checksum text is included only when Compact Text is N.
- **Barcode39CompactText N** - If N, each character is drawn beneath its bar representation. If Y, the text is instead compacted together.
- **Barcode39FullAscii N** - If Y, convert text to use 2-character encoding of all text.
- **Barcode39DerivedText N** - If Y, the text includes any added characters, including from the full ASCII option. Applicable only when compact text is Y.



Code 128

Code 128 is a popular barcode that supports a full alphanumeric character set. In order to achieve compactness in their rendering, they support three different encoding tables known as Code 128 A, Code 128 B, and Code 128 C.

A useful reference:

[How Barcodes Work by University of Washington](#)

Historically you actually had to know that and choose between the different encodings, but DocOrigin does not require you to know that. It uses "adaptive encoding" – choosing the best performing (most compact) encoding automatically.

There is no more explicit dropping into code set C, versus code set B, etc. Just supply the data and DocOrigin will do the rest. As always – supply the data, and let the rendering engine decide on presentment.

GS1-128 barcodes - a variant of Barcode 128.

Support for the GS1-128 subset is enabled by setting the Barcode128-GS1 option.

A useful reference:

[GS1 General Specifications](#)


especially the chapter: GS1-128 symbology specifications.

Merge generates all of the FNC1 characters that are required. Merge may also generate all of the Modulo-10 check digits that may be required. Generation of the Modulo-10 check digit is defined by Barcode128-GS1Checksum.

Merge expects the data for GS1-128 bar code to be supplied in the following format:

```
(AI1)Data1(AI2)Data2...
```

Where AI1 AI2 etc are GS1 Application Identifiers and Data1, Data2, etc. are GS1 Application Identifier data fields, as described by the GS1 specification.

 Note that the data must always include a character at the position the check digit is required, if Barcode-128GS1Checksum is set to Y, the check character will be replaced (not inserted).

SSCC Serial Shipping Container Code

The Serial Shipping Container Code is a GS1-128 code with Application Identifier (00). Unlike most Application Identifiers, the SSCC Application Identifier and its associated data may not be combined with other fields in the same barcode. GS1-128 Above.

Options

- **Barcode128Text Y** - if Yes adds human-readable text below the barcode.
- **Barcode128Esc N** - enable the use of >8 to set a GS1-128 FNC1 code.
- **Barcode128CharacterSet 8859-1** - the encoding character set to use.
- **Barcode128-GS1 N** - switch to GS1 variant of Barcode128. Sets Barcode128Esc to Y.
- **Barcode128-GS1Checksum N** - used when generating GS1 variant to automatically create a GS1 checksum.
- **Barcode128BarWidth 0** - set width of narrowest bar. See [Barcode Bar Widths](#) section above.

DataMatrix

There are no user-configurable options for a DataMatrix barcode.

EAN-13

Options:

- **BarcodeEANBarWidth 0** - set width of narrowest bar. See [Barcode Bar Widths](#) section above.

MaxiCode

(As of 3.0.004.12)

It is expected that the major use of MaxiCode will be for shipping labels to be shipped by UPS.

The MaxiCode specification defines modes 0 through 6, but modes 0 and 1 are now obsolete. DocOrigin can generate MaxiCode in Modes 2, 3, 4 and 5. Mode 6 is used for programming hardware devices and is not supported in DocOrigin.

UPS labels use modes 2 and 3.

Merge expects the data for a MaxiCode field to be supplied in the following format for UPS labels:

```
aaaaaaaa|bbb|ccc|Tracking Number|SCAC|shipper number|day of pickup|shipment id number|Package
n/x|weight|address validation|street address|city|state
```

bbb is a three-digit country code (ISO 3166-1); 840 meaning, USA for example.

Prior to version 3.1.002.06

if bbb is 840 (USA) then aaaaaaaaa is a 9-digit US zip code. 5 digit zip codes must be padded with 4 trailing zeroes.

if bbb != 840 then aaaaaaaaa is a 9-character postal code made up of any combination of digits 0-9 or A-Z, with trailing spaces to pad it to 9 characters long.


Beginning with version 3.1.002.06

if bbb is 840 (USA) then aaaaaaaaa is a 5-digit or 9-digit US zip code. No padding is required.

if bbb != 840 then aaaaaaaaa is up to 9 characters long, made up of any combination of digits 0-9 or A-Z. It may have trailing spaces but they are not required.

ccc is a three-digit class of service.

Tracking Number and SCAC are required by UPS, the remaining fields are optional. Contact UPS for further details about the content.

 The characters shown below as <RS> <GS> and <EOT> are ASCII control characters from the C0 control set. These characters are not printable characters and may present some challenges to scanners. Many scanners will need to be programmed to send ASCII control characters to their host. Specialized software may be required to capture control characters, and display them.

When a Mode 2 or 3 MaxiCode symbol is generated for UPS use, it is required to be pre-fixed with the header:

```
[ ]><RS>01<GS>96
```

The data sequences are terminated with:

```
<RS><EOT>
```

As shown above, Merge expects the data items to be separated with | characters (the pipe symbol). Merge will replace the pipe symbols with the ASCII <RS> character as required. DocOrigin Merge will insert the header and trailer characters as necessary.

Data supplied to Merge, in a MaxiCode field, would look like:

```
123456789|840|001|1Z12345678|UPSN|06X610|159|1234567|1/2|3.1|Y|634 MAIN ST|YORK|PA
```

Data from a scanned, properly generated, Mode 2 MaxiCode symbol might look like:

```
[ ]><RS>01<GS>96123456789<GS>840<GS>001<GS>1Z12345678<GS>
UPSN<GS>06X610<GS>159<GS>1234567<GS>1/2<GS>3.1<GS>Y<GS>634 MAIN ST<GS>YORK<GS>PA<RS><EOT>
```

DocOrigin provides two MaxiCode configuration settings: **MaxiCodeMode** and **MaxiCodeCharacterSet**.

MaxiCodeMode defines which mode of MaxiCode barcode to produce. It can be set in the .prt file or via setParm. The default value is auto. With that setting in effect:

Prior to version 3.1.002.06

DocOrigin Merge will examine the Zip/Postal code provided, if it is numeric it will generate a Mode 2 symbol. If it contains any non-numeric characters it will generate a Mode 3 symbol.

Prior to version 3.1.002.06, we recommend that you explicitly set MaxiCodeMode.

Beginning with version 3.1.002.06

DocOrigin Merge will examine the Zip/Postal code provided, and the Country Code, if the Country Code is 840 (USA), and the Zip Code is 5 numeric digits or the Zip Code is 9 numeric digits, it will generate a Mode 2 symbol. If the Country Code is not 840, It will generate a Mode 3 symbol.

Other valid values for MaxiCodeMode are: 2, 3, 4 or 5 which force the generation of a symbol of the appropriate mode.

MaxiCodeCharacterSet defines the character encoding that the data will be translated to before the MaxiCode symbol is created. The default is "8859-1". Other valid values can be found by running "uconv -l" as supplied with this install of DocOrigin. Encodings which produce NULL bytes will not create correct MaxiCode symbols. MaxiCodeCharacterSet can be defined in the .prt file or set via setParm.

PDF417

There are no user-configurable DocOrigin options for a PDF417 barcode.

Pharmacode

This barcode is used in the pharmaceutical industry.

Options:

- **BarcodePharmacodeBarWidth** - set the width of the narrow bar, wide bar, and gap. See [Barcode Bar Widths](#).
Default is .5mm 1.5mm 1.2mm

Pharmacode 2Track

This barcode is used in the pharmaceutical industry. The 2Track variant features half-height bars as well as full-height bars.

Options

- **BarcodePharmacode2TrackBarWidth** - Widths of bar and gap
Default is .5mm .6mm

PostNet

⚠️ **Deprecated**

This barcode was used by the US Postal Service. It is deprecated. Users are strongly encouraged to use the USPS Intelligent Mail barcode.

PostNet is for U.S. addresses only. It supports zip (5 digits), zip+4 (9 digits), and zip+4 + "delivery point" (11 digits total). All formats encode an internally generated check digit as well.

Options

- **BarcodePostNetBarWidth n** - set the width of a bar. n should be in units understandable by DocOrigin, see [Script Units](#). 0 means auto calculate bar width to make the rendered barcode fit the designed width.
- **BarcodePostNetRatio n** - sets the space/bars ratio. n should be a decimal number somewhere between 1 and 2 to fit width limits. 0 means auto-calculate space width to make the rendered barcode fit the designed width.
(As of 3.1.002.07)

Some useful use cases are:

- Use "BarcodePostNetBarWidth n " and "BarcodePostNetRatio 0" - auto calculate space width to make the rendered barcode occupy barcode designed width.
- Use "BarcodePostNetBarWidth 0" and "BarcodePostNetRatio n " - auto calculate bar and space widths to make the rendered barcode occupy the barcode designed width and keep the desired space/bar ratio.

Note that:

- The width of the bars must be between 0.015" and 0.025".
- The width of the bar/space pairs must be between 0.045" and 0.050".

Consider BarcodePostNetBarWidth .02in BarcodePostNetRatio 1.4

QR Code

Options

- **QRCodeLevel M** - error correction levels - L (Low), M (Medium), Q (Quartile), H (High)
- **QRCodeCharacterSet 8859-1** - the encoding character set to use.
- **QRCodeMode 8BIT** - can be KANJI, 8BIT, ALPHANUMERIC, NUMERIC, FORCEBINARY

UPC

Options

- **BarcodeUPCBarWidth 0** - set width of narrowest bar. See [Barcode Bar Widths](#).

USPS-IM

The USPS Intelligent Mail barcode.

Options

- **BarcodeUSIMText Y** - if Yes adds human-readable text above the barcode.
- **BarcodeUSIMTextBelow Y** - if Yes adds human-readable text below the barcode.
- **BarcodeUSIMBarWidth 0** - set width of narrowest bar. See [Barcode Bar Widths](#).

Zebra Internal Barcodes

Use Zebra Internal Barcodes when generating ZPL output for a Zebra printer. Although "regular" DocOrigin barcodes would work, you would not be leveraging the internal drawing capabilities of your Zebra printer. DocOrigin would instead generate an image of your barcode and send it to the Zebra printer. To use the Zebra printer's internal barcode calculations, choose the "Zebra Internal" setting for your barcode (label or field) in the Object Properties in Design. Likely you will need to set the ZplType using an object tag as well.

DocOrigin includes the fragment library `Default-Zebra.xatwLib` which contains a field and label version of each Zebra internal barcode. Each barcode includes the following applicable chart in the object notes and the ZplType tag is set for your convenience.

Zebra barcode options are also found in the PRT files `Default-ZPL200.prt` and `Default-ZPL300.prt`.

The set of properties varies for each barcode type. The property "CodeValidation" can be applied to any barcode, allowable settings are Yes or No. Turn on code validation to check characters, field length, check-digit and parameters.

Code 11

Property Name	Description	Allowable settings
ZplType	Barcode type	Code11
ZplCheckDigit	Yes - 1 digit, No - 2 digits	Yes or No
ZplNarrow Width	Width of a narrow bar (units: inches, cm, or microns). If set to zero, the width of the narrow bar is calculated based on the object's width.	
ZplRatio	The wide bar to narrow bar width ratio.	2.0 to 3.0
ZplBarHeight	Height that a bar should be drawn (units: inches, cm, or microns). If set to zero, the height is calculated based on the object's height.	
ZplPrintText	Should the text be printed along with the barcode?	Above, Below, or None
ZplPreciseText	Set to Yes if you would like the human-readable text to be drawn the actual size specified in the object. Must be set to NO if you wish to print the printer-calculated check digit.	Yes or No

Code 39 (3 of 9)

Property Name	Description	Allowable settings
ZplType	Barcode type	Code39
ZplCheckDigit	Should the barcode have a check digit?	Yes or No

Property Name	Description	Allowable settings
ZplNarrow Width	Width of a narrow bar (units: inches, cm, or microns). If set to zero, the width of the narrow bar is calculated based on the object's width.	
ZplRatio	The wide bar to narrow bar width ratio.	2.0 to 3.0
ZplBarHeight	Height that a bar should be drawn (units: inches, cm, or microns). If set to zero, the height is calculated based on the object's height.	
ZplPrintText	Should the text be printed along with the barcode?	Above, Below, or None
ZplPrecise Text	Set to Yes if you would like the human-readable text to be drawn the actual size specified in the object. Must be set to NO if you wish to print the printer-calculated check digit.	Yes or No
ZplDerived Text	Should the escape, start, and stop characters be shown in the printed text?	Yes or No
ZplConvert ToFullAscii	Should the text provided be converted to the Code39 full ascii encoding?	Yes or No
ZplIncludeEscapeFor3of9	Include -\$ and +\$ into the data	Yes or No

Code 128

Property Name	Description	Allowable settings
ZplType	Barcode type	Code128
ZplCheckDigit	Should the barcode have a check digit?	Yes or No
ZplNarrow Width	Width of a narrow bar (units: inches, cm, or microns). If set to zero, the width of the narrow bar is calculated based on the object's width.	
ZplRatio	The wide bar to narrow bar width ratio.	2.0 to 3.0
ZplBarHeight	Height that a bar should be drawn (units: inches, cm, or microns). If set to zero, the height is calculated based on the object's height.	
ZplPrintText	Should the text be printed along with the barcode?	Above, Below, or None
ZplPreciseText	Set to Yes if you would like the human-readable text to be drawn the actual size specified in the object. Must be set to NO if you wish to print the printer-calculated check digit.	Yes or No

Property Name	Description	Allowable settings
ZplMode	Mode to draw barcode in	N = no selected mode U = UCC Case mode A = Automatic D = UCC/EAN Mode

Standard 2 of 5

Property Name	Description	Allowable settings
ZplType	Barcode type	2OF5
ZplNarrowWidth	Width of a narrow bar (units: inches, cm, or microns). If set to zero, the width of the narrow bar is calculated based on the object's width.	
ZplRatio	The wide bar to narrow bar width ratio.	2.0 to 3.0
ZplBarHeight	Height that a bar should be drawn (units: inches, cm, or microns). If set to zero, the height is calculated based on the object's height.	
ZplPrintText	Should the text be printed along with the barcode?	Above, Below, and None
ZplPreciseText	Set to Yes if you would like the human-readable text to be drawn the actual size specified in the object.	Yes or No

Interleaved 2 of 5

Property Name	Description	Allowable settings
ZplType	Barcode type	I2OF5
ZplCheckDigit	Should the barcode have a check digit?	Yes or No
ZplNarrowWidth	Width of a narrow bar (units: inches, cm, or microns). If set to zero, the width of the narrow bar is calculated based on the object's width.	
ZplRatio	The wide bar to narrow bar width ratio.	2.0 to 3.0
ZplBarHeight	Height that a bar should be drawn (units: inches, cm, or microns). If set to zero, the height is calculated based on the object's height.	
ZplPrintText	Should the text be printed along with the barcode?	Above, Below or None

Property Name	Description	Allowable settings
ZplPreciseText	Set to Yes if you would like the human-readable text to be drawn the actual size specified in the object. Must be set to NO if you wish to print the printer-calculated check digit.	Yes or No

Industrial 2 of 5

Property Name	Description	Allowable settings
ZplType	Barcode type	2OF5IND
ZplNarrowWidth	Width of a narrow bar (units: inches, cm, or microns). If set to zero, the width of the narrow bar is calculated based on the object's width.	
ZplRatio	The wide bar to narrow bar width ratio.	2.0 to 3.0
ZplBarHeight	Height that a bar should be drawn (units: inches, cm, or microns). If set to zero, the height is calculated based on the object's height.	
ZplPrintText	Should the text be printed along with the barcode?	Above, Below, or None
ZplPreciseText	Set to Yes if you would like the human-readable text to be drawn the actual size specified in the object.	Yes or No

UPC-A

Property Name	Description	Allowable settings
ZplType	Barcode type	UPC-A
ZplNarrowWidth	Width of a narrow bar (units: inches, cm, or microns). If set to zero, the width of the narrow bar is calculated based on the object's width.	
ZplBarHeight	Height that a bar should be drawn (units: inches, cm, or microns). If set to zero, the height is calculated based on the object's height.	
ZplPrintText	Should the text be printed along with the barcode?	Above, Below, or None

PDF417

Property Name	Description	Allowable settings
ZplType	Barcode type	PDF417
ZplBarWidth	Width of a bar (units: inches, cm, or microns). If set to zero, the printer default is used.	

Property Name	Description	Allowable settings
ZplBarHeight	Bar code height for individual rows (units: inches, cm, or microns). If set to zero, the height is calculated based on the object's height.	
ZplSecurityLevel	This determines the number of error detection and correction code-words to be generated for the symbol. The default level provides only error detection without correction.	0-8
ZplColumns	Number of data columns to encode. If set to zero let printer calculate.	1-30
ZplRows	Number of rows to encode. If set to zero let printer calculate.	3-90
ZplTruncate	Truncate right row indicators and stop pattern.	Yes or No

Data Matrix

Property Name	Description	Allowable settings
ZplType	Barcode type	DataMatrix
ZplBarHeight	Height on one row of dots (units: inches, cm, or microns).	
ZplQuality	The amount of data that is added to the symbol for error correction.	0, 50, 80 100, 140, or 200
ZplColumns	Number of columns to encode. If set to zero, the number of columns is calculated based on the amount of data.	9 to 49 when ZplQuality is 0 to 140 (must be an odd integer) 10 to 144 when ZplQuality is 200 (must be an even integer)
ZplRows	Number of rows to encode. If set to zero, the number of rows is calculated based on the amount of data.	9 to 49
ZplEscape	Escape character to be used when embedding special control sequences within the field data. This is used only when ZplQuality is set to 200.	any character
ZplAspect	The aspect ratio to use when laying out this 2-D barcode.	1 = square 2 = rectangle

QR Code

Property Name	Description	Allowable settings
ZplType	Barcode type	QRCode
ZplModel		1 = original 2 = enhanced
ZplMagnification	Magnification factor for barcode dots. If set to zero, the magnification factor is calculated to be the maximum that will keep the barcode within the bounds of the object.	1 to 10
ZplQuality	Reliability quality level.	H = ultra-high level Q = high level M = standard L = high density
ZplMode	Character mode for the data in the barcode. If set to "C" the mode is calculated based on the text provided for the object.	C = calculated N = numeric A = alphanumeric B = 8-bit Latin/Kana character set K = Kanji

RFID

Property Name	Description	Allowable settings
ZplType	Barcode type	RFID
ZplRFIDPrefix		Any

Auto Translate

(As of 3.2.001.01)

The DocOrigin **Auto Translate** feature is a way to create a form whose fixed text labels or fields can be switched from one language to another dynamically. It requires three elements:

- Translate File - A translation table INI file that contains a section for each language to be supported and value pairs for the translate key and value.
-translate "\$F/Translate.ini"
- Language option - The language (or section in the INI file) to use in that batch file or specific document (both are supported)
-Language "FR"
- Translate Key - the name of the value pair that is the "key" for that text label or field

Translate Key

(Option 1) Fixed Text Label with Tilde

The Auto Translate mechanism requires that labels (fixed text objects) on the form be defined with text values that begin with a tilde (~), for example, **~address**.

⚠ Text labels applying Auto Translate must be plain text formatted, not RTF.

When Merge is run, and the -Translate and -Language options (see below) are specified, the resulting documents generated by Merge will have the ~text values (e.g. **~address**) replaced by the appropriate translated text.

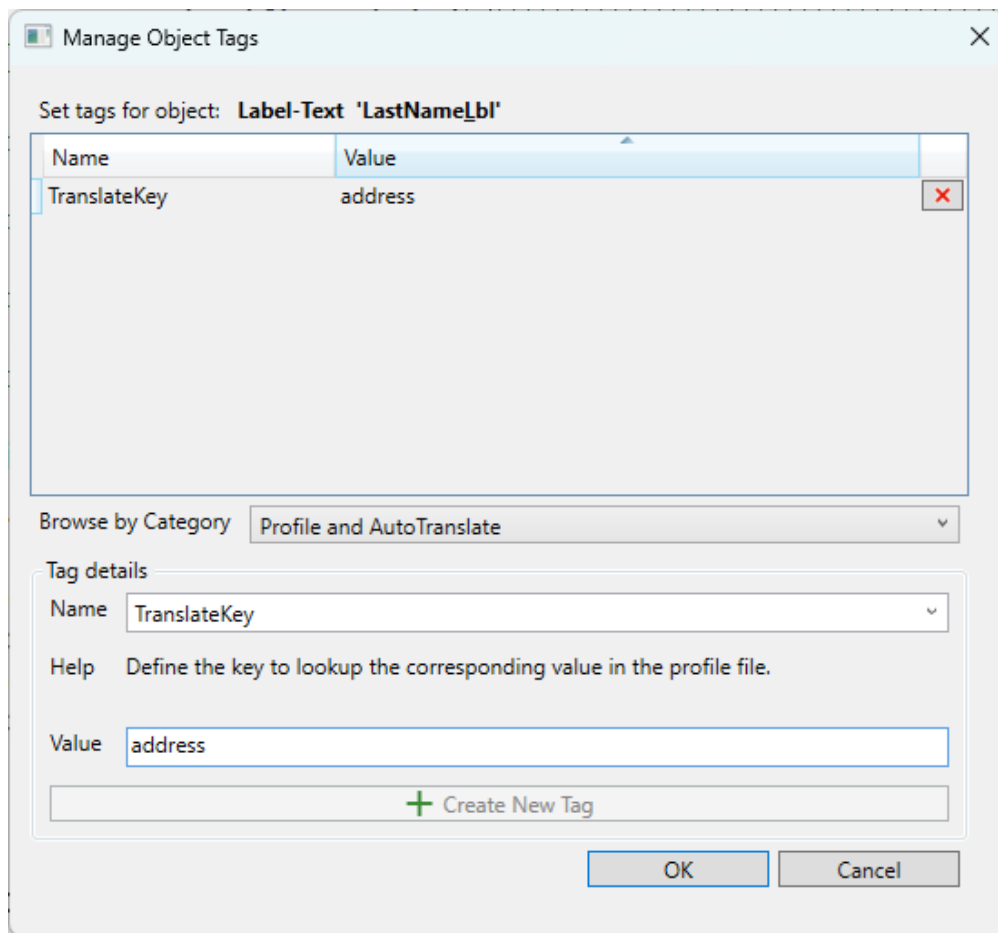
- Merge strips off the leading ~ and looks up the remainder of the text in the nominated translation ini file.
- The ~text value can contain blanks, but no leading or trailing spaces, and no equals (=) sign.
- The ~text value must match an entry in the translation ini file; that match is case insensitive. For example, the label's text value could be **~First Name**, and the INI entries could look like this:

```
first name = Prénom
```

- If no translation is found, Merge leaves the original ~text value in place.

(Option 2) TranslateKey Object Tag

A **TranslateKey** Object Tag can be used to define the name or left side of the value pair of the translation table. For example, you can add the tag **TranslateKey=address** (shown below) instead of using the ~address tilde approach in the sample above.



✓ Asterisk As TranslateKey Value

You can use an asterisk * as the Value of the TranslateKey object tag and then name the object as the name or left side of the value pair of the translation table.

Note: This approach does not support a space character in the name of the value pair since a space is not permitted as an object name.

A Translate Example

An example of this feature, called **Sample Translate**, can be found in the **DocOrigin/DO/Samples** folder. In this form, the **~FirstName**, **~LastName**, **~Address**, and **~Phone** text will be automatically replaced with the appropriate language-specific wording.

The **Sample_Translate.ini** file looks like this:

```
[EN]
; English text
FirstName = First Name
LastName = Last Name
Address = Address
Phone = Phone Number
LanguageName = English

[FR]
; French text
FirstName = Prénom
LastName = Nom de famille
Address = Adresse
Phone = numéro de téléphone
```

```
LanguageName = Français
...
```

And the test form's sample data stream, Sample_Translate.xml, is like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<XmlData>
  <Document>
    <Pane1>
      <First>Marion</First>
      <Last>Weaver</Last>
      <Address>2364 Pollock Rd, Ottawa</Address>
      <Phone>613 555 1515</Phone>
      <LANG>EN</LANG>
    </Pane1>
  </Document>
  <Document>
    <Pane1>
      <First>Elizabeth</First>
      <Last>Lévesque</Last>
      <Address>82 Boulevard de Clichy, Paris</Address>
      <Phone>+33 1 53 09 82 82</Phone>
      <LANG>FR</LANG>
    </Pane1>
  </Document>
</XmlData>
```

The Sample_Translate.xatw form looks like this:

DocTitle	
~FirstName	First
~LastName	Last
uses tag	Address
~Phone	Phone

Labels with ~text values and objects with a TranslateKey tag are automatically looked up in the specified (by -translate) translation ini file. The language to be used is specified by the -language option, in this case "[!Data LANG]", which for this document, evaluates to "[!Data LANG]".

The translate function is triggered by two Merge command-line options:

- **-Translate** - which tells Merge which Translation ini file to use.
- **-Language** - which tells Merge what Language (section) to use.

In the simplest case, you would just set -Language to (say) "FR" to get the French version of the text. This causes Merge to look in the translation ini file for the **[FR]** section and match the names in that section to the ~text on the form. For example, the **FirstName** value under the **[FR]** section is **Prénom**, so the text for **~FirstName** on the form will be set to "Prénom".

The -Language value can also be set to a named Field on the form by using the standard [name] syntax. So if you loaded the field LANG (which you'll notice is in the data file), then set:

```
-Language "[LANG]"
```

This will use the value of the **LANG form field** as the key to the translation.

Alternatively, and likely more easily, you can reference the LANG data node directly by using the syntax:

```
-Language "[!Data LANG]"
```

This finds the data value for the **LANG item in the data stream** directly rather than requiring a Field on the form.

In the **Sample_Translate** example above, the data set has data for several documents - each with a different **LANG** value, so each document is translated to a different language.

Setting Field Values

So far, we've talked about Labels, but you can also configure any form Field to get an automatic translation value by using the following mechanism. In the example form, you'll see a field at the top called **Field1**. In the Design, that field has a Tag of **TranslateKey** set to a value of **LanguageName**.


Any field with a **TranslateKey** tag will use that tag's value (in this case "LanguageName") as the key to lookup in the translation ini file.

In our ini file above, we have

```
LanguageName = English ...
LanguageName = Français
```

So **Field1**'s value will be set to either "**English**" or "**Français**" depending on the language active at the time.

This mechanism can be useful where the ~text mechanism gets too crowded. i.e., where the text label is only a few characters wide and you don't have enough room for meaningful ~text. The disadvantage from the form designer's perspective is that unlike the normal ~text label scheme, when using fields there isn't any helpful text visible on the form to make the form more readable to the designer.

-  Hindsight tells us that the example would have been better served, and less confusing if it had used something like "Title" rather than "LanguageName" and had the translate ini file have:

```
Title = The Document in English
Title = Le Document en Français
```

The TranslateKey has nothing to do with specifying which language to use. It merely identifies the key of the text to lookup in the translation ini file.

Hint

If your data stream doesn't have a simple data field that indicates the language key, you may need a bit of script to determine the key. If so try this simple workaround:
Add an invisible field to your form (perhaps called MYLANG). Add script to this field using the **Start of each Document** event to set the value of MYLANG based on whatever logic you need to make that determination. Now you can use the **-Language [MYLANG]** option to trigger all the other auto-translate features. (Make sure MYLANG isn't a node name in your data stream, or it will get that value instead of your computed one!)

See Also

[-language](#)
[-translate](#)

Auto Email

DocOrigin includes a feature to create sophisticated email messages and send them automatically from Merge. Using DocOrigin Design you can define the message body using many of the same drawing tools as used for other form designs. You can define the Email To, Cc, Bcc, Subject, and Attachment information as well. Email addresses can be pulled directly out of the data stream, or from an applicable form field. The email message can include images, links, and the usual bold/italic/font handling.

The same DocOrigin form file that contains the definition for a PDF document can also include all the necessary information to send that PDF as an attachment to a recipient. Or you can create only the email message itself, with or without additional attachments.

As of version 3.1.002.12, a newer, more integrated version of emailing has been included - as described below. Note that it is also still possible to code the sending of emails via the `_sendmail` script function. The Auto Email feature allows this to be done without employing script.


An example of the Auto Email feature can be seen in the DocOrigin/DO/Samples folder. The **Sample_InvoiceEmail.xatw** form can be opened in Design and tested using the PDF Preview command. If you want to actually send the emails (presumably to yourself for testing), you will need to ensure your system is set up for sending emails - see the [SendMail](#) page for more information on that.

Sample_InvoiceEmail uses its own sample data file, called **SampleInvoice2.xml**. It includes the recipient email address in it's <InvoiceHeader> <Email> data field. In this example, the email goes to a fictional email address (@example.com) - but you can copy and edit that .xml data file to send them to your own email address.

You can also preview the emails right on your own computer by adding the **hold:yes** option to the **Mail Options** setting in the Sample_InvoiceEmail form. The **hold:yes** option saves the email files (.eml) in the **\$U/Outbox/Hold** folder. If you have a local email client on your desktop (such as Outlook) you can double-click on the eml file and view the actual email generated by DocOrigin.

How it Works

If you open the Sample_InvoiceEmail form, you'll see an **_email_5** page in the form. The top of the page looks like this:

Email This Document - 5" wide output 

Use this Page to define a Email to be sent by DocOrigin Merge.
Click help button at top-right for more information.

From:

To:
(multiple email address must be separated by semi-colons)

CC:

BCC:

Subject:

Attachments:
Additional attachment files must be separated by semi-colons.
Use ![PDF] to attached a pdf of the remaining document pages.

Mail Options:
(override default SendMail options. Eg) **hold:yes**. See docs for details)

Plain Text Msg:

In this sample file, we've set the **To** address to the **Email** data field. This could also be a reference to any field within the current form using the usual **[Email]** notation. In this case, the Email data node has not been loaded into the form, so we just reference the data field directly. The To will be set to the value in the Email field, which is Kelly@example.com as shown below.

```
...
<InvoiceHeader>
  <CustomerID>Kelly</CustomerID>
  <Email>Kelly@example.com</Email>
...
```


@example Domain

According to the Wikipedia article: example.com, example.net, and example.org are second-level domain names reserved by the Internet Engineering Task Force through RFC 2606, Section 3,1 for use in documentation and examples. They are not available for registration. So you can use them safely for test email addresses. Of course, you can also specify a `-testMail` in your **DocOriginSendMailServer.prm** as an alternate (an override) address for all calls to `_sendmail()`.

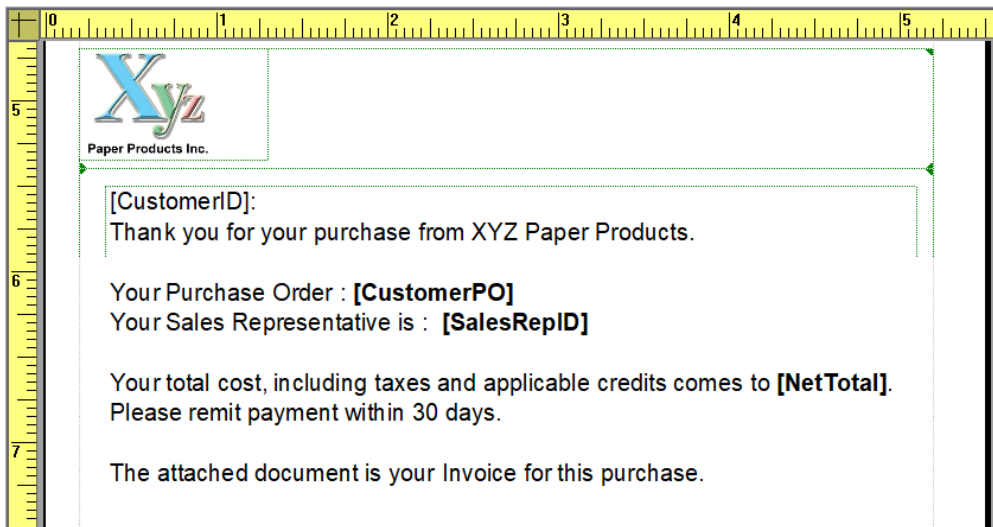
The **Attachments** field has been set to **!PDF**. This is a special embedded text option in Merge that fetches the file name of the PDF file that Merge has just created using the other pages in the form. The result is that when the email is sent, the sample invoice is attached to the email.

The **Mail Options** are for you to pass along any special options to the DocOrigin [SendMail](#) module. As suggested above, you can set **hold:yes** to cause the email to be left in the **\$U/Outbox/Hold** folder instead of emailing it right away. This is handy for testing.

The **Plain Text Msg** is an optional text message to include in the email just in case the recipient has a very old email system that doesn't handle a normal HTML email message.

-  It's good to include a plain text message, as some spam filters may penalize your message if it doesn't include a plain text message.

The bottom portion of the page is where you can create the main email message for your customer.



In the email world, the main body of the email is typically created in HTML. Unfortunately, there are many email client programs your customer might be using to read email, and each supports a different subset of the HTML language. For this reason, DocOrigin has created a special internal **EML driver** that converts the message to a very generic version of HTML suitable for most email systems. When you run Merge against a form such as this **Sample_InvoiceEmail.xatw**, that internal EML driver is automatically used to create an appropriate HTML email

message body from every **pane** in the **_email_5** page (other than the **_header_pane** which it uses for all the To/CC/Subject fields)

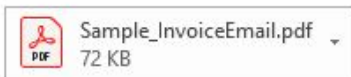
In our example there are two panes - the first just has the XYZ logo in it (more about this later on). The second pane is a typical text message to the customer, with inline substitutions of various field values from within the rest of the PDF document (on Page1 somewhere).

There are **some additional rules** regarding what Design constructs you can use in an HTML email message - see below.

You'll notice that the **page width** of this **_email_5** page is somewhat narrower than most documents. The page width is used to control the final HTML message width. In this case to 5 inches (hence the name!) The message will be centered in the email viewer window (phone, tablet, browser, Outlook, etc.) Statistically today, most emails are read on cellphones. A 5" wide email is fairly convenient.

Below you'll see how the email looks in Microsoft Outlook:

To Kelly@example.com



Kelly:
Thank you for your purchase from XYZ Paper Products.

Your Purchase Order : **20090420-001**
Your Sales Representative is : **Jessica**

Your total cost, including taxes and applicable credits comes to **\$2,038.72**.
Please remit payment within 30 days.

The attached document is your Invoice for this purchase.

Adding Auto Email to a Form

To add the auto email feature to a form, use the **Insert>Fragment** command, or the  button on the toolbar. Find the **Default-Fragments library** and the **_email_5 page** within that library and embed it in your design. (See the [Fragments](#) section for details on using Fragment Libraries.)

 If you are creating an email, but are not creating any attachment PDF to go with it, you can simply delete the existing default blank Page1 from a new form design. You can still Merge this using the PDF configuration (Default-PDF.prt) or use Design's PDF Preview command to generate the email. Regardless of which DocOrigin configuration you use, the **_email_5** page is always processed using the EML driver in Merge.

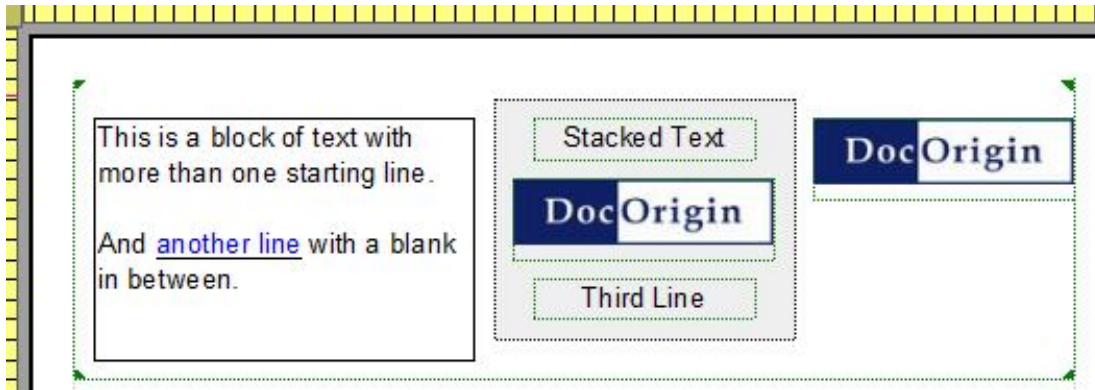
Designing DocOrigin Emails

There are certain rules about building email messages using DocOrigin. As noted above, the HTML supported by email systems is not as flexible as the HTML4 and HTML5 used in web browsers. The following information describes how DocOrigin can be used to create emails that will be supported by most common email clients.

- Emails in Design are constructed using a series of panes. Within each pane, you can use Text Labels, Fields, Groups, and Images. You cannot use Rectangles, Lines, or Barcodes.

- The Labels, Fields, or Images must be drawn as **a single horizontal row of objects within each pane**. You can, of course, have several panes, hence several "rows" of these objects. You can also use the Group object as a container for multiple **vertically-stacked** Labels, Fields, or Images. Note that groups cannot have side-by-side elements, only a vertical stack.

Here is a simple example in Design. It consists of a pane with three blocks of information: 1. a multi-line label; 2. a group (the rectangle) with 3 objects drawn vertically - two labels and an image; 3. finally at the right, a single image.



- i** Note that you can make the borders visible on any of these objects, and you can set a background color on any object (e.g. in this case, light gray on the Group object). Using object borders you can also create a horizontal line by selecting only the top or bottom Edge of the pane or some object within it.

Here is the result as it would appear in an email message:



Images in Emails

When adding images to an email, the image itself must reside on your **website**, i.e. be URL-accessible. It will NOT be embedded into the message. You must add an **Url tag** to the image object in Design, specifying the location where your image can be found on your website. You should also include an **Alt tag** with alternate text to be displayed if the image cannot be loaded for some reason. (Keep in mind that some email recipients may block downloading of images in their email system.) For our sample above, we have set the following 2 tags on the images:

```
Alt The DocOrigin Logo
Url http://www.docorigin.com/DocOriginLogo.jpg
```

Links

Any object can have a web link added to it by setting the **Link tag** on that object. This means you can make images, fields, or text labels "clickable" so as to take the user to a webpage of your choosing.

Hyperlinks within a text label are also supported for Email generation. To do this, just highlight a section of a text label, right-click and select **TagAs > Link** from the popup menu.

Testing Your HTML Email

Like all things, but especially email that can slip out into the world, **YOU MUST TEST THOROUGHLY**. We can't do that for you but we do provide some means for you to 'engage'.

1. Use the aforementioned `hold:yes` option to keep the email from escaping until you have finished testing.
2. Use `somebody@example.com` as test email address to keep email from escaping.
3. If you are forgetful, use the `-testMail` option in your **\$O/DocOriginSendMailServer.prm** file to redirect mail to a test address, hopefully, yours!
4. Enjoy the new buttons on the Design status bar to look at what you're producing.



5. It's always good to pore over the log file!
6. The `<Html>` button will give you a good view of the HTML email body. When the EML output configuration is used (often automatically) it produces **both** .html and .eml.
7. The `<Eml>` button will try to launch the generated .eml file. Hopefully, you have a program (often Outlook) associated with the .eml extension. (*fyi*: Since I don't have Outlook, I have installed the free [CoolUtils MailViewer](#) and have associated that with .eml. So far it has worked 'well enough' for me to see my various issues.) If you have put .eml files in the Hold folder you can review them there after the fact.
8. Don't forget to check out your plain text messages too. You are defining those, right!
9. Of course you can also redirect the email to yourself and see it in your own email client.
10. Get help! The folks at [Email on Acid](#) have lots of advice, including testing services. I'm sure we've run across others during development too. Do test across the variety of email clients that you might expect your recipients to have.
fyi: this link might interest you: [Email Client Market Share](#).
11. A tip. If you text edit .eml files you'll see lots of encoding, lots of =3D character sequences. You can make that easier on yourself by specifying **Encodehtml:8bit** in the Mail Options entry.
12. A tip. During development specify **MailScanFolder:\$F** in the Mail Options and you'll have the generated files more local to your work, not off in some Outbox folder. And you will be even more certain to take your emails out of accidental processing by DocOriginSendMailServer.

Beacons

It's quite common practice to include "beacons" in email. These are typically 1 pixel x 1 pixel transparent images which have an associated URL that automatically "calls back" to the host system to report that the email had been looked at, by whom, and related to what account, etc. Since these are HTML-based emails, that is no problem at all.

For your greater understanding, what is happening is that URL is processed by the email client/browser. It's in a `<img...>` element. The server should (must!) respond with an image. It can be small. The sample below does return a 1x1 pixel transparent image. If you don't return an image your user may get a red **X**.

To insert a beacon into your HTML email design simply put an image label in one of the panes and define an Url tag for that image. The Url tag refers to a web server request handler for the beacon's message. For example:

```
Url: https://example.com/beacon.php?[!Time]&data=[Transaction_ID]&account=[Account_ID]...
    ...&user=[CustomerName]
```

My example uses PHP but you can use whatever web technology you like. As you can see, you can embed information to pass to the "beacon" using any of the usual [Auto/Embedded Fields](#) notation. When Merge resolves the URL, it automatically encodes it such that you don't have to worry about breaking the url-encoded syntax with your field inserts. The PHP will receive it in unencoded fashion.

Presumably, that PHP will extract all the elements in the beacon's message, and likely store them away in a database, possibly initiating further server-side action as befits the host's desires. It is not possible for DocOrigin to supply that custom server-side technology. The following is a mere 'proof-of-concept' PHP to show that the email-embedded beacon is doing its job.

Download the sample **beacon.php** (as a .txt file) below. (Depending on your browser, the text file may open in your browser. Then, right-mouse-click on the page and select **Save As...**)

[beacon.txt](#)

Right-mouse-click on the **1x1transparent.png** image below and select **Save Image As...** (Note: a frame was added to see the small, white image and select)



Just the HTML Please

Maybe you just want this cellphone-friendly HTML to email later, or to post on a website. Clearly, that HTML is generated; you can see it via the <Html> button in the status bar. But normally it resides with an obscure name in the DocOrigin temp files folder. You can get just the HTML into a known place by using the Default-EML.prt output configuration directly and specifying the **-EMLOutputFile=xxxx.html** option. The desired HTML will be placed in the desired location with the desired name. If no value is set then **%T/%u.html** is used.

See Also

[_sendmail](#) scripting alternative

[SendMail](#) for configuring the DocOrigin email subsystem

Global Fields

Global fields may be confusing for the unprepared user since they are different from typical global variables in common programming languages. Actually, they are not variables at all.

As you can see in [The Merge Algorithm](#) global fields processing is done after the Document DOM has been created.

The algorithm is as follows:

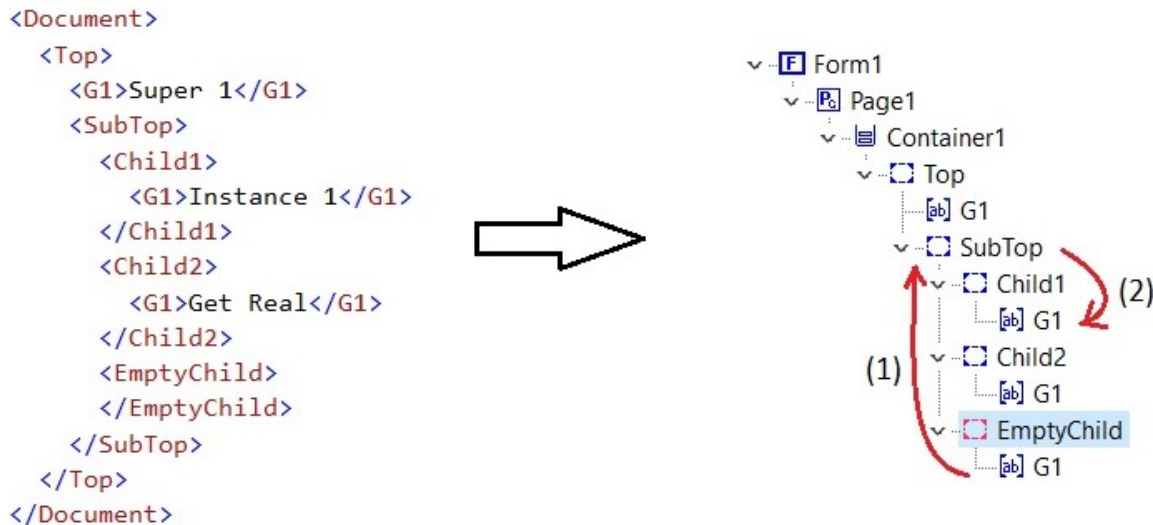
1. Merge loops through all **empty** global fields in a document.
2. If found one it tries to find the **nearest** suitable value for this field.
3. The **nearest** means it gets field parent and tries to find the same global field in a subtree. If successful it copies the value. If failed it gets one level up and retries.

The implication here is that if some global field is missing in an appropriate data location it gets filled with not the very first global value in a document tree but the **nearest** (see above) one.

Note, that if Merge can find the actual value for a field it doesn't do any special handling.

Let's examine one example, see image below.

EmptyChild pane is missing **G1** data so it gets filled with **G1** data from **Child1** pane and not from **Top** pane.



Auto/Embedded Fields

Automatic fields are form fields that automatically supply a field value, such as the current date or current document number. These fields are distinguished by a "\$" (dollar sign) as the first character of the field name. So for example, a field named \$DocNum would automatically display the current document number. Since these fields are filled in at data merging time, values that are known only after pagination is completed are not available as \$-automatic fields, but they are available as [!embedded] field references.

Embedded fields are metatags within a text label where data is automatically inserted. The embedded metatag can be either an existing field name on the form (such as [customerName] or a "!" (exclamation mark) character followed by one of the Automatic Fields specified below:

For example:

- "Today is [!Date MMMM dd, yyyy]." becomes "Today is March 05, 2016."
- "The current page is [!PN] of [!PC]." becomes "The current page is 3 of 5."
- "Dear [customerName]:" becomes "Dear Fred:"

Automatic Field	Embedded Field	Description
	[FieldName]	Value of the form field or label (<i>label added in 3.1.002.12</i>) called FieldName.
	[!PageNum]	Current page# within the document.
	[!PN]	Same as [!PageNum].
	[!PageNumForm]	Current page# within the form. (<i>Added in 3.1.002.02</i>)
	[!PNF]	Same as [!PageNumForm].
	[!PageCount]	Total # of pages within this document.
	[!PC]	Same as [!PageCount].
	[!PageCountForm]	Total # of pages within this form. (<i>Added in 3.1.002.02</i>)
	[!PCF]	Same as [!PageCountForm].
\$DocNum	[!DocNum]	Current document# being generated (starts at 1).
\$DN	[!DN]	Same as \$DocNum or [!DocNum].
\$Date	[!Date fmt]	Today's date. \$Date uses that field's Picture formatting, !Date uses picture format fmt or defaults to yyyyMMdd format.
\$Time	[!Time fmt]	Current time. \$Time uses that field's Picture formatting, !Time uses picture format fmt or defaults to hhmmss.
\$DateTime	[!DateTime fmt]	Today's date and time. \$DateTime uses that field's Picture formatting, !DateTime uses picture format fmt or defaults to yyyyMMddhhmmss format.
\$FormFile	[!FormFile]	Fully qualified form file name.
\$FormPath	[!FormPath]	Path of the form file name (without the form file name).
\$FormName	[!FormName]	Form file name (without the path or extension).
\$FormNameExt	[!FormNameExt]	Form file name and extension (if provided).
\$DataFile	[!DataFile]	Fully qualified data file name.
\$DataPath	[!DataPath]	Path of the data file name (without the data file name).
\$DataName	[!DataName]	Data file name (without the path or extension).
\$DataNameExt	[!DataNameExt]	Data file name and extension (if provided).
	[!Base64 name]	Converts the value of a named Field or Label to an encoded base64 string. (<i>Added in 3.1.002.10</i>)
\$Cachekeyword	[!Cache keyword]	Value from the -cache keyword:value command line option.

Automatic Field	Embedded Field	Description
	[!Data name]	Fetch data directly from the Data file. 'name' can be either a simple node name, or a compound name in dotted notation. If name is '*' the name of the current object (Field or Label) will be used. <i>(Added in 3.2.001.01)</i>
	[!Profile key]	The value from an external profile file. For values under an profile file section, use section.key to reference. See Profile Files .
	[!Session name]	Fetch data from the current Session. If name is '*' the name of the current object (Field or Label) will be used. See Session . <i>(Added in 3.2.001.01)</i>
	[!Env name]	The value from environment variables. If name is '*' the name of the current object (Field or Label) will be used. <i>(Added in 3.2.001.01)</i>
	[!Field name]	Synonym to [FieldName]. <i>(Added in 3.2.001.01)</i>
	[!Auto name]	Synonym to autofields. Added just for consistency. Example: [!Auto DocNum]. <i>(Added in 3.2.001.02)</i>

- All calls to date and time functions use a common date stamp, which is the time and date when the Merge began running. A call to \$Time 20 seconds into the run will return the same time as a call at the beginning.
- The \$Date, \$Time and \$DateTime fields can be formatted using the available field format options for these types of fields. *(As of 3.0.001.02)*
- To have a formatted date embedded in a text label, create a (possibly hidden) \$Date field that correctly formats the date, then references it in the text label as in "Today is [\$Date]".
- File names will not include a file extension unless one was passed on the Merge command line.
- The -cache command line option allows keyword:value pairs to be passed from the calling application to the form. An automatic or embedded reference to "keyword" returns the corresponding value.
- For a table of possible symbols in the ICU documentation https://unicode-org.github.io/icu/userguide/format_parse/datetime/#datetime-format-syntax

-embeddedDefault

If an embedded field could be excluded from the data such as title or address2, use the merge option `-embeddedDefault` to prevent the embedded field syntax from presenting in the output. For example, if the field **title** is optional, then use the merge option `-embeddedDefault ""`

Form Design

Dear [title] [firstName] [lastName],

Output

Dear Sam Smith,

See Also

[Object Tags \(Direct Data Binding\)](#)

[-embeddedDefault](#)

Merge Licensing Info

DocOrigin requires a license key file to enable the product features.

- Without a valid license key file, DocOrigin Merge will emit an "Evaluation" spoiler stripe across all pages of all generated documents.
- Without a valid license key file, some DocOrigin programs will refuse to run.
- A license key file may not allow all features to be accessed -- based on your negotiated contract terms.
- A license key file is meant to **help** you stay in compliance with your contract, however, it is your responsibility to not exceed the terms of your purchase.
- An Evaluation license allows all features to run but for a limited time, e.g. 30 days.
- An Evaluation license can be used on any, and any number of machines, also regardless of O/S. It does report the entity name of the evaluation license holder.
- License keys include a Maintenance and Support (M&S) expiry date. The software will continue to run, without spoilers, after that expiry date. What will result in an 'invalid license key' status, and consequent spoilers, is the attempt to upgrade your version of DocOrigin to one that was released after your M&S expired.
- If the system that is being upgraded allows internet access to docorigindb.com, and the M&S fees have been paid, then during the installation of the new software your license key file will be updated automatically to reflect the latest M&S expiry date (and possibly new feature flags). If internet access is not permitted, contact your distributor before upgrading to a new version. [And for heaven's sake, keep a backup of your old installation file!]

Merge Version

To display the current version number of Merge, use Windows Explorer to locate **Merge.exe** and double-click on it. Typically Merge will be in the **C:\DocOrigin\DO\Bin** folder on a Windows computer. For folks who use the command line, simply run Merge with no parameters at all. The current information on Page and Document counts will also be displayed. You will see a display similar to:

```
DocOrigin Merge version 3.1.002.11
(c) Copyright 2009-2020 - Eclipse Corporation WSL Inc.
Hostname: AcmeServ

This software is licensed for the exclusive use of "Acme Corp.".
License key file used: 'C:\DocOrigin\DO\Bin\Acme_EVAL.key'

Your license expires in 56 days.

Page Count: 135
Document Count: 100
Job Count: 20
```

In script within a form, you can get the Merge version via `_os.getMergeVersion()`.

License Key Renewal

License key renewal is done automatically during installation, assuming the system permits internet access to port 80 at docorigindb.com.

An explicit renewal request can be made at any time using:

```
DO Merge -renew
```

You might wish to do that before you install so as to inspect the .key license key file for its M&S expiry date to see if a recent payment of M&S fees has been recorded and the M&S expiry date updated accordingly.

The .key file includes the expiry date in encrypted form but also includes a comment line of the form: (Changing the comment line has no effect.)

```
* This license expires on yyyy-mm-dd
```

See Also

[License Keys](#)

Merge Output

There are a number of optional ways to control what DocOrigin Merge can generate as output, where it writes it, and in what format. These are summarized here.

The Basics

Merge output is controlled by DocOrigin [Configuration Files](#) (often called .prt files or config files). When you run Merge you must supply a configuration file that governs what fonts are available, when they must be downloaded or embedded (as opposed to assumed to be available), what output language to use (PCL, Postscript, PDF, HTML etc.), and a whole variety of options for these output languages.

Along with the .prt file, you should specify where the output is to be sent. This can be a local printer, a file, a named pipe, or even a program or command to move the output to a designated repository.

When your Merge run contains multiple documents (as in a whole series of invoices), you can also specify whether all those resulting documents are written as individual files or printer jobs, or combined using the `-combinedocuments` feature into a single document. This can be of particular value when generating PDF files, where you might want a single large PDF that contains all (say) invoices.

Finally, you can if you wish, create more than one output stream from the same Merge run. For instance, from a batch of documents create individual PDF files that are each emailed to the client as well as a single combined PDF that you deposit in an archival repository.

Downloaded / Resident Fonts

TrueType Fonts

Typically the fonts that are used are TrueType (TTF) fonts, or they are fonts that are already pre-loaded/pre-installed in the printer or the viewer technology. [PDF has many fonts that are intrinsically known by any PDF viewing technology.] When TTFs are used, Merge knows how to peer into them to discover character metrics and it knows how to subset the font so that only the characters that are used need to be defined in the output stream. If you have or can obtain a TTF version of a font, that is the best course of action to follow.

To indicate that you wish to use a certain typeface in your form designs, use the ConfigEditor tool to simply put a checkmark beside the typefaces of interest. ConfigEditor automatically finds the **TTF** files in your .prt's specified TTFPath.

i Note that you should not select a huge set of fonts, not for technical reasons, but rather pick a subset to standardize on so that your site's forms have a common look and "branding".

Resident Fonts

Some (most?) printers have intrinsic knowledge of a certain set of fonts. DocOrigin includes font metrics files for the most popular of these fonts. You can see those in the **.../DO/Config** folder as .mtx (metrics) files. Even in those cases, the .mtx files most often contain a reference to a .tff file, which will be used if it can be found on the system (in the nominated **TTFPath**). On a Windows-based system, often the referenced .tff file will be found and used. Otherwise, the character metrics defined in the .mtx file will be used. If your printer has a resident font for which no .mtx file is available, you can create your own, since the .mtx files are openly readable text.

Warning: Coming up with all the required information, especially character widths, may be difficult. If possible, try to find a ttf version of the font and use that directly, or try to find a conversion tool to convert whatever format font you have to a ttf format. Even if you have the ttf only temporarily, you can use that to create an mtx file. See TTFUtil in the **.../DO/Bin** folder. Run it with no parameters to get the 'usage' instructions.

To indicate that you have resident fonts in your printer, edit the relevant printer configuration file (.prt) with a text editor and enter in lines such as:

```
<ResidentFont Typeface="Courier New" Bold="*" Italic="*" ID="4099"/>
```

You can find examples of that in each shipped Default-xxx.prt file. Add to the list of resident fonts that are known to be in your printer. There are comments in the .prt to help you.

These "resident" font notes also apply to fonts which, through your own procedures, are always permanently downloaded into the printer.

Soft Fonts (PCL only)

If you have non-TTF fonts which you want to download into the output stream, then first, do consider finding a TTF alternative. However, if you must use these non-TTF (bitmap) fonts, then it is possible to do that. You must have a font definition file; typically a .SFP, or .SFL file, though other extensions are used as well. To make Merge aware of these fonts you would edit the relevant printer configuration (.prt) file with a text editor.

After the "<Printer>" tag, you should enter line(s) such as:

```
<DownloadFont_MICR300>MICR300.SFP</DownloadFont_MICR300>
```

where MICR300 is an example of the typeface being downloaded. The "MICR300.SFP" is the example of the file name that houses the soft font definition.

In many cases, the font may have bold, italic, and bold-italic variations. For this non-ttf technology, you will need to specify the soft font file to be used for each variation. E.g. in the style of:

```
<DownloadFont_typeface>xxxxxxxr.SFP</DownloadFont_typeface>
<DownloadFont_typeface_B>xxxxxxxr.SFP</DownloadFont_typeface_B>
<DownloadFont_typeface_I>xxxxxxxr.SFP</DownloadFont_typeface_I>
<DownloadFont_typeface_BI>xxxxxxxr.SFP</DownloadFont_typeface_BI>
```

Often there is no direct correlation between the typeface name and the soft font definition file name. Hence the example shows xxxxxx rather than typeface. The important point is the <nothing>, **_B**, **_I**, **_BI** tag suffixes. If your form designers will be clicking on the Bold and or Italic buttons for text rendered in this typeface, you will need those entries.

Those entries tell Merge where to get the soft font bits to copy to the output stream. It does that only if that particular typeface variation is actually used in the document.

You must also tell Merge (and Design) that the given soft font is available to use. You would do that by text editing the relevant .prt file and adding line(s) such as:

```
<Font MTX="xxxxxx.mtx"/>
```

There are lots of <Font...> entries present in any .prt so as to guide you as to placement.

You should notice the inference that you also need a font metrics (.mtx) file. As the downloaded font is not a .ttf file, you must supply the font metrics, just as was necessary for resident fonts.

A note about the .mtx files for dynamically downloaded soft fonts... Looking at the shipped **arial.mtx** file as an example, you will see that it starts with:

```
<font sTypeface="Arial" sPostscriptName="ArialMT" iBold="0" iItalic="0" iTTCNum="0" TTF="arial.ttf"/>
```

For dynamically downloaded soft fonts you must not supply the TTF= attribute. Otherwise, it would use that TTF file instead of your downloadable soft font. You also may as well leave off the iTTCNum= attribute. It will be useless.

Do specify the sTypeface= attribute. Also, note that many .mtx files could specify the same typeface name. This would certainly be the case for metrics for the regular, bold, italic, and bold-italic variations of a given typeface. Do not call one ABC**B**old and another ABC**I**ta**I**c, otherwise, both of those names will show up, confusingly, in the Design typeface dropdown. They should both be ABC and the form designer should select the **B** or **I** button(s) as per normal.

Finally, if you wish, you could consolidate several .mtx files into one larger file simply by concatenating multiple <FontMetrics>...</FontMetrics> sections into a single file. To keep it a well-formed XML file you could surround the whole set with <root>...</root>. Your .prt file would then need to reference only that one .mtx file in a line rather than the multiple entries that would have been required otherwise.


MTX File Units

Font metrics (.mtx) files are pretty much internal -- rarely being looked at by anyone. However, if you have resident or dynamically downloaded non-TTF fonts, then their contents can be of interest. Naturally, a "metrics" file contains numbers for widths etc. In the DocOrigin world, the internal unit is a millionth of an inch. Of particular importance here is that in a .mtx file the widths given are for a hypothetical 1-point font. Naturally Merge does the applicable scaling. Note also that as these are in real-world length units, they are not affected by the DPI of the printer. Merge works it all out.

In JetForm Central .L4 files the character width (cwidth) entries varied by whatever point size the .L4 file was for, and the units (even though they had decimals) were in 'dots' with a presumed printer resolution of 300 dpi. Converting a .L4 cwidth to DocOrigin .mtx units requires that you divide by the .L4's point size and multiply by (1000000/300).

Output Filenames / Options

Output files are typically specified as part of the Merge command line options using the `-output` option. The name can have various embedded substitution strings to allow the name to be programmatically modified when it is opened.

 It is important to note that the replacement of these various substitution strings happens at the time the file is opened. This is when the actual "Print" phase of Merge processing begins, immediately after the 'Start Next Print Driver' script event. This can be important when using Field substitution which can be modified by script.

Output files can have 4 types of substitution:

- Printer table substitutions (as of 3.1.002.08). See below for details.
- `$X` command substitutions - see [\\$X String Substitutions](#)
- `[fieldname]` which is replaced by the form field's current value.
- `%x` - one of several automatic field values, typically date and time values. See [File Naming Conventions](#) for details.

Output value may be a logical printer (or file) name. Those values are mapped to real paths via key-value file called `Default-PrinterTable.ini`. Override this file with your own mappings. The logical printer name can not have any of the following characters: `:/\,` or `.`

Some examples:

```
-output $U/Output/Myfile.pdf
... output in C:/DocOrigin/User/Output.Myfile.pdf or equivalent

-output $U/Output/[Account]_%Y-%M-%d.pdf
... substitute value of field "Account" and current date
... C:/DocOrigin/User/Output/12345_2015-03-20.pdf

-output PrinterInRoom404
```

Scripted Output Destinations

You can use a script to set the output file as well. This script can be in any of the Merge events up to and including the "Start Next Print Driver" event.

You can access the current setting of the output string using the `_printer.getOutputFile()` call. If you are using multiple outputs you'll need to qualify that call with the printer name such as `_printer.getOutputFile("PDF2")`.

To set a new output destination, use `_printer.setOutputFile("newname")` or the qualified equivalent `_printer.setOutputFile("newname", "PDF2")`. The new output name can have embedded \$X variables, [fieldname] strings, and %x substitution as well, just as the basic command line -output can have. These substitutions happen just as the actual printing process begins.

If you are using the CombineDocuments option that combines all documents within a multi-document Merge run into a single output file, You can also use the `_printer.setOutputFile()` routine to change the output destination part way through that run.

See the [_printer](#) section for information on other printer-related scripting.

Multiple Output Streams

With DocOrigin it is possible to have a single Merge run (perhaps of several documents) send output to more than one output destination. This might be an application where, for instance, a set of customer statements are generated as PDF and are emailed to the client. At the same time, a single combined pdf is created that will be stored as an archival copy. Or one might have an application that creates PDF statements and emails them to customers who have signed up for electronic delivery and sends the others to a PCL printer for mailing.

The [Multiple Outputs](#) section describes this feature in detail.

Output to Another Program

(As of 3.0.004.05)

If the output destination is preceded by "run:", Merge will treat the rest of the output string as a command to be executed once the entire output file has been created. So for example:

```
-output "run:: curl -T %t -u user:password ftp://mywebsite.com/newfile.pdf"
```

The above output definition would invoke the curl.exe program to send the output file (the %t value) via FTP to a website. This could just as easily be the name of an archive program or any other program or command script on your computer. All the normal \$X, [fieldname], and %x substitutions are available if required.

The program that is called should exit with a return code of 0 if it succeeded. All other return values will be construed as a failure and result in an error written to the Merge logfile.

You could accomplish the same thing by directing the output to a temporary file, then using a script in the End-of-Document or End-of-Job event to invoke `_run`, passing it the needed parameters. Using `run:` as the output "file name" is seen as a simpler, more declarative way to accomplish those operations. You can supply any needed credentials for access to other systems and effect dynamic network connections if that is what your output distribution needs call for.

See Also

[Multiple Outputs](#)
[File Naming Conventions](#)
[_printer](#)

Merge Filters

Merge is necessarily given data to work with — via its `-data` option.

However, that data as originally provided may not be completely suitable for Merge to use with the given form. In fact, it might not even be in XML format.

Merge has the concept of data filters. You can specify a data filter, or a chain of them, to massage the original data stream into something that is more suited to later processing.

The Merge `-filter` command line option is used to indicate the need to convert incoming data to a Merge-compatible XML data stream. The `-filter` option specifies a *Filter Program* — either an executable program or a JavaScript script. DocOrigin includes some standard filters, and is architected so that you may provide additional filters. The filter program is passed the input data file and must return a correctly formatted XML data file. This returned XML file is what is then processed by Merge.

Other sections describe some pre-built filter programs that are part of the DocOrigin installation and later describe how to write and test a JavaScript filter. You can write your own executable filters, if you like, too.

Filter options

When using the `-filter` option, three other options may also be specified on the Merge command line:

- `-filterParm` — allows additional options to be passed to the filter program.

With the advent of multiple filters in one Merge run, these parameters get passed to each filter. No problem, as filters ignore parameters they don't understand. This option is **deprecated** in favour of providing the options as defined in `-filter`.

- `-filterOutput` — will create a copy of the output XML data in a location of your choosing. That is if you want to refer to it later or are debugging. With the advent of multiple filters in one Merge run, by end of job, it will be the last filter's output that will be in this `-filterOutput` specified location.
- `-trace` *tracefile* — writes additional debug information to the file.

All pre-built converters use the standard DocOrigin command line syntax — see [Command Line Processing](#).

Command options are typically stored in a parameter file which is passed to Merge.

All standard DocOrigin command options also are available — see [Common Command Line Options](#).

See also

[-filter](#)

Standalone usage

Note that the filter programs can also be run standalone, independent of Merge. This can be useful for testing or in Job Processing scripts where perhaps Merge is not even involved or the structure of your script leads you to want to do this filtering independently of Merge.


When used in a standalone fashion the filter programs must be passed the `-in` and `-out` parameters explicitly. In a Merge filter context those parameters are supplied automatically.

Executable (non-script) filters

- [Convert JetForm data to XML](#)
- [Older deprecated conversion](#)
- [Sort XML data on multiple keys](#)
- [IBM ICU codepage conversion filter](#)
- [XML Include filters](#)
- Filters that you write

ConvertDatToXml Filter

This is the preferred filter to use to convert Field Nominated Format (FNF) into XML. It is a binary executable, not a script, so it performs faster. Unlike predecessors, it handles multiple documents in one data file. It uses the target form name as input as well and so does a much better job of matching up the case of names used in the FNF file with those used in the form. Internally, it actually creates a document DOM and then dumps that out as an XML file. The result is that the XML file is perfectly tailored for DocOrigin use with the form being used.

 Conversion of .dat files is very common. To that end, if the data file has a .dat, or a .fnf extension then, if no filters are specified, this ConvertDatToXml filter will be applied automatically.

The default options will be used. To override any ConvertDatToXml Filter options:

- Copy the Default-ConvertDatToXml.prm file from \$R\DO\Bin (\$R is the root folder of your DocOrigin installation)
- Paste the file into your \$U\Overrides folder
- Remove the Read Only attribute from the file properties
- Remove the "Default-" file name prefix
- Add any options from the list below.

^Global Processing

^global fields in FNF files are not the same as fields marked "global" in DocOrigin Design. To instantiate ^global fields, see the "-substitute" parameter below.

Additional parameters:

- -in (required)

This is used to provide the name of the input .DAT file. This argument is common to all DocOrigin filter processes. Note that it is supplied automatically when this app is used as a Merge filter. However, if this app is run standalone, the `-in` parameter must be specified.

- -out (required)

This is used to provide the name of the output XML file. This argument is common to all DocOrigin filter processes. Note that it is supplied automatically when this app is used as a Merge filter. However, if this app is run standalone, the `-out` parameter must be specified.

- -form (required)

Used to provide the name of the DocOrigin form file(s) to be used for field name searching. If more than one form is required, the names of the forms should be separated with `;-` as they are when invoking the 'DocOriginCombineForms' executable.

- -formlist (optional)

Used to provide the name of a file into which is written the list of form files referenced in ^form lines within the .DAT file. This file is typically a temporary file whose content is used to re-establish the list of forms for Merge to use when it's actually merging the resulting XML and the forms.

Usage of this option is a little tricky. Please refer to the command line option [-useFilterFormList](#)

- -oneChild *paneName* (optional)

This is used, somewhat rarely, as a means to put into effect the notion of letting the data stream control the order of the panes rather than having the form design specify the order of the panes. See the "Data Order" discussion at the end of this list of options.

- -prefix (optional)

This is used to specify a character other than ^ as the prefix for hat command lines in the .DAT file. *At JetForm, the character was always called 'hat' and not 'caret'.*

- `-newline` (optional) (*as of 3.1.001.13*)

This is used to specify a different alternate newline character. By default, `~` is the alternate newline character, but you can change that to some other character. If you use `-newline` but do not provide a character to use, then the alternate newline character processing will be turned off, i.e. it will not look for `~`.

- `-rename` (optional)

Includes two strings separated by `:` used to populate a table of rename pairs.

Example: `-rename InvoiceNo:InvoiceNumber` - this indicates to `ConvertDatToXml` that it is to replace `InvoiceNo` with `InvoiceNumber` wherever it encounters it used as a field name in a `^field` or `^global` line in the `.DAT` file.

Note that there can be any number of `-rename` arguments on the command line. Well, more likely in a `.prm` file than right on the keyed in command line.

- `-split` *splitText* (optional)

This provides a string that is to be processed as a document separator whenever it is found in the `.DAT` file.

This argument is optional. However, if you do not provide any `-split` argument the entire `.DAT` file will generate a single document in the XML file. That might be what the user wants but more likely it is not.

A popular string to provide is: `^page 1`. That is often used in `.DAT` files to indicate the end of one and the beginning of another document's data.

Example: `-split "^field AccountNumber"` indicates that whenever a line in the `.DAT` file that contains only the string `^field AccountNumber` is encountered a new document should be started in the output XML file. When the string to search for includes spaces, it's necessary to put quotes around the string so that command processing will not treat it as multiple separate command line arguments.

`ConvertDatToXml` always removes leading and trailing blanks and line-ending characters from the line before it checks for a match.

The *splitText* string may contain the special wildcard characters `*` and `?`. In that case, the comparison treats those characters in the same way as Unix and DOS do for file name searching. i.e. `*` matches any number of characters (including no characters) and `?` matches any single character. If you are actually searching for a string that includes a `?` or a `*` you need to precede that character with a backslash in the string. If you are searching for a backslash you need to include two backslashes together in your string.

The string comparison, when checking the line from the `.DAT` file, is case-insensitive.

Note that there can be any number of `-split` arguments on the command line.

- `-substitute` *oldCommandText:newCommandText* (optional) (*As of 3.0.003.11*)

If a `-substitute` specification is given, a command line of the data file can be changed before it is further processed. An example of usage would be:

```
-substitute "^global JF123:^field JF123"
```

Such a change would mean that the field would be treated as a field and not just the resetting of the current value of a global. Thus it could cause a new instance of a pane to be emitted, whereas a mere change in a global value would never do that.

Another example would be:

```
-substitute "^page 2:^subform TsAndCs"
```

This can make it easier to correlate old `.dat` file page numbers with pane names.

The *oldCommandText* is case-insensitive (since JetForm data files are case-insensitive).

It need not be the entire command line; using just a substring of it is permitted. No wildcard characters are supported. This is useful for calls such as `^group` or `^subform`, to substitute the command with a `*field`. Then play a hidden field of the same name in the pane to instantiate the pane.

Note that there can be any number of `-substitute` arguments on the command line. If there are more than one `-substitute` specification, all instances of the first substitution will be done on the `.dat` command line being processed before the next `-substitute` specification is done.

- `-symbolset codepage` (optional)

If your input data is in some codepage other than utf-8 then you should cause it to be converted to utf-8 by indicating which codepage it is in. To see a list of supported codepages run `uconv -l` in the DO/Bin folder of your installation. See also [Codepage Handling](#).

- `-allowRTF Y|N` (optional)

Enable or disable conversion of fields' data to rtf globally for the `ConvertDatToXml` run. By default, this conversion is enabled. Note that it's possible to handle some specific fields that are expected to have rtf content later in Merge using `_inlineToRtf (Central "\x" to RTF)` script function which converts Central inline text commands to rtf.

- `-trim` (optional)

This indicates whether to trim trailing blanks from the value string before storing it for a field. The valid settings are: `left`, `start`, `right`, `end`, `both` and `none`.

- `left` and `start` indicate to trim blanks on only the left end of the string.
- `right` and `end` indicate to trim blanks on only the right end of the string.
- `both` indicate to trim blanks on the left and the right end of the string
- `none` indicates to do no trimming of blanks. `none` is the default setting for `-trim`.

- `-verbose 100` (optional)

If this is provided to the filter, as opposed to Merge itself, then the filter will report which fields were dropped, because of no name match, as it does the conversion.

Data Order

In DocOrigin the idea is that the data stream provides data and the form design provides the layout, including the order of the dynamically instantiated panes. Adobe® (JetForm) Central users are used to the data stream taking full control. Often those data streams were constructed to emit subform X, then subform Y, then subform A, etc. In any order. The form design had no say whatever. `ConvertDatToXml`, via its `-oneChild` option, provides a means to cope with such data streams.

The Central form designs essentially had a sea of unordered subforms. The data stream could reference any of those subforms in any order. Essentially the data stream "drew" the document by continually saying what should come out next. To support this concept in DocOrigin you take all of those unordered subforms, now converted to panes, and place them all inside a parent of a chosen name. (It must be named). For description purposes let's say the chosen name of the parent pane was `DataOrder` -- but you can pick any name.

So you now have a parent pane (`DataOrder`) and a whole bunch of loose, unordered, child panes. The parent pane must be marked as `Allow Multiple`, and probably `Allow Split`. The child panes must always be marked as not `Mandatory`. They may or may not be marked as `Allow Split`. So the structure is like this:

```
DataOrder (Allow Multiple)
  Pane1 (all of these are marked optional)
  Pane5
  Pane3
  PaneX
  PaneQ
  ...
```

When you run `ConvertDatToXml` you would use:

```
-filter 'ConvertDatToXml -oneChild DataOrder'
```

That will very much affect the structure of the XML file that `ConvertDatToXml` produces. It will never allow the chosen pane (`DataOrder` in our example) to have more than one child. So, if the data stream had subforms in the order `PaneX`, `Pane5`, `Pane3`, what would come out in the XML is a structure as follows:

```
<DataOrder>
  <PaneX> ... fields for this pane</PaneX>
</DataOrder>
<DataOrder>
  <Pane5> ... fields for this pane</Pane5>
</DataOrder>
<DataOrder>
  <Pane3> ... fields for this pane</Pane3>
</DataOrder>
...
```

That will cause Merge to create new instances of parent pane `DataOrder` as needed and each time create an instance of a single child pane. In that way, the document output order is not controlled by the form design, but by the data stream.

- ① Of course, we think that the form design should control the layout and output order and the data should be mere data, however, you must deal with the data you are given.

ConvertTxtToXml Filter

(as of version 3.1.002.06)

This is a filter to convert a flat text file (typically a report) to XML. To do this, you must design a .xfilter file using the DocOrigin FilterEditor task. This xfilter is applied to the incoming text file to convert it to an XML file. Internally this program uses the Merge task to process the conversion.

Additional parameters:

- -in (required)

This is used to provide the name of the input .txt file. This argument is common to all DocOrigin filter processes. Note that it is supplied automatically when this app is used as a Merge filter. However, if this app is run standalone, the -in parameter must be specified.

- -out (required)

This is used to provide the name of the output XML file. This argument is common to all DocOrigin filter processes. Note that it is supplied automatically when this app is used as a Merge filter. However, if this app is run standalone, the -out parameter must be specified.

- -xfilter (required)

Used to describe how to find and extract the actual data from the text file. The .xfilter file is created using the DocOrigin FilterEditor program.

- -logfile (optional)

A standard DocOrigin logfile. If not supplied, a default logfile name is used.

Dat2XML Filter

Dat2XML is an executable (not JavaScript) filter which will convert basic Field Nominated Format (FNF) into XML.

NOTE: Dat2Xml is **DEPRECATED**. It is superseded by the far more functional ConvertDatToXml executable.

Dat2xml presumes that the input data file is for only one document.

Dat2XML handles the following FNF commands:

- ^Field **fieldname** - the name of a field on the form.
- ^Global **fieldname** - interpreted as a ^Field. Set global flags in DocOrigin Design to make any field global.
- ^Graph **fieldname** Dat2XML makes the filename the data for the current field.
- ^Form **formname** - adds the form to a list of DocOrigin forms that will be combined into a single form within Merge. Equivalent to `-form "formname1;formname2; ..."` Merge syntax.

Additional parameters:

- **-rename from:to** — This directs Dat2XML to change the field name called 'from' to a new name called 'to'. This allows data to be renamed to match fields in an existing form. You may include multiple -rename commands in the command file.
- **-rtrim Y/N** — `-rtrim Y` tells Dat2XML to remove all trailing blanks from the end of the data.

DocumentSort

This filter allows you to sort your input XML file based on one or more keys. It is quite a fast sort at that! If running the filter standalone it takes the usual `-in` and `-out` parameters. When run as a filter via Merge, those parameters are provided automatically. The input will be either the `-data` file or the output of the previous filter. The output file will be passed on to the next filter, or if none, to Merge.

-keyName

`-keyName=xmlTagName`

Define which XML fields (tag names) to sort on. There may be one or more.

Sort key qualifiers

There are a number of qualifiers that can be applied to each sort key. They must follow the sort key to which they apply. These qualifiers are:

Qualifier	Value	Meaning
<code>-sortOrder</code>	ascending or descending	how key values are sorted
<code>-caseSensitive</code>	*see note	Treat keys with the different case as different
<code>-numeric</code>	*see note	Treat this key as a numeric. E.g. 2 should come after 10.
<code>-rightJustify</code>	*see note	For sort purposes, pad this key on the left with blanks until it is the width of the maximum width key for this sort key

*These are all Boolean settings so, if you are keen on such things, you can add an `=Y`, `=N`, `=T`, `=F`, `=0` or `=1` to each of these. The default is `=Y`. So the setting itself says that you want that action to happen. And the absence of the setting means that you don't want that to happen.

 `-numeric` and `-rightJustify` were introduced in 3.0.002.03.

Filter syntax

Do recall that as a Merge filter, it is the entire set of parameters for a filter that is simply one large `-filter` parameter to Merge. So if you are running this as a Merge filter, as opposed to standalone, you would say, preferably in a `.prm` file, not keyed in by hand on the command line (and notice that the whole set of parameters is within quotes):

```
-filter "DocumentSort -keyName=SortTagA -numeric -keyName=SortTag2 -sortOrder=descending"
```

A powerful usage

By doing a fast sort on, say, Branch, or Region, it is very easy to get all the documents out for a Branch | Region together. If you are producing PDF output, using PDFCombineDocuments Yes, you can easily put all of the documents for a Branch | Region into one PDF for easy dissemination.

See Also

[Multiple Outputs](#)
[Printer options](#)

JavaScript Filters

- [Script-based JetForm data to XML](#) You may use as is, or copy and edit
- [Script based comma separated to XML](#) Use as is, or copy and edit.
- [xfilter scripts via FilterEditor](#) See [Filter Editor](#).
- [XMLAttrs Filter](#) Expose XML attributes as tagged data
- Any filters that you write – they are easy.

Writing filter scripts

Using the JavaScript language you can code your own custom filter to convert data into XML. The `-filter` command must specify a JavaScript file (typically with a `.wjs` file extension). You must specify the script file (`.wjs`) using the `-filter` command as in:

```
... -filter "C:/DocOrigin/User/Scripts/MyFilter.wjs ..."
```

Script Filter Options

When Merge runs a script-based filter it supplies a `-in` option and a `-out` option. If multiple filters have been defined, Merge takes care of passing the 'out' of one filter to the 'in' of the next filter. The first filter normally gets the data file as its 'in'. Even there you can't be sure that Merge didn't have to do some of its own earlier processing which resulted in a temp file. So use the `-in` option as the safer bet.

When run as a filter, the `-in` and `-out` options are NOT found in `_job.command.in` and `_job.command.out`. Instead, they are found in `_job.filter.in` and `_job.filter.out`.

As it happens, one doesn't usually make use of the `-out` option because you are more likely to use the `_xml` object that is also provided to a filter when it is run by Merge. The `_xml` object makes it easier for you to generate XML, and in general, that is the target that all filters are aiming for.

Standalone usage of script filters

It's great that Merge can run filters. It is also useful to be able to run them independently of Merge. Some effort is required to support both usages as a Merge filter and usage as a standalone script. The following script snippet will likely show you the score.

```
var dataFile;
if (typeof _job.filter != "undefined")
    dataFile = _job.filter.in;           // Presume use as a Merge filter
if (typeof dataFile == "undefined")
    dataFile = _job.command.in;        // Fallback for standalone usage
if (typeof dataFile == "undefined")
    dataFile = _job.datafile;          // Well, lazy standalone usage
```

Similarly, one may wish to know the `-out` option that is in force, if for no other reason than to use it in some log messages that you may issue.

```
var outFile;
if (typeof _job.filter != "undefined")
    outFile = _job.filter.out;         // Presume use as a Merge filter
if (typeof outFile == "undefined")
    outFile = _job.command.out;       // Fallback for standalone usage
if (typeof _xml != "undefined")
    outFile = _xml.getFileName();      // A defined _xml is definite 'source of
truth'.
if (typeof outFile == "undefined")
    outFile = _file.getFileNamePart(dataFile, "NoExt") + ".xml"; // A possible default
(you choose)
```

Within a filter, there is a pre-defined object called `_xml` which is used to specify the xml output of the filter. See the `XmlFile` Class for details on the generation of xml output. However, for standalone usage, you will need to create that `_xml` object yourself. As in...

```
if (typeof _xml == "undefined")
    _xml = new XmlFile(outFile, true);    // see above for outFile computation
```

With those code snippets in place you can happily run / test your filter in standalone mode:

```
RunScript -script MyFilter.wjs -in ....
```

Processing within a script

All of the standard DocOrigin scripting extensions listed in the **Scripting** section are available when running a Merge filter script.

A classic filter processing loop would be:

```
var fp = _file.fopen(dataFile, "r");
while (true) {
    line = fp.fgets();
    if (line == null) break;
    ... work out a name / value pair
    _xml[name] = value;
}
fp fclose();
_xml.close();
```

A script Filter can also reset the name of the form that is being processed. Use the

```
_setForm("newFormName");
```

script function to reset the form to be used.

The script's return value

1 (true) means success, 0 (false) is failure

The Filter script must indicate whether the conversion was successful or not by returning `true` if it succeeded or `false` if the file cannot be converted. Use the JavaScript `return true;` or `return false;` statement to return this value.

In most places a `return 0;` is good, **BUT NOT HERE**. For script-based filters, you must `return 1;` or `return true;` to indicate success.

DatToXml Filter (Script-based)

For complete control over the data conversion process, there is no better method than using a script-based solution. Fortunately, you do not need to start from scratch. A `DatToXml.wjs` is included with DocOrigin. It provides a strong basis for converting `.DAT` to `.XML`. In fact, it can often be used as is.

However, if that is the case then please use `ConvertDatToXml` instead.

But you may want to code in some special processing for your situation. In that case, take a copy of the supplied `DatToXml.wjs` and modify it, likely only slightly, to get the result you want. There is additional documentation in the comments at the bottom of the supplied script.

BTW, if you believe the processing option you need is or would be common to your JetForm Central brethren, please do let us know so we can consider it as an additional feature of the `ConvertDatToXml` default filter.

CsvToXml Filter (Script-based)

Comma separated values (or tab or space or whatever delimiter) to .XML. The **CsvToXml.wjs** script is included in the installation. It has a whole slew of parameters which are described in comments at the bottom of the file.

Often this script can be used as is. In fact if a data file has a .csv extension and no filter is specified, then this filter will be applied automatically.

However, if you have some special processing needs, it's probably best to take a copy of this script and make some code edits.

XMLAttrs Filter (script-based)

XML Attribute Processing

DocOrigin does not even consider attributes in the XML it is given as data. That doesn't seem to be a hardship to anyone producing the XML for DocOrigin consumption. The idea is always to keep it simple and ignoring attributes does help do that. However, sometimes one is faced with XML that is already being produced and that does have attributes. What to do then?

Well, one could drag out your XSLT knowledge and several Aspirin and go for it. Yeah, right!

DocOrigin supports the concept of filters. You can specify a `-filter filtername` option on the Merge command line and Merge will run the filter on the fly, converting your data stream in whatever way your filter does, before handing it off for form + data merging. Very handy. Those filters can be written in whatever language and supplied as an executable or written in DocOrigin JavaScript (.wjs) and executed that way.

You are not on your own, in fact it is my fond hope that folks will begin to share the filters that they write. Eclipse Corporation is supplying an **XmlAttrs.wjs** filter that works for us. It's a simple concept. As it filters the input XML file full of attributes, it outputs a new XML file in which every attribute has been turned into a child element of the element that had attributes. For example:

```
<Customer name="Acme Corp">
  ...
```

becomes

```
<Customer>
  <name>Acme Corp</name>
  ...
```

Hence, all of the attributes are now exposed to DocOrigin processing.

All you have to do is specify `-filter $S/XmlAttrs.wjs` as a command line option to Merge (probably buried, in the usual way, in the .prm file for the job). That's it. Now your form has access to those attributes. [\$S is defined in **Default-Paths.prm** as install **dir/Merge/Scripts.**]

At the moment the script does not prettily indent the output XML. Want to update it and share?

Interracial marriage is great, but mixed content in XML is a dumb idea. This script will throw a hissy fit, I imagine, if you supply XML that is not well-formed, or XML that has mixed content -- that is where an element has both child elements and text content. Don't do that!

So what happens when you have:

```
<Customer sla="premium">Acme Corp</Customer>
```

Won't that create the abhorred 'mixed content'? As in:

```
<Customer>
  <sla>premium</sla>
  Acme Corp           <!-- Aack, mixed content -->
</Customer>
```

That would be unacceptable (imo). We have to invent a tag to enclose Acme Corp. Choices abound. For better or worse (Leave a Comment) I have chosen to enclose it with the original tag, so...

```
<Customer>
  <sla>premium</sla>
  <Customer>Acme Corp</Customer>
</Customer>
```

And so, yes, you do end up with Customer.Customer. And your choice would be ...?

Well, you have your choice. You can choose to supply a `-contentTag=aTag` command line option. For example, you might choose `-contentTag=_content_`. As another trick or treat we consider the `!` character to be special. It is interpreted to mean 'whatever' the original element name was. So `-contentTag=_!` would mean `_Customer` in the examples used above. The default `-contentTag` is `!`.

BTW, the script recognizes and faithfully copies: XML comments, XML processing instructions, and CDATA.

Let one of your attributes be *enjoyment* (and another be *content*).


Codepage Handling

Filtering is great but it is a darn sight more effective if you are working in the right character set encoding.

The right character set encoding is utf-8.

If your input data is in utf-8, you're golden, you can skip this topic entirely. If not, then your task is how to get your input data into utf-8, so it can be filtered and processed properly.

If you are running Merge then the simple answer is to use the `-symbolSet` option. When that is present, Merge will automatically use `uconv` to convert your data from the nominated code page to utf-8. In essence, it is the first filter run and allows subsequent filters, and Merge itself, to expect utf-8 based input.

 `-symbolSet` (as of 3.0.002.03)

Before then one would've used `uconv` ahead of time to convert the data given to Merge. Or if using the **ConvertDatToXml** filter, used the `-symbolSet` parameter that it supports (and still supports).

`uconv` can also be run standalone, say in Job Processing scripts if your code structure makes you want to employ it that way.

uconv

Included with DocOrigin is the IBM ICU internationalization library and tools. It is used throughout DocOrigin code. The ICU suite is rightfully a highly regarded suite.

In the `.../DO/Bin` folder you will find `uconv.exe`, (well no `.exe` on unix platforms). You can run:

```
uconv --help
```

and get blown away by all the options. It is impressive. Thank you IBM.

uconv encodings

For more fun, try:

```
uconv -l (that's a lower case L)
```

Reams of codepage identifiers will spew out. I bet your source data file is encoded in one of those codepages. Once you've located the encoding that applies to your data, you are away to the races.

uconv usage syntax

The standalone usage of `uconv` is:

```
uconv -f sourceEncoding -t utf-8 -o outputFile inputFile
```

Where...

- f stands for from
- t stands for to
- o specifies the output file name
- You need to determine your data file's encoding and use it as the *sourceEncoding*.
- The target encoding is always `utf-8`.
- You can nominate any *outputFile* you like.
- And of course, you need to tell it where your *inputFile* is.

Try it! Seriously, try it ... standalone. When you can open the result with a tool that handles UTF-8 text, and you see what you want to see, then you are set -- and not before then. Find the right encoding and try it out.

as a Filter

uconv can be used directly as a filter with the following construct

```
$E/uconv -f sourceEncoding -t utf-8 -o %out %in
```

%in and %out will be automatically supplied by Merge so as to facilitate the chaining of several filters. If %in is used in the first `-filter` option it will be replaced with the original data file name. For subsequent `-filter` options %in will be replaced by the output file name of the previous filter. %out is supplied as an automatically generated temporary file name. The %out of the last `-filter` is what is supplied for Merge to do the data + form merge operation.

See Also

[-symbolSet](#)

Multiple Filters Per Run

A single run of Merge can invoke multiple filters, chained as it were. For example, you might use a `.xfilter` (from FilterEditor) to convert a spool file to XML, then use the DocumentSort filter to sort that XML by whichever keys you like.

Maybe you have custom code already that can be used to convert to `.dat` format, but then want to re-filter that to produce XML.

If you are using multiple filters then your Merge command line must be getting rather long. You should be using `.prm` parameter files, in which case each new filter can be readably entered on a new line. E.g. Syntactically

```
-filter "filter1 -arg11=value11 -arg12=value12 ..."  
-filter "filter2 -arg21=value21 -arg22=value22 ..."
```

within the double quotes, you can use single quotes to deal with spaces in arg names or values.

You will note that each `-filter` option takes one long'ish quoted string. That is interpreted as a single argument to Merge, but its parts are treated as individual arguments to the filter in question.

The first "sub-argument" is the name of the filter to use; e.g. `ConvertDatToXml`, `Delim2Xml`, `myFilter.wjs`, `flatToXml.xfilter`, `myBinaryFilter.exe`.

The remaining sub-arguments are whatever that chosen filter needs. For the `DocOrigin` ones, that will certainly include a `-in` and `-out` pair of parameter names and values. Other options may be desired as well per the foregoing documentation. Note that `-in` and `-out` are supplied automatically by Merge.

You can use a space instead of the `=` sign. Avoiding the `=` helps with auto-completion of file names when typing straight into a command shell.

Merge External PDFs

(As of 3.2.001.01)

It is common for documents to have some fixed pages for things like Terms and Conditions and Standard Policies. As these types of documents are often legally binding, it is imperative that they remain exactly as authored. So, to ensure precise replication, DocOrigin uses "stitching" to support incorporating external PDF pages in your documents. Using stitching can reduce form development and form maintenance costs.

❗ For PDF Stitching, Java 8 JRE must be installed.

❗ File Creator Limitation

PDFs can be created by different programs. If you encounter a PDF that is failing, please send it to support@eclipsecorp.us for troubleshooting.

❗ Windows PRT Limitation

Stitching does not work with the Windows "prt:." scenario because it is a post-processing action that can't be done on the original data (which was already sent to the printer.) If you want post-processing to happen, print to a file and then send that file to the printer.

❗ PDF/UA Limitation

Pages that are included in the output using the stitching method will not be "Read Aloud" in a PDF Viewer. Therefore, the stitching feature should not be used for documents that are required to be PDF/UA compliant.

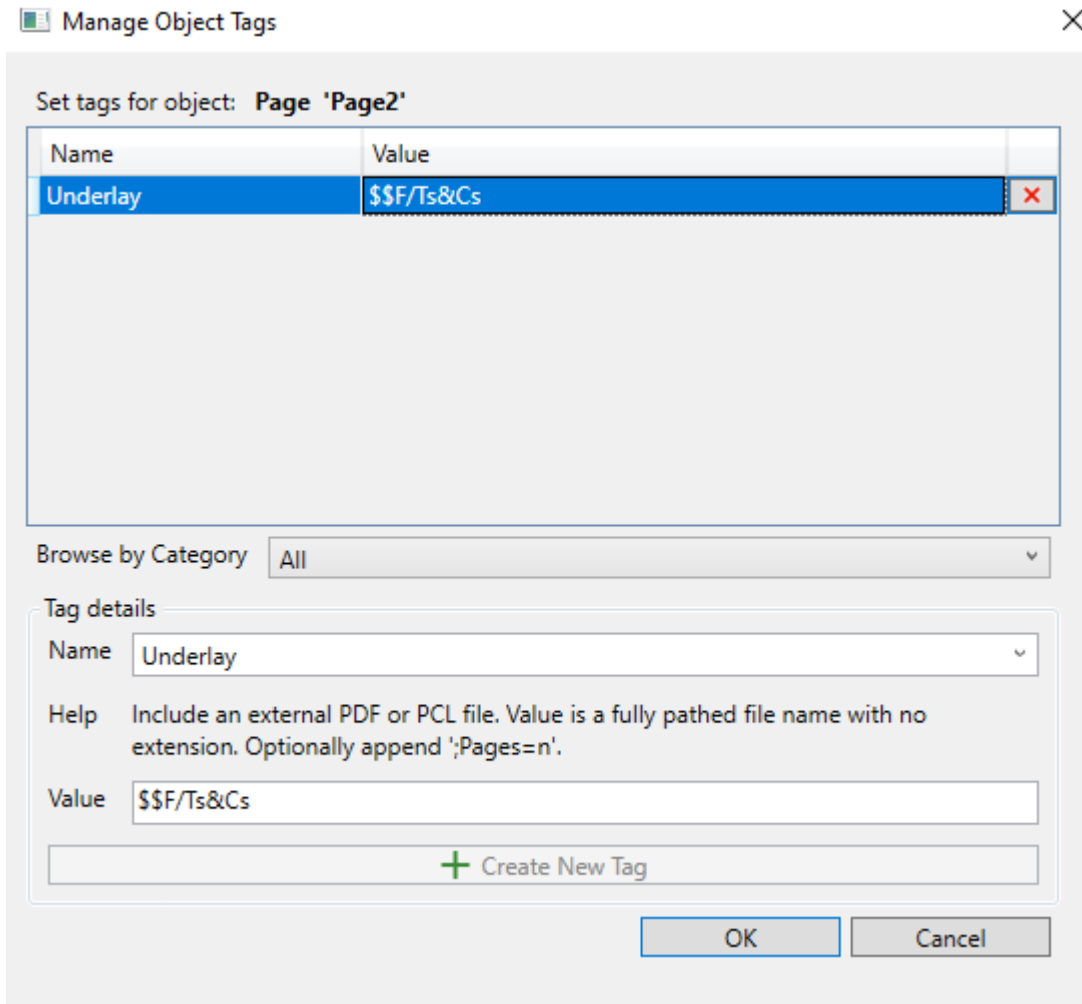
❗ Filename Limitation

External PDF filenames containing some special characters may fail.

What do you do to achieve external PDF integration?

You have a normal DocOrigin form design file (.xatw) and an external PDF, let's say a Ts&Cs.pdf. Your task is to integrate those two source materials into a single output PDF.

To define a place for the external page in the XATW, add a new Page to your form design (often called a "placeholder" page) and add the tag "**Underlay**". The value is the fully pathed filename of the external PDF **with no extension**.



This will stitch all pages from your external PDF. You can use specific pages of your external document by appending `;Pages=n`. `n` may be in form of a single page number, comma-separated list of individual numbers, dash-separated range or a mix of all of them, or an asterisk to indicate all pages.

So in our example case, it could be something like: `$$F/Ts&Cs;Pages=1-3`

Wherever the placeholder page ends up in the output document is where the identified page(s) from the identified external PDF are integrated. The "integration" is such that the external page "underlays" the Merge-generated placeholder page, which might not be blank. E.g. it would be possible to "stamp" a policy number on the integrated external pages or put your own page numbers over Ts&Cs's ones overlaying them with say white rectangle and your own page numbers.

If you are so inclined, you could set these Underlay tag values dynamically, with script. Indeed you can `insertPage`, or clone placeholder pages dynamically as well.

For example:

```
this.setTag("Underlay", "$U/Stitch/PolicyRider23");
```

That is all you need to make it work.

i You can skip the information below unless you are interested in implementation details or need some customization.

What Merge does

Merge will render the document, including the placeholder pages.

Behind the scenes, it will create a temporary trivial control file which identifies the physical page number of the placeholder page that Merge generated plus the name of the external PDF that is providing the page to place underneath the Merge-generated page. And of course the page number from that external PDF. Example:

```
9;"C:/DocOrigin/User/Stitch/CaliforniaRegulations.pdf";1
10;"C:/DocOrigin/User/Stitch/PurchaseTs&Cs.pdf";5
```

Note that the page numbers are always physical page numbers, not necessarily the same as the page number printed on the page.

That control file is passed to a supplied but overridable script named **Default-DocOriginOverlay.wjs**. The supplied script uses the aforementioned third-party technology to do the PDF file integration such that a single integrated PDF results.

The DocOriginOverlay.wjs script

The purpose of this script is to integrate traditional Merge and PDF overlaying tool.


The Default version of this script resides as **\$E/Default-DocOriginOverlay.wjs**. You may override it in **\$O/DocOriginOverlay.wjs**.

The script then goes on to log certain information that may be of interest and it uses the PDFBox facilities to affect the integration of pages per the control file.

The result is written over top of the original Merge-produced PDF document such that Merge is unaware of any changes. Merge carries on as it always has in terms of email production, document counting, etc.

Multiple documents per Merge run

The processing handles multiple documents per Merge run. That is, in either case, `PDFCombineDocuments = Yes` or `No`.

 Warning: The processing entails quite a lot of byte movement and hence dealing with large, multi-document files, the processing time for the documents may be significantly longer than in documents that are not using this new feature.

Merge External PCLs

(As of 3.2.001.06)

- ✓ You can use Eclipse Form Conversion Service (EFCS) to get PCL5S files instead of doing it locally with the PCLExtract tool.

It is very common for business transaction documents to have not only the usual dynamic header, details, and trailer information, but also to have some static content pages for Terms and Conditions or Standard Policies, or similar. As that content is quite possibly part of a legally binding document it is fervently desired that they remain exactly as authored.

DocOrigin supports the direct integration of external PCL pages, without any need for conversion to a DocOrigin form design file. However, for DocOrigin Merge to integrate external PCL files they must be split into separate one-page PCL files and "stripped" with the DocOrigin PCLExtract tool. The resultant PCL5-stripped (PCL5S) single-page files must be named according to a simple rule so that DocOrigin can pick them up automatically.

Achieving External PCL File Integration

Consider: You have a normal DocOrigin form design file (.xatw) and an external document, let's say a Ts&Cs.docx. First, you must print that document into single-page PCL files (see more about that below). Let's say that you get two single-page files out of it - Ts&Cs.1.pcl and Ts&Cs.2.pcl. Second, you must "strip" those files with the PCLExtract tool to get PCL5S files out of it - Ts&Cs.1.pcl5s and Ts&Cs.2.pcl5s. Those files must be placed in a single folder. They must have a .pcl5s extension and have the same base name (Ts&Cs in our example). It will be common practice to put all external files to be merged into documents into a single folder. Perhaps \$U/Inserts -- your choice. The task is to integrate the Merge produced output and those PCL5S files into a single output PCL file.

- ⓘ Even if your external file is only a single page, it still needs to be preprocessed to produce a version that is suitable for dynamic inclusion in a merged document. See more about that below. It is expected that the files to be inserted are quite static -- changing rarely. The preprocessing needs to be done only once, NOT on every run of Merge. It is expected that you would keep the preprocessed files in a single standard folder of your choice.

At the appropriate place (or places) in your form design, you must define what we term a 'placeholder page'. That is quite likely a blank page (though it need not be totally blank). Each placeholder page represents the place where external PCL page(s) are to be inserted.

Each placeholder page is required to have an object tag to identify which external single-page PCL file(s) should be inserted at that point.

The tag name is: Underlay
The tag value is: The external PCL file name;Pages=n

Notes on the external file name

1. The file name must be fully pathed; DocOrigin folder mappings may be used. e.g. \$\$F, or \$U/Inserts.
2. The file name in the tag value must NOT specify the file extension, .pcl5s is assumed, and added automatically. Hence the files to be inserted must have a .pcl5s extension.
3. The file name must NOT include the page number suffix.

Notes on the ;Pages=*n* specification

1. Most commonly you would leave that specification out and get all the pages in the referenced external file.
2. You can use the ;Pages= element to reference specific pages. The ;Pages= value can take the form of: a single page number, a comma separated list of individual numbers, a dash separated range of pages, or a comma-separated mixture of those forms. You may also use asterisk (*) as an explicit indication that you want all pages.

In our example, it could be something like: `$$F/Ts&Cs`. Merge will look for its pages, i.e. `Ts&Cs.1.pcl5s`, `Ts&Cs.2.pcl5s`, etc.

Having defined the placeholder pages with their `Underlay` tags -- what happens? The normal pagination process will happen and your placeholder page(s) will end up as some physical page(s) in your document. Placement depends on how many instances of panes were used to contain all of the data in your data stream. It's dynamic. At the end of this pagination operation, Merge will detect the placeholder pages (because they have an `Underlay` tag) and proceed to do the external file integration. Based on the tag value and the number of relevant external files that it finds, Merge will, at least conceptually, clone additional placeholder pages so as to contain all the referenced external pages. The "integration" is such that the external page is placed first and hence ends up "beneath" (i.e. underlays) the Merge-generated placeholder page, which overlays the external page. (The placeholder page might not be blank: e.g., it would be possible to "stamp" a policy number on the integrated external pages or to put your own page numbers over Ts&Cs's ones overlaying them with an opaque white rectangle and then your own page numbers.)

If you are so inclined, you could set these `Underlay` tag values dynamically, with script. Indeed, you can `insertPage`, or `clone` (placeholder) pages dynamically as well.

For example, the `Underlay` tag value could be set dynamically on a page object with:

```
this.setTag("Underlay", "$U/Stitch/PolicyRiderReFlooding");
```

That tag, and the prepared external files, are all you need to make it work.

Preparing External Files

To function efficiently the external files that you wish to assemble into your documents must be 'prepared' for that operation. These preparations need be done only once to then allow those prepared files to be used in many, many Merge runs.

This is in the context of producing PCL-based documents. For PDF-based document assembly, see [Merge External PDFs](#), these preparations are unnecessary.

The preparations involve a two-step process:

1. Print the external file into single page PCL files
2. "Strip" each single page file of context-altering commands

The Eclipse Form Conversion Service (EFCS) provides the means to quickly accomplish those steps using the EFCS server. However, you may also do the preparations in your own local environment.

Step 0: Get Organized!

You deal with a lot of files. It's sane to organize them into folders rather than have them spread out willy-nilly. We recommend that you create a folder to hold the external files that are targeted to be inserted into documents. Perhaps: `$U/Inserts` or `$U/ExternalDocs` -- please choose a folder and collect your external documents there. As we've just said, these external documents need to have some one-time preparations done with them. You may choose to keep the results of those preparations in the same folder, in a subfolder, or some other folder, but do make that choice and follow an organized plan of file location.

File naming. The naming convention that you **MUST** follow is that given external file: `ABC.xxx`, the single page PCL5S files must be named `ABC.1.pcl5s`, `ABC.2.pcl5s`, etc. (Not `.01.`, `.02.`, ... but `.1.`, `.2.`, etc.). Remember that your `Underlay` object tag value would stop at `ABC`; it does not include the page number suffix. It is extremely helpful to have the prepared PCL5S files in a consistent location so that you can specify your `Underlay` tag values with confidence.

It may be that you will want to invent a 'folder mapping' and define it in your `$0/Paths.prm` file. E.g. `$W=$U/Inserts` so that your `Underlay` tag values can always use `"$W/..."`. Such folder mappings can be very helpful when moving between Dev, QA, and production.

Printing Files in PCL5 Format

The external files need to be printed in PCL5 format, not PCL6. To do that you need to install a virtual PCL5 printer and print your file (docx, pdf, etc.), one page at a time, to that virtual printer. We recommend installing the official HP Universal Print Driver. The last version of that driver that produces pure PCL5 output is version 6.1.0.20062 (upd-pcl5-x64-6.1.0.20062.exe). Do an internet search for that name, download and install it. The driver is quite old and will be found on non-HP sites. DO NOT USE a current PCL6 printer driver. The results are unusable for document assembly purposes.

You should install the PCL5 driver in the usual manner as a regular local printer. Specify settings that target port "FILE (Print to File)". After that you will be able to print as usual, targeting the installed virtual printer and changing print properties such as color, DPI, etc. After each print, the driver will ask you where to store your PCL5 file. Follow the prescribed file naming conventions. Remember to print one page at a time, directing each page to a different file.

Preparing your PCL file for integration

Once you get your one-page PCL file (say Ts&Cs.1.pcl) you have to apply the DocOrigin PCLEExtract tool on it like this:

```
PCLEExtract.exe -strip -in Ts&Cs.1.pcl -out ./stripped/Ts&Cs.1.pcl5s
```

Obviously you must specify the applicable pathing for the files. Do keep in mind the file organization decisions you made in "Step 0".

Shortcut for PDF-based External Files

If your source document is a PDF then DocOrigin can automate this page-by-page printing effort. DocOrigin includes a tool at ...\\DO\\Bin\\Java\\PDFProcessor.jar. For usage information, run

```
java -jar "%DO_ROOT%\DO\Bin\Java\PDFProcessor.jar"
```


One great usage of that tool is to have it list all of your printers. Hopefully, you will see the PCL5-based virtual printer that you installed. Use:

```
java -jar "%DO_ROOT%\DO\Bin\Java\PDFProcessor.jar" -listPrinters
```

For PDF files you can automate page-by-page printing by using the -autoSplit option of PDFProcessor.jar. For example

```
java -jar "%DO_ROOT%\DO\Bin\Java\PDFProcessor.jar" -autoSplit
-printer "HP Universal Printing PCL 5"
-in Ts&Cs.pdf
-out "%DO_ROOT%\User\Inserts"
```

That is a one line command.

 Unfortunately, PDFProcessor.jar is not privy to DocOrigin folder mappings, i.e. it does not understand \$U, \$E, \$F etc. You must specify pathing without the benefit of using those folder mapping \$X names. Given the availability of this shortcut it would not be surprising if you chose, for example, to print your .docx file to PDF format, then use PDFProcessor.jar on the resultant .pdf file to complete the preparations.

The single page PCL files produced in the output folder will be named as *base.n.pcl*. Where base is the base name of the input PDF file. And n is the physical page number from 1 to the last. The extension is always .pcl. These files must still be processed to strip away PCL print environment context altering commands to make them suitable for document assembly. As specified above you would use the PCLEExtract -strip ... tool for that and ensure that the final file names would be *base.n.pcl5s*.

Once you have established your organizational conventions and dev-test-production operational procedures, you will undoubtedly create commands that facilitate these operations.

Limitations

The best results can be achieved when you do not mix document properties. For example, if you want Merge to produce a 600 DPI color PCL document then your external document(s) should be printed using color and 600 DPI. Furthermore, if you design your placeholder page to be Letter and Portrait then your external document should be printed accordingly.

Note, you may choose (experiment?) with adjusting your printer properties as you like. Mixing properties may produce the desired results in some cases, but we do not guarantee consistent results -- after all, they are 'internal constructs unknown' external files.

⚠ Warning: Merging and underlaying PCL is a technically challenging task. To succeed, DocOrigin cannot apply all of its normal PCL optimization techniques. This may result in bigger output files, especially when many external pages are used. You must weigh this cost against the price of repeatedly, as their versions change, converting external documents to design file format and concerns around the accuracy of any such conversions. Why do we insist on single-page PCL files? A multi-page PCL file will use shared resources, fonts, images, etc. It is technically impossible to find and extract these resources. So each page is printed independently and becomes self-sufficient. PCL emulators are often used in testing. These can be great but are not as definitive as actually printing to paper. Indeed there are many models of printer which have their own PCL interpreter code, not always producing the same results. While you may use a PCL emulator initially, a) don't be alarmed if some things are not right, and b) final testing on actual printers is always advised.

See Also

[Merge External PDFs](#)
[domObj.setTag](#)

Merge Scripting

Several of the DocOrigin components (Merge, Control, QueueProcessor etc.) use a common set of JavaScript (Mozilla ECMA 262) conventions and extensions. See section [Scripting](#) for a complete list of script functions and DOM access.

Suggested Scripting Guidelines

1. Whenever possible, scripting should be completed prior to the **Pagination** event.
For most applications, the **Data Merged** event is the most suitable for scripting; at this point, Merge has combined the form and data and performed the word-wrapping, but overflow pages have not yet been established. If a script changes field data after the Pagination event, the field will be re-word-wrapped, but the results cannot affect pagination. For a description of the Merge events, see [Merge Events](#).
2. To simplify object naming and to avoid complications with dynamic tables and panes, the best approach is to associate script with the object it is modifying, or with a "parallel" sibling object or a near parent object.
3. When using the `_document` DOM to access form objects, ensure that the referenced objects have unique names.
4. When Merge splits a table row or a pane between two pages, not all form objects in the Row or Pane may appear on both pages. Fields and text below the split will appear on only the second page; those objects above the split will appear on only the first page. Script on the row-that-references-these-objects will fail in circumstances where Merge has moved the referenced objects. This issue may be resolved in DocOrigin Design by associating script with the form objects.
5. In Design, any table row or pane may be flagged as "Allow Multiple". Within a script, the pane or table row will be treated as an array object, which must be referenced as **Row[2]** or **Pane1[0]**. Even when only one instance of that table row or pane is created, the reference must use a [0] index.

In DocOrigin Design, use the **PDF Preview** or **Merge Test** command on the **Tools** menu to assist in debugging scripts in the form. The commands use Merge to combine the form with test data, and to display the result on-screen or print the output, respectively. For the test data:

1. Click Auto-generate test data, and click Save Data, to create a compatible starting data file. Then edit the data file to test specific conditions that might be encountered.
2. Click Use my test data to provide a test data file created from an application or from an edited auto-generated file.
3. Include [Common Merge Command Line Options](#) which will cause each Merge test run to create a trace file `-trace tracefile.txt`
4. After each test run, review the content of the trace file using a text editor.
5. REPEAT: After each test run, review the content of the trace file using a text editor.

Merge Events

Merge allows for script at each of several key steps or "events" during the processing of a job. The scripts are intended to provide processing to enhance the data presentment. Merge triggers the events both at the job level and during the processing of each document in the data file. For more information, refer to the [The Merge Algorithm](#).

Event scripts are incorporated into the form. Common routines can be stored in a #include file and used by more than one form. Design enables you to associate the script with an event at the object, pane or form level.

NOTE - Several, but not all, Events allow references to the current document's data - for instance by referencing "this._value" in the script. Access to the current document data is possible on only the **DataMerged, PaginationCompleted, ReadyToPrint, StartNextPrintDriver, and EndOfDocument** events. All others will access the Form itself, with undefined field values.

Event	Description
Data Filter Processing	Using the -filter option on the command line it is possible to run JavaScript against the incoming data stream to transform it into a compatible xml data file for merging. See the -filter option in Command Options - Merge for syntax.
Common Code	Common code is a section for reusable scripts shared across a form's objects and events. It allows defining global variables and utility functions, making them accessible throughout the form.
Start of Job	Called once when Merge starts processing the job but before any documents.
Start of each Document	Called at the start of processing of each set of data.
Data Merged	Called once the data and form have been merged together, prior to word-wrapping or pagination.
Pagination Completed	Called at the completion of word-wrapping and pagination. Note that once this event completes, the pagination for the document does not change. If the content is modified, word-wrapping will be corrected, but no attempt will be made to ensure the text still fits in the allowable space. If needed, the function <code>_document.get()</code> can access the page headers and footers that have been inserted by the pagination process.
Ready to Print	Called following the final processing of embedded fields and 'page n/m' fields, and once additional (blank) pages that result from duplexing have been inserted. The document is ready to begin printing.
Start next Print Driver	Merge allows multiple print drivers to be called for each document. This event is triggered prior to each driver being called.
End of Document	Called once all processing of the document is complete, and page and document counts have been updated. Merge is ready to flush this document and begin the next.
Start of Summary	Called when Merge is ready to process a summary page, if required. The script must have used the <code>_summary</code> script functions to generate the summary data file.
End of Job	Called once summary processing is complete.

Event	Description
Finalize	Called once all processing is complete, and Merge is ready to exit. Exists only in Form object.

Hyperlinks

Merge output to PDF or HTML allow the creation of text hyperlinks when the page is displayed.

Any text Label object that has text that starts with `www.` or `http://` or `https://` or `mailto:` is automatically converted to a link in the resulting document.


You can also explicitly add a hyperlink address to any Field on the form. Use the script property `_hyperlink` to attach a hyperlink to a field. For example:

```
Field1._hyperlink ="https://eclipsecorp.us/";
```

This would result in a link to <https://eclipsecorp.us/> whenever Field1's contents are clicked on.

You can select part of the text in a text label and use the **Format > Hyperlink...** menu item to supply the URL to which that text portion hyperlinks.

Fillable Forms

 The Fillable Forms feature is available under only specific licensing arrangements.

DocOrigin enables a number of interactive or "fillable" features to be used with HTML forms. These features allow Merge to generate standard multi-page documents where certain fields are fillable. The document can also contain many interactive controls such as Pushbuttons, Checkboxes, Radiobuttons, and Selection/Choicelist controls. Typically, the document has a "submit" button that posts the collected data to a web server.

Note that DocOrigin does also provide some limited input capability with PDF forms. This includes basic field data capture as well as checkbox handling. At present PDF fillable forms do not support the dynamic addition, removal, hiding and showing of panes (whereas HTML fillable forms do support that). Although the following information is aimed at HTML input forms, the PDF capabilities will be noted when applicable.

This documentation page is about designing a fillable form and then generating that document. There is another part to fillable forms usage; that is the part that is performed outside of DocOrigin, in the applicable viewer technology. Further discussion of that area will be found at [Fillable HTML Forms](#) and [Fillable PDF Forms](#). There is most likely yet another part in the fillable forms puzzle, the part that lives on the web server that is hosting the fillable form. DocOrigin provides some **sample**, unsupported, PHP scripts in the `.../DO/Samples/Web` folder but really the web server portion is left entirely to you.

Single Output File

One of the unique features of DocOrigin Fillable HTML Forms is that all of the components necessary for the html form are output in a single HTML file. Typically, web pages combine a variety of separate server files - style sheets, html, jpg files, etc. With DocOrigin Fillable Forms all these pieces are embedded in a single file. This simplifies deployment and update of the online forms.

Templates

The HTML output structure is defined by a template file. This is a skeleton `.htm` file that defines the structure and base contents of the html file. It has special markers such as **HEADER** in it that are expanded by the Merge driver into the actual contents of the document. Use of a template allows a system integrator greater control over such things as standard headers or footers, Doctype records etc. A standard (default) template is provided in the DocOrigin install in the `.../DO/Bin` folder. A Merge command option of `-template xxx` allows that to be overridden with a custom template if you wish.

HTML Script

DocOrigin uses HTML JavaScript to perform many of the operations. This script gets embedded into the output html file as part of Merge's processing of the **HEADER** and **FOOTER** tags in the template. There is a standard set of DocOrigin supplied script routines that can be accessed in the form design to perform actions such as **submitting** the form data to a server or **inserting** an option detail pane into the document. For convenience, these routine names are all prefaced with "DO." - such as `DO.submitForm()`, `DO.addPane()` etc.

You can also create your own script and add it in a number of ways:

- By defining functions in the Form object's script **Input** event.

In Design go to **File>FormProperties** select the **Script** button then the **Event:Input** option. Any script typed into this section will be inserted into the generated HTML file and can be called by your Input field event processing.

- By creating a custom **Template** and inserting the script into the template's section.

Script is typically stored as JavaScript functions that can be called by the **onclick**, **onblur**, etc. events by setting the **ViewerAttributes** information for a button field. See **Pushbutton Fields** below.

Input Fields

When Fields have the **Input** flag set they are considered to be HTML (or PDF) Input controls or buttons. Input fields then allow the setting of a number of additional settings which control the input process. These are found by clicking on the **Input** button at the lower-left of the Field property page.

All other settings are filled and managed like regular Merge data fields. The input fields are also processed just like all other Merge fields - merging data, affecting layout, etc. prior to being sent to the print driver. In this respect, Input fields are no different than any other Merge Field. It is only at the final print driver stage itself that their special status as Input fields are recognized and acted upon. To set the type of Input field and other options within Design you must click on the **Input** button under the Field's property tab. Note that this button is not activated unless the **Input checkbox** at the top-right has also been checked.

If you select an Input Field (one where you've checked the "Input" checkbox at the top of the Field screen) and you select the **Input** button at the bottom left of the Field Properties screen (or select the **Format > ObjectProperties** menu command) you will see the following screen:

The screenshot shows the 'Common Properties' dialog box with the 'Input' tab selected. The 'Field Name' is 'Field1'. The 'Input Field Type' is set to 'Text'. There are checkboxes for 'Readonly' and 'Show 12Hr (AM/PM) Times'. Below these are several text input fields: 'Required', 'Msg', 'Label', 'Submit Button url', 'Background Image', 'Signature Clear Btn', 'Year Range', 'Date Order', 'Choice', 'Unsupported Msg', 'Tooltip', and 'Viewer Attributes'. There are also checkboxes for 'Short Month' and 'Short Year'. At the bottom are 'OK', 'Cancel', and 'Apply' buttons.

The Field Name field at the top is for information only - you can change the field name elsewhere in the usual manner.

The **Input Field Type** setting indicates how the input is collected. There are several choices. The list of choices is dependent on the field's **Display As** setting on the standard Field screen. For instance, Signatures and uploaded images are only available if the Field Display As setting is "Image". Radio Buttons and Checkboxes work similarly.

The settings that are greyed indicate non-applicable settings for the particular Input Field Type you have selected.

The **Viewer Attributes** section at the bottom of the screen is pass-through information to the browser and is typically used to trigger an action such as the HTML **onclick** event of a Submit button. This is where the **onclick** event would be specified for a submit button for instance.

The various options are listed in more detail below:

Required and Readonly Input

The **Required** and **Readonly** settings are available for most input types. When the **Readonly** option is checked the user is NOT able to change the data. This is roughly equivalent to having placed a similar standard non-input field on the form - it is displayed but not changeable. **Readonly** can be of use when you want the same DocOrigin form to sometimes collect data from the user and other times echo it back to them in the same format. Although the data is not changeable by the user, it is passed through to the web server as it would normally be.

The **Required** setting indicates that some value must be entered in a field. When a form is about to be submitted to a web server, any **Required** fields are highlighted to the user. The form will only post data to the server when all **Required** fields have data values. The associated **Required Msg** field can be used to override the default message with a custom one. This allows for language localization or expanded explanatory information.

Input Field Type

This is the type of input experience you want for your user. This can be as simple as a basic **Text** input or more elaborate such as **Date** where the user selects day, month, and year or a **Signature** input field where you can sign using a tablet/stylus or mouse.

Fieldtype Checkbox

This creates checkbox input field. DocOrigin checkboxes will always Post either 0 (off) or 1 (on) to the server - unlike standard html checkboxes which send nothing when the box is not checked. The Design Initial Value setting, or Merge field data value determines the initial state of the checkbox - a blank, 'n', 'N', or '0' will default to off (0) anything else defaults to on (1).

The value sent to a web server for a checkbox will always be either 0 for unchecked or 1 for checked.

The checkbox **Caption** is set on the standard Field Object Properties screen.

Fieldtype Choicelist

The **Choicelist** input field displays a drop-down list of choices for the user to select from.

The **Choice** option is used to define the list of choices. Each choice consists of a value to be submitted as well as the text to be displayed to the user as the description of the choice. For example:

```
a:"first choice" b:second c:"none of the above"
```

Would display three choices - "first choice", "second", and "none of the above" to the user. If the second choice was made, the value "b" would be submitted to the server.

If the value of the Merge field (using either the Design's Initial Value setting, or Merge data, or Merge script) is one of these submitted values (a, b, c in this case) - that item will be pre-selected when the form is first displayed.

Fieldtype Date

An input Date field allows a user to enter a date. The Date is selected from 3 drop-down lists - year, month, and day of month. The various Date options allow you to control the appearance of these lists.

The **Date Order** option lets you specify which order the year, month, and day are displayed.

```
year month day (the default order)
y m d (can be abbreviated)
d m (omit one or more of the choice lists)
```

The **Label** option is used to assign a default label or title to each of the dropdown selectors. It should be a list of titles in the same sequence as the **Date Order** settings (above). If the entire **Label** setting is left blank, default settings will be provided. The labels are displayed as default, non-selectable, grayed text when the control is first displayed.

The **Year Range** setting is used to specify what year or years to display in the year choicelist.

```
1950-2014
2014-1950 (so 2014 is at the top)
2000 (a single fixed year)
```

The **Short Month** option uses names such as "Jan" or "Feb" rather than "January" and "February".

The **Short Year** option displays "08", "09" rather than "2008" and "2009".

The month names are determined based on the Locale setting of the field. Setting Locale to German will result in German names for the months to be displayed.

If the Merge process results in a recognizably formatted date string (yyyy-mm-dd is a good one), this date is set as the default value.

The submitted value of a Date field is always in the form of yyyy-mm-dd as in 2014-12-25.

Fieldtype Email Address

This displays an input field that expects an email address to be entered. Validation is browser-dependent.

Fieldtype File

This creates an input field for uploading a file's contents to the server.

Fieldtype Hidden

This creates a "Text" field that is not displayed, but does submit its contents to the server. Typically used to pass additional data to the server.

Fieldtype Image

An input Image field is used to upload an image to the server. The image is also displayed or previewed to the user.

Fieldtype Password

This creates a "Password" text field - where the text being typed in is obscured by the browser. Otherwise acts as a Text input field.

Fieldtype Pushbutton or Submit Button

A button is displayed to the user. Typically used to send data to a web server.

The **Label** option is used to specify the text on the buttons (such as "Click Here"). If the **Label** option is not set, the current Merge field value is used as the label. Alternately the **Background Image** option can specify the name of an image file that will be displayed as the button. This filename should have a fully qualified path.

Note that **Background Image** can be forced to reference a file on the server by setting Merge command line "--embed-images N". Otherwise the image file is assumed to be on your computer at the time Merge is run - and is embedded into the html file.

Fieldtype Radiobutton

This creates one of a (potential) set of radio buttons that the user can select from.

Radiobuttons are generated and defined within DocOrigin somewhat differently than they are in standard HTML. Each Radiobutton Field within DocOrigin is a distinct field object. When Merge merges data it expects a data value for every button. (These of course can also be set in Design via the Initial Value setting.) To group these distinct fields so that they act as a typical input radio buttons (click one, the others are turned off) you must give them all *the same DocOrigin Field name*.

When Merge prepares to create Fillable input radio buttons it checks for radiobutton fields which have the same Field Name and are within the same parent Group or Pane. These will be treated as a **radiobutton group**. Alternately any radiobutton which is the only radiobutton within a Pane or Group of that name will get grouped with other similar "lone" radiobuttons of the same name. This allows (for example) a series of repeated Panes each with a single radiobutton the "selects" that pane.

The data stream to Merge typically contains separate field values for each radiobutton. These values control whether the input radiobutton is initially On or Off. Input values of '0', 'N', 'n' or blank are interpreted as Off - anything else is On.

When radiobuttons are Posted (submitted) to a web server each radiobutton will be individually sent with a value of '0', 'N', 'n'. Note that this is different from how standard HTML submits radiobuttons. DocOrigin ensures that every field, whether checked or not, is submitted to the server.

An additional field is also created and submitted for radiobutton groups - with a name of \$DO_ followed by the DocOrigin field name - as in \$DO_Type. This can be used to get an indication of which button was selected - similar to how standard HTML radiobuttons work. On each individual radiobutton field you can specify a **Choice** option string. The \$DO_ field will be set to that choice string if the radiobutton is selected at submit time. Both the \$DO_ field and the individual button field values are sent to the server.

Fieldtype Signature

A Signature input field is one where a user can physically sign their name. This is typically done using a tablet - although a mouse can also be used.

The Signature typically includes as "Clear" button at the top-right for the user to erase what he's drawn and start again. The label assigned to this is an "X" - but you can change this using the **Signature Clear Btn** option.

The **Background Image** option can be set to the name of an image file on your computer. This acts as a "watermark" or background to the signature control. It is NOT sent to the server as part of the user's signature.

When a Signature input field is set to Readonly, the Clear button is NOT displayed and the signature cannot be altered.

Fieldtype Telephone Number

This displays an input field that expects a telephone number to be entered. Validation is browser-dependent.

Fieldtype Text

A normal text input field. Both single-line and multi-line fields are supported - depending on the Field Object Property setting for "Single Line".

Fieldtype Time

Allows user input of a Time

Times are input by selecting from two drop-down lists - hour and minute. If the AM/PM option is selected, a third AM/PM choice list is also displayed, otherwise, the hour selection is of a 24-hour clock.

The **Label** setting can specify a label for the hour and minute choice lists - similar to the Date input field.

Fieldtype URL Address

This displays an input field that expects a syntactically correct URL address to be entered. Validation is browser-dependent.

Tooltip

The Tooltip allows the specification of a Tooltip popup message for a field. When the mouse hovers over the field in either PDF or HTML forms this message is displayed.

Viewer Attributes

The **ViewerAttributes** settings are typically used to set actions to be performed by the browser based on standard html events. For example:

```
onclick="DO.submitForm()"
```

could be used on a submit button to call the standard DocOrigin `submitForm` function when the user clicks on the button. The **ViewerAttributes** text is actual HTML that is inserted as-is into the HTML output stream - it must be correctly formatted HTML.

The **ViewerAttributes** can also be defined within standard Merge script by setting the `_viewerAttributes` property of a field.

PDF Viewer Attributes

For PDF fillable forms, the viewer attributes of a pushbutton field could be simply the URL of form submission target.

It might also take the form of:

```
onclick="myFunction(my args)"
```

That presumes that you have used the PDF Input event of the form object to script in a function named `myFunction`. PDF also uses JavaScript. A fabulous resource for defining JavaScript functions for use in PDF is the: [JavaScript™ for Acrobat® API Reference](#). Do check out its various `submitForm` capabilities and its abilities to enumerate the fields on a form, plus its XML handling.

Data Field Naming Convention

The DocOrigin html driver treats the entire form as a single html `<form>` structure. When data is submitted it is typically POSTed to a server address. Data names are fully-qualified data names. Here is a sample of some data that was echoed back in the browser from a simple php test utility called **echo.php** (found at www.docorigin.com/echo.php):

```
$DO_formName = 'DynaDemo'
page-1:AllDetails:Detail#1:Item = '1'
page-1:AllDetails:Detail#1:Description = 'A fancy item'
page-1:AllDetails:Detail#1:Qty = '1'
page-1:AllDetails:Detail#1:Amount = '$42.00'
page-1:AllDetails:Detail#2:Item = '2'
page-1:AllDetails:Detail#2:Description = 'A standard item'
page-1:AllDetails:Detail#2:Qty = '3'
page-1:AllDetails:Detail#2:Amount = '$38.95'
page-1:AllDetails:Detail#3:Item = '3'
...
```

In this example, you can see that the first data field is `$DO_formName`. It contains the name of the form - the name of the `<Form>` root object assigned in Design, not the form's file name. `$DO_formName` will always be sent for all forms. If you did not rename it in Design it defaults to 'Form1'.

Data fields use the same names as were assigned in Design.


In this example, the form contains a pane called **AllDetails** which in turn contains repeated instances of subpane **Detail**. Because this pane occurs multiple times, the `DO.submitForm()` process gives each instance a unique name - **Detail#1**, **Detail#2**, etc. Within each **Detail** pane there are 4 fields - **Item**, **Description**, **Qty**, and **Amount**.

This naming structure results in unique names for all data fields. Your server application can parse this field naming structure as necessary to map the data to whatever format you require.

See Also

[_viewerAttributes](#)

Fillable HTML Forms


 The Fillable Forms feature is available under only specific licensing arrangements.

Introduction

The [Fillable Forms](#) page described how you can use DocOrigin Design to define input fields for use in HTML (or PDF). That is all within the DocOrigin domain. This page delves into the 'in-browser' side of fillable HTML forms. That means that it is mostly outside of the DocOrigin domain, except for when some DocOrigin provided facilities assist in your browser-side efforts. Mostly, it's up to you.

Within the hosting browser, HTML is wondrously dynamic. Via in-browser JavaScript and CSS you can create a very engaging, dynamic experience. But it is work.

Generally, we expect that for document generation purposes, that you will keep your interactive HTML forms quite simple -- to collect a small amount of data and to submit that data to the URL of your choice. If that is all you need to do then the declarative input field definitions you can make via DocOrigin Design are all you need. You would not have to enter into the in-browser world of things. That is our general premise, but you can do more, if you are up for it.

 **CAUTION:** As soon as you enter into the world of JavaScript within the browser, you are entering into the world of, not form design, not document generation, but **application development**. There, you are catering to the interactive whims of your users and also fighting the lack of standardization across popular browsers. It's a new ball game with new skills required. If you find yourself rueing the effort, don't blame DocOrigin; it is outside of the product's jurisdiction.


Supplying JavaScript

As always DocOrigin is very 'open'. You can choose to employ HTML-world application construction technologies -- e.g. jQuery or other frameworks. Or you can try to keep your application slim. The door to this open world begins with the file **Default-DOHtmlTemplate.htm**, which resides in the **../DO/Bin** folder. As usual, you can override that file by supplying a **DOHtmlTemplate.htm** (no Default- prefix) file in the **.../User/Overrides** folder. The DOHtmlTemplate file is a very simple template which contains placeholders (e.g. *CONTENT*) which DocOrigin Merge replaces with the HTML constructs that it generates. Another way for you to override the **Default-DOHtmlTemplate.htm** file is to supply the `-HtmTemplate` option to Merge, it can specify a template file that you have modified, quite possibly to contain the inclusion of various JavaScript libraries (e.g. jquery) or custom CSS files.

```
-HtmTemplate $0/MySiteHtmTemplate.htm
```

Surely you would not vary this on a form-by-form basis but would build up a standard for your entire site.

The next, and more common, area where DocOrigin opens up to the browser world is on a per form basis. You can enter browser JavaScript, (as opposed to DocOrigin JavaScript with all of its extensions), into the "HTML Input Event" script area of the form object itself, via the Format-Form Properties... menu item. You may put JavaScript code directly into that "event" area, but more likely, especially during development, you would put in a `#include "$$F/ThisForm.js"` construct, and then provide your browser JavaScript in that external file. If you are doing much application development you will likely have to edit it a lot <sigh> and it proves very convenient to have that editing done in an external text editor that keeps the file open, trial after trial.

 Note that this browser JavaScript is not distributed throughout all the objects on the form, but is provided in one place, in the actual form object's HTML Input Event script area. DocOrigin Merge will place whatever JavaScript you provide into a `<script>` element in the `<head>` section of the generated HTML.

Invoking your JavaScript

Now that you can supply JavaScript for your HTML form, how do you get it invoked? The most direct link is if you have supplied a viewer attribute on any of your input form objects of the style: `onClick="MyFunction();"`. Thus, if you supply a function of that name in your JavaScript, it will be invoked.

The aforementioned **DOHtmlTemplate.htm** file includes a `*FOOTER*` placeholder. That is normally replaced by the contents of file `DOHtmlFooter.htm`, which is also delivered in folder `.../DO/Bin`. That file is very well worthwhile looking at. For the topic immediately at hand you might discover in it that if you have provided a function named `Initialize` (note the capital I), then it will be called automatically as soon as the browser has loaded the form.

DocOrigin Assists

While 'in-browser' you are mostly outside of DocOrigin assistance, but some extra functionality is provided to you for your application-building efforts. Within the generated HTML, a **DO** object is created. It has some useful members, most particularly functions you may wish to call from your own JavaScript. You are well advised to peruse the `DOHtmlFooter.htm` file to see what functions are defined within the `DO` object.

- `DO.submitForm([url])` is the most commonly used function. It does some vital housekeeping and then submits the data to the URL of your choice. Before that submit is done it will call a function named `PreSubmit`, if you have provided one. If that function returns `false` then the data submission will not occur.
- `DO.makeVisible(...)` can be used to hide or show a pane based on dynamically determined relevance predicated on user inputs.
- `DO.deletePane(...)` can be used to discard a pane entirely. It will no longer exist. Its data elements cannot be submitted.
- `DO.addPane(...)` lets you add a sibling pane right after the pane that the parameters identify.
- `DO.appendPane(...)` lets you append a sibling pane at the current end of the list of such panes. E.g. Add a pane for an additional dependent or coverage area.
- `DO.clearPane(...)` clears all the input fields in a pane.

As these pane operations occur, the power of HTML dynamism is employed to squeeze the remaining panes together or spread them apart to make room, as necessary.

- `DO.message(text)` is useful for emitting messages to the user (or the developer!) without using an annoying `alert()`. If you were to use this in production, you would probably want to style the presentation to your own preferences.

There are a few other functions in there, though most others are 'internal'; more may be added for developer convenience, but we don't wish to bulk up the amount of JavaScript tagging along in every HTML document we generate.

Accessing Fields

This is not a tutorial on browser JavaScript. Naturally, `document.getElementById()` and `document.getElementsByTagName("input")` are your friends. HTML is open; you can use all of its features to accomplish your goals. However, some understanding of the construction of a DocOrigin HTML form could be useful.

Names and IDs

- Every HTML input element can have an ID and a name.
- If an input element does not have a name, it will not be submitted.
- It is the name property, not the ID, that is submitted along with its value.
- IDs are thought of as being unique, but with multiple pane instances, that is not the case.
- DocOrigin uses the input field's name, as per your design, as the ID.
- DocOrigin modifies the name dynamically to reflect the document structure. That name carries which pane it is in and also the instance of that pane.

So... getting an element by ID is fine, if it occurs only once, i.e. if it is in a "header" pane that occurs only once. [Panels become `<div>`s in HTML.] But if a field occurs multiple times, then its ID will occur multiple times and you will have to take the appropriate steps to ensure that you fetch/update the correct element. It is quite possible that for the expected usage of collecting a small amount of information, rather than being a full-blown application, that the fields of interest will occur only once.

Remember, names change! A great way to get your bearings on the names is to, during development, submit your form to the <http://docorigin.com/echo.php> facility (or your copy of that on your own web server; echo.php is delivered, as an unsupported sample, under `.../DO/Samples/Web`). By using **echo.php** you will see the names of everything that was submitted. That is very instructive.

If you are going to do application development then you really should carefully review DOHtmlFooter.htm to see how it goes about renaming fields -- not necessarily to understand it fully but possibly to reuse some of the internal functions that it employs. Either that or develop your own functions for accessing multi-occurring IDs.

A PHP Wrinkle

PHP is reasonably popular on web servers. We use it in our [example](#) code. Alas, PHP does the nasty thing of translating dots in names to underscores. For that reason, we have chosen to use colons as name hierarchy separators and then, on the PHP side, we translate those colons back to dots.

```
DO submits: Page1:Pane1:GroupName:FieldName
```

```
echo.php reports Page1.Pane1.GroupName.FieldName
```

Having underscores as hierarchy separators is useless since object names themselves can contain underscores. PHP provides no means to look at the raw data as submitted if that data submission is in multipart form encoded format (which it is for DO HTML submissions). Shame on you PHP.

php://input does not work for multipart format.

Checkboxes

Browsers have this annoying practice of not submitting anything if a checkbox is not checked. That is terrible. You really want the submitted/collected data to have a 0 or 1, but to always exist so that the data can be passed on to other applications.

DocOrigin solves that, at a price. It automatically provides a hidden field that is always submitted, and it automatically updates that hidden field whenever the checkbox is clicked. Sounds fine, right? Well, beware. You can get the value of the checkbox from that hidden field, which in fact has the ID of the field name you supplied in Design. However, to change that value you must set the checked property of the previous sibling. Consider a checkbox named, in Design, as Foo.

```
// Setting a checkbox to true
var el = document.getElementById("Foo");
// The checkbox value, 0, or 1 is at el.value.
el.previousSibling.checked = true;
// That set the checkbox as being checked, but did not change the el.value
el.value = "1";
// This occurs automatically when a user clicks on a checkbox.
// But programmatically, you have more responsibilities.
```

Repeating: It is always good practice to use **echo.php** to see what is being submitted.

Radio Buttons

Browsers have this annoying practice of not submitting anything if a radio button is not checked. That is terrible. You really want the submitted/collected data to have a 0 or 1, for each button, so that data can be easily passed on to other applications.

DocOrigin solves that, at a price. Sound familiar? Each radio button is shadowed by a hidden field. All such hidden fields are updated whenever any radio button in the set is clicked. But let's work through an example.

Let's say that you are collecting the user's credit card of choice. You provide three radio buttons in the set: Amex, Visa, and Mastercard. As always, all the radio buttons in the set have to have the same name. Let's say we chose

"CC" as that name. It's a nice idea, if possible, to put all of those radio button input fields into a group; quite likely naming the group CC as well.

Of course, you will immediately use **echo.php** to see what is submitted when you check Visa for example. You will see:

```
Page1:Pane1:CC:CC#1 = '0'
Page1:Pane1:CC:$DO_CC = 'Visa'
Page1:Pane1:CC:CC#2 = '1'
Page1:Pane1:CC:CC#3 = '0'
```

What a bonanza! Each radio button named CC is represented by a **name** of *CC#n*, where that n goes from 1 to however many radio buttons there are in the set. You get all of their individual settings. Additionally, you get a **\$DO_CC** field which holds the value you defined for the chosen radio button, in the "Choice:" property of its input dialog.

This is great for downstream processing of the submitted data (and in a generic way). It permits the creation of XML that can be passed along and even be given to DocOrigin Merge to produce a new document, in HTML, PDF, or whatever.

The price. When you want the logical value of the radio button set named CC, you have to ask for \$DO_cc. But no, not even that works. That is because there are several radio button fields, each with the ID \$DO_cc. Only one gets submitted because that's the way that browsers choose to operate. They submit only the one that was checked.

What can you do? You have several standard browser JavaScript ways to get all the input fields, or all the radio button fields. (`document.getElementsByTagName("input")` or `document.getElementsByClassName('DO_radio')` -- all DocOrigin radio button fields get the class name DO_radio). Anyway, you can then run through that array of elements and look at their IDs. If the ID is \$DO_cc you can then check whether its checked property is true or false. If it is true, you can use its value property as the value of the radio button set. Surely, you will develop a function for that once and use it over and over.

That's fine for fetching the value. What about setting it? Just as for checkboxes it is all done automatically in reaction to user clicks but if you set a radio button to checked via JavaScript, then you have to take on the responsibility of updating the hidden shadow fields as well. As it happens there is a `DO.prepRadio()` function which circles around the whole document and ensures that the hidden shadow fields for radio buttons each reflect their counterpart real radio button field's checked status.

Across Panes Radio Buttons

We like to think that most radio button sets will be inside a single pane and that you will take the effort to group those out of a sense of tidiness. However, fillable HTML forms, (and not fillable PDF forms), support the ability to string out a radio button set across many panes. Typically a recurring pane would have one radio button of the set in each instance of the pane (think of each pane as offering a radio button selectable choice to the user). Naturally, such radio buttons cannot be grouped. However, since the radio buttons do all have the same name (in Design), they will be treated as part of a radio button set. Operations will be as described in the foregoing paragraphs.

Multi-instance Context

Suppose that you have defined an OnClick event on an input field, and it invokes a function you provide. Great, but hey, what context are you in? If this field occurs in a pane that can occur multiple times, just which occurrence are you in? If you want to do an Amount = Price * Quantity calculation, you had better be using elements from the same instance of the pane that reacted to your OnClick event.

I'm sure that you can invent navigation algorithms that let you figure out where you are in the DOM, and then 'walk the DOM' appropriately to find your elements of interest. Getting the list of all input elements (`document.getElementsByTagName("input")`) and wandering that array looking for your this object might be one way. One thing that helps is employing a consistent ID'ing style -- i.e. the names that you provide in Design for pages, panes, and groups. That suggests another approach.

As you have seen, DocOrigin changes the names of fields dynamically to reflect the latest number of occurrences of each pane. It comes up with names such as:

Page1:Pane1#2:CC:CC#1

As it is programmatically produced, it is obviously consistent. If a pane occurs multiple times it has a #n appended to that segment of the name. Those names are updated just before `DO.submitForm()` does a submit. But we can steal some of that. Consider:

```
var form = document.forms[0];
for (var p=form.firstChild; p; p=p.nextSibling) {
    DO.renameFields(p, ""); // give fields an A.B#2.C type of structure
}
```

Your code could do that. Then you could look at the `.name` property of the field that invoked your function (or whatever context you are in), and you would see the hierarchy of your object of interest. You have your context.

You could do just:

```
DO.renameFields(document.forms[0], "");
```

and then ignore the first segment of every name since it will always be the same -- the name of the form object itself. But you would have established your context. On a `DO.submitForm` the fields will be renamed to their proper ready-for-submission text.

Group Visibility

Sometimes it is not only fields that are of interest. A typical case is that of a group of objects. You may wish to make them appear or disappear based on user interaction. Named-in-Design groups get IDs as well, so you can get their element object and set their `.style.display` property as appropriate. Just standard browser JavaScript.

Pages / Tabs

By default, DocOrigin Merge puts each page of the generated HTML document into a tab whose name is the page name, plus the occurrence number if that page layout occurs multiple times. A function named `DO.showPage(pageName)` can be used to switch focus to a specific tab, should you ever wish to wrest that decision away from your user.

A tabbed presentation of pages is not always wanted. You can control that with the command line option:

```
-HtmlNavBar none
```

In which case the generated "pages" will be displayed as a single HTML page through which the user would scroll. The length of each heretofore page is trimmed of white space from the bottom so as to reduce the need for scrolling.

The opposite, and default, setting is:

```
-HtmlNavBar tab.
```

Previewing Your HTML


DocOrigin Design offers a Tools-HTML Preview... menu item. If you don't want to change any options from your last preview, you can just hit **F7**. (You'll be using that a lot.)

What happens is that after Merge creates the HTML output, it directs it to a .bat file named **Default-HtmlPreview.bat**. Naturally, that is delivered in `.../DO/Bin`. You should read that .bat file! Note that it offers you the opportunity to supply your own .bat file in **\$O/HtmlPreview.bat**. (\$O is `.../User/Overrides`). If you read the .bat file, you will see that it makes reference to other sample .bat files that may help you in better testing your generated HTML.

The dialog that invokes Merge's generation of the HTML has a curious little field in its bottom left corner, labeled **HTMLPreview.bat** option:. You can put anything you like in there. It will be passed to your **HTMLPreview.bat** file as an optional third parameter. You can choose to interpret what you enter any way you like. The samples mentioned earlier use that field as a means to choose whether to preview in Chrome, Firefox, or IE. Getting your

forms to look and function properly in all the popular browsers is a challenge. This provides a means to see what's up.

Fillable PDF Forms

 The Fillable Forms feature is available under only specific licensing arrangements.

Introduction

DocOrigin supports the creation of PDF documents that contain Acroform fields to capture input from the user.

Compared to HTML, fillable PDF forms are limited. The big difference is that with HTML you can have an ever-lengthening page, allowing for much greater dynamism in a fillable HTML form while it is being viewed in the browser. With PDF forms, all of the dynamic instantiations of panes and pagination occurs within DocOrigin Merge -- once Merge produces the PDF, that PDF will be constrained to having only the panes that it was generated with. No panes can be added, deleted or even hidden while in your PDF Reader technology. Fields will not grow in width or height, though Adobe Reader does provide scrolling for multi-line input fields. All objects remain the same size -- it's a very static layout.

Still, it is quite useful to have a PDF in which a user can enter data, and then possibly print it out, or submit it to an URL that you have provided in your form design, such that you can collect the information and process it as befits your needs. It's possible that as part of that processing you will update databases, invoke other applications, and even invoke DocOrigin Merge on your web server with the same or a different form with captured and augmented data.

Supported input types: Text fields (single line and multi-line), checkboxes, radio buttons, dropdown choice lists (i.e. one line tall, with a drop list to show the choices), hidden, password, and push/submit buttons.

Unsupported input types: File, Date, Time, Email Address, URL Address, Number, Telephone Number, and Signature. Mostly those are HTML5 field types so their non-support in PDF is not surprising.

Input attributes:

A field may be marked as "readonly" and as such the value provided to it when Merge generated the form cannot be changed by the user.

The "**required**" option is also supported in PDF forms. When a submit is done, fields marked as required will be checked for content and a diagnostic issued if there is none. Any specification of a custom message will be ignored.

Tooltips can be specified; they will be displayed when the user hovers over the relevant field.

Hidden field: Do note the difference between an input field type of hidden versus a design object being marked as invisible. In the latter case, the object won't even be represented in the PDF. In the former case, the field is in the PDF but hidden from the user. You could access it via JavaScript.

Password field: Characters typed into a password field are displayed as asterisks, though the correct values are submitted. NOTE: If a password field is left empty then that field will not be submitted at all.

Responding to User Inputs

The Reader

Definition: "the Reader" -- more than one application exists that can display PDF files. Of course, the most widely known one is Adobe Reader, but there is also Acrobat, Foxit Reader, the Google Chrome PDF extension, ... Below, references to "the Reader" means whichever technology is being used to display the PDF. (akin to "the browser" for HTML documents). Not all PDF Readers perform to the same standard.

Sadly, Chrome's built-in PDF plugin does not support fillable PDF at all. You can neither enter data nor submit it. Useless.

While you can't, with any sort of reasonable effort, change the layout of the PDF once it is displayed in the Reader you can still react to the entries that your user is providing. For the most part that means supplying or altering data that is in the form. And of course, you can cause the submission of the collected data to an URL of your choice.

If all you want to do is collect the data as entered and submit it, then that can all be done declaratively with the DocOrigin Design tool. No further 'in the Reader' effort is required.

If you wish to manipulate the data while 'in the Reader' this can be done using JavaScript. Note that this is not DocOrigin JavaScript, complete with all of its amazing extensions, but rather the JavaScript supported by the Reader. A good reference for this is: [JavaScript™ for Acrobat® API Reference](#). That is a long and sometimes challenging read; you likely have more simple goals in mind. The following tries to grease those wheels, but you can most certainly fall back to that reference manual.

For a developer, a useful resource is always some code to look over. DocOrigin ships a critical piece of JavaScript for use in the Reader. It is called Default-DOPDFScript.js and is delivered in the **.../DO/Bin** folder. Note that it is .js and not .wjs. As usual, you can override that file by supplying your own **DOPDFScript.js** (no Default- prefix) in the **.../User/Overrides** folder. The important thing about this file is that it is included verbatim in every DocOrigin produced fillable PDF document. It is the thread that starts the ball rolling in terms of getting your own JavaScript code operating inside the Reader. And it happens to show some JavaScript code that may be of interest to a developer.

That's nice, but how do you get your own JavaScript code into the PDF; you certainly are not allowed to edit a file that resides in **.../DO/Bin**. You put your JavaScript in the "PDF Input Event" of the form object, via DocOrigin Design's Format-Form Properties menu item. You might well specify a single: `#include "$F/MyScript.js"` line in that location, and then edit the referenced external file. That is often convenient, especially during development. DocOrigin Merge will append whatever you define in the PDF Input Event (and what it `#includes`) into the generated PDF document. Your code will immediately follow the code you see in **DOPDFScript.js**. There... now you can get your own JavaScript into the PDF.

But what can you do using JavaScript? Lots, if you read that whole reference manual. But more likely you simply want read/write access to the values of the fields on the form. This is not overly difficult in a PDF since there won't be any dynamic additions/removals of panes. What you start out with is what you have. [Yes, you, having thoroughly digested the entire reference manual, and really being in the wrong business, could manage to alter the form layout and content while in the Reader, but that is not the goal here.]

Getting a field's value

Consider the following:

```
var fName = "MyInterestingField";
var f = DO.doc.getField(fName);
var v = f.value;
//console.println("Field " + fName + " has value " + v);
f.value = "I clobbered it's value";
```

Well, that's pretty easy. `DO.doc` is literal. Its value is defined by `DOPDFScript.js`. Often you could use this instead. But you can be assured that `DO.doc` always refers to the DocOrigin generated document in the PDF, no matter what context you find yourself in.

Now your only problem is knowing what the names of the fields are. The code in `DOPDFScript.js` shows how you can enumerate the fields; then you could examine their `".name"` attribute. Generally, the names are of the style:

```
Page1.Pane1.GroupName.FieldName
```

Another great way to know the names of the fields in your PDF document is to use, during development, a submit button that submits the data to "<http://docorigin.com/echo.php>". All it does is echo out the names and values of the fields in the form. The `echo.php` file is also provided, as an unsupported sample, in `.../DO/Samples/Web`. You could alter it (or not) and deploy it onto your own web server. We really have no interest in seeing the data that your forms submit! We certainly don't collect it.

We can't emphasize enough just how instructive it is to see the names of the fields as they are / would be submitted. Highly informative. In there, you would see some special \$DO_xxxxx fields that provide some context, and also how radio button group values are submitted.

Debugging

It is extremely useful, during development to use the `console.println` function to debug out the values that you are encountering. Of course, a useful initial operation to perform is:

```
console.clear();console.show();
```

That causes the Reader's JavaScript console to appear so that you can see your debug output.

Radio Buttons

First off, for fillable PDF forms, you **MUST** enclose your radio button sets in a DocOrigin group construct. Of course, all of the radio buttons in the set must have the same name. It may be a fine idea to give that enclosing group that very same name too. The group construct should have no other objects within it; just the same-named radio button fields.

Unlike HTML, PDF radio button sets must be contained within a single pane.

Every radio button input field must be given a "Choice:" value in the field's Input dialog, or via DocOrigin Merge script, before the PDF is actually output. Using the classic credit card example, these choices could be: American Express, Visa, or Mastercard. They define the value to be associated (and submitted!) as the value of the overall radio button set.

Let's say that we have a radio button set in which all of the radio button fields are named: CreditCard, and they are grouped in a group named, for lack of imagination, CreditCard. The leaf (trailing) node of the field that is submitted would be: \$DO_CreditCard. It is a specially manufactured field that contains the current value for the radio button **set**. If there were three individual radio buttons (PDF-ese would call them widgets), they too would have values submitted, named: CreditCard#1, CreditCard#2, and CreditCard#3. Their values would be only either '0' or '1', indicating whether that particular widget was checked or not. This is accomplished through very special DocOrigin Merge fillable PDF construction. It is useful to have that in the submitted data in order to generically produce an XML file that can be reprocessed by DocOrigin Merge. Of course, the overall radio button set value, as submitted in \$DO_CreditCard is quite invaluable too. It, for example, might contain the value Visa.

Set the radio button value

You do not set the value of a radio button set directly, using its `.value` property. Instead, you check one of the buttons within the set. This can be done with the Reader-provided functions:

```
var f = DO.doc.getField("$DO_MyRBSet"); // of course the name would be a hierarchical
name
f.checkThisBox(2); // 0-based, so check the 3rd radio button
```

For what it is worth, you can determine whether any individual button is set by using the `f.isBoxChecked(i)` function. Since the `.value` property of the radio button group object always reflects the button that is checked, using the `isBoxChecked()` function is pretty exotic.

No Radio Button Selected

If you are insane and design a form where it's possible that no radio button is selected, then that is what your user will see. If they do not choose any radio button, then **nothing** will be submitted for that radio button group. Downstream processing tends to have more difficulty with processing the absence of data as opposed to actual data. Don't do that. Use "none of the above" if you have to.

Push/Submit Buttons

There is no difference between Pushbutton and Submit Button. You can provide an URL target in either case. Somewhat more interesting is that you can provide a "Viewer attribute" such as `onClick="MyFunction();"`. In fact, this can be done on any field. When the user clicks on the field (actually when they 'mouse up' on the field) the on-click code will be called.

You can define your functions in your own JavaScript which you know how to get into the PDF. Also, there are a number of actions beyond 'onClick' but that one is the most common one.

Special Functions

Initialize(): If you look at **DOPDFScript.js** you can see that, as soon as the PDF is loaded, it will call a function named Initialize if you have provided one. Do note the uppercase I. It is case-sensitive. Naturally, that is a great place to do, well, initializations.

PreSubmit(): If you attempt to do a form submission, via `DO.submitForm` -- which is the standard way -- then it will call a function named PreSubmit, if you have provided one. If that function returns `false`, then the submit operation will not be done.

Binding Script

Another common case is to bind script to a particular event, e.g. field value change.

This can be implemented using the `setAction()` method. For instance:

```
this.getField("FieldName").setAction("Keystroke", "some JS code");
```

Pagination

Automatic pagination is one of the primary features of any document generation system. Well, "automatic" but with control! Conceptually, pagination is dead simple: when you run out of paper, start a new sheet. When you want control over that process it becomes substantially more difficult than that. But that's the developer moaning. It has to be easy for the user.

Let's back up for a second. What happened just before pagination? Data merge happened. That means that there is now a big DOM full of instantiated panes. Typically poor old "page 1" has a container that is bursting with panes – way more than its container can possibly hold. Data merging doesn't care about that. To it, pages/containers are of infinite capacity. Then along comes pagination to face the real world with its physical limits.

If no control were offered, pagination would/could just see if a pane fits in the container, and if it does, great, get the next pane, but if it doesn't, well start a new page and put the overflowing pane there. Simple. ... but it's not.

What do you mean pane? Sure there are some simple form designs where there is simply one "leaf node" pane after another, but mostly one has a hierarchy of panes where one pane encloses several other panes, etc. Now, what does it mean when we think "if the pane doesn't fit"? Which pane? The outer pane, some inner pane... what?

Besides, even in the simple case of a sweet string of leaf node panes, you won't be happy if pagination does a new page between any old pair of panes. You have to be kind to "widows and orphans". It's a life rule.

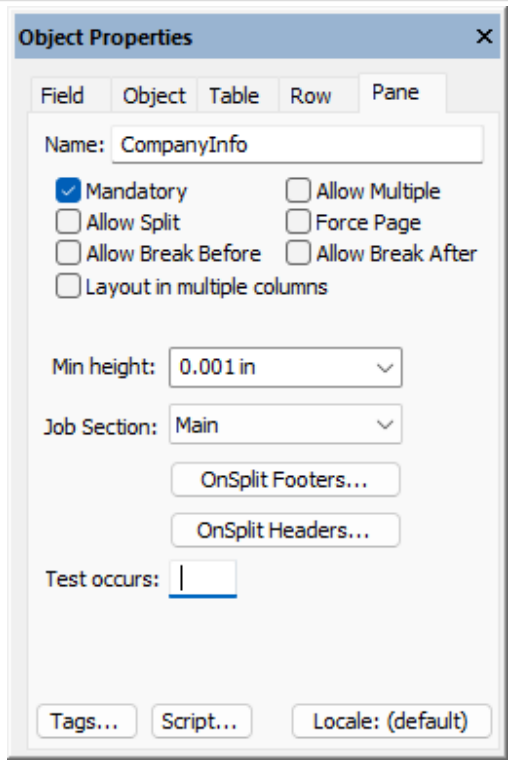
For example, it's unlikely that you want a column header pane to be the last pane on a page – widowed there. It's equally unlikely that you want a totals pane, signature block, or even first of a set of detail lines to be the first thing on a page -orphaned there. In real applications, panes like context.

You need control

Corollary – you need to conscientiously apply that control or face the penalties for inattentiveness.

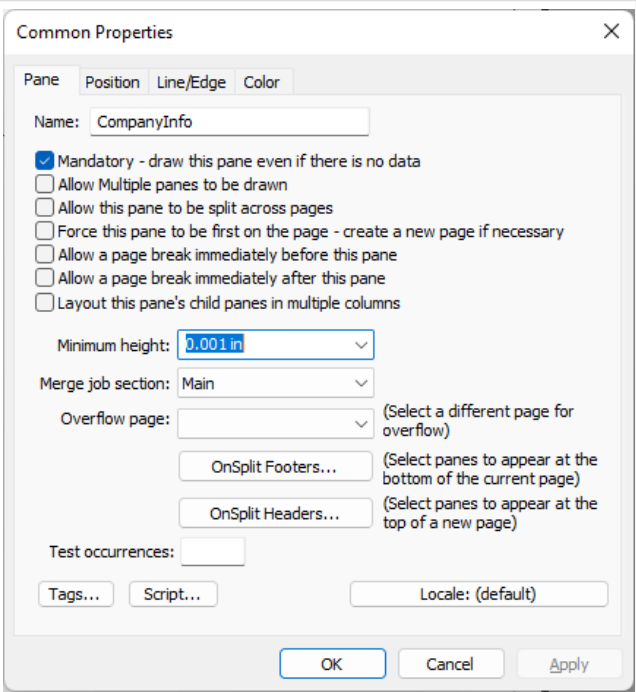
That control is provided to you via the usual modal and modeless property dialogs for pane objects.

Modeless Pane Properties



The Modeless Pane Properties dialog is titled "Object Properties" and has tabs for "Field", "Object", "Table", "Row", and "Pane". The "Pane" tab is selected. It features a "Name" field with "CompanyInfo" entered. Below are several checkboxes: "Mandatory" (checked), "Allow Multiple", "Allow Split", "Force Page", "Allow Break Before", "Allow Break After", and "Layout in multiple columns". There are also input fields for "Min height" (0.001 in) and "Job Section" (Main), along with buttons for "OnSplit Footers...", "OnSplit Headers...", and "Test occurs:".

Modal Pane Properties



The Modal Pane Properties dialog is titled "Common Properties" and has tabs for "Pane", "Position", "Line/Edge", and "Color". The "Pane" tab is selected. It features a "Name" field with "CompanyInfo" entered. Below are several checkboxes: "Mandatory - draw this pane even if there is no data" (checked), "Allow Multiple panes to be drawn", "Allow this pane to be split across pages", "Force this pane to be first on the page - create a new page if necessary", "Allow a page break immediately before this pane", "Allow a page break immediately after this pane", and "Layout this pane's child panes in multiple columns". There are also input fields for "Minimum height" (0.001 in), "Merge job section" (Main), and "Overflow page". Buttons for "OnSplit Footers..." and "OnSplit Headers..." are present, with explanatory text for the latter. There are also buttons for "Tags...", "Script...", and "Locale: (default)".

The modal properties are shown because they have a bit more text so as to explain what the checkboxes mean. One important piece of text is missing though. The context of these checkboxes is **within this pane's immediate parent**. It is not a 'within the entire document' context.

For example, consider the "Allow Multiple" checkbox. Sure, a pane can occur lots of times within a whole document, but what matters is whether it is able to occur multiple times within the pane's immediate parent. Always keep that implicit context in mind.

Pagination Checkboxes

The checkboxes are damnably simple. So simple that one tends to gloss over them, not giving them the attention they deserve. Beware of deadly results. Pay attention to the checkboxes for each and every pane; the high-level ones and the itsy-bitsy bottom-level leaf node ones.

Mandatory

Does this pane have to come out even if there is no data for any of the fields in the pane (or the pane has no fields)?

Careful though, a footer pane may seem mandatory in that you really do want it to come out, but it is forced out by means of an Overflow Footer specification. Think about it. Do you want the pane to come out where it sits in the form design, no matter what?

In the case of a detail type usage pane, must it come out, or are there perfectly valid situations where there might be no detail lines?

Allow Multiple

The only trick here is to realize that the question is being asked in the context of the pane's immediate parent. In that context, is it allowed to come out more than once? Does the tag name for this pane occur multiple times in the XML file – within the context of its parent?

Too often we see an enclosing pane marked as allow multiple and its sole or at least primary child pane also marked as allow multiple. That might be right but please do look at it and decide if perhaps one of them should not be marked as allow multiple.

Allow Split

Containers will fill up with 'too many panes'. Something has to be done. When a pane, first at the outermost level of the panes in the hierarchy, overflows its container you want to control whether to push the whole pane over to a new page or to allow the pane to be split leaving as much as can fit on the current page stay on the current page, and only the rest move to the new page.

Panes that enclose other panes are more likely to need the "Allow Split" setting turned on. Not always, as sometimes there is a tightly knit group of panes that are meant to stay together. Look carefully.

Where will it split the pane? You are in control of that using these same checkboxes throughout the child panes of the overflowing pane. First of all, if the pane that overflows the container is a leaf node pane (has no child panes) but is marked as allow split, then it will be split at the lowest possible place where there is a gap in the extents of the objects in that pane. E.g. between a row of fields. By 'lowest' we mean a split point that keeps as much as possible on the current page; that makes maximum use of the paper stock.

How much room is there? Nothing is easy for the code. Enjoy the fact that you need to only click a few boxes. How much room is available is determined not only by the container size but also by the size of the overflow footers that apply to the pane being split. The footers for pane A are not necessarily the same as the footers for Pane B. So the available room varies.

But of course, the pane that overflows is likely going to be an enclosing pane, a pane with child panes. At that point the pagination code will recurse down a level looking for a good spot to split between the child panes or possibly within a leaf node child pane. The applicable footers may change.

Finding that split point is controlled by your settings of "Allow Split", "Break Before" and "Break After".

Allow Break Before

That is "Allow break before". This is not forcing the pagination code to break before this pane is used but merely allowing this pane to become the first pane on a new page.

I like to personify the pane this way. If I check Allow Break then I am saying I am so strong that I can leave the nest, start out on a new page all by myself. I don't need any elder sibling or mama. I'm good to go. Conversely, if I don't check Allow Break, it is saying that I am too weak to stand on my own. I need an elder sibling or my mama. Don't orphan me. I need my context.

As it happens, leave Allow Break Before off almost all the time. You do not want a break before headings. You don't want a break before the first in a series of detail lines – you want to cuddle up under the detail line headings. You probably don't want a break before a totals type usage pane. It would just be odd to have a total with no context as to what it is totalling, so usually you want to drag along the last detail line. "Usually", you can choose to put Break Before on if you want. Similarly, a signature type usage pane probably wants some context dragged on its page.

But there are times when a document contains, oh, sections. These sections are strong. They can start out on their own. You could easily want to mark the first pane of a section as "Allow Break Before".

Allow Break After

That is "Allow break after", not force break after. This is the protective mama end of the spectrum. Leaving this off says that you don't want your kids, well, your younger siblings, to be sent off in the world on their own. You want to wrap them to your bosom by keeping them on the same page where you can "look after them" by providing your context. Or like saying "Don't abandon me, or leave me a widow". Conversely, checking Break After means that you have faith in your younger siblings, faith that they can stand on their own.

Less personified, the example is of the details header pane that says "don't break after me", my, at least eldest detail line has to be on my page – or move me too.

Like Allow Break Before, you pretty much always have Allow Break After unchecked. Think about it. Where would you really want to say it's ok to break after this pane? Certainly not immediately after a header pane – it's a header for something that wants that header context. Continue thinking like that as you imagine the various pane usage types and you'll see that allowing break after is a rarity.

Allow Break Between

What? There is no "Allow Break Between" checkbox. Correct. Merge pagination **always** believes that it can break between two successive instances of the **same** pane. That's automatically always true.

It is this implicit Allow Break Between setting that takes all the steam out of Allow Break Before and Allow Break After. Allow

Break Before really means "Allow Break Before the first occurrence of this pane" (in its parent's context), and Allow Break After really means "Allow Break After the last occurrence of this pane" (in its parent's context). Both of those cases are generally false, but break between – well, that's a winner. There's context on both pages so breaking feels great.

Force Page

This is the pagination code's friend. No hassle, no thought, do as commanded. If it encounters a pane with this checked, it will immediately start a new page. One exception is if the current page's container has nothing in it, then Force Page will do nothing.

You almost never set Force Page. Let pagination do its thing.

Layout in multiple columns

If pane has "Layout in multiple columns" checked then Pagination Algorithm tries to arrange the pane's children panes in multiple columns interpreting this pane as a kind of a "container".

Footers/Headers defined for the "Layout in multiple columns" pane (or its parents) are applied when the pane itself is being split between pages. If you want footers/headers for the columns then you should define footers/headers for the children panes keeping in mind that in this case "OnSplit" occurs when the pane is being split between columns.

Column balancing

By default children, panes are evenly balanced between the columns. As of version 3.2.001.09, it's possible to disable balancing with a tag "BalancedColumns No" set for the "Layout in multiple columns" pane. It can also be set globally when used as a Merge command line option.

Minimum Height

What is this about anyway? Well, when drawing out a pane on the Design canvas it can be convenient to leave space at the bottom of the pane, just to have room to mouse around in. You more clearly see the objects in the pane without confusing them with the bottom edge of the pane. Whitespace is your design-time friend.

But do you want that whitespace to show up in the rendered documents? *Probably* not. So by setting Minimum Height to something really small you are allowing Merge to throw away all the whitespace below the lowest object in the pane. That is generally what you want to do.

But what if you do want whitespace at the bottom of your pane? Merge will always respect your bottom margin specification. So specify all the whitespace you want via the bottom margin setting. Alternatively, you can draw out your pane to the exact height you want it to be, including whitespace at the bottom. Then you can select the "(full height)" setting for the Minimum Height setting. You don't need to set a bottom margin in that case.

Min Height of Panes that wrap other panes

The setting "(full height)" means the full height as seen at Design time, i.e. the height that you drew the pane(s) out at. An enclosing pane may include all sorts of optional panes. It makes no sense to set the Minimum Height of an enclosing pane to "(full height)". They should always have a very small minimum height and let them be as tall as they need to be. Use a bottom margin as applicable.

A *small* number? Zero is small isn't it? Yes, it is, but zero is actually the encoding for "(full height)", so don't use zero, use a small number.

Pagination Generalities

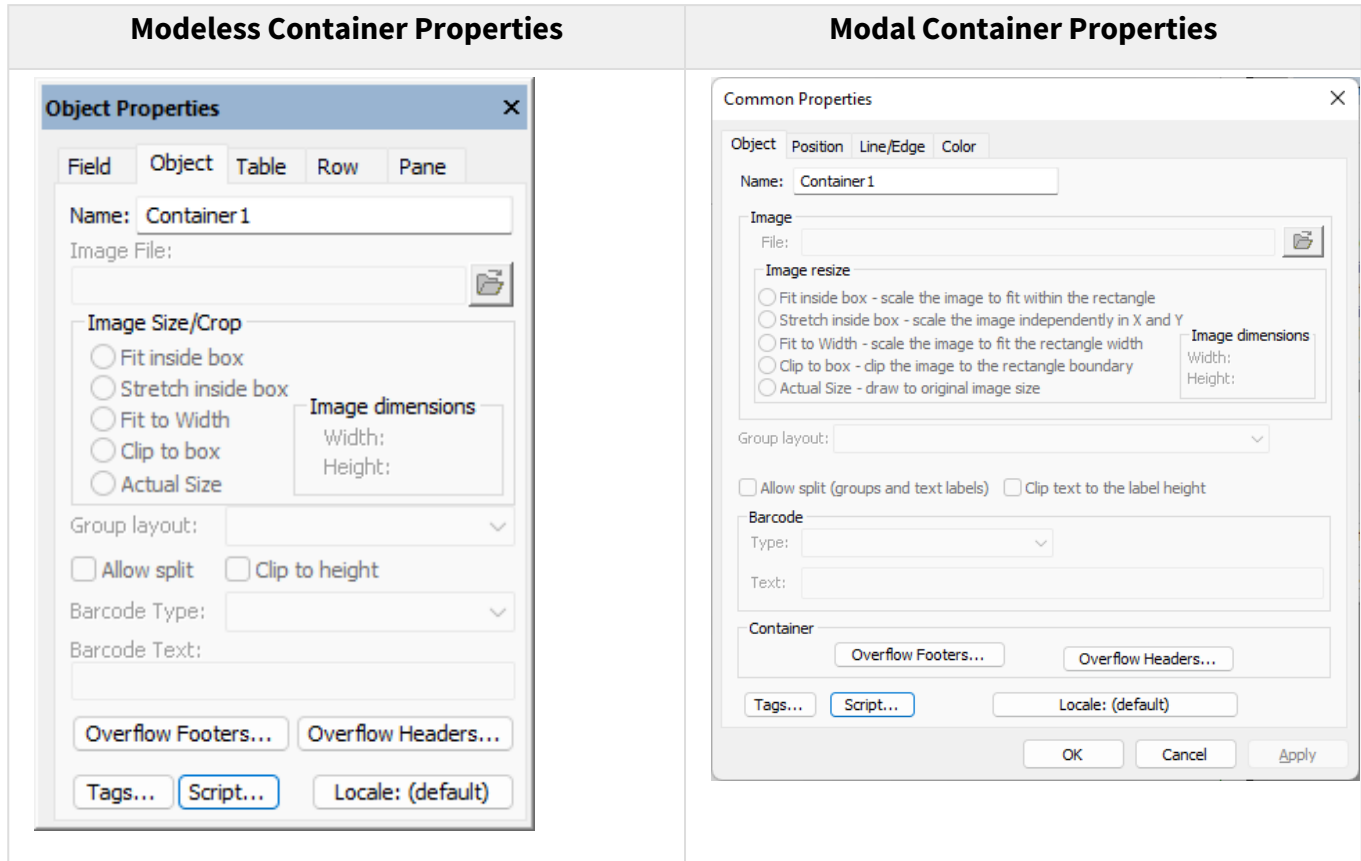
But don't even dream about not paying attention every time.

Setting	Value
Mandatory	On a lot of the time
Allow Multiple	Mostly off, unless it's a repeating detail type pane
Allow Split	On a lot, especially for enclosing panes, and panes over 1 "row" high
Force Page	Almost never on
Allow Break Before	Rarely on, but sometimes
Allow Break After	Rarely on, but sometimes
<i>Allow Break Between</i>	Automatically on all the time

Setting	Value
Minimum Height	Mostly a very small size (.001)

Footers/Headers

Note that besides OnSplit Footers/Headers for a pane, it's also possible to define Overflow Footers/Headers for a container. Though only the closest ones to the split point are applied, i.e. footers/headers are not accumulated going up the objects hierarchy from the split point.



- i** There is a difference in how footers/headers are applied depending on the pagination version. See [Pagination Algorithm](#).

Pagination Algorithm

As of 3.1.002.06 version of "Pagination Algorithm" is the property of the form. The difference between different pagination versions is how footer/headers are applied in the split point.

Version	Description
Version 1 (relaxed)	Pagination Algorithm looks for the footers/headers defined for the pane being moved to the next page (first one in the set of moved panes).
Version 2 (strict)	Pagination Algorithm looks for the footers/headers defined for the pane being split.

Pagination Algorithm gets complex really fast but some high-level insight might help.

- Determine the container height
- Walk down the list of panes until one causes overflow (consider footers)
- Walk back up the list of panes looking for...
 - A pane that is splittable
 - Or a pane that is marked as Allow Break Before **and** has an elder sibling that is marked as Allow Break After.
- If no split point is found...
 - Walk back up the list of panes looking for...
 - A pane that is marked Allow Break Before **and** simply has an elder sibling, regardless of its Break After setting.
- If no split point is found yet...
 - Walk back up the list of panes looking for...
 - A pane whose elder sibling has Allow Break After set.
- If still no split point is found...
 - Split before the pane that overflowed regardless, as long as it is not the only pane in the container. Else log a message.

When dealing with a pane that is marked as "Allow Split", then the algorithm for finding the split point in it is much the same as that for a container. In extremis, it may have to split a leaf node pane somewhere between its objects.

Splitting a leaf node pane between its objects has its own set of difficulties when one considers long vertical lines, tables, borders and the like.

Processing Instructions

(As of version 3.2.001.09)

Merge supports Processing Instructions (PI) located in XML data file.

It is a usability feature which may be used if at data generation time you know all the parameters Merge needs and do not want to specify them for Merge later.

Merge PI are similar to FM PI but related to Merge, not FM. In FM world PI help to identify job name or run Merge directly without any scripts involved. In the Merge world, PI allow to run Merge without specifying additional parameters.

For example, put this line right after tag of your data file and run Merge as "Merge -data yourData.xml" to get data merged with **\$F\Invoice1.xatw** form, PCL output created and sent to OfficePrinter5.

```
<?DocOrigin form="$F\Invoice1.xatw" config="Default-LJ4.prt" output="prt:OfficePrinter5"?>
```

Note, that Merge PI have "medium" priority, they overpower properties from PRM files but user-specified CLO overpower PI.

For example, process the same data file as "Merge -data yourData.xml -config Default-PDF.prt" and you will get PDF (instead of PCL) sent to OfficePrinter5.

"Do PI always applied?", "How does Merge know when to use PI?" you may ask. Merge applies PI from data file only when:

- User specified -data parameter but didn't specify -form.
- All PI items are valid Merge parameters.

Some options are more likely to be seen in Merge PI than others. For example, it is mandatory to have -form in PI but it makes almost no sense to specify something like -logfile there.

Probably needless to say that you have to be careful with relative and absolute path used in PI. Your data file may be processed on a different computer than planned, computer config or file system may change over time and your data will have wrong PI hardcoded. There are pros and cons, it is up to you to decide if you need this feature and find a balance between flexibility and simplicity.


See Also

[Job Name Discovery](#)

Profile Files

Merge allows the use of simple text Profile files (or "INI" files) to provide data that can be load (or preloaded) into your Merge run. This mechanism provides a simple way to configure or customize a form definition externally, without needing to alter the actual form template in Design. This can be helpful when a single form is customized for different user groups such as branch offices or departments.

A Profile file is one that is typically loaded via the Merge command line option `-profile` prior to any document or script processing. There is also a `_file.readIniFile` function which is recommended if you wish to simply read a `.ini` file into your JavaScript code.

 Note that these are NOT DocOrigin/Merge configuration files, they are for you, the programmer, to use for building your form/document application.

Profile File Format

See the [_profile \(Access Profile Files\)](#) section for details on the file format. Profile files look like `.ini` files but have additional capabilities as described below.

Opening a Profile File

As well as using the script `_profile.load()` function you can specify a Profile file to be opened by Merge by using the command line option

```
-profile filename
```

This will open the specified profile file at the start of the Merge process. The data will remain available throughout the Merge run unless a different file is opened using the script `_profile.load(filename)` call.

One profile file at a time!

The contents of only one profile file are available at one time. Referencing another profile file overwrites all the info from a prior profile file. See the `_file.readIniFile` function for reading and retaining the info for multiple generic `.ini` files (but with no special Profile "actions"). The contents of the last loaded profile are available throughout the Merge run. They survive document boundaries.

Profile Values as Merge Embedded Fields

The DocOrigin Merge program allows additional Profile features that are not available in other DocOrigin Modules. Profile settings can be referenced via the DocOrigin embedded field mechanism. See [Auto/Embedded Fields](#). This allows values to be automatically inserted into text Labels and Fields when Merge is run. For example, a text Label might be defined as the string:

```
Label = The city of [!profile Address.city] in [!profile Address.state].
```

which at runtime, using our example, would be converted to:

```
The city of San Jose in California.
```

Note that the `[!profile]` definition uses a dot to separate the profile section name from the key. This embedded substitution may also be used in Field initial values as well as actual Field values.

Using a Profile to Modify Form Objects

Learn to use the power of true profile files (not just .ini files)

Note that Profiles are more powerful than generic .ini files. Please examine the available actions below and consider the script-free opportunities.

A powerful feature of the Profile is its ability to modify or customize a Form template in Merge. You can use a profile to modify text Labels, fonts, and colors as well as re-position certain objects on the page. These features can be used to allow a single form design to be customized to a particular customer's name, address, logo, etc. without the need of either JavaScript or modifying the form in Design. While not as powerful or flexible as actual form re-design, this feature can be used where a standard, stock form requires minor tailoring for various clients or uses.

In essence, these are actions that modify a Form's design as Merge is initializing. Each action performs some sort of action on a selected object in the form.

Another use of these customization features is the ability to create a single form which is to be presented in several languages. Each of the fixed titles, field names, etc. can be re-assigned a new (translated) value when Merge runs. Using a simple bit of scripting one can even dynamically choose the language at run-time based on an incoming field value.

There are several pre-defined profile section names that trigger these dynamic actions. All these special section names are preceded by an asterisk - as in [*Font].

[*Assign] - assign new text to a Label or Field

```
[*Assign]
; Assign a value to a Field, Label, or Image.
Panel.Prompt = "Approved by:"
Address = DocOrigin Corp.\n2364 Pollock Rd\nYourtown, Ont.
Logo = $U/Logos/YourTown.jpg
```

The above would search for a Field or Text Label called **Panel.Prompt** and change its value to "Approved by:". Similarly, the **Address**, a multi-line label, is reset. The static image object called **Logo** is changed to display the given image file.

If you use the *Assign mechanism to modify a Field value, it effectively sets a (possibly new) default value for the Field. The subsequent merging of data into the form will of course override this default provided that there is in fact data for that specific field name.

[*Color] - Change an Object's Color(s)

```
[*Color]
; Reset color of an object.
; <foreground>, <background>, <border>, <shading pattern>
; Patterns are horizontal vertical crosshatch diagonal reversediagonal diagonalcrosshatch

Footer = white,black
Declaration = , #ddd, #ddd, diagonal
```

The above settings will change a label named Footer to display white text on a black background. The Declaration label will change its background to a diagonal-stripe of color #ddd (a light gray) and also turn on a border of the same color. See the Colors page for information on built-in colors and defining arbitrary colors.

Note that *Colors can set the color of any object - rectangles, panes, lines, etc. as well.

[*Font] - Change Field or Label Font Settings

```
[*Font]
; Reset font information for a Field or Label
; <typeface>,<point size>, <B/I/BI>, <horizontal justify>, <vertical justify>
; if value omitted, it remains the same as in the design.
; <horizontal justify> can be left, right, center, spread, or spreadall
; <vertical justify> can be top, middle, or bottom

Address = Courier New, 9pt, B, right
Pane1.Prompt = ,12
```

The above will change the **Address** string to use Courier New font, 9 points, and bolded. It also right-justifies each line of the address. The Pane1.Prompt string is changed to 12-point text, with other font settings left untouched.

Note that *Font changes will NOT work for Text Labels that have embedded font changes (RTF) already in them.

[*Visible] - Show or Hide an Object

```
[*Visible]
; Change an object's visibility
; = N or No - to make invisible
Footer = No
Declaration = Yes
Address =
TearOffLine = N
```

The above settings make the **Footer** invisible, **Declaration** visible, and leaves **Address** unchanged. The object (presumably a line object) named **TearOffLine** is also made invisible.


Wow! Variants!

Power! Variants allow you to rename objects to establish use of whole alternate sections of a form. And this is done dynamically on a data-driven, per document, basis.

[@Variant]

(As of 3.1.001.25)

Select a variant of an Object

 Note - this is preceded by an '@' character, unlike the other profile options! (See [Profile Files](#))

Section names starting with an '@' character are processed at the start of each document (rather than the usual start-of-job) This means they affect each document individually.

The @Variant action is to rename an object. It is typically used with Panes when a table or structure may want a slightly different layout of a table row or other collection of fields for different customers - but all using the same form template.

Consider a form that displays a repeating row of data called 'Detail'. Normally you would create a Pane or Row in the form design called 'Detail' and put the data fields inside it. But in this case, the layout of that 'Detail' row must appear differently depending on some customer/subsidiary field called 'Style'. This can be achieved with a single form using the [@Variant] profile option.

The designer creates (say) 3 different variants of the pane - called 'OptionOne', 'OptionTwo', and 'OptionThree'. We add a field named, for example, 'Style' into the data file with a value of 'One', 'Two', or 'Three'. This appears in each <document> of the file. The data for the pane continues to exist in the data stream under the name 'Detail'.

In the profile, the following appears:

```
[@Variant]
Detail = Option[Style]
```

Now when the data file is processed, the value of field '**Style**' is appended to the string '**Option**' to become a name such as '**OptionTwo**'. The resulting named object in the design is renamed to the name '**Detail**'. The resulting Merge of the data that follows then uses variant '**OptionTwo**' of the pane.

Each <document> can have a different value for the field '**Style**', and hence, generate a different variation of the Detail row.

See Also

[_profile \(Access Profile Files\)](#)
[_file.readIniFile](#)

Transparent Color Image Support

- Select an image with a transparent background
Tip: Both sample images above contain a transparent background. To save the image, select the image and press **Ctrl+Shift+E**. You will be prompted to save the image.
- Add the following script to your image
`this._transparent = true;`

i Note: The **Common Properties > Color > Transparent** Background applies to the image label object only, not the actual image itself.

- Set the Blend printer options
 - Using a form script
`_printer.setOption('Blend', 'Normal');`
 - Using a merge option (in your merge command or enter in your **\$U/Overrides/merge.prm**)
`-blend Normal`
 - Setting in your Printer Configuration File (PRT)
`<Blend>Normal</Blend>`
- Set the SupportAlpha printer options
 - Using a form script
`_printer.setOption('SupportAlpha', 'Yes');`
 - Setting in your Printer Configuration File (PRT)
`<SupportAlpha>Yes</SupportAlpha>`

MultiMerge

(As of 3.1.001.07)

MultiMerge is a Merge pre-processor that combines the output of separate Merge runs into a single output file. At this time this program is used to create only Adaptive HTML output.

Html-Adaptive Processing

The Html-Adaptive processing results in an html output that will present distinct browser layouts depending on the width of the user's browser. It is typically used to allow mobile users a separate, more vertical view, of a web page that might otherwise become too small on a phone display.

With the MultiMerge adaptive processing, the form designer must create separate form layouts for each variation of the browser display they require. These can be in **separate DocOrigin xatw files** (one xatw file for each view), or **a single xatw** where each page in the xatw represents a different view. Up to 10 distinct views can be defined. In practice 2 or 3 are likely all that you will require.

Processing begins by MultiMerge internally invoking the DocOrigin **Merge** program separately for each of the form files (or pages if there is only one form file specified). The command line `-data`, `-config`, `-logfile` and `-mergeargs` are all passed to each of these Merge runs along with the individual form definition of a single html "view". All Merge runs get the same options except for the `-form` option.

The output from these separate runs is then restructured into a single output html file (the `-output` arg). This output file contains each of the distinct "views" together with some html logic that determines which view should be displayed to the user.

Html-Adaptive Template Files

The creation of the combined output file is controlled by a **MultiMerge Template** file - an html file with special **Placeholder** tags in it where the various pieces of information from the Merge runs are placed. For example, the output of the first view is placed in the **BODY1** placeholder, the second view in the **BODY2** placeholder, etc.

There are default files in the **DocOrigin/DO/Bin** folder that ship with DocOrigin. You can override these on the MultiMerge command line with the **-AdaptiveTemplate** option.

The default templates are named **Default-DOHtmlAdaptiveTemplate#.htm** where '#' is the number of views you specify in the **-form** argument. The most common template is **Default-DOHtmlAdaptiveTemplate2.htm**. If you decide to customize this, just copy this default file. Give it a new filename and store it in your overrides folder (DocOrigin/User/Overrides/).

MultiPart Output

MultiMerge also enables the generation of traditional multi-part documents. These are documents where 2 or more variants of the same document are created from a single set of data.

For MultiPart processing, the form designer must create separate form layouts for each "part" of the document they require. These can be in **separate DocOrigin xatw files** (one xatw file for each "part"), or a **single xatw** where each page in the xatw represents a different "part".

MultiPart processing internally invokes DocOrigin **Merge** separately for each of the form files (or pages if there is only one form file specified). The command line `-data`, `-config`, `-logfile` and `-mergeargs` are all passed to each of these Merge runs along with the individual form definition of a single html "view". All Merge runs get the same options except for the `-form` option. The result of these separate Merge runs is then combined to create a series of multi-part documents written to the `-output` file name.

The specific type of output - PDF, PCL, Postscript - must be specified using the `-action` command line argument. See the Command Options to the left.

See Also

[Command Options - MultiMerge](#)

Command Options

MultiMerge has the standard DocOrigin command line options as described in the [Common Command Line Options](#) section of this document. In particular, it requires that `-logfile` and `-message` be defined either explicitly or in one of the default PRM files.

The basic MultiMerge commands are similar to that of Merge. These work the same as they do in Merge - see [Command Options - Merge](#) for details.

- `-config` **filename** (defaults to DefaultHTM.prt)
- `-data` **filename** (required)
- `-form` **formfile/formfilelist** (required)
- `-logfile` **filename** (required)
- `-output` **filename** (required)

In addition, there are some MultiMerge-specific command options listed at the left.

-action

Type of MultiMerge action to perform.

Syntax

-action *actionType*

Description

Available *actionTypes*:

- *html-adaptive* - This create Adaptive Html output.
- *pdf-multipart* - Create a multi-part document with PDF output.
- *pcl-multipart* - Create a multi-part document with PCL (HP Laserjet) output.
- *ps2-multipart* - Create a multi-part document with Postscript output.

The default *actionType* is *html-adaptive*

See Also

[Command Line Processing](#)
[MultiMerge Command Options](#)

-adaptiveTemplate

Template used to create final output file of MultiMerge.

Syntax

-adaptiveTemplate *filename*

Description

This allows you to override the standard template used by MultiMerge. The default template is either one provided in the DocOrigin install or defined in your Overrides folder. See [Html-Adaptive Processing](#) for details.

See Also

[Command Line Processing](#)

[Command Options - MultiMerge](#)

-mergeargs

Additional parameters passed to Merge calls

Syntax

-mergeargs *args*

Description

This command allows the specification of any of the standard Merge args to be passed to each invocation of Merge from within MultiMerge.

The `-config`, `-data`, `-form`, `-logfile`, `-output` and `-trace` options do not need to be included in this list - they are set as normal MultiMerge command options and passed to Merge as required. You need only use `-mergeargs` if there are non-standard options to be sent to Merge.

See Also

[Command Line Processing](#)

[Command Options - MultiMerge](#)

MultiPage

(As of 3.2.001.01)

 MultiPage only works with DocOrigin-generated PDF files!!!

The MultiPage feature lets you output two or more logical PDF pages on a single physical PDF page.

Usage

```
MultiPage -in DocOrigin.pdf -out MultiPage.pdf [-ini MultiPage.ini]
```

There is **Default-MultiPage.ini** in the **DocOrigin/DO/Bin** folder. Please study this .ini file as it contains a lot of valuable information.

Before using MultiPage, you must define mandatory settings either in **\$O/MultiPage.ini** or in the custom MultiPage .ini file which can be passed to MultiPage via ini command line parameter.

There are several **MultiPage*.ini** examples in the DocOrigin **DO/Samples** folder.

You will find that you may choose from three "pre-configured" page sizes for the output (Letter, Legal, and A4).

The **mandatory settings** in the .ini are

- TargetPageSize=Letter|Legal|A4
- TargetOrientation=Portrait|Landscape
- GridCellsInX=Number of pages across
- GridCellsInY=Number of pages down

You can also choose to have a border drawn around the original pages. To do this, set the [Border] style, line type, line color, line thickness, and line inset accordingly.

You can generate MultiPage output directly from Merge by using Merge's run:: option in the -output argument, e.g.

```
-output "run::MultiPage -in %t -out MultiPage.pdf -ini MultiPage_A4_Portrait_2x2.ini"
```

MultiPage Command Options

MultiPage has the standard DocOrigin command line options as described in the [Common Command Line Options](#) section of this document. In particular, it requires that `-logfile` and `-message` be defined either explicitly or in one of the default PRM files.

- `-in filename` (required, DocOrigin generated input PDF)
- `-out filename` (required, MultiPage output PDF)
- `-ini filename` (optional, custom MultiPage spec which overrides `Default-MultiPage.ini` and `$/MultiPage.ini`)

Notify

(As of 3.0.003.13, Windows-only)

DocOriginNotify is a standalone feature that facilitates watching for the existence of certain files.

Once run, it will continue to monitor for the existence of files matching the file mask you have specified and will popup a "toast" message should any such files be detected. At any point you can view a list of the detected files and perform operations on each file; operations defined by you for a typical right-mouse-click context menu.

You may run multiple instances of DocOriginNotify, each specifying a different file mask to monitor.

Each instance of DocOriginNotify will have an icon in the Windows "systray". *[Well, that's the old term, that I think most of us still recognize.]*

Purpose/Uses

Jobs that are run via DocOriginFolderMonitor can direct their output to various locations. One such possibility is to direct output to a folder assigned to the intended recipient. These jobs may have been submitted by other people or directly from an ERP or CRM or some LOB application. If the recipient is running DocOriginNotify they can be alerted whenever output arrives in their folder, or perhaps even a folder assigned to a group of individuals. If the files being detected are the results of fillable form production or some other "task description" the perception of these monitored files as "completed output" or "work item" is in the eye of the beholder.

Under error conditions, output (or failed input) may be directed to certain system folders. By monitoring such folders an interested party can be warned of such happenings.

Infrastructure

The usual scenario is that of a DocOrigin server, and several client machines busily engaging in the overall company business. Jobs are sent to the DocOrigin server and output is produced.

DocOriginNotify is unusual in that it is often meant to be deployed on a client system, away from the DocOrigin server, on a system with no other DocOrigin software on it. Of course, DocOrigin can be used directly on the DocOrigin server as well, but that may not be its primary usage point.

At this time, there is no client install for DocOriginNotify. However, it requires very little. Simply extract the **DocOriginNotify_x32.zip** archive from the server's .../DO/Bin folder to some convenient location on each client machine whose user wishes to run DocOriginNotify. Note, that archived executable is 32bit so you may run it on older 32bit systems. It may require you to install "Visual C++ redistributable 2017" on a client machine, same thing you do when installing DocOrigin.

Configuring

When a user runs DocOriginNotify, they specify either a series of command line options and/or, much more likely, they specify the name of a .prm file that is a modified clone of the delivered **DocOriginNotify.prm** file. That file has copious comments within it to describe the options that are available.

Principle among those options is: `-fileMask`

That is a fully path-qualified file name mask which this instance of DocOriginNotify is meant to monitor. At present, these paths must be accessible, network-mounted folders.

See also the DocOriginNotify.prm file.

Command Options

Notify uses the standard DocOrigin command line option handling as described in [Common Command Line Options](#).

-context

Define a context menu to present when an identified file is right-clicked.

Syntax

```
-context{n} name, command
```

Description

This allows you to define a context menu to be presented when an item identified by the `-fileMask` is right-clicked on. `-context1` is very special as it also defines the action to be taken when the item is simply double-clicked. Each `-context{n}` item is defined in terms of a Windows command. As per the exemplar `DocOriginNotify.prm` file, the typical `-context1` context operation is to simply "Windows start" the file and let the program associated with the file's extension be run.

A 'delete the file' context item may be useful too so as to readily "be done with a task that you have completed".

As per the exemplar, it is useful to call command prompt files (.bat) so that the processing associated with a context item can be more readily tailored as ongoing needs change.

See Also

[Command Line Processing](#)
[Command Options - Notify](#)

-fileMask

File to look for.

Syntax

-fileMask *mask*

Description

A wildcard file mask such as -fileMask *\$U/Output/*.pdf* that indicates which files to scan for.

See Also

[Command Line Processing](#)
[Command Options - Notify](#)

-themeColor

Display color.

Syntax

-themeColor *color*

Description

By default, the toast comes up in a "bluish" color. If you have multiple DocOriginNotify instances running, you may wish to specify a different theme color so that you can recognize instantly just "which toast popped". Color is specified using the standard DocOrigin color specifications - see [Colors](#).

See Also

[Command Line Processing](#)
[Command Options - Notify](#)

-waitTime

Time, in seconds, to wait between checking to see if more files have arrived.

Syntax

-waitTime *seconds*

Description

Time, in seconds, to wait between checking to see if more files have arrived.

See Also

[Command Line Processing](#)
[Command Options - Notify](#)

PCLExtract

PCLExtract is a standalone tool to allow the **extraction** of documents from a DocOrigin PCL file.

Extracting Documents from a PCL

Given a hypothetical multi-document PCL named multiDoc.pcl you could extract document 7 from it by using:

```
DO PCLExtract -in multiDoc.pcl -docs 7 -out justDoc7.pcl
```

Rather than extracting only a single document, you could extract a comma-separated list of documents (e.g. `-docs "2,5,18,19,34"`). You can also specify a dash-separated range of document numbers. (e.g. `-docs "1,20-30,18"`). Or extract by document bookmarks (e.g. `Name=Fred OR CustomerID=123,124,125`).

Other options

PCLExtract contains a set of other options to work with PCL file. For instance: extract input data, count documents, etc.

✔ Hint: If you ever want to know the syntax for PCLExtract, just run it with no arguments.

PDFExtract

PDFExtract is a standalone tool to allow the **extract**ion of pages or documents from a DocOrigin "combined PDF", **or to combine** multiple documents from multiple PDFs into one PDF.

Combined PDFs

Merge very often deals with data streams that have multiple documents' worth of data in them. Merge can produce an individual PDF for each document (record, transaction) in the data, and/or it can produce a 'combined PDF' of multiple documents into a single PDF. The latter is often used for archiving, for sending out to a print service bureau, or with bookmarks for looking up documents.

A combined PDF is produced by using the `-PDFCombineDocuments Yes` directive. That is in fact the default setting in the `Default-PDF.prt` configuration file.

So now you could have PDFs with lots of documents in them. DocOrigin encodes special markers and a special index section in its PDFs. These markers allow PDFExtract to know where the document boundaries are and, with the document index, is able to have direct access to any given document.


This is quite different from other PDF software where all you have is a great number of pages with no built-in understanding of which pages belong to which document.

Combining PDFs using PDFExtract

Another way to get a 'combined PDF', i.e. a multi-document PDF, is to use PDFExtract in the following manner.

```
DO PDFExtract -in file1.pdf -in file2.pdf -out output.pdf
```

You can specify as many input PDF files as you like and they will all be combined into a single output PDF. Perhaps you would want to combine a 'Statement' and an invoice. Recall that DocOrigin JavaScript (in the form, or in a Job Processing script, or even standalone via RunScript) can use its `_run` function to run any application. A script could easily be crafted to run PDFExtract so as to combine PDF documents on the fly, and under scripting logic control.

 PDFExtract operates on only DocOrigin-produced PDF files.

Extracting Documents from a PDF

- ✓ Hint: If you ever want to know the syntax for PDFExtract, just run it with no arguments.
DO PDFExtract

Given a hypothetical combined PDF named multiDoc.pdf, you could extract document 7 from it by using:

```
DO PDFExtract -in multiDoc.pdf -docs 7 -out justDoc7.pdf
```

Rather than extracting only a single document, you could extract a comma-separated list of documents (e.g. `-docs "2,5,18,19,34"`). You can also specify a dash-separated range of document numbers. (e.g. `-docs "1,20-30,18"`)

Extracting Pages from a PDF

Extracting pages is identical to the above but with `-pages` used instead of `-docs`.

This is not meant to be page numbers within a document but pages in the entire multi-document PDF starting at 1 and going up to however far it goes. You can get the number of pages in a PDF via:

```
D0 PDFExtract -in multiDoc.pdf -pagecount Y
```

The selections done by `-docs` and `-pages` are totally independent. They are not an intersection of those criteria, nor a union of them. If those selections happened to overlap then such objects selected would come out twice. It is not recommended that you mix them.

Selection by Bookmark

When creating a multi-document PDF you will often specify bookmarks that are to be inserted into the PDF. See [_document.bookmark](#) Using that script function in your forms when you create the combined PDF, you can insert bookmarks of chosen names and values as applicable. Then, rather than having to know the ordinal number of a document you could select/extract by bookmark value.

```
DO PDFExtract -in multiDoc.pdf -docs "SalesRep=Sam"
```

View Bookmarks

You can get a very simple, but therefore easily able to be processed via script, list of all the bookmarks in a PDF by using:

```
DO PDFExtract -in multiDoc.pdf -dumpIndex -quiet > bookmarks.txt
```

or

```
DO PDFExtract -in multiDoc.pdf -dumpIndex -out bookmarks.txt
```

You would likely then use a simple script to read that output file and take whatever actions you desired to take. Do look at the Sample_Extract sample that is provided in the installation.

```
DO sample 2
```

Command Options

PDFExtract uses the standard DocOrigin command line option handling as described in [Common Command Line Options](#).

-attach

Add an attachment to the PDF.

Syntax

-attach *filename* [*;description*]

Description

filename is the file to attach.

description can be any text. It will be shown in PDF viewer as the description for the attachment file.

This option can be specified multiple times to PDFExtract so you can attach more than one file. You can attach any file, a typical example might be of a spreadsheet or some background document.

See Also

[Command Line Processing](#)

-docs

Select specific DocOrigin documents from the source PDF.

Syntax

-docs *name=value*

-docs *docNumber* [,*docNumber* [,*docNumber* ...]]

Description

Use the `-docs` option to select one or more DocOrigin documents from within a multi-document PDF generated by the Merge `-combineDocuments` option.

name=value command option specifies a name/value pair as defined by these bookmark features.

docNumber [,*docNumber* [,*docNumber* ...]] - Alternately you can specify the document as a list of document numbers - *1,2,6,10*... etc.

Documents are added to DocOrigin pdf files using the Bookmark feature - see [_document.bookmark](#)

See Also

[Command Line Processing](#)

[Command Options - PDFExtract](#)

[View Bookmarks](#)

-dumpIndex

Extract the DocOrigin Bookmark name/value table.

Syntax

-dumpIndex **Y|N**

Description

When DocOrigin generates a PDF it adds a special Bookmark index to the file. See [_document.bookmark](#). This option extracts that index to the output file.

See Also

[Command Line Processing](#)

[Command Options - PDFExtract](#)

-dumpObjects

Dump the internal PDF objects to a series of text files.

Syntax

`-dumpObjects` **Y|N**

Description

This option extracts all internal PDF objects into a series of .txt files. Each PDF object is written to its own file.

See Also

[Command Line Processing](#)

[Command Options - PDFExtract](#)

-in

Defines the input PDF file to process.

Syntax

-in *filename*

Description

This is the base file that PDFExtract works on.

See Also

[Command Line Processing](#)

[Command Options - PDFExtract](#)

-multiSelect

Select all matching documents with the PDF.

Syntax

`-multiSelect` **Y|N**

Description

When the `-docs` option is being used to select documents by name/value it is possible that more than one document within the PDF matches the criteria. `-multiSelect` determines whether to extract only the first one or all matches.

See Also

[Command Line Processing](#)

[Command Options - PDFExtract](#)

-onto

Defines one or more destination pages onto which the `-overlay` pages are overlaid.

Syntax

`-onto` *page#*

See Also

[Command Line Processing](#)

[Command Options - PDFExtract](#)

[-overlay](#)

-out

Output file for PDFExtract.

Syntax

-out *filename*

Description

The name of the file to be created by PDFExtract.

See Also

[Command Line Processing](#)

[Command Options - PDFExtract](#)

-overlay

Overlay another PDF over selected pages in the PDF.

Syntax

`-overlay filename`

Description

The pages of this file will be combined (overlaid) onto the source (`-in`) file. The `-onto` option can be used to specify which input pages to be overlaid.

See Also

[Command Line Processing](#)

[Command Options - PDFExtract](#)

[-onto](#)

-pageCount

Count the pages in this PDF.

Syntax

-pageCount **Y|N**

See Also

[Command Line Processing](#)

[Command Options - PDFExtract](#)

-pages

Select specific pages from the input PDF.

Syntax

-pages *page#* [,*page#* [,*page#* ...]]

Description

Extract these specific PDF pages from the file. You can specify a list of page numbers or a range such as **11-15** or combinations of the two.

See Also

[Command Line Processing](#)

[Command Options - PDFExtract](#)

-quiet

Suppress information messages to the console.

Syntax

-quiet **Y|N**

See Also

[Command Line Processing](#)

[Command Options - PDFExtract](#)

RunScript

RunScript is a tool to run JavaScript scripts. It executes the JavaScript script that is provided in *fileName*.

Syntax

```
RunScript -script fileName
```

Parameters

fileName is the file to run such as a WJS.

The script files, by convention, almost always have an extension of .wjs. The script files must be saved in UTF-8 format.

RunScript can be used to develop and test scripts that may eventually be part of one of the other DocOrigin applications. Or it may be used as a standalone application to carry out virtually any activity.

Since RunScript can invoke other applications via its `_run` object, it can be used to orchestrate business processes involving many steps. Note that RunScript can even invoke other instances of itself such that processing steps can be broken down into parts and then have a master script that runs each part, or runs each part conditionally. We find that technique particularly handy for running a suite of tests.

RunScript includes all of the DocOrigin features described below in Common DocOrigin Script Functions.

Predefined Vars

When RunScript executes, all the \$X settings that are defined either explicitly or by default on the command line are made available as pre-defined javascript variables. (All DocOrigin programs automatically include the **Default-DocOrigin.prt**). So, for example:

```
var fn = $E + "/Default-DocOrigin.prm";  
_message("fn=%s", fn); // displays "fn=c:/DocOrigin/D0/Bin/Default-DocOrigin.prm"
```

All the parameters set on the command line of RunScript are also available within the script as the javascript object `_job.command`. So for example:

```
_printf("script=%s", _job.command.script); // will display -script setting
```

RunScript allows you to set any command line options of your own. Consider the following file "myscript.wjs" contains

```
_printf("mydata=%s", _job.command.mydata); // display command option -mydata
```

Running `RunScript -script myscript.wjs -mydata=Fred.txt` will print "mydata=Fred.txt".

Command Options

RunScript has the standard DocOrigin command line options as described in the [Common Command Line Options](#) section of this document. In particular, it requires that `-logfile` and `-message` be defined either explicitly or in one of the default PRM files. You may also define any command line options of your own and retrieve their value within a script. For Example:

```
RunScript -script yourscrip.wjs -file yourfilename.txt
```

Within your script, you can fetch the value of `-file` using:

```
var file = _job.command.file; // set to "yourfilename.txt"
```

-allowDJScript

Use the built-in DocOrigin script processor.

Syntax

```
-allowDJScript Y|N
```

Description

By default, RunScript always uses the ECMA SpiderMonkey script engine. By setting `-allowDJScript Y` you can request that the optimized DocOrigin script engine be used instead. The DJ script engine is faster but does not support some of the less frequently used JavaScript features. If the DJ script engine does not recognize any JavaScript feature you may have used, RunScript will automatically revert to the more complete (but slower) SpiderMonkey processing.

See Also

[Command Line Processing](#)

Runscript -script

[Mandatory] Specify the JavaScript file to execute.

Syntax

-script *filename*

Description

This tells RunScript where to find your JavaScript to be executed.

See Also

[Command Line Processing](#)
[RunScript Command Options](#)

FormConvert

DocOrigin supplies a non-GUI program called "FormConvert" which can be used to convert IFD, XDP, or PDF files to XATW format. The same prerequisites apply for batch conversion as for conversion using DocOrigin Design, see [Form File Import/Conversion](#)

Command Options

The FormConvert command line options are simple:

```
FormConvert -in xxx.<ifd|xdp|pdf> [-out xxx.xatw] [-saveXdp xxx.xdp]
```

If you do not supply a -out then it will use the same base name as the input file. When converting ifd or pdf it can be useful to save intermediate xdp with -saveXdp option, see also [DO_SAVE_XDP](#).

-adjustLabelWidth

Adjust text label width during conversion.

Syntax

-adjustLabelWidth *value*

Description

Sometimes you may need to adjust text label width after conversion, e.g. to make it slightly bigger to avoid extra line wrap. This can be automated with this option. The parameter's value can be a number (then it means microns) or a number with a suffix (in, cm, mm, pt). It can be both positive and negative. If you want to use negative then use `-parm=value` syntax, so minus will not be treated as a start of a new option, e.g.

```
-adjustLabelWidth=200  
-adjustLabelWidth=-200  
-adjustLabelWidth=0.01in  
-adjustLabelWidth=-0.01in
```

See Also

[Form File Import/Conversion](#)

-config (FormConvert)

Option to specify DocOrigin configuration file used by FormConvert at XDP to XATW conversion step.

Syntax

-config *prt*

Description

Output configuration file (.prt) specifies information such as available fonts, page sizes, etc. Default configuration is .../D0/Config/Default-PDF.prt.

See Also

[-setConfig](#)

-containerShift

Option to set left and right shift of container border in the converted form.

Syntax

-containerShift *amount*

Description

When converting forms (*before 3.2.001.02*), DocOrigin tried to introduce 0.25in container shift if possible (recalculating the position of nested elements). (*As of 3.2.001.02*) Elements use original positions delegating all the required adjustments to the form designer. If you still want DocOrigin to shift the container's borders you can configure it using -ContainerShift option. When set, **FormConvert** will attempt to shift the container border from the page border. Value should be in DocOrigin units (microns, in, cm, mm, pt.) Default is 0.

See Also

[Form File Import/Conversion](#)

-fieldCorrection

The option controls global field synchronization used by ConvertIFDShell at IFD to XDP conversion step.

Syntax

-fieldCorrection *global|once*

Description

This is ConvertIFDShell parameter -fieldCorrection with value either *global* or *once*. If the value is *global*, it specifies that fields with the same names should be global. The default value is *once*.

See Also

[Form File Import/Conversion](#)

-ics

Option to specify Output Designer configuration file used by ConvertIFDShell at IFD to XDP conversion step. Same as `-target`.

Syntax

`-ics path`

Description

Output Designer configuration file (`.ics`) specifies information about available fonts. Default configuration is `pdf.ics`.

See Also

[Form File Import/Conversion](#)

`-target`

-importFilter

Option to control subforms import used by ConvertIFDShell at IFD to XDP conversion step.

Syntax

`-importFilter` *fieldsOnly* | *none*

Description

ConvertIFDShell parameter `-importFilter` with value either *none* or *fieldsOnly*. If value is *fieldsOnly* then all unnamed subforms are removed from the hierarchy. Default value is *none*.

See Also

[Form File Import/Conversion](#)

-in (FormConvert)

Option to specify FormConvert input file.

Syntax

-in *path*

Description

Input file can be either Adobe Output Designer Form (.ofd), or Adobe PDF (.pdf), or Adobe XML Form (.xdp).

See Also

[Form File Import/Conversion](#)

-out (FormConvert)

Syntax

-out *path*

Description

Output file is resultant DocOrigin Form (.xatw).

See Also

[Form File Import/Conversion](#)

-retainSpaces

Control spaces compression in the text labels during conversion.

Syntax

-retainSpaces **Y|N**

Description

This is an option to control the compression of multiple spaces in a row to a single space for text labels. Sometimes, source forms could use spaces for text alignment, and therefore it may be handy to keep them after conversion. Default is **N**, i.e. multiple spaces are replaced with a single space.

See Also

[Form File Import/Conversion](#)

-saveXdp

Option to specify output for intermediate Adobe XML Form file.

Syntax

-saveXdp *path*

Description

This option is useful for debugging purposes to analyze intermediate Adobe XML Form file produced by ConvertIFDShell at IFD to XDP conversion step, or ConvertPDF at PDF to XDP conversion step.

See Also

[Form File Import/Conversion](#)

-setConfig

Option to specify that configuration file name (file name only) should be stored in the resultant DocOrigin Form (.xatw).

Syntax

-setConfig **Y|N**

Description

If value is **Y**, then the name of DocOrigin configuration file used for conversion is stored in the resultant DocOrigin Form (.xatw) as this form configuration file. Default value is **N**.

See Also

[Form File Import/Conversion](#)

-target

Option to specify Output Designer configuration file used by ConvertIFDShell at IFD to XDP conversion step. Same as `-ics`.

Syntax

`-target path`

Description

Output Designer configuration file (.ics) specifies information about available fonts. Default configuration is `pdf.ics`.

See Also

[Form File Import/Conversion](#)

SendMail

DocOrigin supports the sending of emails through two main mechanisms:

- Using a Form template (`_email_5` for instance) that lays out the email header information as well as the actual email message body directly within Design. This is covered in more detail in the [Auto Email](#) section.
- Using the JavaScript `_sendmail` function to send emails. This can be used in all DocOrigin script-enabled functions: Merge, RunScript, FolderMonitor, etc.

Both schemes use the same underlying DocOriginSendMailServer executable to send the actual messages.

The DocOrigin `_sendmail` function relies on the use of the cURL external executable to interface with an email server (via SMTP) and send the actual messages. This package is provided in the DocOrigin Install for the Windows environment and is freely available online for other platforms.

Settings must be configured via an overridable `DocOriginSendMailServer.prm` file or via command line parameters (as of 3.2.001.01). Your SMTP host, access user/passwords, port number, and default "from" address must be configured in order for `_sendmail` to work.

When `_sendmail` is called you can specify whether the message is to be sent immediately or whether it can be queued to send later. This option is part of the `_sendmail` function parameter settings. This option is typically adjusted for performance reasons in certain email environments.

When messages are queued for sending, a DocOriginSendMailServer is automatically run to process these message files and do the actual email transmission via an SMTP email server. The queue option results in much better overall email throughput.

See the `_sendmail` function description for more details on using the JavaScript `_sendmail` function and [Auto Email](#) for details on embedding automatic emailing into your DocOrigin Design.

SendMail Configuration

SendMail must be specifically configured for each user environment. The default settings are defined in the `DefaultDocOriginSendMailServer.prm`. (Note that despite the "Server" at the end of this file name, the settings are in fact for both the immediate sending of emails and the queued sending of emails.) This default file is supplied with your installation. To make changes you should make a copy of any required sections of the file and store it in your `.../User/Overrides` folder under the name `DocOriginSendMailServer.prm`. This new file is automatically referenced after DocOrigin reads `Default-DocOriginSendMailServer.prm`. You can make any changes you like to the version of the file in `.../User/Overrides` - they will override the default version of the file.

The `DocOriginSendMailServer.prm` file is a text file and contains vital internal comments as to what each of the settings is for. You will, at the very least, need to change the following:

- `-smtp`host to be your email server
- `-smtp`user and `-smtp`password to your account
- and possibly `-smtp`port if your email server uses a different port number.
- You should also modify the `-from` from the default setting to be one of your email accounts. The `-from` can also be set explicitly on each call to `_sendmail`.

Alternatively (*as of 3.2.001.01*), for more flexibility, you may specify command-line parameters which will override settings found in the PRM file.

`DocOriginSendMailServer` will take the information found in the message scan folder, typically `.../User/Outbox`, establish a connection to your specified SMTP server, and send the message. Upon completion, the message files are deleted or moved per your configuration settings. When `DocOriginSendMailServer` has processed all messages from the scan folder it shuts down.

Immediate or Queued

Email may be sent either 'immediately' or 'queued'. Sending email immediately is rather slow since a connection to the email server needs to be made and the calling script must then wait until the mail is actually sent. For this reason, it is often preferred to send email in a queued fashion. This puts the needed files in a folder, usually `.../User/Outbox`, and a separate program is launched to look after the sending of the email (or possibly many emails); meanwhile, the calling script will carry on processing with no waiting.

The folder where queued emails are put is defined by the `-mailscanfolder` option as specified in the `DocOriginSendMailServer.prm` override file. Typically it is `.../User/Outbox`.

You specify your wish to send emails immediately via a `-sendmail immediate` directive. More likely, for performance reasons, you would specify `-sendmail queue` in order to queue the sending of email while your script carries on processing. These options can also be specified on a per email basis by supplying the `.queue` property as true or false in your `_sendmail` function call.

```
var args = {};
args.To = ...
args.Subject = ...
args.etcetera ...
args.queue = false; // Send this email immediately, I'm willing to wait
_sendmail(args);
```

Similarly, with the Auto Email system, set the Mail Options to either **queue:Y** or **queue:N**.

Messages that are sent immediately are processed by the `_sendmail` function itself. No scanning of folders is involved. `_sendmail` launches cURL on its own, waits for it to complete, and then does the necessary housekeeping.

Messages that are queued, are put, by `_sendmail`, into the `.../User/Outbox` folder. There is a pair of files. One has a `.eml` extension and contains the entire email package including any attachments. The other file has a `.scan` extension and contains the instructions for cURL to use. Once the files are placed in that folder, `_sendmail` will launch `DocOriginSendMailServer` so that it can process any messages that are in the nominated scan folder, while

the script that called **_sendmail** continues on without waiting. DocOriginSendMailServer scans the folder for ***.scan** files and processes each email accordingly.

Aside: As it happens, emails sent immediately do also put a pair of files in the **Outbox**. One is the usual .eml file and one has a .config extension. DocOriginSendMailServer never sees those files. `_sendmail` handles them directly by using cURL on them itself.

Creating but not Sending and Email (-hold)

You can tell the email system to hold an email and not send it automatically at all using the **-hold** options (`arg.hold = true`; in script, or **Mail Options** of **hold:Y** for the Auto Email feature) This causes the .eml file and an associated .scan file to be placed in a separate **/Outbox/Hold** subfolder. This can be handy if you are testing a form or script and you don't want to actually send it - just look at it. If you have a desktop email client (such as Outlook) you can double-click on the .eml file and see the email as it would be displayed if it had been sent.

If you move the .eml file and its associated .scan file back to the parent Outbox folder, it will then be sent the next time DocOriginSendMailServer is run. (It requires no parameters, so it is easy to run!)

Post-Send Housekeeping

If the transmission of a message is not possible due to some parameter error or email system error, the pair of files is (*as of 3.0.003.24*) moved to the subfolder as defined via the `-mailErrorFolder` directive in your **DocOriginSendMailServer.prm** file. The default is to go a sub-folder named **Error**. This folder will be immediately under the scanned folder, hence it is typically **.../User/Outbox/Error**. *Prior to 3.0.003.24*, the .scan (or .config) file would've been left in place but renamed to have a .ERR extension.

If the email send is successful then the pair of files will either be deleted or renamed/moved. Normally they would be deleted. This is controlled by the **-keep** option in the **DocOriginSendMailServer.prm** file. If `-keep` is not specified, it is assumed to have the value `No` and hence the successfully sent email files will be deleted. If you are not queuing the emails (i.e. `args.queue = false` this can also be controlled on a per `_sendmail` (or [Auto Email](#)) basis by specifying the `.keep` property as either `true` or `false`.

If you do choose to keep the files even after a successful send... *before 3.0.003.24*, the .scan file would've remained in place but be renamed to have a .DONE extension. *As of 3.0.003.24*, the pair of files would be moved to the subfolder identified by your `-mailSentFolder` option. By default, that has the value `Sent` and hence the files are moved to folder **../User/Outbox/Sent**.

During initial development, it may be useful to specify `-keep Yes` in order to look at the email files that were produced. However, if you do so, do remember to empty out that folder from time to time as folders with many many files can slow the system down. The folder name specification can have the usual date substitution `%` variables in it (e.g. `%U` is replaced with the current week number). In that way, multiple "Sent" folders can be created each covering a certain period of time. See [File Naming Conventions](#) for file/folder name syntax.

Prior to 3.0.003.24, if DocOriginSendMailServer encounters a timeout from the SMTP server, (either it's not available or may be overloaded), the message remains in the scan folder and DocOriginSendMailServer will attempt to resend it at a later time. The **-pause** setting below can be used to control when this resend is attempted.

As of 3.0.003.24, cURL is directed to retry the send `-retry` times. If it fails, the files are copied to the `-mailErrorFolder` folder.

Parallel Connections

Prior to 3.0.003.24, many instances of DocOriginSendMailServer may have been running simultaneously. However, their invocation of cURL resulted in only a single connection to the email server being made. Other overheads were encountered as well.

As of 3.0.003.24, only one instance of DocOriginSendMailServer is allowed to run. If `_sendmail` launches a second instance, that second instance will detect the first instance and simply exit. The one instance of DocOriginSendMailServer scans the outbox folder for .scan files and will launch up to **-maxProcesses** instances of cURL **in parallel** to process the messages. If the software generating the emails gets ahead of

DocOriginSendMailServer, the latter will find many email messages to send but will process them in a parallel fashion. There is a hard limit of 64 simultaneous connections, though your email server may have a lower limit.

As always, DocOriginSendMailServer exits when it finds no further email messages to send (i.e. no more .scan files). `_sendmail` launches DocOriginSendMailServer only after it has created the pair of files (**.eml**, **.scan**) in the outbox folder.

Option List

The following command line parameters control the DocOriginSendMailServer process. There is a **Default-DocOriginSendMailServer.prm** file located in the **DO/Bin** folder of your install. This prn file will automatically be loaded when DocOriginSendMailServer starts running. It is also loaded by any call to `_sendmail`. There are some default settings that come with your installation. You must make a copy of this file and place it in the **.../User/Overrides** folder under the name **DocOriginSendMailServer.prm** – do notice the dropping of the Default- prefix. This becomes your custom settings file. Put any overrides to the default settings in this Overrides file (*if one exists, be sure to remove the @@ reference at the bottom of the copied file*).

The following parameters are in **addition** to those listed in the `_sendmail()` function description and are specific to the Windows platform. Fields marked as MANDATORY must be defined either in the default parameter file or passed via a `_sendmail()` args setting or via an included parameter file.

⚠ You **MUST** update the **smtpHost** and related parameters so as to not use or rely on the docorigin.com server. As of 3.0.003.24 the default `-smtpHost`, `-smtpUser`, and `-smtpPassword` settings are fictitious and **MUST** be overridden.

Setting	Description
<code>-checkAddress</code> <i>Yes/No</i>	Do basic email address format validation.
<code>-encode</code> <i>password</i>	If DocOriginSendMailServer is run with this as its only command line parameter, (not from the prn file), an encrypted version of the supplied password is displayed. You can use this in conjunction with the <code>smtpPassword</code> setting below to hide the true user password.
<code>-from</code> <i>emailAddress</i>	The default "from" address to be used. The smtp server may require that this be a valid address for your email server. This setting can be overridden in the <code>_sendmail()</code> function. Using Microsoft Exchange? See the note in Sending Mail .
<code>-header</code> <i>header record</i>	Add this additional header to all emails sent. This setting can be overridden in the <code>_sendmail()</code> function.
<code>-logfile</code> <i>filename</i>	Standard DocOrigin logfile entries will be written to this file. We think it is a good idea to nominate a specific logfile for DocOriginSendMailServer log messages, rather than use the omnibus, ad hoc, \$L/DocOrigin.log . You probably won't want your DocOriginSendMailServer log messages interleaved with other simultaneously running processes. MANDATORY
<code>-message</code> <i>filename</i>	Standard DocOrigin error message file. MANDATORY, but automatically supplied in the DefaultDocOrigin.prm file.
<code>-mailscanfolder</code> <i>fileFolder</i>	This is the name of a folder where the messages are stored by calls to <code>_sendmail()</code> in Merge etc. MANDATORY
<code>-pause</code> <i>seconds</i>	This causes the DocOriginSendMailServer to wait a few seconds before trying to resend messages that have failed to send because of an smtp timeout. The default is 60 seconds. (<i>Obsolete as of 3.0.003.24</i>)

Setting	Description
-proxyport <i>number</i>	If you are using an SMTP server that requires a proxy or SSL link, you must supply a ProxyPort number. This is a port# on your computer and can be any currently unused number such as 8801.
-smtphost <i>hostname</i>	This is the SMTPHost name provided by the email server. MANDATORY
-smtpuser <i>name</i>	Many SMTP servers require authentication - a way to ensure that only valid users of their mail server can use it for sending messages. Typically this can be any email user/password combination that has been defined on the server. If your server does not require authentication override this setting with -smtpuser= (nothing after the equals sign).
-smtpPassword <i>password</i>	The associated password for the SMTPUser setting. The password can be entered in either plain (readable) text or as a specially encrypted string to protect it from casual viewing. If the password is entered in plain (un-encrypted) text, it must be preceded by an exclamation mark as in: <pre>-smtppassword !mypoorsecret</pre> <p>To create an encrypted password string you must run the DocOriginSendMailServer program in a command shell or bat file with the command line option of -encode followed by your password. DocOriginSendMailServer will display the encrypted string that you should then use with this smtppassword option. If no password is required, override this setting with -smtppassword=</p>
-smtpport <i>number</i>	This is the SMTP port# specified by your SMTP service provider. Often this is port 25, but many servers do use other port numbers. MANDATORY
-maxProcesses <i>number</i>	DocOriginSendMailServer will employ multiple parallel connections to your email server, by invoking multiple instances of cURL, one per message. This has a dramatic effect on email send performance. The maximum <i>number</i> of processes is 64. A value of 20 seems to perform quite well.
-maxRate <i>number</i>	If set, message rate is limited by the <i>number</i> per minute. Throttling state is maintained via "throttling.state" file in mail scan folder (usually \$U/Outbox). (As of 3.1.002.06)
-queue <i>true/false (or Yes/No)</i>	This tells _sendmail whether to run cURL on its own or give the email files to DocOriginSendMailServer's outbox queue.
-keep <i>No/Yes</i>	This tells DocOriginSendMailServer whether to delete the .eml package after it is sent, or rename it to a .DONE extension (as of 3.0.003.24, the files are moved to the -mailsentfolder folder). Leaving -keep out entirely has the effect of a No setting. Handy for testing / debugging. See also the args.hold = true; argument in _sendmail. It is quite different in that it holds the email from being sent, so you can investigate the packages being created.
-hold <i>No/Yes</i>	This tells _sendmail to <u>not</u> launch DocOriginSendMailServer after producing the files in the outbox. Generally, this is for testing purposes only, wherein you generate the email messages but don't send them. This gives you a chance to look to see if your email message files are being created, and created properly, and allows you, if desired, to invoke DocOriginSendMailServer explicitly to actually send the emails at some point in time.

Setting	Description
-testMail <i>emailAddress</i>	If set, all To, CC, and BCC values are set to this email address. This allows you to redirect all mail to yourself when testing your DocOrigin-powered solution. This can be easier than modifying the data stream for testing.
-trace <i>filename</i>	If set, additional trace information will be stored in the named file. If the filename is omitted, a Trace.txt file will be created in the same folder as the <code>-logfile</code> entry.

See Also

[_sendmail](#)
[Auto Email](#)

Sendmail Verification

The DocOrigin sends mails via 3rd party tool called curl (<https://curl.haxx.se/>).

In order to verify that you can send emails, we provide you with the following simple test steps:

1. Store the following to an **email.txt** file, adjusting as needed:

```
From: John Smith <john@example.com>
To: Joe Smith <smith@example.com>
Subject: an example.com example email
Date: Mon, 7 Nov 2016 08:45:16
```

```
Dear Joe,
Welcome to this example email. What a lovely day.
```

2. Execute the following command after adjusting the SMTP server, user, password, and port options:

```
C:\DocOrigin\DO\Bin\curl smtp://mail.example.com:587 -u SMTP_USER:SMTP_PASS
--mail-from myself@example.com --mail-rcpt receiver@example.com
--upload-file email.txt -v --ssl
```

3. Note, you may or may not need to add additional security flags, see <https://everything.curl.dev/usingcurl/smtp> for details.

In a short period of time, you should get an email into your inbox.

If you don't get mail, you either didn't specify parameters correctly or your network configuration is blocking you from doing that.

Signing and Encrypting mail

(As of 3.2.001.02)

DocOrigin supports email signing and encryption. External tools may be used to do the actual signing. We propose to use OpenSSL v.1.1.1 tool since this one is able to handle S/MIME v3.1 mail. You should get this tool and make it available to DocOrigin. You may do this by adding it to PATH or specify the full path to OpenSSL in DocOrigin config as we'll show below.

Note, signing and encryption are independent tasks. While you may do encryption it is unlikely that you need it because to encrypt mail for say 1000 different recipients you need to have a 1000 public keys. But for signing, you need to have only one (your own) digital ID and all your recipients should trust it.

That is what you need to do to organize mail signing:

1. Get your digital ID. If you do not have one there are many ways to do this, you may just google "create digital id certificate". In our demo we will use Signer.cer and password-protected Signer.private.key in PEM format.
2. If you are going to encrypt a mail for some user then get that user certificate. In our demo, we will use User.cer in PEM format.
3. Get the OpenSSL tool. In our demo, we will use the one which is shipped with GIT - "**c:\Program Files\Git\usr\bin\openssl.exe**"
4. Specify signing and encryption options in your scripting. See the demo below.

Imagine that you put your keys in the same folder along with the following script:

```
var args = new Object();
args.from = "signer@domain.com";
args.subject = "SubjectTest";
args.to = "user@domain.com";
args.text = "messageBody";
args.signingParams = "-signer Signer.cer -inkey Signer.private.key -passin
pass:XXXYourPassXXX";
args.encryptionParams = "User.cer";

_sendmail(args);
```

Notice the last two parameters: **signingParams** and **encryptionParams**. If those are specified then the corresponding routine is activated. The rest of the settings (which are unlikely to be changed) are located in **Default-DocOriginSendMailServer.prm** file.

i If you are on Linux, you may want to override the default "`-signingTool $P/openssl.exe`" and "`-encryptionTool $P/openssl.exe`" from **Default-DocOriginSendMailServer.prm** file.

When you run that script (with parameter `$P="c:\Program Files\Git\usr\bin"`) your user should get signed and encrypted mail. The look and support will vary from one mail client to another. But for example in Outlook you should see "lock" and "red ribbon" badges on the top right, you may click on them for details.

You are free to use another signing tool or customize all parameters, see <https://www.openssl.org/docs/man1.1.1/man1/cms.html> for details.

i Microsoft Outlook Example

If you are using encryption then your recipient should add his private key to mail client. For example in Outlook it is under **File > Options > TrustCenter > Settings > EmailSecurity > DigitalIDs > Import...**

Email on Linux


DocOrigin supports the `_sendmail` function on Linux. This support is accomplished through the use of the extremely popular **curl** facility. In general, Linux OSes include curl in their distributions. curl supports many different protocols. For sending mail, the **smtp** protocol must be supported. The version of curl installed on your Linux OS may or may not include support for the smtp protocol. If it does not, you will need to update your version of curl to a version that does support **smtp**.

At the time of this writing, RHEL 6 (Red Hat Enterprise Linux 6) shipped curl version 7.19.7, and that does not include support for **smtp**. At present, the current version of curl is 7.37.0. curl has supported **smtp** as far back as version 7.21. RHEL 7 ships a curl (7.29) that does support smtp.

curl is available for free from [The curl Download site](#).

curl is available for many platforms. The most popular Linux platform for DocOrigin is commercially supported Red Hat or its community-supported mirror: CentOS.

Nothing remains static. Linux variations and feature development are constant. DocOrigin cannot distribute curl and the libraries it uses for every platform and version. Distribution licensing is also very complicated, but curl is free for you to download, install and use.

 Before getting too engrossed in these details, perhaps you should skip down to the 'Updating cURL via yum' topic that follows this.

At the curl download site, you can use the Download Wizard or search the long table for your OS. At least for RHEL6/CentOS6, this will lead you to another page with an incredibly long list of versions available for download.

CAUTION

The detailed version numbers cited here change often. Verify the latest numbers at the applicable download site.

If you are after the latest version, 7.37.0, for RHEL6, 64-bit you will arrive at:

http://mirror.city-fan.org/ftp/contrib/sysutils/Mirroring/curl-7.37.1-3.0.cf.rhel6.x86_64.rpm

Download it, or use wget to fetch it to your target system.

On the same page is the corresponding version of libcurl:

http://mirror.city-fan.org/ftp/contrib/sysutils/Mirroring/libcurl-7.37.1-3.0.cf.rhel6.x86_64.rpm

Download it, or use wget to fetch it to your target system.

The comments at the top of this page of links inform you that curl needs some additional libraries: c-ares, libidn, libmetalink, and libssh2. And it gives you a location to fetch those libraries:

<http://mirror.city-fan.org/ftp/contrib/libraries/>

When you go to that page you can find the RHEL6, x86_64 versions of those libraries. Note libidn versions stop at RHEL5. That's because as of RHEL6, that library is included with the OS. But the rest are available as:

http://mirror.city-fan.org/ftp/contrib/libraries/c-ares-1.10.0-3.0.cf.rhel6.x86_64.rpm

http://mirror.city-fan.org/ftp/contrib/libraries/libmetalink-0.1.2-6.rhel6.x86_64.rpm

http://mirror.city-fan.org/ftp/contrib/libraries/libssh2-1.4.3-15.0.cf.rhel6.x86_64.rpm

There are lots of other libraries. Try not to get distracted. As time moves on, depending on your OS and its version, you will likely need to select slightly different rpm files.

Having downloaded the required rpm files, you now need to install them. This will require root privileges as it does install into **/usr/bin** and **/usr/lib64**.

```

cd /tmp
wget http://mirror.city-fan.org/ftp/contrib/sysutils/Mirroring/
curl-7.37.1-3.0.cf.rhel6.x86_64.rpm
wget http://mirror.city-fan.org/ftp/contrib/sysutils/Mirroring/
libcurl-7.37.1-3.0.cf.rhel6.x86_64.rpm
wget http://mirror.city-fan.org/ftp/contrib/libraries/c-
ares-1.10.0-3.0.cf.rhel6.x86_64.rpm
wget http://mirror.city-fan.org/ftp/contrib/libraries/libmetalink-0.1.2-6.rhel6.x86_64.rpm
wget http://mirror.city-fan.org/ftp/contrib/libraries/
libssh2-1.4.3-15.0.cf.rhel6.x86_64.rpm
#
sudo rpm -Uvh c-ares*.rpm
sudo rpm -Uvh libmetalink*.rpm
sudo rpm -Uvh libssh2*.rpm
sudo rpm -Uvh curl*.rpm libcurl*.rpm

```

Of course, you will have to pay attention to all messages. Perhaps it will report the need for other libraries which you will have to resolve for your OS and version. At the end of it all, you should have a **/usr/bin/curl**. Perhaps you will get a message about an installed library being incompatible. One recourse is to uninstall (erase) it. But that's at your risk. E.g.

```
sudo yum erase libcurl-devel-7.19.7-37.el6_4.x86_64 --assumeyes
```

A good test is to run:

```
/usr/bin/curl --version
```

and to look at the version of curl and the supported protocols that it supports.

Once curl is installed, please cd to wherever you have DocOrigin Merge installed. Typically, `/var/DocOrigin/DO/Bin` but that may vary depending on your version of DocOrigin. If you have an older version of DocOrigin, then you must:

```

cd the-location-of-DocOrigin-Merge
rm -f curl
rm -f libcurl*
ln -s /usr/bin/curl curl

```

That will provide a symbolic link to the just installed curl and it will remove older, local DocOrigin versions of **libcurl**, if any exist.

At that point, you should be able to use the DocOrigin `_sendmail` function. If you have a post 3.0.003.15 version of DocOrigin installed you may wish to run the `email.sh` shell script that is in `.../DO/Bin` for it to do some checking on your email support via curl.

Updating cURL via yum

The posting at <http://serverfault.com/questions/321321/upgrade-curl-to-latest-on-centos> provides a potentially easier way. But again, things change. Use this as only a solution that worked 'at one time'. By using yum you don't have to worry about dependencies; it's all worked out for you.

```

sudo yum clean all
cd /etc/yum.repos.d
sudo gedit city-fan.repo

```

Put in the following content, but do update that baseurl. Perhaps you use `rhel5`, for example/

```
[CityFan]
name=City Fan Repo
baseurl=http://www.city-fan.org/ftp/contrib/yum-repo/rhel6/x86\_64/
enabled=1
gpgcheck=0
```

Save.

```
sudo yum install curl
```

And it loads and installs whatever is necessary. Booyah!

Session

DocOrigin Inter-process Communication

(As of 3.1.002.10)


DocOrigin maintains an internal mechanism for passing information between various DocOrigin programs that are being spawned internally or explicitly using the DocOrigin scripting functions. This mechanism is called Session.

A Session is any chain of DocOrigin programs that run to completion. Such as when using FolderMonitor to spawn Merge to generate a document. Or RunScript calling Merge, or perhaps other DocOrigin tasks. The Session mechanism provides an alternate way to pass data between these DocOrigin tasks.

You can use script to access the current Session and read or write entries to it. So, for example, if you had a RunScript program you launched which then called Merge to create a document, you could arrange for Merge to add some entries to the Session, which you could then access back in your Runscript file. Or you could have your RunScript program add some additional information to the Session before running Merge and be able to retrieve that information within the Merge run.

Session File Format

Session information is stored as basic name:value pairs, similar to ini/profile entries or the cache mechanism of Merge. Values are currently strings or numbers. Names are standard alphanumeric names. DocOrigin does have some internally generated names such as `_LastApp` and `_LastUpdated` which are preceded by the underscore character - so it is best that you do not create names with a leading underscore.

 Note: There is no need for you to know the internal format of the session file, or even where it is located. The format could change at any time. Use `_session` to access it.

Script access to Session data

Within DocOrigin tasks which support JavaScript, you can set Session values or fetch Session values using the `_session` functions. See `_session` for details.

Merge Command Line Options

DocOrigin Merge has a command option called **-setSession** which tells Merge to save named field values to the session file. You can specify a single field name to be saved or a semicolon-separated list of field names, e.g.

```
-setSession "field1;field2;field3"
```

Specifying the wildcard

```
-setSession *
```

will save all form fields to the session file. This means that you can return some or all of the document field name/value pairs to the task that called Merge. In all cases, it is the leaf node name that is saved. E.g. City, not ShipTo.City or BillTo.City. Also / consequently it is the **last** instance of each field that is saved/passed along.

Typically Merge is called in a Job Processing Script (JPS) and thus those returned session variables (*fyi: sometimes we call them settings*) are available not only to the JPS but also to any subsequent DocOrigin programs called by that JPS.

Using the command

```
-setSession *metadata
```

will cause selected metadata fields such as total DocumentCount, PageCount, and SheetCount to also be stored in the Session.

Hint: Use the `_session.object()` script function to look at all the names/values saved for the session.

See Also

[_session scripting](#)

[Auto/Embedded Fields](#)

See the [\[!Session xxx\]](#) embedded field

Command Line Processing

All of the non-interactive programs in the DocOrigin program suite (i.e. other than Design, FilterEditor and ConfigEditor) use a common command-line syntax and set of defaults. See [Invoking Command Line Apps](#)

Programs that run from a command line have the following basic syntax:

```
ProgramName -option1 value1 -option2 value2 ...
```

Option names are documented in the [Common Command Line Options](#) as well as the Command Line Options of individual programs - see [Command Options - Merge](#) for instance. Generally, the option names can be mixed upper and lower case and are case insensitive. The following two are equivalent:

```
Merge -PDFCombineDocuments Yes ...
Merge -pdfcombineddocuments yes ...
```

All **Option** names and **values** are separated by one or more blanks. The **values** must be surrounded by double-quotes if they contain embedded blanks. The syntax **Option=value** can also be used. In that case, it is likely best to surround that whole pair with quotes.

⚠ When a quoted string starts with a dash, an equals sign must be used so that DocOrigin does not interpret the value as a new option. For example:
`-mergeargs="-cache mail:no -profile $$F/my_lang.ini -documentTag 1"`

ℹ If the same -option name occurs multiple times on the command line only the last setting takes effect. This allows you to override default settings made in the Default PRM files (see [Indirect Parameter Files](#)). Note also that the `-parm`, `-cache`, and `-filter`, (and as of 3.1.001.26 `-attachment`) option settings of Merge are exceptions — multiple occurrences of those options are all retained. Other exceptions will be noted in the documentation.

Indirect Parameter Files (.prm files)

As well as listing the command options directly on the command line, groups of them can be stored in special parameter files and referenced on the command line:

```
ProgramName @myoptions.prm -option value - @mysecondoptions.prm -
```

Any number of indirect parameter files can be specified. The use of these .prm files can simplify the setup of applications — such as Merge — which have large numbers of parameters. The .prm file is a plain text file in which each option/value pair set is listed on a separate line.

Any line whose first real (non-whitespace) character is either a *, ;, or a [is ignored (treated as a comment). Also, blank lines are ignored. As an example:

```
* Test file for myform Merge
*
-form C:/DocOrigin/User/Forms/myform.xatw
-data C:/DocOrigin/User/Forms/mydata.xml
-output C:/DocOrigin/User/Output/myform.pdf
```

Note that double-quotes around values with embedded blanks become optional in parameter files. The above note applies to simple name/value options. In situations of compound options such as the `-filter` option, values of sub-options, e.g. `-split=""^page 1"`, will need quoting.

These command files can also include other command files by using the syntax `@filename` within an existing command file.

Conditional PRM Files

You may also use the syntax `@@filename` to reference an indirect .prm file. An `@@` reference is conditional — if the file does not exist no error is reported, it is simply ignored. You will see these in many of the default .prm files described below.

Text Substitution

(As of 3.1.002.04)

Indirect files support a mechanism to create shorthand names for longer paths or other text that is often repeated within the files. The `-define` command is used for this purpose

```
-define mypath c:/DocOrigin/User/Forms/Mytests

-form {mypath}/myform.xatw
-data {mypath}/mydata.xml
-output {mypath}/myform.pdf
```

Everywhere within the command file processing where a name is found between {braces} will be replaced by the text defined by an earlier `-define` statement. If the name within the braces is not found, the original text (and braces) will remain as-is.

Default Parameter files

Whenever any DocOrigin application starts it will look for a special default parameter file, `Default-DocOrigin.prm`, for options to be included in the command line automatically.

It will then look for an application-specific default parameter file. That file's name is `Default-` followed by the application name (e.g. Merge), and a `.prm` file extension. This `Default-` `.prm` file is located in the same file folder as the program itself.

For example, with a standard DocOrigin install, Merge is located at:

```
C:/DocOrigin/DO/Bin/Merge.exe
```

and its associated *Default prm* file is at:

```
C:/DocOrigin/DO/Bin/Default-Merge.prm
```

These files are optional. If the file is present, the parameters in the file will be processed before any others on the command line.

Default parameter files can be useful in defining site-wide default configuration of email options, standard logfiles, error message files, etc. Note that certain programs that are run automatically in the background such as the DocOriginSendmailServer program will rely on its default parameter file for its required settings — in this case, SMTP settings, etc.

The standard DocOrigin install creates Default `.prm` files for most of the DocOrigin applications. These files are *readonly* when installed. They **WILL be overwritten** when you re-install DocOrigin.

DocOrigin allows you to set up your own set of default settings. *Prior to 3.1.001.23*, the default `.prm` files ended with a *soft* `@@` reference to a similarly named `.prm` file e.g. `@@$O/Merge.prm`, in the `$O`, i.e. **.../User/Overrides**, folder. *Starting in 3.1.001.23*, no ending `@@` line is required or desired. Instead, whenever a DocOrigin executable loads a `.prm` file with a `Default-` prefix in its name, and residing in the `DO/Bin` folder, then that program will automatically look for a possible user override `.prm` file of the same name, but without the `Default-` prefix in the `$O` folder.

FYI: This was automated because users, to create their override file, would copy the entire `Default-xxxx.prm` file to `$O` and would forget to remove the trailing `@@$O` line. The result was a near endless loop that made program start times soar.

The `User/Overrides` folder is **NOT** overwritten when DocOrigin is re-installed. This allows developers to conveniently alter or add additional default settings without modifying the standard `Default-ApplicationName.prm` file.

Design.prm

Define settings used by Design by adding the Design.prm file to their /User/Overrides folders (\$O).

Data File Filter

Syntax

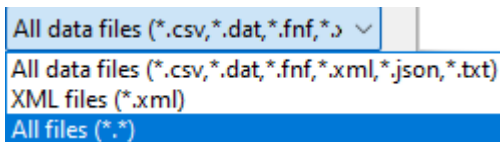
-DataFileFilter=*File Type Description to Display* | *Filter(s) with wildcard*

Example

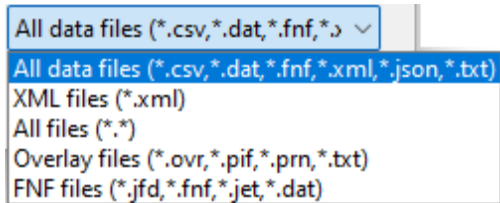
```
-DataFileFilter=Overlay files (*.ovr,*.pif,*.prn,*.txt)|*.ovr;*.pif;*.prn;*.txt|FNF files
(*.jfd,*.fnf,*.jet,*.dat)|*.jfd;*.fnf;*.jet;*.dat|
```

Description

Often, users have custom file extensions for their data. When developing, that extra step of selecting the **All Files (*.*)** filter or manually entering the wildcard and data file extension such as (*.JET) can become taxing.



Add the DataFileFilter entry to get a customer display. The result of the Example is shown below.



Form Explorer Font Size

Syntax

-FEPointSize=*point size*

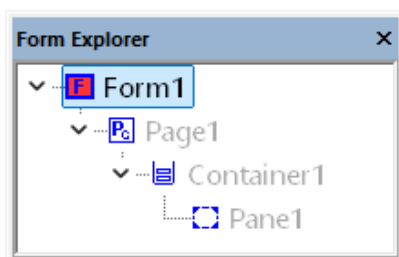
Example

```
-FEPointSize 12
```

The default is 8 points.

Description

Change the point size of the Form Explorer. This is useful if you have trouble reading the objects or clicking on the expand and collapse arrows.




Order of Precedence

Merge command-line options can come from the configuration file (.prt), the command line, or a command file. Many options can appear in more than one of these places. Some of the options may be overridden at runtime by JavaScript calls. The order of precedence is:

1. Runtime javascript overrides all other sources of command-line options.
2. The command line and command file(s).
3. The configuration file.

Command-line options are processed sequentially, left-to-right. A command file reference (an @parmFile.prm option) is expanded when it is encountered. It is exactly as if all entries in that file are directly in the command line, replacing the reference itself. If a command-line option is set more than once, the last setting takes precedence.

 In a sense, \$X option definitions (see [\\$X String Substitutions](#)) are *executed immediately*. That is, as soon as a \$X *path-of-choice* option is encountered, the meaning of \$X is immediately changed. This means that subsequent use of something like @@\$X/myParms.prm will employ the most recent definition of \$X.

The use of the "@\$X/..." example above is reflective of an *active* command-line option, not just a static setting. One could set "-form \$X/myForm.xatw" and it matters not what the value of \$X is at the time that setting is recorded. It will take on the value of \$X as it happens to be at the time that the form file is opened. Settings just 'set'. @ and @@ options are *active*; they take an action; they employ the latest \$X value option settings in effect, at the time of taking the action.

\$X type options are usually used in file names/paths. The resolution of those \$X type options is not made until the nominated file name is about to be opened. We call that operation 'hardening the path'. That hardening is done at the last possible instant so that files can be referenced more flexibly. See also [_file.resolveName](#).

There is one other command-line option that has immediate effect, as the options themselves are being processed. That is the **-cd newFolder** option. If you were to supply **-cd myProjectfolder** on the command line then a subsequent reference to **@@myProjectParms.prm** would look for that file in the most recently cd'ed to folder.

Invoking Command Line Apps

First of all, customers do not usually run DocOrigin Merge directly by keying a command on the command line. *Truly, with some other applications, notably RunScript, that is more likely. See the Invoking RunScript section below.* Merge though, is generally run, in development mode, via a *preview* operation from within Design, or in production mode, via FolderMonitor's detection of a data file to process and subsequent invocation of a Job Processing Script. Customers also have the option of launching Merge from their own application code. But it is possible, from time to time, that you will want to invoke a DocOrigin application directly from the command line. There is no mystery:

Programs that run from a command line have the following basic syntax:

```
programName -option1 value1 -option2 value2 ...
```

.prm Files

First a reminder about .prm files. They are extremely useful. Yes, there is a bit of indirection in their use — instead of putting the parameters on the command line itself, you put them in a file and simply reference the .prm file as-is:

```
programName @myParams.prm
```

Having created that .prm file you so often want to go back and change it a little, this way and that, back and forth, test after test. That is so much easier to do than re-entering a full command line. Before/While going down the road of planning typed-in command lines, do keep considering those super flexible .prm files. They let you put in comments, you can nest them, you can escape much of the tyranny of parameter quoting, each line is not super-wide so you can grasp its meaning so much more quickly. When thinking about invoking Command-Line Apps, think prm files.

Convenience Invocation

The following is about mere convenience features that you may choose to use and which provide us a means to hide away the details of where you may have happened to install DocOrigin. This really has no place in production usage but might be useful during development.

... On Windows

DO_ROOT

When DocOrigin is installed, it does update your system and current user PATH variables with `.../DO/Bin` folder. Also it adds an environment variable named **DO_ROOT** to your system. It contains the fully qualified path to where you installed DocOrigin. Our favorite such place is: `C:\DocOrigin`.

DO.bat

We also deliver a convenience .bat file named **DO.bat**. It is deposited into the `.../DO/Bin` folder of your chosen installation folder. However, we urge you to copy it out to somewhere in your PATH. The end result then is that at a Windows command prompt you can code:

```
DO Merge merge-params-as-needed
```

That's pretty easy.

There's not really much to it. In fact you could've taken advantage of the fact that the DO_ROOT environment variable was defined and instead code:

```
"%DO_ROOT%\DO\Bin\Merge" merge-params-as-needed
```

But the needed quotes, \DO\Bin, and %s are pretty ugly and tedious.

Naturally, this applies to other apps such as RunScript, uconv, ConvertDatToXml, ... whatever it is that you wish to run in a standalone, command line, way.

DO.bat Parameter Count Limit

There is a caveat here. DO.bat is just a plain old .bat file. Have a read. It is limited by the arcane wonders of Windows command prompt processing. Normally that means that it can reference only 10 parameters, %0 through %9. (And the %0 is 'DO' itself!). As of 3.0.003.28, that limit has been brute force increased to 29 parameters. If you are in the habit of typing in command lines with that many parameters then I suggest that you explore other methodologies.

As many parameters to DocOrigin applications are of the format -name=value be aware that that equal sign, unless quoted as "-name=value", will come through as a blank, and hence the construct will count as two parameters.

Again, remember .prm files. A real treat is:

```
DO app @myParms.prm
```

there is no parameter overflow there!

Other DO.bat Features

DO.bat has in it recognition of some special options. These allow it to more easily take you to:

Usage	Syntax
The HTML documentation that is installed on your system	DO doc
The samples	DO sample
A specific sample	DO sample 2
Invoke the sample fm.bat file that some people use during development to test out their usage of FolderMonitor.	DO fm start stop list
Invoke a .bat or .exe file, but check for an override in the .../User/Overrides folder before using the .../DO/Bin folder.	DO xxx.bat

... On Linux

DO_ROOT, LD_LIBRARY_PATH

When DocOrigin is installed, it does not use root access. It does not set any environment variables. However, we really do like to have an environment variable named: **DO_ROOT**. It is to contain the fully qualified path to where you installed DocOrigin. Our favorite such place is: /var/DocOrigin3.0.

We also require that the LD_LIBRARY_PATH be augmented to include the .../DO/Bin folder of the chosen installation location.

We accomplish this by supplying a shell script named DO. That is provided in the usual .../DO/Bin folder. At install time it is dynamically constructed and contains the name of the installation folder.

 That's why you really, really need to run the POSTINSTALL shell script after you do an install.

We urge you to copy that DO shell script somewhere into your PATH. If you put it in your PATH then you don't have to continually give a fully qualified path name in order to invoke it. The end result then is that at a shell prompt you can code:

DO Merge *merge-parms-as-needed*

That's pretty easy.

That will define and export DO_ROOT every time and will adjust the LD_LIBRARY_PATH as needed before invoking the named application.

Other DO Shell Script Features

The DO shell script has in it recognition of some special options. These allow it to more easily take you to:

Usage	Syntax
The HTML documentation that is installed on your system	DO doc
The samples	DO sample
A specific sample	DO sample 2
Invoke the sample fm shell script that some people use during development to test out their usage of FolderMonitor.	DO fm start stop list
Start up the viewer (often Firefox) associated with a file — e.g. a PDF or HTML file	DO viewer file name

It may be that you will want to customize the DO shell script to better address your recurring development desires.

Invoking RunScript

Something of a personal digression...

In my experience, during development, I often want to run RunScript standalone from the command line. It's just too handy! I do this A LOT. For me, typing out:

```
DO RunScript "-script=abc.wjs" -form xxx.xatw -in vvv.txt ...
```

is just too much effort. Most of the time, during development, my script has some hard-coded test values as defaults, at least until I get by the initial prototype working. My most common invocation of RunScript is as follows:

```
rs foo
```

Where `foo.wjs` is the script I am developing. It may seem an absurd amount of abbreviation, but did I tell you that I run this A LOT? The `rs.bat` script in my path is awfully simple, but it affords me a place to do all the things that I want to do every time. I like to delete all my log files. It just makes it easier to cope. My `rs.bat` file allows me to specify or not specify the `.wjs` extension. I could worry about more parameters or other features but this is simple and I like simple.

My rs.bat

```
@setlocal
@echo off
if exist "%DO_ROOT%\User\Logs\*.log" del /f "%DO_ROOT%\User\Logs\*.log"
set script=%~1
set ext=%~x1
if "%ext%" == "" set script=%script%.wjs
"%DO_ROOT%\DO\Bin\RunScript" -script="%script%" %2 %3 %4 %5 %6 %7 %8 %9
```

On Linux, the least I can do is to define an alias of:

```
alias rs='/var/DocOrigin3.0/D0/Bin/RunScript -script '
```

I really do encourage you to take some time to set up a development environment that is nice for you. Do use GUI apps when you can but when some of the underlying tools that you use are command-line based you are going to want to be able to invoke them easily. Once you have developed your custom data filter script or your special Job Processing Script, you will deploy them in ways suitable to production, and these development-time command-line invocation tricks won't matter.

See Also

[Common Command-Line Options](#)

Common Command Line Options

The non-GUI DocOrigin programs all use a standard set of command-line options listed here. The following style conventions are used:

Convention	Meaning
monospace	options, code, commands, file names, paths and extensions
<i>green</i>	values and dynamic parts of options
{ }	option modifier
[]	optional part of value
	between possible values
<i>(As of <version>)</i>	indicates what version an option or feature of an existing option starts at
<i>(Mandatory)</i>	indicates a mandatory option
<i>(<Program> Only)</i>	indicates the option applies only to a certain program or tool

For convenience, most of the Default PRM files will set default logfile and message files but you are free to override them as you wish.

See [Command Line Processing](#) for more information on command line syntax.

For application-specific command options, see the individual DocOrigin program's section.

See Also

[Command Options - Merge](#)

[Command Options - FolderMonitor](#)

-alert

Turns email alerts on or off.

Syntax

-alert **Y|N**

Description

Turns email alerts on (**Y**) or off (**N**). By default, alerts are off. If you specify an `-alertTo` option, likely in your `DocOrigin.prm` override file, use `-alert` to turn it off or on as desired.

See Also

- [-alertLevel](#)
- [-alertMsg](#)
- [-alertSubject](#)
- [-alertTo](#)

-alertLevel

Indicate which level of messages will trigger email alerts.

Syntax

`-alertLevel` *Information* | *Warning* | *Error* | *Fatal*

Description

Indicate which messages will trigger email alerts. Each message has an associated severity level – *Information*, *Warning*, *Error*, or *Fatal*. When `-alertLevel` is set to one of these levels, all messages with this severity or greater will trigger email alerts. The default is severity *Error*.

See Also

- `-alert`
- `-alertMsg`
- `-alertSubject`
- `-alertLevel`

-alertMsg

Provide additional message text to accompany the alert.

Syntax

```
-alertMsg text
```

Description

Email alerts automatically set the email message to the same message that is normally logged to the logfile, including the time and date stamp.

The `-alertMsg` is an additional optional text message to add to the message. The text can be multi-line by separating each line by a semicolon as in:

```
-alertMsg "This is an alert;this is the second line;this is the third"
```

Alert messages (and the `-alertSubject`) can also contain certain system information strings. These are specified by the following syntax:

```
-alertMsg "Error: [MsgID];Registration ID: [RegID]"
```

The processing will replace `[MsgID]` with the current message ID, and `[RegID]` with the customer's registration ID. The following substitutions are possible:

[App]	- The name of the program that's running - for example Merge
[Keyfile]	- The name of the registration keyfile if it can be found. This file is often named using the customer name or some system name, making it useful in identifying exactly which computer sent the alert.
[MsgID]	- the error/message identifier. This uniquely identifies the error within each program.
[RegID]	- the customer's DocOrigin Registration ID if it can be found.
[SysID]	- the computer's DocOrigin System ID if it can be found.

See Also

- [-alert](#)
- [-alertLevel](#)
- [-alertSubject](#)
- [-alertTo](#)

-alertSubject

Specify the email subject line for the alert.

Syntax

-alertSubject *text*

Description

Specify the email subject line for the alert. This is a single line of text. You can also use the same substitution strings as for the `-alertMsg` option. Structuring the subject line with key information can facilitate the identification of critical alerts when viewed within an email inbox.

See Also

- [-alert](#)
- [-alertLevel](#)
- [-alertMsg](#)
- [-alertTo](#)

-alertTo

Identify the email addresses that an email alert is to be sent to.

Syntax

-alertTo *address1;address2;...*

Description

These are the email addresses that the email alert is sent to. If there is more than one address, separate the addresses with semicolons. This parameter is mandatory if you are using email alerts.

See Also

- alert
- alertLevel
- alertMsg
- alertTo
- alertSubject

-cache

This command-line option presets `_cache` variables that can be accessed via script - see script object `_cache`.

Syntax

```
-cache name1:value1;name2:value2;...
```

Description

This provides application programmers the opportunity to pass data directly to their form scripts. `-cache` options set `-cache` variables. `-cache` variables survive across document boundaries (unlike global fields). In JavaScript, using the above syntax example `_cache.name2` returns `value2` as its result. Outside of JavaScript, `-cache` variables can be referenced directly in forms or data via the automatic/embedded field reference syntax of `[!Cache name]`. For example, using the above syntax example, `[!Cache name2]` would have `value2` substituted.

The name string must contain only alphanumeric characters or the '_' (underscore) character. You can specify multiple name/value pairs by separating each pair with a semicolon. Also, you can specify multiple `-cache` options and they will be combined, not overridden. Note that if any string contains blanks the entire `-cache` parameter must be enclosed in double quotation marks, for example:

```
-cache "first:this is the first value;second:and this is the second"
```

See Also

[Command Line Processing](#)
[Auto/Embedded Fields](#)
[_cache script object](#)


-cd

Set the current directory.

Syntax

-cd *directory*

Description

 This is a highly atypical command. It is operated upon immediately as soon as it is encountered and then it is thrown away.

By putting a -cd *directory* option near the start of your Merge command line, you may be able to save having to specify the same path for several other command-line options.

At any rate, it establishes the current directory for the Merge operation. Note that \$B is unaffected by this and will remain constant throughout as the directory that was the current directory when the Merge executable was run.

See Also

[\\$X String Substitutions](#)

-clearLog

(As of 3.2.002.04)

Empty the log before the run begins.

Syntax

-clearLog **Y**

Description

Using -clearLog or -clearLog Y will empty the log before the run begins.

This can be handy during dev/test.

-debug

Ask for debug output from suitably instrumented JavaScript code.

Syntax

-debug

Description

Sets a JavaScript variable called `_DEBUG` to true. This command-line option is intended to assist developers in debugging scripts. Using this you can conditionally have your `_logf` and `_tracef` messages written to the file specified by the `-logfile` and `-trace` command-line options.

For example:

```
if (_DEBUG) _message("iDataLen=%d", iDataLen);
```

See Also

[Command Line Processing](#)

[_logf](#)

[_printf](#)

[_tracef](#)

[-logfile](#)

[-trace](#)

[Debugging Script](#)

-define

Defines shorthand names for longer paths or other text that is often repeated within the files. Valid only in PRM files.

See Also

[Indirect Parameter Files](#)

-fileMask

(As of 3.0.003.13) (Windows only) (DocOriginNotify only)

Specify the path and file mask of files that DocOriginNotify should watch for.

Syntax

-fileMask *path/fileMask*

Description

Specify the path and file mask of files that DocOriginNotify should watch for. This needs to be to a local or network-mounted file system.

```
-fileMask M:/DO/Outputs/MyName/*.pdf
```

See Also

[DocOriginNotify](#)

.../DO/Bin/DocOriginNotify.prm

-include

Include the contents of the specified file in the command parameters.

Syntax

-include *filename*

Description

Include the contents of the specified file in the command parameters.

On the command line itself, this can also be specified as *@filename*.

See Also

[Indirect Parameter Files](#)

-locale

Specify the name of the Locale to be used in the program.

Syntax

`-locale localename`

Description

This command explicitly sets the locale used for language/currency/date/number formatting for the run. If not set, the locale set on the computer will be used. Also note that in Design you can explicitly override this default locale for specific Fields, Labels, or other objects. Locale names are a 2-character Language Code, followed by an underscore, followed by a 2-character Country Code. As an example, "fr_CA" indicates the French language in Canada. DocOrigin uses the IBM ICU tools to do all locale conversion. They use standard Language Codes specified in the [ISO-639](#) standard. The Country Codes are from the [ISO-3166](#) standard.

See Also

[Command Line Processing Internationalization](#)
[_locale \(Format Currency, Number, Date/Time\)](#)

-logfile

Specify the name of the log file where Merge will write job-related information.

Syntax

`-logfile filename`

Description

Specify the name of the log file where Merge will write job-related information, such as start and end times, and error messages. This is a mandatory command-line option, however, it is supplied automatically by the `Default-DocOrigin.prm` file. Of course, you may override it on the command line or in your own `.prm` files. See the [File Naming Conventions](#) section for additional restrictions and options. This includes the ability to inject various date-time-related placeholders in the logfile name, as well as referencing file values. If the specified log file already exists, DocOrigin programs append to the end of the file, otherwise, the file will be created.

Emergency log file

Sometimes issues arise before the log file has been established. Typically this is during the initial processing of all the command line parameters. In that case, messages will be appended to the `.../DO/Bin/DocOrigin.log` file. This emergency log file will also be resorted to if we are unable to write to your nominated log file.

By the way, if needed, we make ten slightly spaced attempts at accessing your log file but if all those fail we will then use the emergency log file. If you detect that your processing times have really slowed down, it may be because we can't access your log file and all those retries do slow you down. Consider checking `.../DO/Bin/DocOrigin.log` from time to time. And yes, even though it is under the DO tree, you may choose to delete it from time to time.

Most issues reported to this log file will be related to bad/unrecognized parameters. If you get a "bad arg list" return code such as **101**, **201**, or **708** strongly consider looking in this emergency log file.

Finally, if we cannot access this emergency log file either, we will attempt to send an email alert (up to 3 times). However, you must have `-alertTo` set, as is recommended, for an email alert to occur.

See Also

[Command Line Processing](#)

[File Naming Conventions](#)

[-alert](#)

[Return Codes](#)

-logSeverity

(As of 3.2.002.04)

Specify the minimum severity level of log messages to be logged. This can improve system performance.

Syntax

-logSeverity *info|fatal|error|warning/warn*

Description

This option can be used to reduce the number of log messages produced. Specify the minimum severity level of log messages to be logged. A message's severity is defined in the message file, DocOrigin_EN.xml.

This sample filters out all log messages with a severity below *Warning*, so no *Information* severity messages are logged:

```
-logSeverity warning
```

-message

Identify the file that contains all error messages issued by DocOrigin.

Syntax

-message *filename*

Description

Identify the file that contains all error messages issued by DocOrigin. This is a mandatory option, however, it is supplied automatically by the `Default-DocOrigin.prm` file. Do not specify it unless you are pointing to an override message file, possibly for a different locale than EN.

-monitor

(Windows DocOriginFolderMonitorSVC.prm only)

Define a Folder Monitor instance name and indicate if it is started automatically or manually.

Syntax

```
-monitor "instanceName:Automatic|Manual"
```

Description

On Windows, the DocOriginFolderMonitorConsole application can start/stop any of a user-defined set of FolderMonitor instances. These instance possibilities are made known to the DocOriginFolderMonitorSVC Service by way of its parameter file: DocOriginFolderMonitorSVC.prm. This is done by entering in as many -monitor lines as desired into that file. If the monitor instance is designated as **Automatic** then when the FolderMonitor service is started, that monitor instance will also be started. Since, as installed, and as is typical, the FolderMonitor service itself is started automatically when a reboot is done, that means that all of the FolderMonitor instances designated as **Automatic** will also start on a reboot. If the instance is designated as **Manual**, then the FolderMonitorConsole application can be used to start/stop such an instance when desired.

See Also

[FolderMonitor](#)

-phase

(As of 3.1.002.03)

The phase option is used to identify whether the code is running in production, dev, or test. It affects, at the time of introduction, the `!!` conditional comments that may appear in the script.

Syntax

`-phase production | development | test`

Description

At the time of introduction, the setting is sensitive to only whether its value begins with "*p*" or not.

During development, it is often the case, (and advisable) to include logging statements in the script code. That might be `_logf`, `_logPrintf`, `_tracef`, or others. They can be extremely helpful during development. However, there is a tendency to simply leave them in and hence slow down production. Or you take them out, or comment them out, and are dismayed when maintenance has to be done on the script.

As of 3.1.002.03, you can "conditionally comment" your code by starting such lines with `!!`. To be effective, that must be at the very start of the line, not indented at all. As long as you are in the development phase, those `!!` characters will be replaced with blanks before the code is compiled. However, if `-phase p` is provided as a command line option then the `!!` characters will be replaced with `//` before the code is compiled. Hence all of those logging lines will be commented out, without your physically changing any code, and yet be instantly available should any maintenance be required. This provides development efficiency and also improves production performance. Other functions like `_dlogf` and `_dmessage` are still useful since, unless `-debug` is specified, they will avoid I/O out to the log file. Nevertheless, evaluation of their arguments will wastefully occur before the function can discover that it is not to do the actual logging.

It is expected that `-phase production` will be inserted into the override `$0/DocOrigin.prm` on `production` DocOrigin servers, but not be so specified on "dev" machines. Of course, a development person could occasionally specify `-phase p` for final testing runs. Note that these `!!` conditional comments need not be applied to only logging functions. It might be that in development you would, for example, override a production output location with a stand-in for use during development. This processing applies not just to Merge but to all areas where scripting is used.

See Also

[Debugging Script](#)

-setSession

Put field values into the Session table

Syntax

-setSession ***

-setSession *field1*

-setSession "*field1;field2[;...]*"

-setSession **metadata*

Description

This option allows you to populate the [Session](#) table with field name/value pairs. It is particularly useful for inter-process communication.

See Also

[Session](#)

-themeColor

(As of 3.0.003.13) (Windows only) (DocOriginNotify only)

Specify the color theme of the toast issued by DocOriginNotify and also the color of the DocOriginNotify icon that appears in the "systray".

Syntax

```
-themeColor color name | #rrgbb
```

Description

This is most useful when running multiple instances of DocOriginNotify. The theme color can be used to instantly know which monitored file set has had activity.

```
-themeColor LightGreen
```

Recognized color names are: *Beige, Black, Blue, Brown, Coral, Cyan, DarkGray, GreenYellow, Gray, Green, LightBlue, LightCyan, LightGreen, LightGray, LightPink, LightYellow, Magenta, Orange, Pink, Red, Salmon, Turquoise, Yellow,* and *White*.

Otherwise use #rrgbb where each r, g and b is a valid hex digit.

The default is "*Steel blue*" *#4682B4*

See Also

[DocOriginNotify](#)

[Command Options - FolderMonitor](#)

-trace

Cause debug trace output to be written to the specified file.

Syntax

-trace *filename*

Description

Write trace, log and debug messages to a file. *filename* is the fully qualified name of the file where the trace, log and debug information will be written. When *filename* is not fully qualified, Merge will assume the file is to be in the same folder as where Merge is running. Note that the JavaScript function `_tracef()` will write to this *filename* as well, but will not write anything if `-trace` is not set.

See Also

[-tracelevel](#)
[_tracef](#)

-verbose

Cause more output to appear in the log file.

Syntax

-verbose **100**

Description

One can increase the amount of message output one receives by increasing the level parameter. One message where this applies is about the appearance in the data stream of fields that do not exist in the form design. This certainly might be expected or it might be the result of an inadvertent mismatched case usage in a name. By setting -verbose to **100** or more, each such field will be reported. If the -verbose level is less than 100 then at most one message will be issued indicating the presence of some unused data fields. At the time of writing this (Nov 2013) that is the only case for the usage of -verbose and **100** is the only applicable level to use.

-\$X

Set a value for a \$X-type variable.

Syntax

-\$X *value*

\$X *value* // the dash is optional

Description

Set the value for a \$X-type variable. These variables start with \$ and are followed by a single upper case character. A few such characters are reserved for system use. See [\\$X String Substitutions](#).

Typically the values assigned to these variables are file paths. It is never a good idea to hard code a file path deep down in a script (within or outside of a form design). Better to use a \$X-type variable and define that on the command line or in a parameter (.prm) file, as befits the scenario, or the server on which the script is deployed. Such paths may vary from development to test to production.

While the values are typically paths (see `Default-Paths.prm`), they need not be restricted to only paths. They can be any value that the application's script wishes to use. For the most part, folks would use a `-cache name:value` setting for those purposes but using a \$X-type variable is fine too, though a single letter doesn't carry much meaning into an application.

See Also

[\\$X String Substitutions](#)

[_cache](#)

Scripting

Most DocOrigin applications use JavaScript to enhance the processing of document data. This scripting capability includes many extensions to the standard JavaScript language through the addition of functions and the pre-definition of certain variables. When run within Merge a document DOM (Document Object Model) exists which allows access to most of the document objects and their properties. Functions that greatly exceed the normal 'reach' of JavaScript include file read/write functions, web access functions, send mail, generic running of any external application, and business charting. Together with all of the other extension features and normal JavaScript logic facilities, this greatly enhances the document generation and processing capabilities of the DocOrigin system.

DocOrigin has extended the scope of the built-in scripting engine in a number of ways:

- Many additional functions or object methods have been added to specifically benefit DocOrigin document processing. See the [Script Functions](#) section for a list of available objects and functions.
- Certain pre-defined variables such as `_DEBUG` and the `$` variables are automatically initialized when the script is run. These variables typically give access to command-line settings in a simple and effective manner.
- Additional [Merge DOM](#) (Document Object Model) extensions allow direct access to values, settings, and objects within the document template.

Script Files

Scripting for Merge is done via embedded script within the template (.xatw) files. For all other applications, the script is defined in separate script source files - typically with a .wjs or .wjsinc file extensions. These files may be nested by using the #include and ##include statements. The #include statement loads the contents of a specified file. The file must exist or an error is triggered. The ##include by contrast is a conditional include of the specified file. If the file does not exist the statement is ignored.

The syntax for #include and ##include is:

```
#include "filename"  
##include "filename"
```

Where *filename* is any valid filename on your computer. The filename can have any file extension, but in most DocOrigin examples and documentation we use .wjsinc. If *filename* is not a fully qualified name, DocOrigin will assume it's a name within the same file folder as the source file. The #include and ##include statement must start on the first character of the line.

The Merge DOM

When scripting within Merge a DOM (Document Object Model) is normally available. This provides access to the various document structure and components and allows you to dynamically modify that structure or output. The Merge DOM is described under [DOM Properties Accessing the Document Structures](#) section of this manual.

\$Variables

All non-interactive DocOrigin programs use a common command-line processor. This includes the availability of a number of **\$X** folder mapping settings that are used as shortcuts for various file paths. See [\\$X String Substitutions](#). The \$ variables tend to be defaulted in the standard .prm files that come with the DocOrigin system but can be redefined by you. These variables are all automatically defined within the DocOrigin JavaScript system as pre-defined javascript variables.

For example, in a typical DocOrigin installation **\$F** is a shortcut to the forms folder (C:/DocOrigin/User/Forms on Windows).

```
var mfile = $F + "/myfile.txt";
var mfile2 = "C:/DocOrigin/User/Forms/myfile.txt"; // same as mfile
```

Note that if the string is to be used in a context where a file name is expected then you can actually embed the **\$X** reference in the string. E.g. `var fp = _file.fopen("$0/myFile.ini");`. That is, you can avoid the tedium of **\$X** + "..." concatenation constructs. It is highly recommended that you make use of these \$ variables when designing your scripts as they will create code that is easier to manage when moved to a different computer system. The makers of DocOrigin promise never to use **\$V**, **\$W**, **\$X**, **\$Y**, or **\$Z**. You may use those without fear of a subsequent release coming along and redefining their meaning.

Defaults:

\$var	Meaning
\$R	Your installation folder -- never override this!
\$E	Where the current executable resides -- set automatically, do not override
\$N	The base name of the executable, e.g. Merge, RunScript, ... It is set automatically. Do not stray from this meaning.
\$B	The current directory at the very beginning of execution. The code sets this for you but does not itself reference it.
\$U	Where the User folder tree is; (\$R/User) -- we wish that you not override this. The meaning itself is fixed.
\$C	Where the *user* configuration files are. This is the meaning that \$C has, where you define it to point to is your business.
\$L	Where log files are to be kept. Set as you like, but make sure it has this stated meaning.

As of 3.0.001.11...

\$var	Meaning
\$\$F	Where the current form resides, not forms in general, but the current form. (Merge-only).
\$\$D	Where the current data file resides, not data files in general, but the current data file. (Merge-only).
\$\$X	Re FilterEditor: In a .xfilter file refers to the location of the .xfilter file

Script Functions

DocOrigin script extensions consist of a number of additional functions and JavaScript objects that are predefined in the JavaScript environment. Some of these additional features are only available within certain programs (such as Merge) and are noted as such. Others are available for all DocOrigin programs.

Naming Conventions

In an attempt to reduce "name collision" between DocOrigin defined routines and ones you might write, most DocOrigin functions or objects use an underscore at the beginning of their name, such as `_message()` or `_chart.hBar()`. This convention is also used extensively in the Merge handling of DOM variables. See [The Merge DOM](#) section for more information.

Many of the DocOrigin extensions are provided under an umbrella JavaScript object whose properties and methods are used to access a common set of information. An example would be the `_printer` object - which has several methods (functions) such as `_printer.getDocNum()`, `_printer.getPageCount()` etc. Another object might use a JavaScript property such as `_auto.dataFile`, or in the case of the Merge DOM, `Field._value`.

Functions and JavaScript Objects

Function/Object	Description						
<code>_auto</code>	<i>(Merge only)</i> Script equivalent of the form automatic variables.						
<code>_cache</code>	<i>(Merge only)</i> Save data between Merge events or documents.						
<code>_cacheInt(s)</code>	<i>(As of 3.0.003.28)</i> Returns cache variable 's' as an integer.						
<code>_cacheIncr(s)</code>	<i>(As of 3.0.003.28)</i> Increments cache variable 's' by one. Returns incremented value.						
<code>_chart</code>	<i>(Merge only)</i> Create a BarChart, PieChart, or LineGraph.						
<code>_data</code>	<i>(Merge only)</i> Access the Merge data file (read-only).						
<code>_dmessage</code>	<i>(Windows only)</i> Display debug messages to the user.						
<code>_document</code>	<i>(Merge only)</i> Access the Merge document that's being created.						
<code>_file</code>	Read and write files.						
<code>_isDJ()</code>	Is optimized DocOrigin script running?						
<code>_job</code> (Current Job and Command Line Parameters)	Access command-line options and other job processing information. <table border="1" data-bbox="760 1486 1437 1751"> <tbody> <tr> <td><code>_job.command</code></td> <td>For command-line options</td> </tr> <tr> <td><code>_job.filter</code></td> <td>For filter options</td> </tr> <tr> <td><code>_job.options</code></td> <td>For job options set in a Job Name Discovery script or found on a <code><? DocOrigin ...?></code> Processing Instruction. (See Job Name Discovery).</td> </tr> </tbody> </table>	<code>_job.command</code>	For command-line options	<code>_job.filter</code>	For filter options	<code>_job.options</code>	For job options set in a Job Name Discovery script or found on a <code><? DocOrigin ...?></code> Processing Instruction. (See Job Name Discovery).
<code>_job.command</code>	For command-line options						
<code>_job.filter</code>	For filter options						
<code>_job.options</code>	For job options set in a Job Name Discovery script or found on a <code><? DocOrigin ...?></code> Processing Instruction. (See Job Name Discovery).						
<code>_locale</code> (Format Currency, Number, Date/Time)	Internationalized access to currency and date formatting.						
<code>_logf</code>	Write a message to the current logfile.						
<code>_logfEx</code>	Write a message to the current logfile with the severity level.						
<code>_logPrintf</code>	Write a message to both the logfile and stdout (console).						

Function/Object	Description
_merge (Call DocOrigin Merge)	Invoke DocOrigin Merge.
_mergeEmbedded	Do string substitution of field data
_message	<i>(Windows only)</i> Display a message to the user.
_metaData	Access metadata embedded in data streams.
_os (Operating System Functions)	Access to select operating system functions.
_page (The Page DOM)	<i>(Merge only)</i> Access Merge document current page.
_parseInt(s)	<i>(As of 3.0.003.28)</i> DocOrigin equivalent of JavaScript parseInt() but with no hex or octal options. Always assumes a decimal-based integer string even if it starts with zero(s).
_parser	Convert comma/tab-delimited data and fixed record data.
_printer	<i>(Merge only)</i> Access/modify printer driver settings.
_printf	Write a message to stdout (console).
_profile (Access Profile Files)	<i>(As of 3.0.004.02)</i> Access Profile (.ini) files.
_prompt	<i>(As of 3.1.002.06)</i> Prompt user for input.
_resolve	Do string substitutions in a provided <i>template</i> text string.
_run (Execute Another Program)	Run any program.
_runNoWait	Run any program. Do not wait for the result.
_sendmail	Send an email message.
_summary	<i>(Merge only)</i> Access/store summary data for Merge summary processing.
_system	<i>(Merge only)</i> Access to certain Merge options.
_toDOUnits	Convert various measurements to internal DocOrigin units (microns).
_fromDOUnits	Convert internal DocOrigin units (microns) to various common measurements.
_tracef	Write a message to the Trace file. (See -trace)
_xml	<i>(Merge only)</i> Save xml data from Merge -filter processing.
XmlFile Class (Write XML Files)	Create xml output files.
XMLHttpRequest	Used to send requests to an http website and get a response.

Predefined Variables

[\\$X String Substitutions](#) - all pre-defined \$X-type variables.

`_auto` (Automatic Field Values)

(Merge only)

The `_auto` object allows access to the set of automatic field values for Fields. These values can also be used to embed data directly into text Labels. See the [Auto/Embedded Fields](#) section.

Description

The `_auto` object is a pre-defined JavaScript object which has several pre-defined [properties](#) that you can access (read-only). These property names are out-of-character in that they are case insensitive. But don't try that anywhere else. Note also the aberration in that they do not start with an underscore. This is all in aid of compatibility with their use as embedded and automatic fields. Many of these values can also be accessed using other similar functions within DocOrigin.

Properties

<code>_auto.dataFile</code>	The full path and filename of the current data file.
<code>_auto.dataName</code>	The filename only (no path, no extension) of the current data file.
<code>_auto.dataNameExt</code>	The filename and extension (when specified) of the current data file.
<code>_auto.dataPath</code>	The file path of the current data file.
<code>_auto.date</code>	The current date, in the format <code>yyyyMMdd</code> .
<code>_auto.datetime</code>	The current date and time, in the format <code>yyyyMMddHHmmss</code> .
<code>_auto.docNum</code>	The current document number being processed. Numbering starts at 1.
<code>_auto.formFile</code>	The full path and filename of the current form file.
<code>_auto.formName</code>	The filename only (no path, no extension) of the current form file.
<code>_auto.formNameExt</code>	The filename and extension (when specified) of the current form file.
<code>_auto.formPath</code>	The file path of the current form file.
<code>_auto.pageCount</code>	The total number of pages within this document.
<code>_auto.pageNum</code>	The current page number within the document (also <code>_auto.PN</code>), atypically case-insensitive. This is <u>not</u> a property, per se, of the <code>_auto</code> object, but rather a function, despite the lack of <code>()</code> , that reports the page number of the current global object, i.e. the object in which the script is defined. It makes absolutely no sense to use this on the form object itself. The form object has no "current page" context and can report only 0 . You would be better off learning to use <code>this._pageNumber</code> which expresses more clearly what you are getting the current page number of. Note that page numbers are not defined until the Pagination Completed event. They are of no use at Data Merged time.
<code>_auto.time</code>	The time that Merge began running, as a string in the format <code>HHmmss</code> .

`_cache` (Scripting Object)

Description

The `_cache` object provides a way to store information at various points in a Merge run and retrieve it later on. The `_cache` data does not disappear between Merge events nor between Merge documents. Stored data is simply associated with a given name. `_cache` variables and their values are often defined via the `-cache` command line option. In JavaScript, you can assign a value to a cache variable either by directly assigning it to the variable as in:

```
_cache.foo = "some value";
```

or by using the `_cache.set()` function as in:

```
_cache.set("foo", "some value");
```

Similarly, you can retrieve values from `_cache` by:

```
var value = _cache.foo;           // sets value to "some value"
var value2 = _cache.get("foo");  // sets value2 to "some value"
```

Embedded references can be made to `_cache` variables via the syntax `[!Cache name]`, as in `[!Cache foo]` using the above example.

Note `_cache` variables are always stored as strings. A side effect of that is that the `+` operator in JavaScript implies concatenation, not addition. It's likely that you may want to make liberal use of the construct `parseInt(_cache.name)`. `_cache` variables are often used in counting operations; if you use the standard JavaScript notation of `++` that is recognized as an integer operation and the `_cache` variable will be automatically converted to a number before being incremented and then stored back as a string.

```
_cache.counter = 0; ...
_cache.counter++;
```

Advanced `_cache` usage

Sometimes you wish to cache arrays of values. There are various techniques that you could use. The following is a rarely used method. It's rare because people rarely use `_cache.set("name")` when they can simply use `_cache.name`. However, there is the further option of:

```
_cache.set("name", someValue, index);
```

which lets you supply an array index, hence permitting the creation of an array of values. Naturally, there is the corresponding:

```
_cache.get("name", index);
```

to retrieve an array element.

Functions

<code>_cache.logf()</code>	Dump <code>_cache</code> content to log file.
<code>_cache.get(<i>name</i> [, <i>index</i>])</code>	Get value of <code>_cache</code> property <i>name</i> .
<code>_cache.set(<i>name</i>, <i>value</i> [, <i>index</i>])</code>	Set <code>_cache</code> property <i>name</i> to <i>value</i> .

<code>_cacheInt(<i>name</i> [, <i>index</i>])</code>	Returns <code>_cache</code> property <i>name</i> as an integer.
<code>_cacheIncr(<i>name</i> [, <i>index</i>])</code>	Increments <code>_cache</code> property <i>name</i> by one. Returns incremented value.

See Also

[Auto/Embedded Fields](#)

[-cache - command-line option](#)

_chart (Draw Bar Charts and Pie Charts)

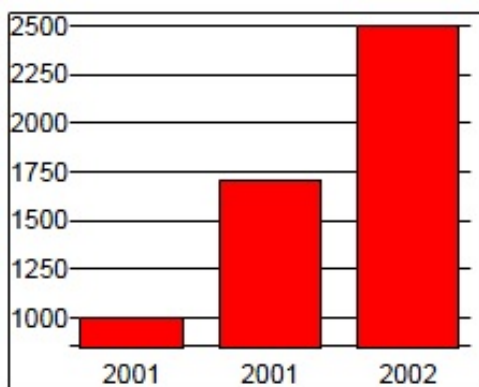
(Merge only)

Description

Merge enables you to create bar charts, pie charts, and line graphs and embed them into the documents. The data can be embedded in the XML data stream or created dynamically using JavaScript. The charts are created using the `_chart` JavaScript routines within Merge. The `_chart` routines draw within the area of (typically) a Group object in Design. The Group object defines the physical size and position of the chart. The chart is automatically scaled to fit within the margins of the Group. A simple chart description might be coded as follows:

```
var data = [['Year', 'Sales'], [2001, 1000], [2001, 1700], [2002, 2500]];
_chart.vBar(data, null);
```

Resulting in:



Usage

All charts are generated within Merge. They use standard Merge objects including fonts and various filled objects. They are NOT raster images.

Charts are created by calling one of the `_chart` methods listed below. Each takes a data parameter (see `_chart Data`) and a formatting options parameter (see `_chart Display Options`). The data is either generated by your javascript program or by reading from an external file or by loading from the Merge data stream.

Functions

<code>_chart.barLine(data [, options])</code>	Draw a bar chart using vertical bars for 1st data set and line graphs for all other data sets. A combination of <code>_chart.vBar()</code> and <code>_chart.line()</code> functionality.
<code>_chart.hBar(data [, options])</code>	Draw a bar chart with horizontal bars.
<code>_chart.line(data [, options])</code>	Draw a line graph.
<code>_chart.pie(data [, options])</code>	Draw a pie chart.
<code>_chart.vBar(data [, options])</code>	Draw a bar chart with vertical bars.

`_chart` Data

The `_chart` routines all use a common method of passing data to the various routines. Data is passed to the charting routines as a JavaScript array of data sets. For example:

```
var data = [
  ['Year', 'Estimate', 'Actual']
  , [2001, 1.5, 2]
  , [2002, 10, 11]
  , [2003, 1, 5]
  , [2004, 1.5, 2]
  , [2005, 10, 11]
  , [2006, 1, 5] ];
```

In the above example, the variable 'data' has been assigned a series of 6 data sets (rows) plus a header row. Each data set is itself an array of names and/or values. The first row is a header. It must have the same number of array elements as each of the following data rows. This header row lists the names to be given to each column of data - in this case the first column is a 'Year', the second is called 'Estimate', and third is 'Actual'. These names may be used when labeling legend data. The values in the first column are the "primary data" values. These are considered to be the name of the row and can be either a number or a string (2001 and '2001' mean the same thing). All other data should be numeric.

For a simple bar chart with vertical bars the primary data represents the values along the bottom X-axis of the chart. The second and successive columns represent the Y-axis height of the bars. A rotated bar chart (`vBar`) would display the primary data along the vertical (left) side of the chart, with bars extending horizontally. In a pie chart the primary data is the name of each pie slice and the second column value represents the slice size. The primary data values (2001, 2002, etc.) are used as labels for the primary axis of bar charts and line graphs. They can also be used to label individual slices of a pie chart. Bar charts and line graphs allow multiple data columns (columns 2 and up). They produce sets of bars or multiple overlapping lines. Pie charts only take a single data column.

Charting data should always be positive values.

Creating Data

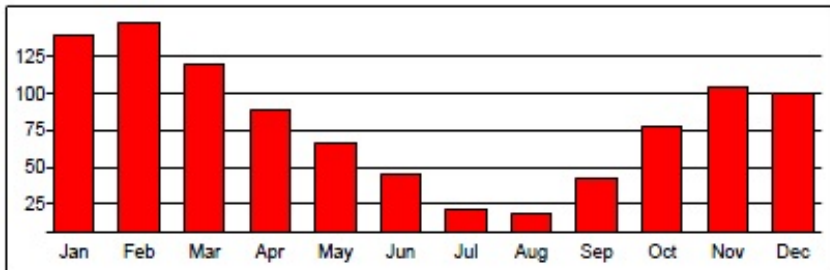
The simplest way to provide data is by directly creating a JavaScript array in the appropriate structure (see above). Data can also be extracted from a Merge document's data stream and loaded into an appropriate data array. This can be done using the `_data._loadAsArray()` function - see the `_data` functions. For example, consider an XML data file with the following data somewhere within it:

```
<usageHistory>
  <period>
    <month>Month</month>
    <usage>Usage</usage>
  </period>
  <period>
    <month>Jan</month>
    <usage>139</usage>
  </period>
  <period>
    <month>Feb</month>
    <usage>148</usage>
  </period>
  <period>
    <month>Mar</month>
    <usage>119</usage>
  </period>
  ...
</usageHistory>
```

The following script could be used to load that data and display a bar chart of the data:

```
data = _data._loadAsArray("usageHistory");
_chart.vBar(data);
```

Resulting in:



In this example there is an entry with the data name information. i.e.

```
<period>
  <month>Month</month>
  <usage>Usage</usage>
</period>
```

Quite often that leading entry is not provided. You can handle that by providing the first set of "name" data right in the `_loadAsArray` function call. In this example you would use:

```
data = _data._loadAsArray("usageHistory", ["Month", "Usage"]);
_chart.vBar(data);
```

The optional second argument to `_loadAsArray` is an array giving the names of the data being loaded. It must have the same number of elements as is in the loaded data.

i (Before 3.1.001.08) You could use the second argument but you may as well supply simply `[]` (an empty array), and following the `_loadAsArray` call, do an explicit set of the `[0]` element. Per this example:

```
data = _data._loadAsArray("usageHistory", []); // pre 3.1.001.08
workaround
data[0] = ["Month", "Usage"]; // set the real [0] entry.
_chart.vBar(data);
```

By using the second argument, room was provided in the array for the `[0]` element, but it was not loaded properly. *Fixed in 3.1.001.08.*

_chart Display Options

The `_chart` routines for bar charts, pie charts, and line graphs all use a common set of display options. These allow you to customize the appearance of the individual charts.

All options have a default setting that is noted below. You only need to explicitly set options when you wish to override these default settings.

The Options Object

All options are defined as properties of a JavaScript object. For example:

```
var options = {
  BarGap: '.15in',
  LegendFont: {bold:true, size:10},
  colors: ['orange', 'coral', '#FF0000', 'brown']
};
```

Each option, such as `LegendFont` or `colors`, is followed by either a single value, an array of values surrounded by square brackets, or a set of sub-options enclosed in curly brackets. Measurements, the gaps between objects, thickness, etc. use standard DocOrigin script units (an integer size in microns or a string with units, such as 250000, '.25"', '.25in', '.25cm' etc.) Colors are standard DocOrigin script colors (either a 6digit hex RGB code or one of the default color names). See the [DocOrigin Colors](#) section for more information.

All option names such as `BarGap`, `ChartGap`, etc. are case-insensitive. For example, `BarGap`, `bargap`, and `BARGAP` are all acceptable.

- **AutoZero: *false***
(As of 3.0.002.05) When set to '**true**' bar charts will always start to 0.0 rather than their minimum value. Or, more precisely, charts whose minimum value is greater than zero will instead be drawn with zero as the minimum. charts with a maximum value less than zero will be drawn with zero as the maximum. Only applies to bar charts. You may also explicitly set the min/max values using the `MinValue` and `MaxValue` options.
- **BarGap: *'.15in'***
This is the space between each set or group of bars on a bar chart.
- **ChartGap: *'.05in'***
This is the "margin" around the actual chart or graph. This space separates the chart from the X-axis and Y-axis labels and from the legend if it is present. You can set all four gaps using the `ChartGap` setting, or set them individually using `ChartGapBottom`, `ChartGapTop`, `ChartGapLeft` and `ChartGapRight`.
- **Color: [*'red', 'green', 'blue', 'yellow', 'magenta', 'cyan', 'orange', 'lightgreen', 'lightpink', 'lightblue'*]**
This sets the colors of bars, pie slices, or lines (depending on the type of chart). Colors are used sequentially. If more colors are required than have been defined the system will automatically re-use the defined list of colors but make them slightly darker. You can also use the Hex color codes such as `Color: ['#B467DC', '#007CDC']`. You can also set `Color: 'grayscale'` to cause only shades of gray to be displayed.
- **ColorColumn: *0***
(As of 3.1.001.12) (*Bar charts only*) Specify a column in the data array (first data value is column 1) which specifies the bar color for each row of data. This specifies a particular bar color for each bar, as opposed to the normal assigning different colors to each column of data in a multi-column data set. The number in column `ColorColumn` is an index into the `Color` array above. For example, a value of **2** in the `ColorColumn` would cause that bar to be displayed in **blue** (color 0 is **red**, color 1 is **green**, color 2 is **blue** etc.) See also the `LegendLabel` command which allows you to specify what each of these colors means.
- **DatasetNames: {*start:1, every:1, tickStart:1, tickEvery:1*}**
(As of 3.1.001.04) This setting controls how the main axis of data records is labeled. This is the bottom axis of a vertical bar chart or line chart, or the left axis of a horizontal bar chart. You can control which axis labels are displayed and which are left out. This can be useful if the labels tend to run together or overlap due to space constraints.

(As of 3.1.001.07) The **tickStart/tickEvery** settings affect the small tick marks displayed on the bottom axis of line charts. This is of use when there are many data samples and the tick marks start to blur together. For example, if you have several data records labeled '2005', '2006', '2007', etc., you can specify that every 3rd label starting at 2006 be displayed, by setting:

```
DatasetNames: {start:2, every:3}, // labels 2006, 2009, 2012, etc.
Or set tick marks every 5 data items
DatasetNames: {tickStart:1, tickEvery:5}, // tick marks every 5th item.
```

- DrawEdges: **true/false**
Should bar edges be drawn? If **true** the border of bars in a bar chart will be outlined in black. If **false** they will not be. When drawing normal bar charts, this setting default to true. When drawing 3D barcharts, this setting defaults to **false**. Note that drawing the edges of a 3D bar chart may be visually incorrect if the bars overlap.
- Feature: **null**
Used with pie charts to highlight or "feature" selected pie slices. This is done by offsetting the slice outward from the center of the pie chart by a distance of FeatureOffset. To feature a single data item, set Feature to the data item name. If you want more than one item featured, set Feature to an array of data item names. For example:

```
Feature: '2003', // features single data item named '2003'
Feature: ['2003', '2006', '2007'], // feature several items
```

If you want all items featured, set Feature to '*'. Bar charts will also allow Feature values. The bar chart FeatureFont will be used typically to change the color.

- FeatureColor: **(none)**
If Feature: above is set and FeatureColor is set, a bar chart will display the bar in this color. Use this to highlight one or more bars in a bar chart. If not set, the bar will be in its normal color. See also FeatureFont.
- FeatureFont: **{size:8, face:'Arial', bold:false, italic:false, color:'black'}**
This is the font settings used to label the featured data specified in the Feature setting above.
- FeatureOffset: **0**
Explicitly specify the distance to offset a featured pie slice from the center of the pie chart. If this value is **0** (zero) a reasonable offset is automatically applied.
- GridColor: **'black'**
This is the color of any grid lines that are drawn on bar charts or line graphs.
- GridThickness: **'.01in'**
This is the line width for any grid lines that are displayed.
- LegendFont: **{size:8, face:'Arial', bold:false, italic:false, color:'black'}**
This font is used for all Legend text.
- LegendFormat: **'[label]'**
The text displayed next to each color square on the legend is formatted according to this setting. Formatting is done by taking the LegendFormat text string and replacing any text displayed in [square brackets] with the named data or value. For example, setting LegendFormat: **'[label]'** will display the label text associated with the specified item. LegendFormat: **'[data]'** will display the actual data value. Not all substitutions are available or make sense for every type of chart. The following substitution parameter are available:
 - **[data]** - the actual data value represented by the legend color.
 - **[data format]** - as above, but with an explicit format specification.
The format part is an actual number formatting designation. It can be a standard C-like format string such as %f, %f6.2, %d, etc. or "D", "I" or "\$" to cause formatting using the current Locale of the chart canvas object. "D" formats as a decimal number, "I" as an integer, and "\$" as currency.
 - **[label]** - the data name.
 - **[percent]** - for pie charts, the percentage represented by an individual pie slice.

- **[percent1]** - same as [percent] but displayed to 1 decimal point.

Note that if LegendFormat is set to null a legend will not be displayed.

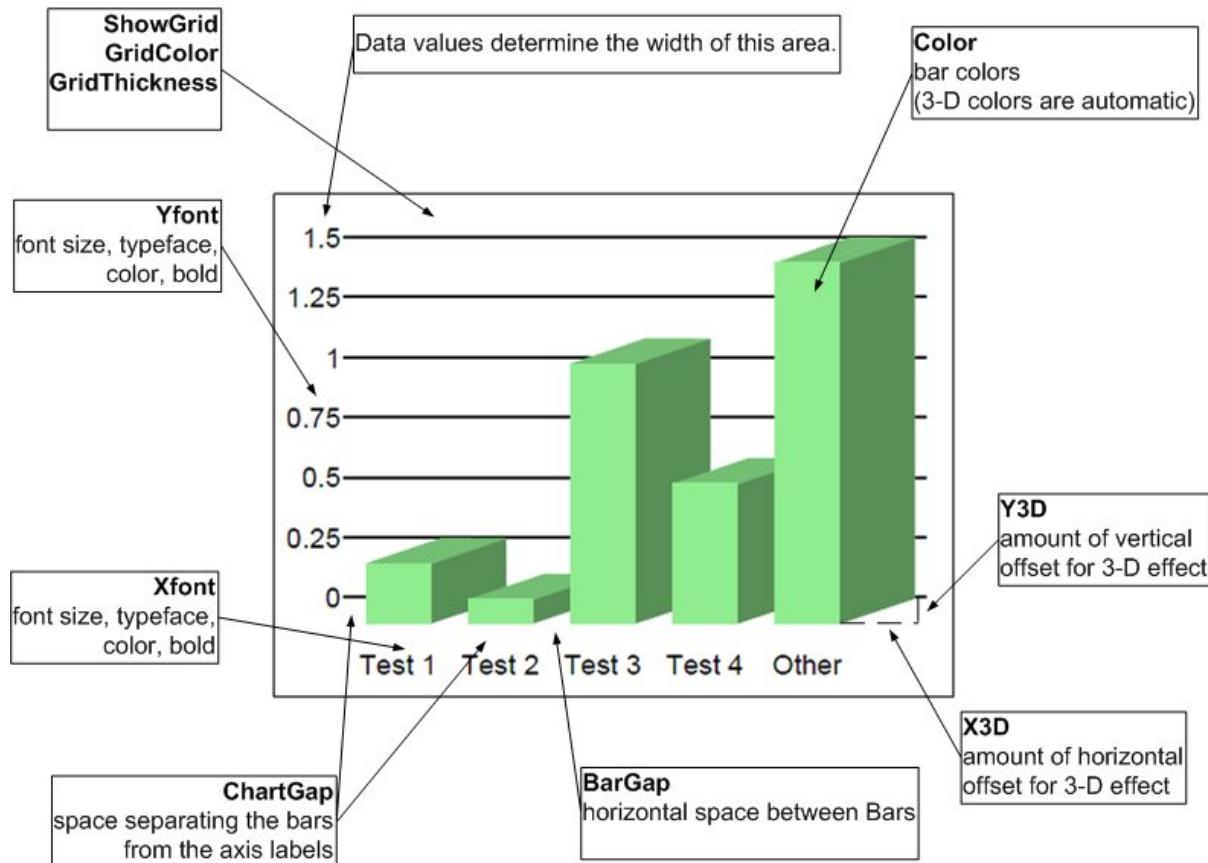
- LegendGap: **'.05in'**
The vertical space separating legend items.
- LegendLabel: [**'name0'**, **'name1'**, **'name2'**, ...]
(As of 3.1.001.12) Assign text to be used on each legend entry. The array of names must correspond to the Color array. When using the ColorColumn option, this option is required if you want to display a Legend.
- LegendSwatchSize: **'.2in'**
The size (width and height) of the color swatch drawn as part of the legend.
- LineThickness: [**'.02in'**]
(Line graphs only) This is the thickness of each line drawn. This must be an array with one entry for each data set (line) that is drawn. If there are more graph lines than settings, the first value in this array will be repeated for all missing settings.
- LineStyle: [**'solid'**, **'dash'**]
(Line graphs only) A list of line styles for each data set (line) to be drawn. Choose from styles of **'solid'**, **'dash'**, **'dot'** or **'dotted'**.
- MarkerColor: [**'black'**]
(Line graphs only) You can specify that a marker symbol be drawn at each data point. MarkerColor is the color of that marker. This is an array with one entry for each data set (line) that is drawn. If there are more graph lines than settings, the first value in this array will be repeated for all missing settings.
- MarkerSize: [**'.05in'**, **'.08in'**]
(Line graphs only) The size (width and height) of the marker symbols. This is an array with one entry for each data set (line) that is drawn. If there are more graph lines than settings, the first value in this array will be repeated for all missing settings.
- MarkerStyle: [**'box'**, **'star'**]
(Line graphs only) The style of marker symbol to display at each data point. Choose from style of **'none'**, **'box'**, **'star'**, **'openbox'**, **'diamond'** or **'vline'**. This is an array with one entry for each data set (line) that is drawn. If there are more graph lines than settings, the first value in this array will be repeated for all missing settings.
- MaxValue:
(As of 3.0.002.05) Explicitly set the value of the "top" or maximum range of the data values displayed on a bar chart. By default, if this value is not set, the topmost position on the bar chart is defined by the maximum data value in the data array. This option allows you to override that value. The AutoZero option can also influence the range of values of a bar chart.
- MinValue:
(As of 3.0.002.05) explicitly set the value of the "lower" or minimum range of the data values displayed on a bar chart. By default, if this value is not set, the lower position on the bar chart is defined by the minimum data value in the data array. This option allows you to override that value. The AutoZero option can also influence the range of values of a bar chart.
- Pattern: **'None'**
Color bars or pie slices can be displayed as crosshatch styles rather than solid colors. (This may be desirable on monochrome printers). If a single pattern is specified it will apply to all colors. If an array of pattern names is specified it will be applied in sequence to each of the data colors. Pattern choices are **'None'**, **'Horizontal'**, **'Vertical'**, **'Crosshatch'**, **'Diagonal'**, **'ReverseDiagonal'** and **'DiagonalCrosshatch'**.
- PieRound: **true**
If set to **true** pie charts will be round and scaled to fit the available space to the maximum. If set to **false** the pie chart will be elliptical - stretched horizontally and vertically to fit the available space.
- PieStartAngle: **0**
For pie charts, this specifies where the first data item starts visually on the chart. This is an angle measured in degrees

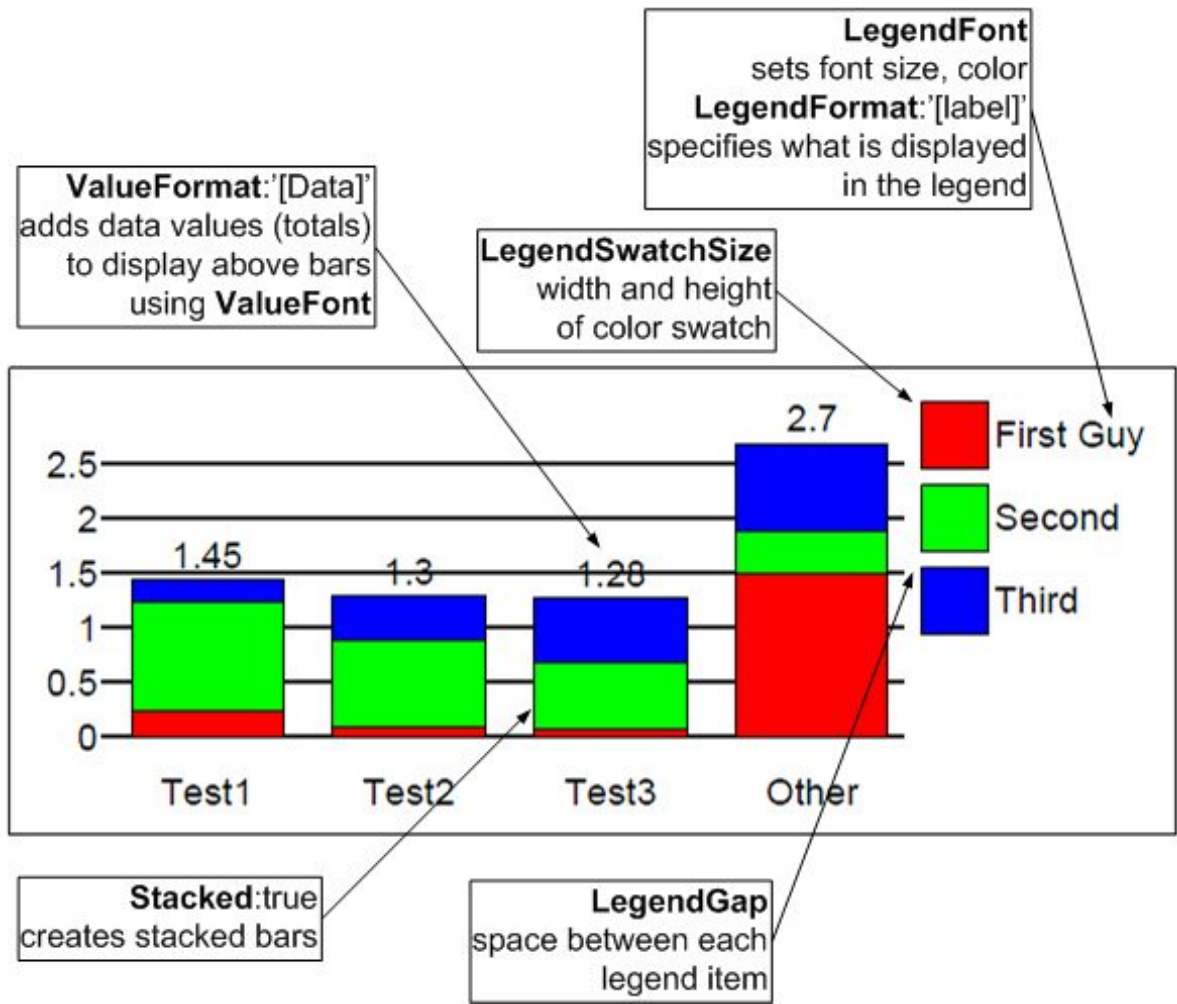
clockwise from the top or positive Y position.

- **RangeFormat: *null***
The left Y axis of a vertical bar chart or the bottom X axis of a horizontal bar chart is known as the Range axis. The numerical values displayed on these axes are formatted according to the RangeFormat. Null indicates a default integer/decimal number in standard North American dot-for-decimal notation. You can also set RangeFormat to an explicit c-like format such as **%f6.2**. Set RangeFormat to **\$** to format as currency in the Locale of the chart canvas object. Set RangeFormat to **D** to format as a decimal number in the Locale of the canvas object. Set RangeFormat to **I** to format as an integer number in the Locale of the canvas object.
- **ShowBarValue: *false***
(As of 3.1.001.04) Set to **true** to display text value of barcode over top of the bar. ValueFormat must also be defined. The text will not be displayed if it doesn't fit inside the bar. You can change the font size etc. using the ValueFont setting.
- **ShowBarValueAtEnd: *true***
(As of 3.1.001.04) Set to **true** to display text value of barcode at the end of the bar (above it for vBar, to the right for hBar). ValueFormat must also be defined.
- **ShowGrid: *true***
Should grid lines be drawn on the chart or graph?
- **ShowXAxis: *true***
(As of 3.1.001.04) Set to **false** to suppress displaying any X-axis (bottom of chart) labeling.
- **ShowXTicks: *true***
(As of 3.1.001.12) Whether to display small tick marks above the labels on the X (bottom) axis. For backward compatibility, vertical bar charts have a default of false and do not show tick marks.
- **ShowYAxis: *true***
(As of 3.1.001.04) set to **false** to suppress displaying any Y-axis (bottom of chart) labeling.
- **ShowYTicks: *true***
(As of 3.1.001.12) Whether to display small Tick marks beside the labels on the Y (left) axis. For backward compatibility horizontal bar charts have a default of **false** and do not show tick marks.
- **Sort: *false***
Should the data items be sorted? Sorting will be from the largest to smallest value. If the data set has more than one data value (multi-line charts or multi-bar charts) only the first data "column" is sorted.
- **Stacked: *false***
For bar charts with multiple data values for each set of bars you can choose for each to be a separate adjacent bar or that all values be stacked in a single bar.
- **SuppressColumn: []**
(As of 3.1.001.09) When you have multiple columns of data in the data array, you can cause one or more to NOT be charted using this option. Supply an array of column numbers of the data array that should not be displayed. For example **[2,3]** will remove the 2nd and 3rd column of data. Use this option when the same dataset might be used for different charts, or removal of a column of data is inconvenient.
- **ValueFont: {size:8, face:'Arial', bold:false, italic:false, color:'black'}**
This is the font used to display the actual value of each bar, pie slice, or line data point.
- **ValueFormat: *null***
String to be displayed above each bar in a bar chart, or within each slice of a pie chart, or above each data point in a line graph. Formatting is done by taking the ValueFormat text string and replacing any text displayed in square brackets with the named data or value. For example, setting ValueFormat: "**[data]**" will display the data value associated with the specified item.

Not all substitutions are available or make sense for every type of chart. The following substitution parameter are available:

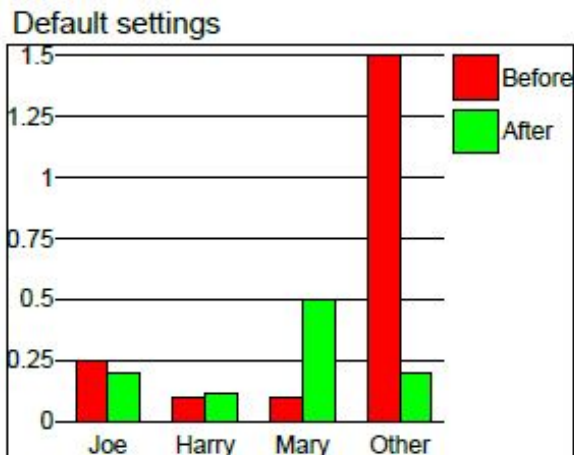
- **[data]** - the actual data value represented by the bar, pie slice, or line data point.
 - **[data format]** - as above but with an explicit format specification. The format part is an actual number formatting designation. It can be a standard C-like format string such as `%f`, `%f6.2`, `%d`, etc. or `D`, `I` or `$` to cause formatting using the current Locale of the chart canvas object. `D` formats as a decimal number, `I` as an integer, and `$` as currency.
 - **[label]** - the data name.
 - **[percent]** - for pie charts, the percentage represented by an individual pie slice.
 - **[percent1]** - same as **[percent]** but displayed to 1 decimal point. If `ValueFormat` is set to `null`, no text will be displayed. If the text doesn't fit totally inside a bar chart bar or a pie chart slice, it will not be displayed.
- **X3D: 0**
If non-zero, this sets the amount of "shadow" or 3D effect in the X (horizontal) direction. A typical setting might be `".3in"`. Bar charts are drawn as 3D bars when either X3D or Y3D are non-zero.
 - **XFont: {size:8, face:'Arial', bold:false, italic:false, color:'black', angle:0}**
This is the font used to annotate the X-axis along the bottom of bar charts and line graphs. Note that for vertical bar charts and line charts you can specify a rotation angle for the text in the range of -90 to 90 degrees.
 - **Y3D: 0**
If non-zero, this sets the amount of "shadow" or 3D effect in the Y (vertical) direction. A typical setting might be `".1in"`. Bar charts are drawn as 3D bars when either X3D or Y3D are non-zero. Pie charts are drawn as 3D using only Y3D.
 - **YFont: {size:8, face:'Arial', bold:false, italic:false, color:'black'}**
This is the font used to annotate the Y-axis along the left side of bar charts and line graphs.





_chart Margins and Spacing

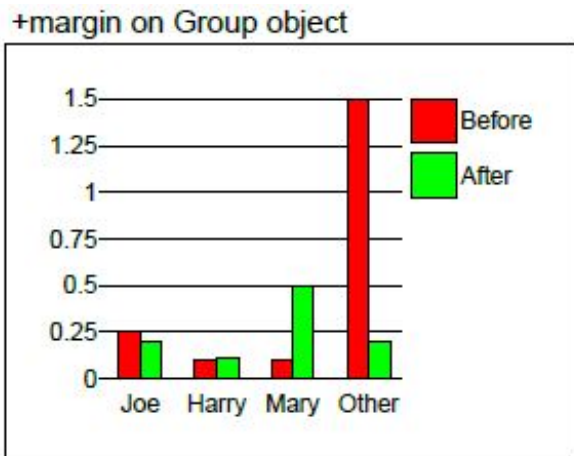
The spacing of various objects within a chart can be modified by using a variety of settings. For example, consider this basic bar chart:



This chart uses the default setting for charts and is displayed inside a default Group object whose margins are 0.

Group Object Margins

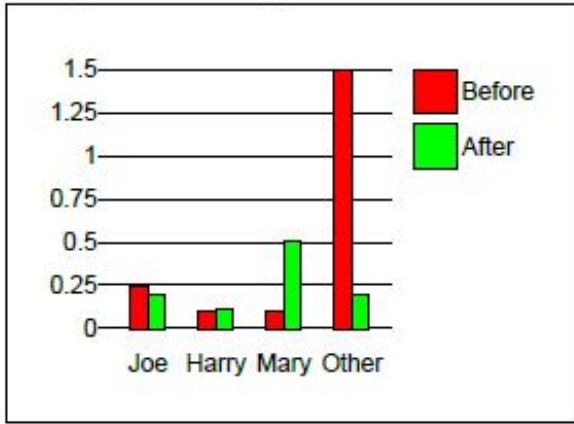
By introducing a margin setting on the object (typically a Group) that contains the chart, you can change the amount of space between the borders of the group and the drawing. Of course, each margin can be set individually.



ChartGap settings

The ChartGap (or individual ChartGapTop, ChartGapLeft, etc.) settings define a bit of space between the actual chart and the surrounding captions and legends. Note for instance how the legend is separated from the chart grid. Also the captions at the bottom are a bit further down, and the left captions are a bit further away from the actual bars. This of course shrinks the bars slightly in order for it all to appear inside the fixed containing Group. The default ChartGap is **0.05"**.

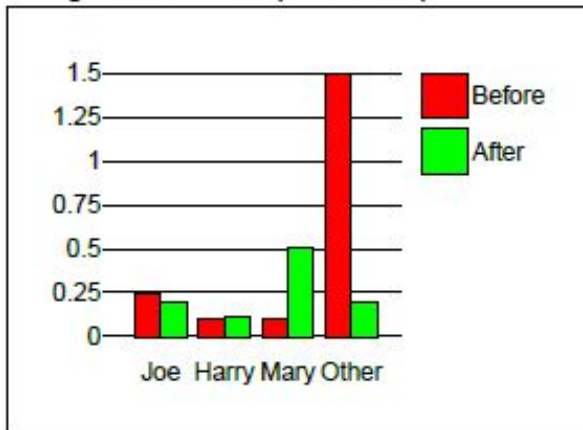
margins + ChartGap



BarGap setting

The `BarGap` setting changes the amount of space allotted between each set of bars and the ones next to them. The gap gets smaller, and the bars themselves get slightly wider. The default `BarGap` is **0.15"**.

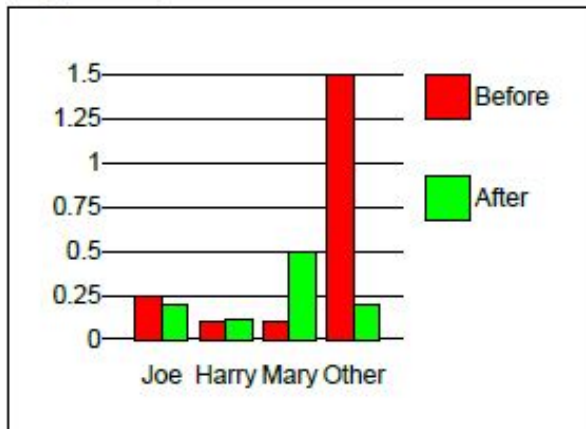
margins + ChartGap + BarGap



LegendGap setting

The `LegendGap` is the space between the two colored legend rectangles. The default `LegendGap` is **0.05"**.

margins + ChartGap + BarGap + LegendGap



`_chart.barLine`

(Merge only)

This method draws a bar chart with overlaid line graphs.

Syntax

```
_chart.barLine(data [, options])
```

Parameters

data is a script array of data points to be charted. See [_chart Data](#) for formatting details. ***options*** is a script object whose properties define various presentation options. See the [_chart Display Options](#) for details.

Returned Value

True if successful, false if an error occurred.

Description

The data array contains a list of data sets to be drawn on the chart. The first data set (column) is drawn as a series of vertical bars. All other data columns create line charts that overlay these bars. See [_chart.hBar](#) and [_chart.Line](#) for details on their construction and options. The `_chart.barLine()` routine must be called from a "containing" object, typically a Group object. The chart is drawn within this object. Bar widths, scaling etc. are all automatically calculated so that the chart completely fills the containing object. See the [_chart Display Options](#) for details on how to modify colors, legends, fonts, etc.

Example

```

options = {
  MarkerStyle: ['none']
};

var data = [
  ['Month', 'Electricity (kWh)', 'Last Year', 'Average']
, ['SEP', 440, 420]
, ['OCT*', 421, 430]
, ['NOV', 469, 445]
, ['DEC*', 460, 440]
, ['JAN', 386, 430]
, ['FEB*', 423, 425] ];

var avg = 0;
for (i=1; i<data.length; i++) {
  avg += data[i][1];
}

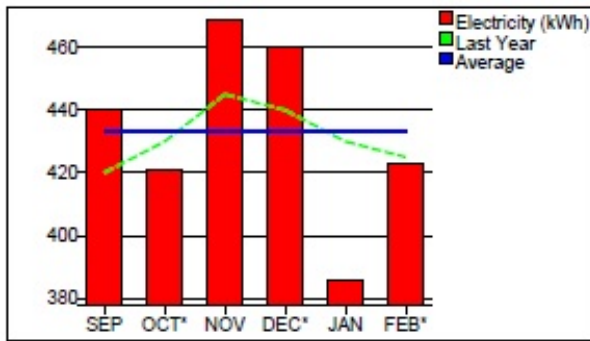
avg = avg/(data.length-1);

for (i=1; i < data.length; i++) {
  data[i][3] = avg;
}

_chart.barLine(data, options);

```

Results in:



This chart shows a set of bars representing monthly electricity usage. The previous year's data is shown as a green line. The blue "Average" line is computed and added using a script. Colors, fonts, and other options are controlled by the *options* parameter. See [_chart Display Options](#) for details.

`_chart.hBar`

(Merge only)

This method draws a bar chart with a series of horizontal bars.

Syntax

```
_chart.hBar(data [, options])
```

Parameters

data is a script array of data points to be charted. See [_chart Data](#) for formatting details. ***options*** is a script object whose properties define various presentation options. See the [_chart Display Options](#) for details.

Returned Value

true if successful, **false** if an error occurred.

Description

The ***data*** array contains a list of data sets to be drawn on the chart. The first data set (column) is drawn as a series of vertical bars. All other data columns create line charts that overlay these bars. See [_chart.hBar](#) and [_chart.Line](#) for details on their construction and options. The `_chart.barLine()` routine must be called from a parent object, typically the Group object. The chart is drawn within this object. Bar widths, scaling, etc., are all automatically calculated so that the chart completely fills the container object. See the [_chart Display Options](#) for details on how to modify colors, legends, fonts, etc.

Example

```

options = {
  MarkerStyle: ['none']
};

var data = [
  ['Month', 'Electricity (kWh)', 'Last Year', 'Average']
, ['SEP', 440, 420]
, ['OCT*', 421, 430]
, ['NOV', 469, 445]
, ['DEC*', 460, 440]
, ['JAN', 386, 430]
, ['FEB*', 423, 425] ];

var avg = 0;
for (i=1; i<data.length; i++) {
  avg += data[i][1];
}

avg = avg/(data.length-1);

for (i=1; i < data.length; i++) {
  data[i][3] = avg;
}

_chart.barLine(data, options);

```

Results in:



This chart shows the use of two parallel sets of bars for each Y-axis data point. Colors, fonts, and other options are controlled by the *options* parameter.

See [_chart Display Options](#) for details.

`_chart.line`

(Merge only)

This method draws a horizontal line graph.

Syntax

```
_chart.line(data [, options])
```

Parameters

data is a script array of data points to be charted. See [_chart Data](#) for formatting details. ***options*** is a script object whose properties define various presentation [options](#). See the [_chart Display Options](#) for details.

Returned Value

true if successful, **false** if an error occurred.

Description

The ***data*** array contains a list of data sets to be drawn as overlapping horizontal line graphs. The `_chart.line()` routine must be called from a containing object, typically a Group object. The chart is drawn within this object. Lines are drawn horizontally from the left to right on the chart. Data is provided as a series of primary (X-axis) data sets, each with one or more data values (Y-axis). Each X-axis data set creates a single connected line graph. See the [_chart Display Options](#) for details on how to modify colors, legends, fonts, etc.

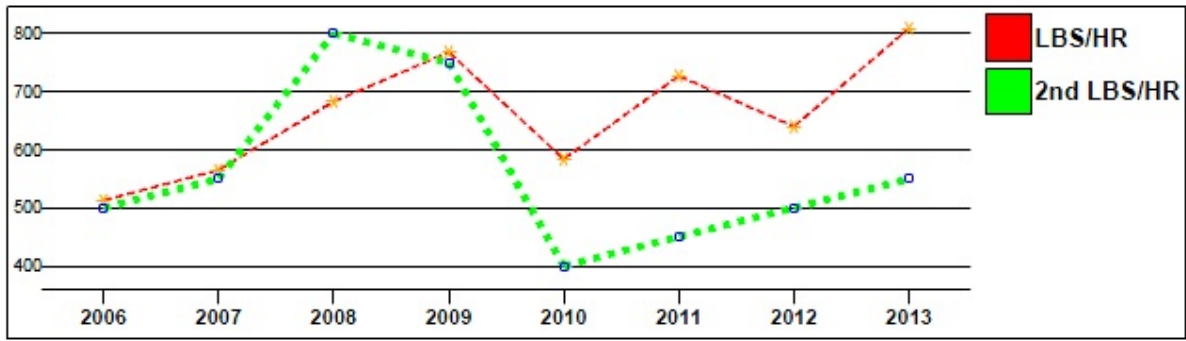
Example

```

var data = [['Date', 'LBS/HR', '2nd LBS/HR']
, [2006, 513, 500]
, [2007, 565, 550]
, [2008, 683, 800]
, [2009, 768, 750]
, [2010, 584, 400]
, [2011, 727, 450]
, [2012, 640, 500]
, [2013, 809, 550] ];
options = {
  chartgap:100000
  , LegendFont:{Size:12, Bold:true}
  , XFont: {Size:10, Bold:true}
  , LineThickness: [20000, 50000]
  , LineStyle: ['Dash', 'Dot']
  , MarkerStyle: ['star', 'openbox']
  , MarkerColor: ['orange', 'blue']
  , Markersize: [100000, 50000]
  , ValueFont :{Size:10, Bold:true }
};
_chart.line(data, options);

```

Results in:



`_chart.pie`

This method draws a piechart for a set of data.

Syntax

```
_chart.pie(data [, options])
```

Parameters

data is a script array of data values to be charted. See [_chart Data](#) for formatting details. Unlike bar charts, the pie chart can contain only a single set of data. These values must all be positive values. *options* is a script object whose properties define various presentation options. See the [_chart Display Options](#) for details.

Returned Value

true if successful, **false** if an error occurred.

Description

The ***data*** array contains a list of data sets to be drawn on the pie chart. The `_chart.pie()` routine must be called from a containing object, typically a Group object. The chart is drawn within this object. Bar widths, scaling, etc. are all automatically calculated so that the chart completely fills the container object. Each data element consists of a Name and a data Value. The pie chart represents the proportion or percentage of the total of all Values that each element represents. Each pie "slice" can be color-coded or distinctly shaded. Selected slices can also be "featured", causing them to be offset from the center of the chart to accentuate them. Data is provided as a series of primary (Y-axis) data sets, each with one or more data values (X-axis). See the [_chart Display Options](#) for details on how to modify colors, legends, fonts, etc.

Pie charts can be made "3D" by setting the `Y3D` option which adds depth to the chart. It is suggested that `PieRound` also be set to ***false*** which creates an oval pie chart.

Example

```

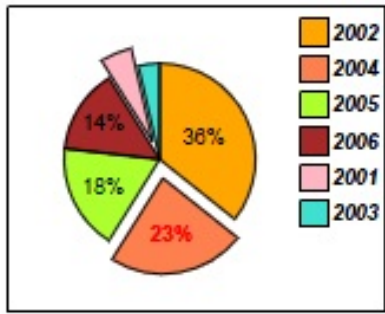
var data = [['Year', 'Data']
, [2001, 1.5]
, [2002, 10]
, [2003, 1]
, [2004, 6.5]
, [2005, 5]
, [2006, 4] ];

var options = {
  chartgap:250000,
  colors:['orange', 'coral', 'greenyellow', 'brown', 'lightpink', 'turquoise'],
  legendfont:{bold:true, italic:true},
  sort:true,
  valueformat:'[percent]',
  feature:['2002', '2005'], pieoffset:'2mm',
  featurefont:{bold:true, color:'red'}
};

_chart.pie(data, options);

```

Results in:



`_chart.vBar`

This method draws a barchart with a series of vertical bars.

Syntax

```
_chart.vBar(data [, options])
```

Parameters

data is a script array of data points to be charted. See [_chart Data](#) for formatting details. ***options*** is a script object whose properties define various presentation options. See the [_chart Display Options](#) for details.

Returned Value

true if successful, **false** if an error occurred.

Description

The ***data*** array contains a list of data sets to be drawn on the bar chart. The `_chart.vBar()` routine must be called from a containing object, typically a Group object. The chart is drawn within this object. Bar widths, scaling etc. are all automatically calculated so that the chart completely fills the container object. Bars are drawn vertically. Data is provided as a series of primary (X-axis) data sets, each with one or more data values (Y-axis). Each X-axis data set creates a cluster of one or more vertical bars for that primary value. See the [_chart Display Options](#) for details on how to modify colors, legends, fonts, etc.

Example

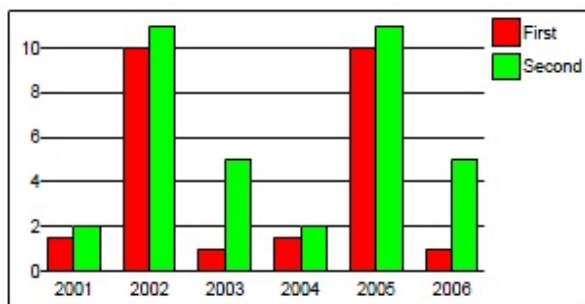
```

var data = [['Year', 'First', 'Second']
, [2001, 1.5, 2]
, [2002, 10, 11]
, [2003, 1, 5]
, [2004, 1.5, 2]
, [2005, 10, 11]
, [2006, 1, 5] ];

_chart.vBar(data);

```

Results in:



This chart shows the use of two parallel sets of bars for each X-axis data point. Colors, fonts, and other options are controlled by the options parameter. See [_chart Display Options](#) for details.

`_data` (The Data DOM)

These properties allow the scripter read-only access to the Data DOM (as opposed to the Form or Document DOM). The Data DOM represents the XML file that is read by Merge and whose data is merged into the document. The `_data` DOM represents the data for each single document in turn, not the entire data file all at once.

Property	Description
<code>_data</code>	This is the root object of the Data DOM. It is not yet populated at Start-of-Job , but is at Start-of-Document .
<code>._value</code> <code>.value</code>	Return the raw (text string) value of a data item in the XML data file. This is the only Data DOM attribute for which one can leave off the leading underscore (<code>_</code>), though that is not recommended as it is quite possible that a data file will have a <code><value></code> element in it.
<code>._name</code>	Return the name of the current Data DOM object, i.e. the tag name of an XML element.
<code>._fullName</code>	Return the fully qualified name of the current Data DOM object, all the way from <code>_data</code> down to this object's basic name, all in dotted notation. E.g. <code>_data.Document.Invoice.Header.CustomerId</code> .
<code>._parent</code> , <code>._firstChild</code> , <code>._nextSibling</code> , <code>._prevSibling</code> , <code>._nextNode</code>	These allow the scripter to walk the Data DOM in a logical fashion. Each function returns a Data DOM object with which one could access <code>._value</code> , <code>._name</code> , <code>._fullName</code> , or one of these traversal functions.
<i><code>dataDomObject</code></i> . <code>ancestor(name)</code>	(As of 3.2.001.03) Find an ancestor (not necessarily just an immediate parent) of a given name.
<i><code>dataDomObject</code></i> . <code>descendant(name)</code>	(As of 3.2.001.03) Find a descendant object with a given name.
<i><code>dataDomObject</code></i> . <code>childNamed(name)</code>	(As of 3.2.001.03) Find an immediate child (not further down descendant) of a given name.
<i><code>dataDomObject</code></i> . <code>name</code>	For any Data DOM object one can get the object that represents one of its children by using that child's <i>name</i> or <i>tagName</i> (case-sensitive).
<i><code>dataDomObject</code></i> . <code>attribute("attrName")</code>	Get the value of the named attribute (or "" if it doesn't exist) for this data node. <i>attrName</i> is case-sensitive.

It's a bit of a philosophical debating point as to whether the above are properties or functions/methods. They certainly act like properties in that they give the appearance of a static value being made available for the asking, even though background processing may have to occur to 'come up with' the answer. The following are definitely functions.

Function	Description
<code>_loadAsArray(datafield[, firstrow])</code>	Load up a series of values in the data file and represent them as an array. The optional <i>firstrow</i> is an array of column names for the subsequent data, usually used when loading data for a chart. See _chart Data for an example of its usage.
<code>_loadAsObject(datafield)</code>	Load up a single structure in the data file and represent it as a JavaScript object. See the example below.

`_loadAsObject` Example

Suppose your data file had this construct somewhere in it:

```
<ShipTo>
  <CompanyName>Perfect Printers</CompanyName>
  <Address1>425 Lansing Drive</Address1>
  <Address2>Moline, Illinois</Address2>
  <Address3>USA 61265</Address3>
  <Address4>Tel:800.555.2323 Fax:309.762.4411</Address4>
</ShipTo>
```

You could use: `var o = _data._loadAsObject("ShipTo");` to load up that structure into a JavaScript object. We could print it out with: `_logf("%s\n", o)` and we'd see these values (manually split into multiple lines here for display purposes).

```
{CompanyName:Perfect Printers, Address1:425 Lansing Drive, Address2:Moline, Illinois,
Address3:USA 61265, Address4:Tel:800.555.2323 Fax:309.762.4411}'
```

Notes

Notes

1. `_data` does not exist yet at Start-of-Job. It does by Start-of-Document.
2. If you really want to read the raw XML data file you can always:

```
var fp = _file.fopen(_job.command.data, "r");
while (true) {
  line = fp.fgets();
  if (line == null) break; // end-of-file
  // do whatever you like
}
fp.fclose();
```

That's not a recommended course of action, just advising of an available 'port in a storm' if you become desperate. It might be used as a way to get at header/control information that occurs outside of the `<Document>...</Document>` boundaries. Of course, you could also use the `XmlInput Class`.

3. In a field's event you can refer to `this._dataSource` to get the `_data` DOM object that was used to populate the field. That is a generality that should be taken with a grain of salt. Some fields are populated with script. Some other fields, likely in error, have no associated data file element. Beware of null.

See Also

[XML Data Files](#)

`_document`

This is the Document Object Model (DOM) object for the current document being processed. It enables access to field data and form field attributes. It contains a hierarchy that reflects the design's nested structure of Pages, Containers, Panes and Groups with their Fields and Text Labels, and it also reflects the various instances of those objects created as a result of merging data into the form design. For example:

```
_document.main_pane.header_pane.customer_name._value
```

The DOM is specific to the current combination of form design and data, and it reflects the current design's structure. `_document` reflects the top-level object in the document, i.e. the Form object.

Optional and Multiple Panes

One thing to be careful of is that if a Pane is optional and the data stream does not force its instantiation, then a reference to that Pane or to objects within that Pane will result in a runtime error. When scripting, you should keep the possibility of 'non-existence' of Panes in mind. See [domObj.DOM](#) and [domObj.DOMValue](#).

Another thing is that you will have multiple instances of the same Pane and in those cases you **must** take care to specify which occurrence you wish to reference. Remember that when pagination occurs (and for all events after that), there is an even greater likelihood of multiple occurrences of Panes, even ones not marked as being **Allow Multiple**. This occurs because page headers and footers may have been injected multiple times. Multiple instances of a Pane can also occur if it had to be split across pages. In that case, the Pane object will exist in the DOM on both (or more) pages, with a part of the Pane on each page. Typical error messages in such cases are `ScriptExtendedSyntax` and `DJ-GetResolveValName`.

Use of `this._parent`, `this._ancestor()` and `this._descendant()` can relieve a lot of the "have to know the occurrence number" headaches. Another technique is to stuff a value of interest, using a local reference, like `this._value`, into an `_cache` variable and then later referencing that `_cache` variable.

The following script can help you understand the current document structure:

```
for (var o=_document; o; o=o._nextNode) {
  if (pn._type == "Pane") _logf("%s\n", o._fullName);
}
```

Note that the results of that will not be the same at all events so be sure to explore the document structure in the event where you are puzzling out an issue.

Object Names with Colons

Design and Merge allow us to define object names with a colon in them. This is useful to match XML that uses namespace prefixes. However, with JavaScript we cannot use a dotted SOM notation that has colons in it. If your form contains an object with a name that includes a colon (like "Colon:test"), you can't use dotted notation, such as `_document.Colon:test`. Instead, you should use square brackets around the object. For example:

```
_document["Colon:test"].
```

Or, you can use the [DOMValue\(\)](#) or [DOM\(\)](#) functions. For example:

```
_document.DOMValue("Colon:test")
_document.DOM("Colon:test")
```

Dom Object Attributes

Fields and Text Labels have values and attributes (font, color, margins, and so on) which can be accessed and modified in script by specifying the name of the Field or Label followed by a dot followed by the attribute name.

The value of the Field or Label is accessed as just another attribute named `_value`. The DOM properties topic (See [DOM Properties](#)) identifies the many, many properties that are available for reference. In that list the attributes are marked as return or set. return indicates that the attribute value may be accessed from script, while set indicates that it may be modified using script. There are versions of these attributes which do not have the leading underscore(`_`) in their names. However, the use of the leading underscore is highly recommended so as to avoid potential conflict with the object names in a document.

Other DOMs

See also the `_page` DOM. It is rarely used and reflects only a page subset of the overall `_document` DOM.

Note that the `_data` DOM is a completely different DOM, reflecting the XML data that is applied to the form design.

`_document.abandon`

(Merge only)

Don't produce output for this document.

Syntax

```
_document.abandon()
```

Parameters

None

Returned Value

None

Description

⚠ It's really easy to put Merge into an infinite loop when using `_document.reprocess`. We suggest this option for advanced users only.

`_document.abandon` is almost certainly used with `_document.reprocess`.

This function sets an internal flag such that at the end of the current Merge processing stage, Merge will give up on this document's data set and move on to the next one unless `_document.reprocess` was also called. This may be desirable, without `reprocess`, if validations detect absurdities in the data. We feel it should be the job of the application producing the data to validate, but this could perhaps be used as some last-second failsafe. Where it is more likely applicable, and even then only rarely, is in taking action based on information that only the presentation software (Merge) knows. For the most part that is the page count (see the example below). The internal `abandon` and `reprocess` flags are set to false at the start of every Merge processing loop.

Examples

If data is detected to be bad...

```
if (this._value == "reeks") _document.abandon();
_logPrintf("Document ...identifying data... was rejected, and not output\n");
```

Consider that one and two-page documents are folded to fit into a simple business envelope, but three or more require a large envelope and either no or different folding. Well, the amount of paper needed depends on the size of the headers used. So the idea is to switch headers based on the number of pages being generated. But we don't know the number of pages until we paginate. Generally, that's too late to substitute in a different sized header. So we may want to abandon the current rendering and do a new one with a different header in use.

At the start of job and end of document events...

```
_cache.useHeader = "normal";
```

At the pagination completed event...

```
pageCount = _printer.getPageCount();
if (pageCount > 4 && _cache.useHeader == "normal") {
  _cache.useHeader = "manyPagesHeader";
  _document.reprocess(); // Set flag to start this document over again
  _document.abandon(); // Set flag to abandon at the end of the current stage
}
```

In the Data Merged event for each header...

```
this._visible = (_cache.useHeader == "the one that applies");
```

Certainly, you should pick the most common scenario as the default rather than incurring reprocessing overhead.

One could "go wild" by reprocessing to handle repagination if you change the document contents significantly at the Pagination Complete stage. This is not recommended.

`_document.bookmark`

Create PDF bookmarks.

Syntax

```
_document.bookmark(key, value)
```

Parameters

key specifies the main bookmark category, for example, "CustomerContact".


value specifies a sub-bookmark below the category, for example, "ABC Company".

Both parameters must be strings.

Returned Value

None.

Description

 Note that the PDF driver bookmark feature must be enabled either with the PRT file by setting the `<Bookmark>Yes</Bookmark>` option, or by using the `-PDFBookmark Yes` command-line option. Without that setting, the bookmarks will still be encoded in the PDF and could be used by the PDFExtract tool, but they would not appear in the Adobe Reader bookmark navigation pane. To see the bookmark navigation pane in Adobe Reader use **View > Show/Hide > Navigation Panes > Bookmarks**. In Foxit Reader use **View > Navigation Panels > Bookmarks**.

Each time the `_document.bookmark` routine is called the current page is added to a table of bookmarks that the PDF viewer can display. A nested hierarchy of bookmarks can be defined by setting `key` to a list of key names separated by the "|" (vertical bar character). For example, "CustomerContact|ABC Company". The "CustomerContact" bookmark would open a list of companies that included "ABC Company". The "ABC Company" bookmark would open a list of values, for example, contact information, as in:

```
_document.bookmark("CustomerContact|ABC Company", "Tony Blue 555.123.4567");
```

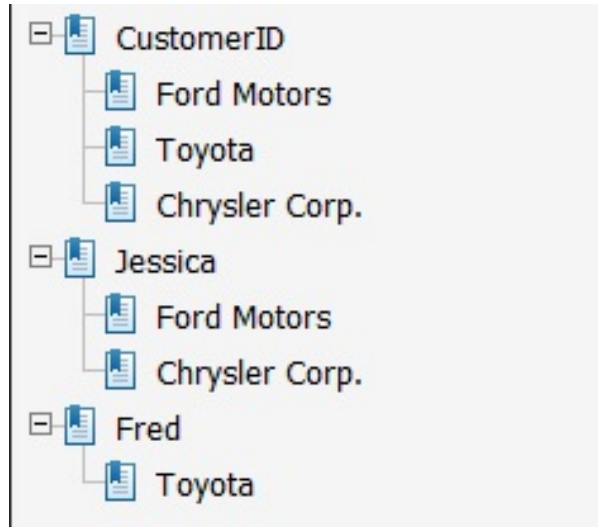
Each individual value would go to the specific first page of the corresponding document. In the PDF viewer, when the bookmark associated with ***value*** is selected, the user will be repositioned to the first page of the document associated with that bookmark.

Examples

Consider a scenario where you have an invoice document. Each invoice header page has a "CustomerID" Field which is the customer's name and a "SalesRep" Field which is the company's sales representative. Merge has been configured to run a large batch of documents and to combine them into a single PDF file containing all the Invoices. On the "CustomerID" Field you add the following script:

```
_document.bookmark("CustomerID", this.value);
_document.bookmark(SalesRep.value, this.value);
```

The first line causes the PDF viewer to show a bookmark group call "CustomerID". Under that name, there will be an entry for every customer that takes you to the invoice for that customer. The second line creates a bookmark group for each unique SalesRep name. Under each sales rep there will be a list of customer names assigned to that sales rep. The result in the PDF viewer would appear as:



`_document.reprocess`

Do this same document data again.

Syntax

```
_document.reprocess()
```

Parameters

None

Returned Value

None

Description

⚠ It's really easy to put Merge into an infinite loop when using `_document.reprocess`. We advise that only experts in DocOrigin use this.

`_document.reprocess` is almost certainly used with `_document.abandon`.

This function sets an internal flag such that at the end of the current Merge processing stage, Merge will process this document's data set over again. This may be desirable, without `abandon`, as some obscure way to produce multiple copies. You could set an `_cache` variable to "Customer Copy" or "Store Copy", for instance, and use that `_cache` variable to populate a footer, but there are other better ways to achieve that goal. Where it is more likely applicable, and even then only rarely, is in taking action based on information that only the presentation software (Merge) knows. For the most part that is the page count (see the example below). The internal `abandon` and `reprocess` flags are set to false at the start of every Merge processing loop.

Examples

Consider that one and two-page documents are folded to fit into a simple business envelope, but three or more require a large envelope and either no or different folding. Well, the amount of paper needed depends on the size of the headers used. So the idea is to switch headers based on the number of pages being generated. But we don't know the number of pages until we paginate. Generally, that's too late to substitute in a different sized header. So we may want to abandon the current rendering and do a new one with a different header in use. At the **Start of Job** and **End of Document** events...

```
_cache.useHeader = "normal";
```

At the **Pagination Completed** event...

```
pageCount = _printer.getPageCount();
if (pageCount > 4 && _cache.useHeader._value == "normal") {
  _cache.useHeader = "manyPagesHeader";
  _document.reprocess(); // Set flag to start this document over again
  _document.abandon(); // Set flag to abandon at the end of the current stage
}
```

In the **Data Merged** event for each header...

```
this._visible = (_cache.useHeader == "the one that applies");
```

Certainly, you should pick the most common scenario as the default rather than incurring reprocessing overhead. One could "go wild" by reprocessing to handle repagination if you foolishly change the document contents significantly at the **Pagination Complete** stage. Not recommended. And watch for loops.

`_document.serializeForm`

(As of 3.1.002.03)

Serialize this document to the file.

Syntax

```
_document.serializeForm(file)
```


Parameters

file specifies the output file. Should be an XATW file name.

Returned Value

(As of 3.2.001.09) **true** if successful, **false** if an error occurred.

Description

 This function is not for everyday use but may be useful for advanced users in some specific scenarios.

Your script might make any number of property changes to the form template DOM (like visibility, margins, etc.) Then you can call `serializeForm(file)` and it will write out the XATW with all of the current form DOM. In the context of this function it makes sense to modify the form DOM only up to and including the **Start of each Document** event. After that event, all your changes will be reflected in a document being generated, but not in the form template, although you may call this function on any event.

`_file` (Read/Write Files)

Description

`_file` is a JavaScript object which has a number of functions related to file handling. One of these functions is `_file fopen`. This function opens a file for either reading or writing. It creates a new JavaScript object which is returned as the value of the function call. That object, in turn, has functions associated with it to read from and write to the file.


Semi-automatic File Path/Name Resolution

DocOrigin likes to discourage the use of hard-coded file paths. Consequently, we urge you to use the `$X` folder mappings as described in [\\$X String Substitutions](#). Script functions with file or folder name parameters will automatically resolve those `$X` folder mapping references. That takes care of a large number of cases automatically.

However, if your file name parameter also uses the various `%` placeholders as described in [File Naming Conventions](#), those `%` placeholders will not be automatically resolved. The user is expected to have control over their file names and should use `_file.resolveName` or `_resolve` if required when passing such templated file names to a script function.

Functions

- `_file.copy(fromFile, toFile)`
Copy a file.
(As of 3.2.001.02) **toFile** can be just the destination directory, and not include the file name. **fromFile** cannot include wildcards. You copy one file at a time.
(As of 3.3.002.01 on Windows) The “to” parameter can handle the “prt:.” prefix as well. The return value is the result code of the copy operation.
- `_file.date(fileName)`
Return the 'last modification date' of a file. The returned format is "yyyy/mm/dd hh:mm:ss". This can be used directly in a new `Date(...)` construct.
- `_file.delete(fileName)`
Delete a file.
(As of 3.0.005.07) The file name can contain the wildcards:
* - matches any number of characters
? - matches a single character in a specific position
No error is issued if no such file exists.
- `_file.errno()`
Returns the last error code from a previous file operation.
- `_file.exists(fileName)`
(As of 3.0.005.07) Determine if the named file exists or not.
(As of 3.2.001.02) Supports file name wildcards (like "`$E/* .prn`").
- `_file.findClose()`
Terminate a wildcard file search.
- `_file.findNext(fileSpec [, directories])`
Begin or continue a wildcard file search. Returns "" at end-of-list.
- `_file.gets()`
Read a single line from the [console](#).

 Note that this is `gets`, not `fgets`. Do not confuse this with `fp.fgets` which reads from a file.

- `_file.fopen(fileName, fmode[, "BOM"])`
Open a file. Returns a file pointer object (`fp`). For write mode purposes you can force the writing of a BOM by supplying the optional third parameter with a value of `BOM`.
- `_file.fromBase64(base64, fileName)`
(As of 3.2.001.01) Decode a base64 string into a file. Use this for binary data with NULLs. Returns a Boolean value representing the operation's success/failure.
- `_file.getFileNamePart(fileName, part)`
Return a selected portion of a file name. Valid part names are: `Full`, `Drive`, `Path`, `DrivePath`, `Name`, `Ext`, `NameExt`, and `NoExt`. Note that `DrivePath` and `NoExt` require build 3.0.001.10 or greater. If the `fileName` does not have a full path, the full path will be computed based on current circumstances.

```
var f = "C:/DocOrigin/D0/Samples/empty.xml";
s = _file.getFileNamePart(f, "Full");      // "C:/DocOrigin/D0/Samples/empty.xml"
s = _file.getFileNamePart(f, "Drive");    // "C:"
s = _file.getFileNamePart(f, "Path");    // "/DocOrigin/D0/Samples"
s = _file.getFileNamePart(f, "DrivePath"); // "C:/DocOrigin/D0/Samples"
s = _file.getFileNamePart(f, "Name");    // "empty"
s = _file.getFileNamePart(f, "Ext");     // ".xml"
s = _file.getFileNamePart(f, "NameExt"); // "empty.xml"
s = _file.getFileNamePart(f, "NoExt");   // "C:/DocOrigin/D0/Samples/empty"
```

- `_file.getOpenFileName([folder], [title], [fileTypes], [defaultFile])`
Display the **File>Open** dialog. Returns the user-chosen file name. If the user clicks Cancel, `null` is returned.
- `_file.getSaveFileName([folder], [title], [fileTypes], [defaultFile])`
Display the **File>Save** dialog. Returns the user-chosen file name.
- `_file.getOpenFolderName([initFolder], [title])`
(As of 3.2.001.02) (Windows only) Display the Folder Open dialog. Returns the user-chosen folder name. If the user clicks Cancel, `null` is returned.
- `_file.isDir(dirPath)`
(As of 3.0.005.07) Determine if the provided path is a directory or not.
- `_file.mkdir(dirPath)`
Create a file directory.
(As of version 3.0.003.19) This will also handle multi-level paths such as "C:/A/B/C". Return codes: **0** (Ok), **1** (path was already there), **-1** (a failure).
- `_file.readFile(fileName[, convertLineEndings])`
Read the contents of `fileName`. Returns the file contents as a JavaScript string. If the file does not exist, `null` is returned.
(As of 3.2.001.01) `convertLineEndings` (defaults to true) requests line ending conversion from Windows to Unix (`\r\n` to `\n`). Note that not doing line-ending conversions may result in faster processing but newlines and linefeeds are left in the returned string.
- `_file.readIniFile(fileName)`
Read an INI file. Returns a JavaScript object with all the data from the INI file.
- `_file.readPdfData(fileName)`
(As of 3.0.003.13) Open a DocOrigin generated PDF and extract the embedded data. See the `-{prt}option` `"-PDFembedData Yes"`. Returns a text string of the embedded-in-the-PDF XML data used to produce the document.

- `_file.readPrmFile(fileName)`
Read a DocOrigin PRM file. The returned value will be a JavaScript object with name/value properties corresponding to the PRM file entries. If the file does not exist or is not properly formatted, **null** is returned.
- `_file.remove(fileName)`
Synonym for the `_file.delete` function.
- `_file.rename(fromFile, toFile)`
Rename a file. Returns **0** if successful. Note that you cannot change the path of the file, only its name.
- `_file.resolveName(template)`
Do variable substitution into a *template* intended to be used as a file name. See `_resolve` for details. In this "file name template" context, `[]` references to fields which don't exist will be deleted.
- `_file.rmdir(directoryName)`
Delete the specified directory (which must be empty).
- `_file.tempname(dir)`
Create an empty temporary file and return its name as a string. Note that *dir* is ignored. On Windows, it uses the TMP environment variable. Under Unix, it uses the TMPDIR environment variable if it is set, otherwise, it uses /tmp. The folder pointed to by %TMP% or \$TMPDIR must exist. In either case, it uses an AtwTempFiles folder under the nominated temp folder; that subfolder is created automatically. Doing an `_file.resolveName("$T/xxxx_%u.ext")` may be a preferred option in that it gives you more control over the name of the file including its file extension. Use %u to ensure it has a unique name.
- `_file.toBase64(fileName)`
(As of 3.1.002.10) Convert the contents of a file to a base64-encoded string.
- `_file.unzip(zipPath, destPath [, overwrite])`
(As of 3.1.002.06) Extract files from a zip archive.
- `_file.writeFile(fileName, string [, "-BOM"] [, mode])`
Writes string to the designated file. This function opens *fileName* for write (overwriting if necessary); writes the string, and closes the file. The output is always written in UTF8 format. Normally a BOM is written.
(As of 3.0.005.07) You can suppress that by supplying the optional third parameter with the value **-BOM**.
(As of 3.2.001.02) the *mode* parameter was introduced. Use the value **a** to initiate append mode (creates the file if it does not exist). The `_file.writeFile` function returns a Boolean **true** if successful and **false** if something went wrong.
- `_file.zip(zipPath, filePath [, addMode])`
(As of 3.1.002.06) Add a file to the zip archive.

See Also

[_file.fopen](#)
[fp.fprintf](#)
[fp.fclose](#)
[fp.fgets](#)
[fp.fputs](#)
[_message](#)
[_printf](#)
[_tracef](#)
[_toBase64](#)
[_fromBase64](#)

`_file.fopen`

Open a file for reading or writing.

Syntax

```
_file.fopen(filename, fmode [, "BOM"])
```

Parameters

filename the name of a file to be opened.

fmode The type of open. This setting is a standard C file open mode, typically *r* for "read", *w* for "write" or *a* for "append".

Supplying a third parameter of value **BOM** (Byte Order Mark) will indicate that a BOM is required on output. Otherwise, no BOM will be written. Note that this is a different default than what applies to `_file.writeFile()`. If the open mode is *w* or *a*, i.e. for output, then the file will be immediately created. For mode *w* it will silently overwrite an existing file of the given *filename*.


Returned Value

If successful, `fopen` returns a new object that represents the File Pointer `fp` of the open file. You can then use this new `fp` object to read or write. If the file cannot be opened, **null** is returned.

Functions

You must use the returned `fp` object from `fopen` in order to read or write a file. The following functions can be used with this file pointer:

- `fp.fclose()`
Close the file specified by `fp`.

 **WARNING** - Files are NOT automatically closed by a DocOrigin application until the application itself exits. If your application repeatedly opens files and does not explicitly close them when it has finished with them, you may experience performance degradation. In extreme cases the application may fail due to having too many files open (even if it is the same file opened over and over without being closed; close your files!).

- `fp fflush()`
Force the file's memory buffer to be written to the file.
- `fp fgetc()`
Read a single byte (an integer between 0 and 255) from the current open file specified by `fp`. Returns **-1** when at end-of-file. Multi-byte characters will be returned one byte at a time. Users of these byte-oriented functions will likely be interested in the basic JavaScript functions of `charCodeAt()` and `String.fromCharCode()`.
- `fp fgets()`
Read a single line from the current open file specified by `fp`. Returns **null** when at end-of-file. The file is expected to be encoded in UTF8 (7-bit ASCII is a subset of UTF8).
- `fp fprintf(format [, p1 [, p2 ...]])`
Write a formatted line of text to the file. See `_printf` for details.
- `fp fputc(an integer number)`
Write a single byte to the file. The number should be $0 \leq n \leq 255$. Note that no automatic UTF8 encoding will occur. It's very raw.
- `fp fputs(string)`
Write a line of text to the file. The line will be encoded in UTF8. (Internally, strings are in multi-byte Unicode.)

- `fp.fread([count])`
Read up to `count` bytes of data from the input file. This returns a string containing the bytes read. If `count` is not specified, everything up to and including the next newline is read. This bypasses any implied assumption that characters are encoded as UTF8. Instead, the raw bytes are read. In theory you could read binary files with this function but in practice null bytes will terminate the returned string and processing non 7-bit ASCII characters will be nigh impossible. See `fp.fwrite()` as well.
- `fp.fseek(offset, 0|1|2)`
Move to a new location in the file. `0` is start-of-file, `1` is current offset, `2` is end-of-file. `offset` can be negative.
- `fp.ftell()`
Returns the next read/write position (i.e. the current offset) in the file.
- `fp.fwrite(string[, count])`
Write `count` characters to the file. This writes the data without the usual conversion to UTF8. If `count` is not specified, the entire string is written out.

Example

```
var fp = _file.fopen("myfile.txt", "w"); // open for write
if (fp) {
    fp.fputs("Hello World");           // write to file
    fp.fclose();
}
```

Naming Conventions

`fp` is an extremely common variable name to use for a file pointer. It's readily understood by maintainers of your code; great for when dealing with only one file. As a suggestion, you might consider using `fpIn` and `fpOut` for a classic case where you have an input file and an output file. Another useful method is to prefix `fp` to the file's purpose. E.g. `fpForm`, `fpData`, `fpControl`. Using that convention helps make your script more readily understood (and thus maintained). This is akin to the habit of using what is called "Hungarian notation" wherein strings would start with "s" (e.g. `sName`); Booleans start with "b" (e.g. `bFound`); integers start with "i" if they are 0-based, or with "n" if they are 1-based (e.g. `iIndex`, `nCount`). Similarly, it is useful to adopt the convention of prefixing your file name variables with `fn`. E.g. `fnForm`, `fnData`, `fnControl`. Correlating your `fn`'s and your `fp`'s is pretty helpful.

`_file.findNext`

Cycle through a list of files that match a file name mask

Syntax

```
_file.findNext(nameMask[, directories])
```

Parameters

nameMask is a wildcarded file (or directory) name mask. Typically something like `"*.xml"` or `"subFolder/*.txt"`


(As of 3.1.002.06) ***directories*** is an optional Boolean parameter indicating to look for directories instead of files. The default behavior is to look for files.

Returned Value

The name of the next file (or directory) that matches the name mask. Returns `""` if there are no more matches for the name spec.

Description

This function is used to determine the names of all the files or directories that match a given name mask.

 It is an extremely good idea to call `_file.findClose()` before beginning a loop of calls to `_file.findNext()`. `_file.findClose()` ensure that you start a new search rather than continuing on from some previous `_file.findNext()` search.

Example

See [domObj.appendInstance](#) for an example of usage of these functions.

`_file.getOpenFileName`

Prompt user for a file to Open.

Syntax

```
_file.getOpenFileName([folder], [title], [filetypes], [defaultfile])
```

Parameters

All parameters are optional or can be replaced by null.

folder - the local file folder that should be displayed to start with.

title - alternate text to appear in the Open dialog's title area.

filetypes - a string that customizes the dialog with default file types to display. If omitted, all files are displayed. The string must be formatted as a sequence of pairs of descriptors - a text (display) string followed by one or more wildcard file extension specifications. Each item in the list must be separated by the vertical bar character.

Example:

```
"Form Files|*.xatw|Data Files|*.xml;*.dat|All Files|*.*"
```

The above string displays 3 filetype options: one for all XATW files, one that displays DAT and XML files, and one that displays all files. By default, all files are shown.

defaultfile - a default filename to be displayed in the dialog. Only the filename and extension (no folder path) is required.

Returned Value

If the user selects a file, the full path+filename is returned. Otherwise returns **null**.

`_file.getSaveFileName`

Prompt user for a file to Save.

Syntax

```
_file.getSaveFileName([folder], [title], [filetypes], [defaultfile])
```

Parameters

All parameters are optional or can be replaced by **null**.

folder - the local file folder that should be displayed to start with.

title - alternate text to appear in the Save dialog's title area.

filetypes - a string that customizes the dialog with default file types to display. If omitted, all files are displayed. The string must be formatted as a sequence of pairs of descriptors - a text (display) string followed by one or more wildcard file extension specifications. Each item in the list must be separated by the vertical bar character.

Example:

```
"Form Files|*.xatw|Data Files|*.xml;*.dat|All Files|*.*"
```

The above string displays 3 filetype options - one for all XATW files, one that displays DAT and XML files, and one that displays all files. By default, all files are shown.

defaultfile - a default filename to be displayed in the dialog. Only the filename and extension (no path) are required.

Returned Value

If the user selects a file, the full path+filename is returned. Otherwise returns **null**.

`_file.readIniFile`

(As of 3.1.002.10)

Load an INI file into a JavaScript object

Syntax

```
_file.readIniFile(filename)
```

Parameters

filename is the name of the INI file to be read. Note that the usual `$X`-type of substitution is also applied. The file is required to contain either UTF8 or plain ASCII characters.

Returned Value

The contents of the INI file is converted to a JavaScript structure. **null** is returned in case of an error, e.g. file not found.

Description

For INI file entries without a `[section]` (i.e. in the default `""` section), these are simple `name:value` entries in the JavaScript object. Entries within a `[section]` are stored as `section:{array of name:value pairs}`. See the Example below.

Example

```
; sample ini file

first="ABCDEF"
second="12345"

[Section1]
third=123
fourth=456
```

The above can be loaded by:

```
var oIni = _file.readIniFile("thefilename");
```

The resulting value of variable `oIni` will be:

```
oIni={
  "first":"ABCDEF",
  "second":"12345",
  "Section1":{
    "third":"123",
    "fourth":"456"
  }
}

first = oIni.first; // simple reference to a value
home = oIni["home address"]; // when the keyword contains a non-alphanumeric char.
third = oIni['Section1'].third; // reference to a section
```

ⓘ DocOrigin Extended ini Support

Please see [_profile \(Access Profile Files\)](#) for descriptions of DocOrigin's extended support for INI files. These include:

1. Conditional and unconditional includes of other INI files
2. Special character handling within values
3. "heredoc" support
4. Default section continuation using `[]`
5. Auto text substitution using the `[Define]` section
6. Key names may be any string not just a single word
7. Values can optionally be surrounded by `"` or `'`.

⚠ The `readIniFile()` function is similar to the `_profile.load()` function, but more general-purpose, and is recommended unless you are specifically wanting to reset the internal Profile settings. `readIniFile` will not do any of the automatic actions that may be specified in a true profile file.

`readIniFile` does not have specific sibling functions for enumerating sections and/or keys within sections. As the returned object is a standard JavaScript object, the usual

```
for (var x in oIni)...
```

constructs should suffice for enumeration needs. If `typeof x` is an object then `x` is a section name.

INI files are often used for language translation tables. In such cases, it is not unusual that the key is a multi-word string. Eg.

```
Continues on the next page=Fortsättning på nästa sida
```

You will not be able to reference `oIni.Continues on the next page` but you can (must) reference `oIni["Continues on the next page"]`. Since this example is likely from the `[Swedish]` section you might be referencing `oIni.Swedish["Continues on the next page"]`. All the usual JavaScript capabilities apply.

See Also

[_profile \(Access Profile Files\)](#)
[_file.readPrmFile](#)

`_file.resolveName`

String substitution

Syntax

```
_file.resolveName(text)
```

Parameters

text - a text string with all manner of placeholders to be replaced. These include [fieldname] or [!autofield] placeholders, %x placeholders (see [File Naming Conventions](#)), and \$X placeholders (see [\\$X String Substitutions](#)).

Returned Value

A new string with substitutions is done.


Description

This routine is a script equivalent of the [fieldname] and [!autofield] substitution available in normal text labels. See the [Auto/Embedded Fields](#) section for a list of available [!autofield] variables. If used in the context of Merge, i.e. in the script in a Form, where there is a Document DOM available, `_resolveName` looks for substrings of the form [fieldname] and replaces them with the Field's value from the current document. This function uses the same substitution logic as the Merge embedded fields, including [!pagenum] etc. If the text between the left and right square brackets is not a known name the text and square brackets are left unchanged in the text string.

Unlike `_mergeEmbedded`, in any context, e.g. Merge, RunScript, or Job Processing scripts, `_file.resolveName` will also do substitution of the % variables (typically date elements) that is done for output file names. See any PRT file.

Further, any \$X style variable will be substituted as well. Note that the string is presumed to be a file name, so replacement of characters that are invalid for file names is done. Also, backslashes are converted to forward slashes. By contrast, see `_resolve`.

`_file.resolveName` is generically related to files and hence is a function of the `_file` object itself. It is not a function of an opened file instance, i.e. (not) `fp._resolveName`.

 CAUTION: `_file.resolveName` will create any paths that are referred to in the provided name template string. It does not create the file, but it does create the path. If you do not want that to happen, use `_resolve`.

Example

In RunScript one might use:

```
myFileName = _file.resolveName("$T/CriticalStats_%U.txt");
fp = _file.fopen(myFileName, "w");
```

See Also

[Auto/Embedded Fields](#)

[_auto](#)

[_resolve](#)

[_mergeEmbedded](#)

`_file.zip`

(As of 3.1.002.06)

Add a file to zip archive.

Syntax

```
_file.zip(zipPath, filePath[, addMode])
```

Parameters

zipPath is the name of a zip file to be created or modified.

filePath is the name of a file or folder to be added to zip archive. As of 3.2.001.05, wildcard for filenames are supported (like `in/file*.pdf`).

addMode is a string that defines whether archive is truncated or appended. Possible modes are **create** (default) or **add**. If mode is **add** but zip file doesn't exist then **create** mode is used.

Returned Value

Boolean value represents the success of the operation.

See Also

[_file.unzip](#)

`_file.unzip`

(As of 3.1.002.06)

Extract files from a zip archive.

Syntax

```
_file.unzip(zipPath, destPath[, overwrite])
```

Parameters

zipPath is the name of a zip file to be read.

destPath is the path where to put extracted files.

overwrite is a Boolean flag indicating whether to overwrite the destination file or not in case of conflict. Default is ***false***. If set to ***false*** and there is a conflict, then the function fails.

Returned Value

Boolean value represents the success of the operation.

See Also

[_file.zip](#)

fp fclose

Close a previously opened file.

Syntax

```
fp fclose()
```

Parameters

None

Returned Value

None

Description

Closes a currently open file.

Example

```
var fp = _file.fopen("myfile.txt", "w"); // open for write
fp.fputs("Hello World"); // write to file
fp fclose();
```


fp.fgets

Read a line from a previously opened file.

Syntax

```
fp.fgets()
```

Parameters

None

Returned Value

Returns the next line of text in the file. **null** if at end of file.

Description

A previous call to `_file fopen` must have been made. That `fopen` returns a new file pointer object `fp`. That object must then be used to call `fgets`. The file being read is expected to be encoded in UTF8.

Example

```
var fp = _file.fopen("myfile.txt", "r");// open for read
if (fp) {
    var s = fp.fgets(); // read first line
    _message("First line='%s'", s);
    fp.fclose();
}
```

fp.fprintf

Writes formatted text to a file

Syntax

```
fp.fprintf(format [, p1 [, p2 ...]])
```

Parameters

format is a template string to be written to the file. It is similar to a C-format string with embedded %s, %d, and %f markers that get replaced by parameters passed to `fprintf`. For a more complete description of the formatting, process see [_printf](#).

p1, *p2*, ... are parameters to be substituted into the *format* string.

Returned Value

None

Description

A previous call to `_file.fopen` must have been made. It returns a new file object `fp`. That object must then be used to call `fprintf`.

Example

```
var fp = _file.fopen("myfile.txt", "w"); // open for write
var sName = "Wayne Hall";
fp.fprintf("Name='%s'\n", sName);
fp.fclose();
```

See Also

[_logf](#)
[_message](#)
[_printf](#)
[_tracef](#)

fp.fputs

Write a string to a previously opened file.

Syntax

```
fp.fputs(string)
```

Parameters

string is a string of characters to be written to the output stream. fp a file object previously created by `_file fopen`.

Returned Value

None

Description

A previous call to `_file fopen` must have been made. It returns a new file object fp. That object must then be used to call fputs.

Example

```
var fp = _file.fopen("myfile.txt", "w"); // open for write
fp.fputs("Hello World"); // write to file
fp.fclose();
```

`_fromBase64`

(As of 3.1.002.10)

Convert a Base64-encoded string to plain text.

Syntax

```
_fromBase64(string [, bytesperchar])
```

Parameters

string is a base64-encoded string value to be converted.

bytesperchar is the coding size of the characters in the *string* parameter. The default is **1** indicating that each string character is to be converted to a single unicode output character. If the source string contains two encoded bytes that represent a single unicode character, then set *bytesperchar* to **2**.

Returned Value

Returns the resulting unencoded string

Example

```
var str = "DocOrigin 2020";
var b64 = _toBase64(str); // "RG9jT3JpZ2luIDIwMjA="
var txt = _fromBase64(b64); // "DocOrigin 2020"

var str2 = "保险单号";
var b64 = _toBase64(str2, 2); // "T92WaVNVU/c="
var txt = _fromBase64(b64, 2); // "保险单号"
```

See Also

[_toBase64](#)

[_file.toBase64](#)

[_file.fromBase64](#)

`_fromDOUnits`

(As of 3.1.002.10)

Convert a Base64-encoded string to plain text.

Syntax

```
_fromDOUnits(number[, units, decimals])
```

Parameters

number is an integer value of internal DocOrigin units to be converted.

units is a string representation of a common units code (in, ", cm, mm, pt, pts).

decimals is an integer value of the number of decimals to report. The default is **3** so as to match what the Design GUI reports.

Returned Value

Returns a floating point number of the specified common units, with the unit code appended, as a string.

Description

This routine converts an internal DocOrigin coordinates value (microns) into a string such as "1.25in". An invalid or missing common unit code results in the use of the default locale measurement (cm in case of Metric and inch otherwise).

Example

```

_fromDOUnits(2000000, "in");           // returns "2.000in"
_fromDOUnits(this._absTop);           // returns cm or inch depending on current locale
_fromDOUnits(2020000, "cm", 5);       // returns "5.13080cm"

```

The expected usage here is simply to aid in debug output where you use `_logf` to report an object's height or position but do so in units that are more in keeping with the units you used in designing the form.

See Also

[_toDOUnits](#)
[Script Units](#)

`_fromXmlString`

(As of 3.2.001.06)

Convert an XML-encoded string to plain text.

Syntax

```
_fromXmlString(string)
```

Parameters

string is an XML-encoded string value to be converted.

Standard set of XML symbols that has to be escaped are handled - `<>'"&`.

Returned Value

Returns the resulting unencoded string

Example

```
_fromXmlString("<>'\"&"); // " <>'\"&"
```

See Also

[_toXmlString](#)

`_inlineToRtf` (Central "\x" to RTF)

(As of 3.1.001.01)

Converts Central inline formatting to RTF.

Syntax

`_inlineToRtf`(*Central data with \x style inline commands*)

Parameters

An Adobe (JetForm) Central data stream string.

Returned Value

That input string with inline commands is converted to RTF format.

Description

In Adobe (JetForm) Central, the data stream could include "inline commands" that could be used to style text, e.g. bold, underline, italicize. DocOrigin uses the official RTF syntax for such operations. This function strives to convert the "Central" syntax to the RTF way. This is used internally in the `ConvertDatToXml` filter but has been made available for usage in a script. This conversion handles:

- `\b1` and `\b0` (bold)
- `\i1` and `\i0` (italic)
- `\u1` and `\u0` (underline)
- `\fs` (font size)
- `\fn` (font name)
- `\push (rtf {)`
- `\pop (rtf })`
- `\li` (indent -- all we manage is to insert a tab)
- `\f#` (select font by number -- discarded since those numbers are not correlated to any typeface name)
- `\u+xxxx.` (Unicode character) (as of 3.1.001.08)
- `\n` - becomes a newline
- Other `\???` sequences are discarded, with efforts to detect Windows-style file names so that they are not discarded.

The above handles very much the vast majority of text style inline commands. If you encounter others you will have to pre- or post-process the data.

Example

```
var rtf = _inlineToRtf("The \\b1.quick\\b0. fox...");
```

See Also

[ConvertDatToXml Filter](#)

`_job` (Current Job and Command Line Parameters)

Whenever a DocOrigin script program is run a `_job` object is created. This object stores all command line settings in a subObject called `_job.command`. In some situations, it also carries job processing options and settings. The information is all `ReadOnly`. When used by the FolderMonitor program, some additional information is available that indicates the FolderMonitor job status.

Properties

<code>_job.command.x</code>	A sub-object that contains all command line settings. All command line name/value pairs are available. For instance, the current logfile name can be found at <code>_job.command.logfile</code> . The Form passed to Merge is <code>_job.command.form</code> .
<code>_job.datafile</code>	<i>(FolderMonitor only)</i> The name of the current data file being processed.
<code>_job.name</code>	<i>(FolderMonitor only)</i> The current job name. This is the job name as determined by the FolderMonitor Job Name Discovery .
<code>_job.options.x</code>	<i>(FolderMonitor only)</i> Other options from the Job Name Discovery.

Functions

Returns the resulting unencoded string:

<code>_job.logf()</code>	Dump <code>_job</code> content to log file.
<code>_job.setCommandOption(name, value)</code>	Set one of the <code>_job.command.x</code> options and the global option it represents.

See Also

[Job Name Discovery](#)
[_printer](#)

`_locale` (Format Currency, Number, Date/Time)

These functions format numbers, currency, date, and time values in localized language/country format.

Functions

<code>_locale.create([code])</code>	Create new or get default locale object.
<code>_locale.setDefault(code)</code>	<i>(As of 3.2.001.01) (Merge only)</i> Set default locale.

Usage

To use the `_locale` features you must create an instance of the `_locale` object as follows:

```
var loc = _locale.create("fr_CA"); // French-Canadian
```

This new `loc` object can now be used to do various conversions to the specified locale (French Canadian in this example). The parameter to `_locale.create()` is a string comprised of a 2-character language code, followed by an underscore, followed by a 2-character country code. If you do not supply a locale name (`_locale.create()`) then you will get a default locale object. DocOrigin uses the IBM ICU tools to do all locale conversion. They use standard Language Codes specified in the [ISO-639 standard](#). The Country Codes are from the [ISO-3166 standard](#).

Object functions

<code>loc.country()</code>	Returns a 3-character country code. For example when using locale 'en-US' this routine will return "USA".
<code>loc.currency()</code>	Returns a 3-character currency code. For example when using locale 'en-US' this routine will return "USD".
<code>loc.currentDate([style[, timezone]])</code>	Returns the current date.
<code>loc.currentTime([style[, timezone]])</code>	Returns the current time.
<code>loc.formatCurrency(num)</code>	Returns a text string formatted according to the current locale's currency conventions.
<code>loc.formatDate(num[, style])</code>	Creates a localized date string.
<code>loc.formatNumber(num)</code>	Returns a number formatted into a text string according to the specified locale's conventions.
<code>loc.name()</code>	Returns the locale's name (e.g. "fr_CA").
<code>loc.parseCurrency(currencystring)</code>	Convert a formatted currency value to JavaScript number. This routine will remove currency symbols and deal correctly with comma vs. dot separators as dictated by the locale.
<code>loc.parseDate(datestring[, style])</code>	Convert formatted date to a number.

<code>loc.parseNumber(<i>numberstring</i>)</code>	<p>Convert formatted number to a JavaScript number. Note that this routine assumes the <i>numberstring</i> is correctly formatted for the locale. For example:</p> <pre>var loc = _locale.create("fr_FR"); // France n = loc.parseNumber("12.345,67"); // returns 12345.67</pre>
<code>loc.spellOut(<i>number</i>[, <i>caps</i>])</code>	<p>Convert a number to words. This function is only available for a limited number of languages. It converts a number like 154 to "one hundred fifty-four". If <i>caps</i> is set to <i>true</i> the text will be "One Hundred Fifty-four".</p>
<code>loc.spellOutCurrency(<i>number</i>, <i>format</i>[, <i>case</i>])</code>	<p>convert a currency number to words. <i>format</i> is a text string with an embedded %s to mark where the currency "dollars" should be embedded. Optionally, a second %s or %d can be used to embed the "cents" as either a string or 2 decimal digits. <i>case</i> can be <i>upper</i> to shift all spelled-out text to upper case or <i>mixed</i> to make the first letter of each word upper case. Default is all lowercase. Example:</p> <pre>var loc = _locale.create("en_US"); this._value = loc.spellOutCurrency(123.46, "%s Dollars and %d Cents*"); // results in *one hundred and twenty-three Dollars and 46 Cents* this._value = loc.spellOutCurrency(123.46, "%s Dollars and %d Cents*", "mixed"); // results in *One Hundred and Twenty-three Dollars and 46 Cents* this._value = loc.spellOutCurrency(123.46, "%s Dollars and %s Cents*", "upper"); // results in *ONE HUNDRED AND TWENTY-THREE Dollars and FORTY-SIX Cents*</pre>

`loc.currentDate`

Format the current date

Syntax

```
loc.currentDate([style [, timezone]])
```

Parameters

style is the style to format the date string, possible styles are:

Short	Default style. The most terse form, as in "3/2/09"
Medium	"Mar 2, 2009"
Long	"March 2, 2009"
Full	"Friday, March 2, 2009"

timezone, if present, indicates which international time zone to get the current date for. Time zones are specified by a string of the form "GMT+08:00" or "Australia/Perth" or "EST". If omitted, the time zone of the computer on which it is running will be used.

Returned Value

A formatted date string.

Description

Return the current date formatted for the language and country of the locale object `loc`. *style* indicates the display format. The *style* variants may not all be unique for some locales.

See Also

[_locale \(Format Currency, Number, Date/Time\)](#)
[loc.formatDate](#)

`loc.currentTime`

Format the current time

Syntax

```
loc.currentTime([style [, timezone]])
```

Parameters

style is the style to format the time string. Possible styles are:

Style	Description
<i>Short</i>	Default style. The most terse form, as in "1:59 PM"
<i>Medium</i>	"1:59:26 PM"
<i>Long</i>	"1:59:26 PM GMT-05:00"
<i>Full</i>	"1:59:26 PM GMT-05:00"

timezone, if present, indicates which international time zone to get the current time for. Time zones are specified by a string of the form "GMT+08:00" or "Australia/Perth" or "EST". If omitted, the time zone of the computer on which it is running will be used.

Returned Value

A formatted time string.

Description

Return the current time formatted for the language and country of the locale object `loc`. *style* indicates the display format. The *style* variants may not all be unique for some locales.

See Also

[_locale \(Format Currency, Number, Date/Time\)](#)

[loc.formatDate](#)

[loc.currentDate](#)

`loc.formatDate`

Format a date

Syntax

```
loc.formatDate(num [, style])
```

Parameters

num is a numeric representation of the date. Typically this is from a previous call to `loc.parseDate`.

style is the style of this date string. Possible styles are:

<i>Short</i>	Default style. The most terse form, as in "3/2/09"
<i>Medium</i>	"Mar 2, 2009"
<i>Long</i>	"March 2, 2009"
<i>Full</i>	"Friday, March 2, 2009"

See different [local patterns here](#). For instance "Date.3" pattern means "Short" style.

Returned Value

A formatted date string.

Description

Convert a date value "num" into a formatted date string. ***style*** indicates the display format. The value of ***num*** can be generated by the `loc.parseDate` function. The ***style*** variants may not all be unique for some locales.

See Also

[_locale \(Format Currency, Number, Date/Time\)](#)
[loc.parseDate](#)

`loc.parseDate`

Converts date string to numeric date.

Syntax

```
loc.parseDate(datestring [, style])
```

Parameters

datestring is a formatted date such as "March 29, 2013".

style is the style of this date string. Possible styles are:

Short	Default style. The most terse form, as in "3/2/09"
Medium	"Mar 2, 2009"
Long	"March 2, 2009"
Full	"Friday, March 2, 2009"

See different [local patterns here](#). For instance, "Date.3" pattern means **Short** style.

Returned Value

A numeric value that represents the date. This value can be used in `loc.formatDate` to re-format to a different locale.

Description

The returned date value can be used as an argument for `formatDate()` to re-format the date using a different locale or formatting style. Note that to be parsed correctly the *datestring* must be in the format specified by the style parameter (default is **Short**).

See Also

[_locale \(Format Currency, Number, Date/Time\)](#)
[loc.formatDate](#)

`_logf`

Format a line of text and write it to the current logfile (and console if `_logPrintf` is used).

Syntax

```
_logf(format [, p1 [, p2 ...]])
```

```
_logPrintf(format [, p1 [, p2 ...]])
```

Parameters

format is a template of the text to be written to the file. It is similar to a C-format string where embedded `%s`, `%d`, and `%f` markers that get replaced by parameters passed to `_logf`. For a more complete description of the formatting process, see `_printf`.

p1, *p2*, ... are the optional parameters to be substituted into the *format* string.

Returned Value

None

Description

The `_logf` function formats a message and writes it to the current logfile. There is also a `_logPrintf` function which writes the same message to both the logfile and console. And there is a `_dlogf` function which is exactly the same as `_logf` except that it will write out to the log only if `-debug` has been set. See [Debugging Script](#)

Example

```
_logf("name=%s, city=%s", "Tony", "Paris"); // writes 'name=Tony, city=Paris'
```

See Also

[fp.printf](#)

[_message](#)

[_printf](#)

[_sprintf](#)

[_tracef](#)

[Debugging Script](#)

`_logfEx`

Format a line of text and write it to the current logfile with a severity level.

Syntax

```
_logfEx(severity, format [, p1 [, p2 ...]])
```

Parameters

severity can be one of **INFO**, **WARN**, **ERROR**, or **FATAL**. Actually, only the first character (case insensitive) matters. If it is not recognized, **INFO** is used.

format is a template of the text to be written to the file. It is similar to a C-format string where embedded `%s`, `%d`, and `%f` markers that get replaced by parameters passed to `_logf`. For a more complete description of the formatting process, see `_printf`.

p1, *p2*, ... are the optional parameters to be substituted into the *format* string.

Returned Value

None

Description

The `_logf` function formats a message and writes it to the current logfile. There is also a `_logPrintf` function which writes the same message to both the logfile and console.

And there is a `_dlogf` function which is exactly the same as `_logf` except that it will write out to the log only if `-debug` has been set. See [Debugging Script](#)

Example

```
_logfEx("WARN","name=%s, city=%s", "Tony", "Paris"); // writes 'name=Tony, city=Paris'
```

See Also

[fp.printf](#)
[_message](#)
[_printf](#)
[_sprintf](#)
[_tracef](#)
[Debugging Script](#)

`_merge` (Call DocOrigin Merge)

Runs Merge, passing the argument list as command line parameters.

Syntax

```
_merge(args)
```

Parameters

args is either a string listing all the Merge command-line parameters or a JavaScript object whose properties define the command-line parameters. If *args* is a string, the string will be passed as-is to the Merge program. If *args* is a JavaScript object, each object property name is converted to a DocOrigin-style `-name` parameter and the property's value becomes the parameter's value. The property names mirror the command line option names.

Returned Value

The return value from Merge. Merge will return zero (**0**) if it runs successfully. If the program cannot be found or cannot be run the return code is platform-specific. On Windows `_merge` returns a value of **-801** to indicate that the Merge program could not be found or executed. For Linux and other Unix platforms, it returns **-1**. As of 3.0.002.02, the return code when Merge is not found is **111**.

Description

Your application will wait until Merge has completed before resuming. As a convenience, the `-logfile` specification that was provided to the app that is running your script will automatically be passed along to the Merge app instance that you are launching. Of course, if you specify the `-logfile` option explicitly, your option will be the one in effect.

Example

```

var arglist = "-form=$F/myform.xatw -data=$D/mydata.xml";
var rc = _merge(arglist);
if (rc != 0) _printf("The Merge return code was: %d\n", rc);
var args = new Object();
args.form = $F + "/myform.xatw"; // or "$F/myform.xatw";
args.data = $D + "/mydata.xml"; // or "$D/mydata.xml";
var rc = _merge(args);
if (rc != 0) _logPrintf("The Merge return code was: %d\n", rc);

```

This presumes that you have set up definitions for where the `$F` and `$D` folders are in your installation's `Paths.prm` override file. See [\\$X String Substitutions](#). In the first variation, if the form name had a space in its name, e.g. `Sample_Invoice.xatw`, then you would have to quote it as in:

```
-form="$F/Sample_Invoice.xatw"
```

It has to be double quotes, not single quotes, and hence the overall quoting around the whole `arglist` would use single quotes. Don't forget to put a slash (`/`) at the start of the base file name. In the second variation, it is safe to use `$X` path substitutions on all platforms. If you pass `$X` in the first variation it will be incorrectly interpreted as an environment variable by Linux systems. Do check your return codes.

See Also

[_run](#) (Execute Another Program)
[_runScript](#)

`_mergeEmbedded`

(Merge only)

String substitution in data fields

Syntax

```
_mergeEmbedded(text)
```

Parameters

text a text string with [fieldname] or [!autofield] placeholders to be replaced.

Returned Value

A new string with substituted data.

Description

This routine is a script equivalent of the [fieldname] and [autofield] substitution available in normal text labels. See the [Auto/Embedded Fields](#) section for a list of available [autofield] variables.

`_mergeEmbedded` looks for substrings of the form [fieldname] and replaces them with the field's value from the current document. This function uses the same substitution logic as Merge embedded fields processing including [!pagenum], etc. If the text between the left and right square brackets is not a known name the text and square brackets are left unchanged in the text string.

Example

If the Field "LastName" has the value "Smith" and the Field "AccountNo" has the value "123456", then:

```
merged = _mergeEmbedded("Dear Mr [LastName]:\n re your account#[AccountNo]");  
// becomes "Dear Mr Smith:\n re your account#123456"
```

Note that a potentially common usage of this facility is in scripts run at the **Data Merged** event. At that event, data values have not had embedded field substitution done yet. Hence one may want to use:

```
merged = _mergeEmbedded(this._value);
```

to get the current field's value as it would be when embedded field substitutions (if any) has been done.

See Also

[Auto/Embedded Fields](#)
[_auto](#)

`_message`

Format a line of text and display as a Windows popup message.

Syntax

```
_message(format [, p1 [, p2 ...]])  
_dmessage(format [, p1 [, p2 ...]])
```

Parameters

format is a template of the text to be written to the file. It is similar to a C-format string where embedded %s, %d, and %f markers that get replaced by parameters passed to `_message`. For a more complete description of the formatting process, see `_printf`.

p1, *p2*, ... are the optional parameters to be substituted into the format string.

Returned Value

None

Description

The `_message` function displays a pop-up message on a Windows system. On non-Windows platforms it will display the message to the `stderr` console and wait for the user to press a key. The `_dmessage` function is identical but operates only when the `-debug` option is set. In some cases, it may block your execution without notice. For example, if you run as a Windows service, you will not see a message but the execution is blocked. Consider using `_logf` or `_printf` or setting `-RunFromService` input parameter to avoid interaction.

Example

```
_message("name=%s, city=%s", "Tony", "Paris"); // displays 'name=Tony, city=Paris'
```

See Also

[fp.fprintf](#)
[_logf](#)
[_printf](#)
[_sprintf](#)
[_tracef](#)

_metaData

Access data file metadata

Description

DocOrigin can automatically embed certain additional metadata into data streams that it passes from one program to another. This data is automatically made available to the called routine. The metadata is placed at the very end of the data stream and looks something like this.

The DocOrigin processing instruction actually occurs on a single line but is folded here for display purposes:

```
<!--!#$%* DocOrigin - MetaData *%$#!-->
  <?DocOrigin _origFileName="C:\DocOrigin\D0\Samples\Sample_Invoice.xml"
    _source="CopyDataFile 3.1.001.08" _history="CopyDataFile 3.1.001.08"
    _submitTimeUtc="2016-09-11 20:25:08 UTC" _submitTimeLocal="2016-09-11 16:25:08"
    _userName="Dev_PC\Andy"?>
<!--0000000309-->
```

You can see in the above, the meta-information that is provided. The 309 in the above example is the byte length of the metadata (it varies). This is used to help us select (and remove) the exact amount of introduced material from the data stream. Often job data files are small, one document, data streams but DocOrigin also processes very large, multi-document, data files. Being able to effect rapid access to information is critical.

i These metadata facilities are rarely used. They do involve processing overhead in order to hide or expose their existence as required by the process-of-the-moment. You should not use this feature unless you have a justifiable value proposition.

How to get metadata - QueueMonitor

The prime place where metadata is desired is when data is being funneled to DocOrigin via QueueMonitor, i.e. via a queue associated with a "virtual printer". It can be good to know when a file was submitted. Note that by the nature of spool queues, QueueMonitor is not provided with the original file name but rather receives simply a stream of bytes, so the metadata may not be as valuable as hoped. To get metadata delivered by QueueMonitor use the `-metadata Y` option in the applicable QueueMonitor port's PRM file (e.g. `DocOriginQueueMonitor9100.prm` for port 9100).

How to get metadata - CopyDataFile

(Windows only, also Linux as of 3.1.001.09) Another way to get metadata added to your data stream is to use the CopyDataFile program. By using the syntax

```
D0 CopyDataFile -metadata Y inputFile outputFile
```

the output file will have the metadata appended. A standard operation is to have that output file actually refer to a QueueMonitor-managed virtual printer. Of course, if direct access is available, it may refer to a FolderMonitor watched folder. You may wish to experiment with this to see how the `_history` attribute gets expanded as the data stream goes through multiple metadata-aware processes. The `-metadata Y` command line options, if provided, must be provided as the first two arguments. If not provided, no metadata will be appended to the **outputFile**. It will be a simple direct copy operation.

Getting metadata processed

By default, Merge will not expect to have to handle metadata. It can be instructed to do so with the command line (or more likely PRM file) option:

```
-metadata On
```

Note that the setting value is **On** (not **Y**). If that setting exists then Merge will note the existence of any metadata and will construct the `_metaData` JavaScript object. That object will have the expected properties: `_metaData._origFileName`, `_metaData._source`, `_metaData._history`, ... i.e. all the attributes that you see in the sample above. Do note that the JavaScript object name is case sensitive and is `_metaData`, not `_metadata`.

Metadata for non-XML

Naturally, we expect that most data streams will be XML-based. However, this metadata, even though it is in XML format, can be appended to any data stream. It will be temporarily removed and re-added as necessary as the data flows through filters and whatnot.

_odbc (Database Access)

This object enables simple [ODBC](#) (Open Database Connectivity) access.

Introduction

(Linux supported as of 3.1.001.14)

In general, users have their own applications that generate data files that are then merged with an applicable form design. Almost always, those data files will be generated via access to various corporate databases. The apps use speedy computer languages that can maintain database connectivity efficiently across millions of operations. That continues. In no way is this simple JavaScript interface for database access meant to supplant that. What it does allow a user to do is to augment the provided data with some additional elements from an applicable database. That is the extent of the intended usage. It is not intended to maintain databases or "discover" all of the data for a given document. That is still the purview of the customer application.

ODBC requires the use of "Connector/ODBC" drivers that are specific to the database technology in use at the customer site. These connectors are not provided by DocOrigin -- they are a customer's responsibility. The underlying DocOrigin C++ code is expecting ODBC version 3 drivers to be used. DocOrigin does not do any ODBC administration. Setting up DSN (Data Source Name) definitions that enable access is again the responsibility of the customer.

Background

The ODBC functionality involves the use of several JavaScript objects:

The <code>_odbc</code> object itself	The <code>_odbc</code> object serves as the anchor. It is available to you 'out of the box'. It allows you to connect to a given database 'data source' via a DSN name that you provide. The <code>_odbc</code> object also lets you terminate your ODBC usage so that all database connections are closed.
The ODBC connection object	The connection object, often named <code>oConn</code> , is used to execute SQL statements. Typically those would be <code>select</code> statements, though other SQL statements such as <code>insert</code> or <code>describe</code> can be executed as well. In theory, though unlikely, you could have multiple connections open while processing a single document.
The ODBC statement object	The statement object, often named <code>oStmnt</code> , is used to cycle through all of the rows that were selected by the executed query. In theory, though probably unusual, you could have multiple statements active simultaneously.
The ODBC row object	The <u>row</u> object, often named <code>oRow</code> , provides all of the columns and their values for a single row. This is often the most convenient means of processing the query results.
The ODBC column object	The <u>column</u> object, often named <code>oCol</code> , can be used to look at individual columns either by name or column number.

Usage / Contexts / Performance

Remember that the intended usage is not to write your business application in JavaScript in a document presentation tool. This is meant to add a few data items to what your main data-producing application provided. Establishing a connection to a database is quite slow - e.g. hundreds of milliseconds. That's a lifetime compared to normal document generation processing. It is surely best if the data provided by the program, specifically meant for that purpose, is complete. This JavaScript API for database access offers a port in a storm, especially performance-wise, to augment that data.

Often a "Merge run" processes data for only a single document. In such a case, it's simple, you, of necessity, make an ODBC connection, do your thing, and then terminate your ODBC access. There are no magic options. That one document will pay the entire overhead of establishing a database connection. If your Merge run processes data

for multiple documents then you run afoul of the fact that at the end of each document the JavaScript context is destroyed, all your JavaScript objects are gone and you start anew. However, for the special (and 99%) case, of connecting to a single data source, that data source can be kept open for the entire run -- across all switches of JavaScript context. In such a case you would make the `_odbc.connect(DSN)` call in a **Start-of-Job** event script. In the **Start-of-Document** (or **Data Merged**) event you would use the special reconnect call: `oConn = _odbc.connect()`. Notice that no DSN argument is supplied and hence can only mean to reconnect to a ('the') prior connection. This is done without any disconnect/reconnect overhead. Hence the overhead of establishing the database connection is amortized over all documents, over the entire Merge run.

The API

The API is deliberately simple. The normal flow is as follows:

```
var oConn = _odbc.connect("MyDSN");
var oStmt = oConn.exec("select * from MyTable where ...");
while (true) {
    var oRow = oStmt.fetchRow();
    if (oRow == null) break;
    // User code to examine oRow.columnName as desired
}
oStmt.close();
oConn.disconnect();
_odbc.term();
```

If you prefer to do individual column access you could use:

```
...
while (true) {
    var rc = oStmt.fetch(); // fetch, not fetchRow
    if (rc != 0) break; // at end-of-fetch it will be ODBC standard 100
    var oCol = oStmt.col(3); // or some column number ... or
    var oCol = oStmt.col(colName);
    // User code to examine oCol.value
}
...
```

_odbc Object Details

Prop/Func	Remarks
init()	Initialize the ODBC system, get "handles" etc. In practice, this is not called explicitly since any .connect call will invoke it as needed.
connect(DSN)	Connect to the database as identified by the DSN which you setup previously using an ODBC Administrator tool. Returns a connector object or null on failure (see rc).
term()	Close off all ODBC access. Close all statements. Disconnect all connections. Release all ODBC handles.
rc	The latest return code from the underlying ODBC call. Often 0 .
diagnostic	A message related to the underlying ODBC call. Often "".

Connector Object Details

Prop/Func	Remarks
exec(SQL)	Execute an SQL statement on this connection. Returns a statement object or null. Non-query statements return null. Check rc.
disconnect()	Disconnect from this connection. Close any statements. Free resources.
rc	The latest return code from the underlying ODBC call. Often 0 .
diagnostic	A message related to the underlying ODBC call. Often "".
DSN	Returns the DSN string, in case you forgot it.
rowCount	Returns the number of rows that were affected by your non-query statement. (-1 if a query statement)
colCount	Returns the number of columns in your query statement.

Statement Object Details

Prop/Func	Remarks
fetch()	Advances to the next row in the query result. Returns a return code. 0 - Ok, 100 - end-of-fetch, else a standard ODBC error code.
col(col#OrName)	Returns a column object or null if there is no such column. Column numbers go from 1 to colCount.
fetchRow()	Advances to the next row in the query result and constructs and returns a row object with the column names and values. Returns null on end-of-fetch.
close()	Free any ODBC resources related to this statement.
rc	The latest return code from the underlying ODBC call. Often 0 .
diagnostic	A message related to the underlying ODBC call. Often "".
colCount	Returns the number of columns in your query statement.
row	Returns the row number (starting at 1) of the just-fetched row.

Row Object Details

Prop/Func	Remarks
columnName	columnValue
columnName	columnValue
...	...

Column Object Details

Prop/Func	Remarks
colNumber	The column number: 1 to oStmnt.colCount
name	The column name. May be "" for unaliased expressions.
maxLen	The maximum length for this column.
value	The value of this column. Known only after a fetch is done. The other properties are known as soon as the connector object executes the statement.

ODBC on Linux

The needed ODBC library `libodbc.so` is not shipped with DocOrigin. It is your responsibility to create the needed infrastructure. That requires the 64bit ODBC library and the applicable ODBC connector(s) for whichever database system(s) you intend to use. And you must create/maintain the configuration files: i.e. `odbc.ini` and `odbcinst.ini`. DocOrigin does not provide guidance on how to setup your ODBC configuration.

There is more than one implementation of ODBC support for Linux. As a small start you could try:

```
sudo yum install unixODBC
```

but as said before, you will also need the applicable ODBC connector(s) and do the needed configuration.

The DocOrigin post-installation process (POSTINSTALL) looks for the ODBC library in `/usr/lib64/libodbc.so`. If found, it creates a symbolic link from `D0/Bin/libodbc.so` to that library. If your installation of ODBC puts the library in some other place be sure to create the `D0/Bin/libodbc.so` symbolic link to wherever it is located. DocOrigin does a dynamic load of that library via that symbolic link.

_os (Operating System Functions)

These functions access certain operating system features.

Functions

<code>_os.exit(<i>rc</i>)</code>	Exit program immediately with this return code value.
<code>_os.getClipboard()</code>	<i>(As of 3.1.002.07)</i> Fetch the text from clipboard.
<code>_os.getCurDir()</code>	Fetch the program's current working directory <i>name</i> .
<code>_os.getenv(<i>name</i>)</code>	Fetch the value of environment variable <i>name</i> .
<code>_os.getMergeVersion()</code>	Fetch version of Merge that is running, e.g. "3.0.005.07". <i>Prior to 3.1.002.04</i> , this was available in only Merge. <i>As of 3.1.002.04</i> , any script-supporting process can use this function to get its version string. Also, the synonym function name <code>_os.getVersion()</code> was introduced.
<code>_os.getpid()</code>	<i>(As of 3.0.003.19)</i> Get the operating system's process identifier for the running process. This can be useful in coming up with unique identifiers/file names since process ids don't get reused quickly — often not until a reboot occurs.
<code>_os.getHostname()</code>	<i>(As of 3.2.001.04)</i> Fetch the hostname.
<code>_os.getProgramPath()</code>	Fetch the full path and filename of the currently executing program.
<code>_os.getustr()</code>	<i>(As of 3.0.003.19)</i> Get a fairly short unique string, based on time to the millisecond. The string uses only those characters that are allowed in file names.
<code>_os.setClipboard(<i>text</i>)</code>	<i>(As of 3.1.002.07)</i> Set <i>text</i> to clipboard.
<code>_os.setCurDir(<i>newdir</i>)</code>	Change the program's current working directory.
<code>_os.sleep(<i>millisecs</i>)</code>	Pause execution for <i>millisecs</i> milliseconds.
<code>_os.type</code>	Returns either "Windows" or "Unix" as applicable.
<code>_os.userName</code>	Returns the login id of the user running the DocOrigin application. Note that this could be a network user, a services user, a web server user, ... not necessarily the human user who clicked a mouse or hit enter.

`_page` (The Page DOM)

`_page` is a JavaScript object which represents the DOM for a given Page object. This object is created dynamically and automatically for the page on which the object using `_page` exists. `_page` is undefined for the Form object as a Form is not on a page. If you choose to use the `_page` object it must be on a Page object or somewhere in a page's descendant tree of objects. The `_page` object does not refer to all pages in the document. Rather, it refers to only those instances of pages on which the object referring to `_page` resides. For example, if the object whose script references `_page` is, say the "Amount" Field, and that Field is on a Page named "LayoutTwo", then that `_page` reference will be to only the various instances of the Page named "LayoutTwo". As usual, `_page[0]` would refer to the first such instance and `_page[1]` would refer to the second such instance, etc.

To refer to all the pages in a document, you do not use the `_page` object. The following script excerpt shows one way to refer to all the pages in a document. By looking in the Form Explorer panel you can see that all Pages are the immediate children of the Form object. Hence, when wanting to refer to all the pages in a document you are wanting to refer to all the immediate children of the Form object.

```
for (var r = _document._firstChild; r; r = r._nextSibling) {
    _logf("Page '%s'\n", r._fullName);
    // ... do your desired page processing
}
```

When working at the Page level it is good to remember that every object (except the Form object) has a `_pageNumber` property. Of course, that is valid only on or after the **Pagination Completed** event. There is also an `_auto.PageNumber` property but its usage is discouraged. Clearly, it does not provide the page number of the `_auto` object -- that would be nonsensical. Rather, it provides the page number of the current global object, the `this` object, i.e. the object in which the `_auto.PageNumber` reference is made. If you want to know the page number of something then use the `_pageNumber` property reference on that something object.

Also, consider `_auto.PageCount` as a means to know how many pages exist in the document.

- ⓘ Notice the unusual property naming -- no leading underscore. That is because `_auto` is a "synthesized" object, not a user object on the form, and hence no collision can occur with the property name **PageCount**. As it happens, in another atypical way, **PageCount** is a case-insensitive property name of the `_auto` object. `_auto.PC` is a valid synonym.

Somewhat curiously, you can also use `this._pageCount` or `this._numberOfPages`. It seems that everywhere you look you can find the number of pages in the document.

`_parser`

Parse delimited or fixed-format records. These routines are used to convert data from either fixed-column format or delimited (eg tab-delimited, comma-delimited) to a useable JavaScript format.

Functions

The routines are passed descriptions of the format and names of the data items. The returned value is a JavaScript object which provides a convenient name/value list of all fields in the record.

- `_parser.delimited(record, names[, delimiter[, quote[, trim]])`
Convert a delimited (eg tab-delimited) record.
- `_parser.delimitedToXml(fromfile, tofile, names[, delimiter[, quote[, trim]])`
Convert a delimited file to an XML file.
- `_parser.fixed(record, format)`
Convert a fixed-column data record.
- `_parser.fixedToXml(fromfile, tofile, format)`
Convert a fixed-column data file to an XML file.
- `_parser.parms(record)`
(As of 3.1.001.03) Convert a string with name="value" syntax to a JavaScript object of properties and values.

`_parser.delimited`

Convert comma or tab-delimited records.

Syntax

```
_parser.delimited(record, names[], delimiter[], quote[], trim[])
```

Parameters

record is the text to be parsed. Trailing newline and carriage returns are ignored (not returned as data).

names is an array of field names. The record is assumed to have these fields in it in this order. This allows the resulting output to be treated like an array of parsed values. If any individual names array element is set to `null` rather than a field name, the corresponding data field is excluded from the output, i.e. the field is ignored. If a ***names*** parameter is set to `null` the fields will instead be named `0`, `1`, ... sequentially.

delimiter is optional and defaults to `' , '` (a comma). Set this to `'\t'` for tab-delimited data records or `' '` for blank-delimited files.

quote character is optional and defaults to the double-quote character. This is the type of quotation character that surrounds strings in the data stream. You might want to set to `"'"` to use single-quotes.

trim parameter is optional (defaults to ***false***). If present and set to ***true***, returned values will have leading as well as trailing blanks removed.

Returned Value

The returned value of this function is a JavaScript object whose properties are the names from the ***names*** parameter and values are the associated values from the data ***record***. If the record is incorrectly formatted and cannot be parsed, the return value is **`null`**.

Description

This routine is used to parse a single delimited record and assign names to all the values in that record.

Example

```

var names = ["first", "second", "third", "fourth"];
var record = "one","two","three";
var obj = _parser.delimited(record, names);

if (obj) {
    _message("obj.first=%s obj.second=%s", obj.first, obj.second);
} else {
    _message("Error parsing line");
}

```

Of course, normally one would have read in a record rather than having it hard-coded. This is equivalent to having set:

```

obj.first = "one";
obj.second = "two";
obj.third = "three";
obj.fourth = null;

```

See Also

[_parser.delimitedToXml](#)
[_parser.fixed](#)
[_parser.fixedToXml](#)

`_parser.delimitedToXml`

Delimited data to an XML file.

Syntax

```
_parser.delimitedToXml(fromfile, tofile, names[], delimiter[], quote[], trim]]])
```

Parameters

fromfile existing file containing delimited records.

tofile name of a new file that will get created with the XML equivalent of the ***fromfile***.

names is an array of field names. The record is assumed to have these fields in it in this order. This allows the resulting output to be treated like an array of parsed values. If any individual names array element is set to `null` rather than a field name, the corresponding data field is excluded from the output, i.e. the field is ignored. If a ***names*** parameter is set to `null` the fields will instead be named `0`, `1`, ... sequentially.

delimiter is optional and defaults to `,` (a comma). Set this to `'\t'` for tab-delimited data records or `' '` for blank-delimited files.

quote character is optional and defaults to the double-quote character. This is the type of quotation character that surrounds strings in the data stream. You might want to set to `"'"` to use single-quotes.

trim parameter is optional (defaults to ***false***). If present and set to ***true***, returned values will have leading as well as trailing blanks removed.

Returned Value

A return value of zero indicates a successful conversion. Other possible error return codes:

- **-904** - could not open the ***fromfile***.
- **-905** - could not open the ***tofile***.
- **-906** - a conversion error occurred converting one of the data records.

Description

This routine is used to convert an entire delimited file into XML. Each line of data in the source ***fromfile*** is considered a single record in the output ***tofile*** and will be surrounded by a `<document> ... </document>` XML structure.

See Also

[_parser.delimited](#)
[_parser.fixed](#)
[_parser.fixedToXml](#)
[_parser.parms](#)

...

The following code will read all records in a file and convert it to an equivalent XML file.

```

var sFormat = { Quantity:{from:6, width:6, trim:'both'},
                Code:{from:16, width:7, trim:'both'},
                Description:{from:27, width:28, trim:'both'},
                Price:{from:64, width:10, trim:'both'},
                Discount:{from:74, width:10, trim:'both'},
                Charged:{from:84, width:10, trim:'both'},
                Net:{from:94, width:12, trim:'both'} };

var fp = _file.fopen("C:/DocOrigin/test/Fixed.txt", "r"); // Open the data file
if (!fp) _message("File open error");

var xml = new XmlFile("C:/DocOrigin/test/Fixed.xml"); // Open the xml output file

var sRec;
var obj;

while (sRec = fp.fgets()) { // read each record
    if (sRec.length < 5) continue;

    obj = _parser.fixed(sRec, sFormat); // converts record
    if (obj) {
        xml.DetailLine = obj; // format and write as xml
    }
}

fp.fclose();
xml.close();

```

This will produce a new file C:/DocOrigin/test/Fixed.xml that looks like:

```

<?xml version="1.0" encoding="UTF-8"?>
<XMLData>
<Document>
  <DetailLine>
    <Quantity>1</Quantity>
    <Code>AK036</Code>
    <Description>Happy Cat Activity Centre Bl</Description>
    <Price>23.12</Price>
    <Discount>0.00</Discount>
    <Charged>23.12</Charged>
    <Net>23.12</Net>
  </DetailLine>
  <DetailLine>
    <Quantity>0</Quantity>
    <Code>SC180</Code>
    <Description>S/P Swing N Play Cat Toy*DEL</Description>
    <Price>1.10</Price>
    <Discount>7.00</Discount>
    <Charged>0.00</Charged>
    <Net>0.00</Net>
  </DetailLine>
  <DetailLine>
    <Quantity>3</Quantity>
    <Code>AH589</Code>
    <Description>Atlas 10 Pet Carry Entry Lvl</Description>
  </DetailLine>
</Document>
</XMLData>

```

```
<Price>22.00</Price>
<Discount>7.00</Discount>
<Charged>20.46</Charged>
<Net>61.38</Net>
</DetailLine>
<DetailLine>
  <Quantity>4</Quantity>
  <Code>AH590</Code>
  <Description>Atlas 20 Pet Carry Entry Lvl</Description>
  <Price>27.50</Price>
  <Discount>7.00</Discount>
  <Charged>25.58</Charged>
  <Net>102.30</Net>
</DetailLine>
<DetailLine>
  <Quantity>2</Quantity>
  <Code>VP611</Code>
  <Description>VP Super All Wormer Cat 4 Ta</Description>
  <Price>9.25</Price>
  <Discount>7.00</Discount>
  <Charged>8.60</Charged>
  <Net>17.20</Net>
</DetailLine>
...
</Document>
</XMLData>
```

See Also

[_parser.delimited](#)
[_parser.delimitedToXml](#)
[_parser.fixedToXml](#)
[_parser.parms](#)

`_parser.fixedToXml`

Fixed-column data file to XML.

Syntax

```
_parser.fixedToXml(fromfile, tofile, format)
```

Parameters

fromfile is the existing file containing fixed-column data records.

tofile is the name of a new file that will get created with the XML equivalent of the ***fromfile***.

format is a JavaScript object that describes the structure of the data record. This will typically be a JavaScript constant description such as:

```
{first: {from:10, width:30, trim:'right'},
  second: {from:50, width:5, trim:'both'},
  ...}
```

As can be seen from the above example each data field (`first`, `second`, etc) has an associated definition of how to extract the requisite data. ***from*** defines the data column where the data field begins (first column is number 0). ***width*** is the field's column width. ***trim*** is an optional setting that indicates whether leading or trailing blanks should be removed automatically:

left removes leading blanks.

right removes trailing blanks.

both removes leading and trailing blanks.

none does not remove any blanks. This is the default setting.

Returned Value

A return value of zero indicates a successful conversion. Other possible error return codes:

-904 - could not open the ***fromfile***.

-905 - could not open the ***tofile***.

-906 - a conversion error occurred converting one of the data records.

-907 - the ***format*** definition contains invalid or incorrect information. A field could not be extracted.

Description

This routine will convert an entire file of fixed-format records into an equivalent XML file.

Example

Consider a file **C:/DocOrigin/test/Fixed.txt** that contains fix-format records. Here's a sample of some of those records:

```
          1          2          3          4          5          6          7          8
9
01234567890123456789012345678901234567890123456789012345678901234567890123456789
0123456789012
          1          AK036          Happy Cat Activity Centre BlEach          23.12          0.00
23.12
          0          SC180          S/P Swing N Play Cat Toy*DELEach          1.10          7.00
0.00
```

20.46	3	AH589	Atlas 10 Pet Carry Entry LvlEach	22.00	7.00
25.58	4	AH590	Atlas 20 Pet Carry Entry LvlEach	27.50	7.00
8.60	2	VP611	VP Super All Wormer Cat 4 TaEach	9.25	7.00
	...				

The following code will convert that file to an equivalent XML file.

```
var from = "C:/DocOrigin/test/Fixed.txt";
var to = "C:/DocOrigin/test/Fixed.xml";

var sFormat = { Quantity:{from:6, width:6, trim:'both'},
  Code:{from:16, width:7, trim:'both'},
  Description:{from:27, width:28, trim:'both'},
  Price:{from:64, width:10, trim:'both'},
  Discount:{from:74, width:10, trim:'both'},
  Charged:{from:84, width:10, trim:'both'},
  Net:{from:94, width:12, trim:'both'} };

_parser.fixedToXml(from, to, sFormat);
```

This will produce a new file C:/DocOrigin/test/Fixed.xml that looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLData>
  <Document>
    <Quantity>1</Quantity>
    <Code>AK036</Code>
    <Description>Happy Cat Activity Centre Bl</Description>
    <Price>23.12</Price>
    <Discount>0.00</Discount>
    <Charged>23.12</Charged>
    <Net>23.12</Net>
  </Document>
  <Document>
    <Quantity>0</Quantity>
    <Code>SC180</Code>
    <Description>S/P Swing N Play Cat Toy*DEL</Description>
    <Price>1.10</Price>
    <Discount>7.00</Discount>
    <Charged>0.00</Charged>
    <Net>0.00</Net>
  </Document>
  <Document>
    <Quantity>3</Quantity>
    <Code>AH589</Code>
    <Description>Atlas 10 Pet Carry Entry Lvl</Description>
    <Price>22.00</Price>
    <Discount>7.00</Discount>
    <Charged>20.46</Charged>
    <Net>61.38</Net>
  </Document>
  <Document>
    <Quantity>4</Quantity>
    <Code>AH590</Code>
    <Description>Atlas 20 Pet Carry Entry Lvl</Description>
    <Price>27.50</Price>
```

```
<Discount>7.00</Discount>  
<Charged>25.58</Charged>  
<Net>102.30</Net>  
</Document>  
<Document>  
<Quantity>2</Quantity>  
<Code>VP611</Code>  
<Description>VP Super All Wormer Cat 4 Ta</Description>  
<Price>9.25</Price>  
<Discount>7.00</Discount>  
<Charged>8.60</Charged>  
<Net>17.20</Net>  
</Document>  
...  
</XMLData>
```

See Also

- [_parser.delimited](#)
- [_parser.delimitedToXml](#)
- [_parser.fixed](#)
- [_parser.parms](#)

`_parser.parms`

(As of 3.1.001.03)

Convert name=value records

Syntax

```
_parser.parms(record)
```

Parameters

record is the text to be parsed. Trailing newline and carriage returns are ignored (not returned as data).

Returned Value

The returned value of this function is a JavaScript object whose properties are the names parsed from the *record* and values are their associated values. If the record is incorrectly formatted no messages are issued and a best efforts basis is made to parse the record.

Description

This routine is used to parse a `name1=value1 name2=value2 ...` style record and build up a JavaScript object.

Example

```
var record = 'foo=bar fullName="Jane Doe" party:'O'Brian' IExist"
var obj = _parser.parms(record);
```

Of course, normally one would have read in a record rather than having it hard-coded. This is equivalent to having set:

```
obj.foo = "bar";
obj.fullName = "Jane Doe";
obj.party = "O'Brian";
obj.IExist = null;
```

Either single quotes or double quotes can be used. But the end quote must match the starting quote character in any given value. If the starting quote character needs to appear in the value then it must be doubled up. See 'O'Brian' in the example. Name/Value pairs can be separated by space, tab, comma, or semicolon. Assignment operators can be either equals (=) or colon (:). If a name is not followed by an assignment operator it will be given a value of `null`. If an assignment operator and a value are not preceded by a name, they will be discarded.


See Also

[_parser.delimited](#)
[_parser.delimitedToXml](#)
[_parser.fixed](#)
[_parser.fixedToXml](#)

_printer (Output Configuration)

(Merge only)

The `_printer` set of functions give access to a variety of Merge output configuration settings. DocOrigin Merge allows one or more printer output configurations to be used at the same time. Each has a unique "printername" such as PDF or LJ4.

 Note that output configurations have both a "printername" and a "printertype". Several of the `_driver` routines use **printername** as a parameter. The printertype is a generic name for the general type of output (PDF, PCL, HTML, etc.) that may apply to several distinct printer names. For example your `-config` option might list multiple PDF related PRT files. Each would have a printertype of PDF but each would have a distinct printername such as PDF and PDF2. Each output configuration could have separate settings such as having one driver combine all documents into a single output file while the other creates individual files.

The `_printer` functions are most likely used in the **Ready to Print** or **Start Next Print Driver** script events. When multiple output configurations are used, some of these functions allow you to control which will be "active" or "inactive" at any given time.

Functions

<code>_printer.first()</code>	Return name of first active output configuration. See also <code>_printer.next()</code> .
<code>_printer.getDocNum([printername])</code>	Return the number of documents printed to this output configuration.
<code>_printer.getDriver()</code>	Returns the printername of the current active output configuration.
<code>_printer.getPageCount([printername])</code>	Return the number of pages printed for the current document for the named (default current) output configuration.
<code>_printer.getTotalPageCount([printername], [blIncludeNumberedPages], [blIncludeUnnumberedPages])</code>	Return the total number of pages printed for the named (default current) output configuration. (As of 3.0.002.06) By default, blIncludeNumberedPages is true , and blIncludeUnnumberedPages is false . Typically, the unnumbered pages are for preamble material, separator pages, or terms and conditions pages. The numbered pages are typically the core of the document. By default, you get back the count of the numbered pages only.
<code>_printer.getOutputFile([printername])</code>	Return the name of the file that this output configuration is writing to.
<code>_printer.isDefined(printername)</code>	Is the named printer defined in this Merge run? true/false
<code>_printer.next()</code>	Return the name of next active output configuration. This assumes that <code>_printer.first()</code> has been called. Returns null if no further active output configurations have been specified.
<code>_printer.select(printername)</code>	Make this printername active, all others inactive.
<code>_printer.setActive(true false [, printername])</code>	(As of 3.1.002.01) Make an output configuration active or inactive. Use "*" for printername to apply to all output configurations.

<code>_printer.setOption(<i>name</i>, <i>value</i> [, <i>printername</i>])</code>	Set printer-specific options. Typically overriding config PRT file settings. eg: <pre>_printer.setOption("Select_Arial", "\E(s1p10v0s0b16602T", "LJ4"); _printer.setOption("spoolerDocName, "Invoice "+InvoiceNumber._value);</pre>
<code>_printer.setOutputFile(<i>filename</i> [, <i>printername</i>])</code>	Set the name of the file to write the output to.
<code>_printer.setPDFAllowAccess(<i>true false</i>)</code>	Set the PDF 'Allow Access' permission.
<code>_printer.setPDFAllowChange(<i>type</i>)</code>	Set the PDF 'Change' permission. Options are Insert, Fields, Commenting, or All.
<code>_printer.setPDFAllowCopy(<i>true false</i>)</code>	Set the PDF 'Allow Copy' permission.
<code>_printer.setPDFAllowPrinting(<i>type</i>)</code>	Set the PDF 'Printing' permissions. Types are Yes HiRes , No , or LoRes .
<code>_printer.setPDFInfo(<i>name</i>, <i>value</i>)</code>	Set name/value data for the 'PDF Info' section. This is for informational purposes only.
<code>_printer.setPDFModifyPassword(<i>password</i>)</code>	Specify the PDF file modification password.
<code>_printer.setPDFOpenPassword(<i>password</i>)</code>	Specify the PDF file open password.
<code>_printer.addPDFAttachment(<i>name</i> [, <i>description</i>])</code>	Add an attachment to PDF attachments list for current document.
<code>_printer.clearPDFAttachments()</code>	Clear list of PDF attachments for current document.
<code>_printer.resetPDFAttachments()</code>	Init PDF attachments list from command line <code>-attachment</code> options for current document.

`_printer.setOutputFile`

Syntax

```
_printer.setOutputFile(filename [, printername])
```

Parameters

The setting of the output file name can take effect only when output generation for the current document, if any, finishes, and output generation for the next document begins.

filename - The file names provided here can use embedded Field name references to dynamically name the output file based on the current data values. See also % substitutions in [File Naming Conventions](#). Further, output file names can be prefaced with: `prt::`, `file::`, or `run::` to more explicitly identify output destinations.

printername - In some systems, a printer name can include a slash (/). That makes the name look like a file name and hence it is a good idea to preface actual printer names with `prt::`. And conversely, it could help to specify that a file name is meant to be a real, on disk, file name, by prefacing the name with `file::`.

Rare

If `file::` is used with a WIN-based output configuration, it signifies that Merge is to instruct the Windows printer driver to send its raw output to a file on disk. This works for only some Windows printer drivers but has been known to be of use to capture such things as the raw ZPL destined for a Zebra label printer. When using this option with a WIN output configuration you need to identify that actual Windows printer that the output would normally go to if you weren't choosing to direct it to a file. You do that by using the `-outputRef` command line option. It identifies an actual printer, whose Windows driver will be asked to send its output to your nominated `file::` location instead. Your mileage may vary.

The use of the `run::` prefix is described in [Output Filenames / Options](#). It is exceptionally handy, especially during form+data development where quick onscreen results can be seen by passing, for example, PCL output to GhostPCL to render it to a PDF format, or any other variation you might think of re: doing post-processing on whatever output was generated.

On Linux systems, it is also possible to suffix the output destination name with `,p`. This instructs Merge to interpret the destination name as a program name (e.g. `lpr`) and to pipe the generated output to that program.

`_printf`

Format and write text to the console

Syntax

```
_printf(format [, p1 [, p2 ... ]])
```

Parameters

format is a template of the text to be written to the file. It is similar to a C-format string where embedded `%s`, `%d`, and `%f` markers that get replaced by parameters passed to `_printf`.

p1, *p2*, ... are the optional parameters to be substituted into the *format* string.

Returned Value

None

Description

The string *format* may contain substitution formatting codes:

%d - gets replaced by an integer number.

%D - gets replaced by an integer number that will have commas every 3 digits as in 1,234,567 (*as of 3.0.004.05*).

%s - gets replaced by a text string.

%S - same as `%s` except when the data value is a JavaScript structure or array. In these cases, the result is 'prettified' by adding indents and carriage returns (*as of 3.1.002.06*).

%f - gets replaced by a decimal or floating number.

%F - gets replaced by a decimal or floating number with commas every 3 digits (*as of 3.0.004.05*).

%x - gets replaced by an integer as a series of hexadecimal digits (lowercase)


%X - gets replaced by an integer as a series of hexadecimal digits (uppercase)

%E - gets replaced by the name of the script event currently being processed. Example: "OnEndMerge" (*as of 3.0.004.09*).

%% - gets replaced with a single `%` character.

Between the `%` and the formatting code, you may add a single number that specifies the character width of the resulting text. For decimal numbers (`%f` code), you may also specify a format such as `%10.4f` - where the first number indicates the overall output width and the second the number of digits to display after the decimal point.

Each `"%"` code must be matched by a parameter *p1*, *p2* etc. The parameter will get converted to the appropriate format and inserted into the output string. There is a reasonable string limitation which can be formatted, about 5000 characters. If a resulting string is too long it gets truncated and appended with `". . ."` in the end. Do not format very long strings, it is not a fast process.

 Note that when Merge is run from within Design using **PDF Preview...** or **Merge Test...** in the **Tools** menu, the console window output is suppressed. Instead, `_printf()` output is redirected to the Merge log file.

Example


```
_printf("Hello World"); // prints one line to the console
_printf("name=%s, city=%s", "Tony", "Paris"); // prints 'name=Tony, city=Paris'
_printf("value=%4.2f", 3.1415); // prints 'value=3.14'
_printf("value=%4.2F", 12345.67); // prints 'value=12,345.67'
_printf("num=%D", 1234567); // prints 'num=1,234,567'
```

See Also

- `fp.fprintf`
- `_logf`
- `_message`
- `_sprintf`
- `_tracef`

`_profile` (Access Profile Files)

DocOrigin allows the use of simple text Profile files (".ini" or "INI" files) to provide data that can be loaded via script. This mechanism can be used to customize your form design using an external file.

 Note that these are not DocOrigin or Merge configuration files, they are for you, the programmer, to use for building your form/document application.

A Profile file is one that is typically loaded via Merge command line option (`-profile`) prior to any document or script processing. (As of 3.1.002.10) There is also a `_file.readIniFile` function which is recommended if you wish to simply read a INI file into your JavaScript code.

True profile files offer "automatic actions". See [Profile Files](#). Please differentiate, in your mind, ordinary INI files from true, action-oriented, profile files. You can continue to use `_profile` with ordinary INI files but remember that each `_profile.load()` wipes out any earlier profile info. It may curtail the functionality of the data-driven, per document, actions specified in a previously loaded profile. It may cause `[!profile xxx]` embedded field definitions used in the form, but long forgotten by the scripter, to fail. Definitely consider using the `_file.readIniFile` function.

On the other hand, it may be that the true profile file that you want to load will be determined by values in the data stream. E.g. you may wish to load the profile for `Acme2` instead of the usual one for `Acme`. You cannot specify the needed profile file name on the Merge command line. So it still makes perfect sense to use `_profile.load()` to load a true profile file. Do so during the **Start-of-Job** event as it is in only that scenario that the `[*action]` automatic actions will be performed. (Of course, they are also performed if the `-profile` command-line option is used.)

Profile File Format

Profile files are simple text files. The file is required to contain either UTF8 or plain ASCII characters. The files consist of lines of `key=value` settings. One can optionally add a `[section]` name which indicates that all key assignments that follow are to be considered part of this section name. For example:

```
; An example profile file. (any text following a semicolon is a comment)
; Blank lines are ignored

Name=Joe Smith

[Address]
city = San Jose
state = "California"
```

Any `name=value` pairs before the first (if any) `[section]` line are considered part of the "default section", which internally has the section name `""`. You can continue the definition of the default section after other sections by using a section specification of `[]`. The key names can be any string of characters, although it is recommended that they be alphanumeric for clarity. Blank lines and comments can be interspersed throughout the file. The value can optionally be contained within quotes (double or single). The value strings are stripped of leading and trailing blanks or tabs. If leading/trailing blanks are required you must quote the string in order to preserve the blank. Within the value string you can use the following special sequences:

```
\\ - is replaced by a single \
\n - is replaced by a newline (hex 10) character
\r - is replaced by a linefeed (hex 13) character
\t - is replaced by a tab character
\E - is replaced by the escape character (hex 1B).
\xnnnn - the 4 digits nnnn are interpreted as a single hex unicode character.
```

(As of version 3.1.002.06) Multi-line values can also be defined using "heredoc" notation, as in:

```
Address = <<<END
123 Sesame St,
San Jose,
California
END
```

The `END` in the above example can be any text you like. All text up to the first line with that same text string at the start of a line is assigned to the key name.

Profiles can also contain directives to "include" another file:

```
; the following loads the contents of master.ini as well.
@master.ini

; this conditionally loads 'secondary.ini'
; it does not complain if the file does not exist.
@@secondary.ini
```

If an explicit file path is not given, the file is assumed to be in the same directory as the profile file it was referenced from.

Text Substitution [Define]

(As of 3.1.002.04)

Profile files support a mechanism to create shorthand names for longer paths or other text that is repeated, possibly many times, within the files. The `[Define]` section is used for this purpose

```
[Define]
mypath = //server3/share7/AppName/Archives/MyTests

[]
myfile = {mypath}/myOutput.pdf
```

Everywhere within the profile file processing where a name is found between `{ }` (braces), it will be replaced by the text defined within an earlier `[Define]` section. If the name within the braces is not found, the original text (and braces) will remain as-is.

Opening a Profile File

```
_profile.load(filename)
```

Opens a new profile file and replaces any currently stored profile settings. No return code is provided. You may want to check for the file's existence beforehand.

Accessing Profile File Settings

You can access a profile file setting with the following script function:

```
_profile.get(key[, section]);
```

The *section* name is optional and required only for values that are stored under a profile section name. Section names and *key* names are case-insensitive.

```
this._value = _profile.get("Name"); // set to "Joe Smith" using the above profile file.
this._value = _profile.get("city", "Address"); // set to "San Jose" using the above
profile
```

If the desired key does not exist then **null** will be returned.

Enumerating Section keys

You can enumerate the names of all keys within a section using the following two calls:

```
_profile.getFirstKey(section);
```

Returns the first key in the section or **null**.

```
_profile.getNextKey();
```

Returns the next key in the section or **null**.

Typical usage might be:

```
for (k=_profile.getFirstKey('abc'); k; k=_profile.getNextKey()) {
  ... do something with key k
}
```

To enumerate the keys before the first (if any) [section] entry, use "" as the section name. Note that only one _profile enumeration may be in progress at a time. Calling _profile.getFirstKey will cancel any current enumeration processing.

Enumerating all Sections

```
_profile.getFirstSection();
```

Returns the first section name in the profile or **null**.

```
_profile.getNextSection();
```

Returns the next section name or **null**.

Note that the default section name is "", so the section name returned may be a zero-length string. Your test for completion must distinguish between a null/empty string and null itself.

```
for (s=_profile.getFirstSection(); s != null; s=_profile.getNextSection())
  _message("next section name is '%s'", s);
```

Extended Profile usage within Merge

See the [Profile Files](#) section of Merge for additional Merge-only features and handling of profile files.

See Also

[_file.readIniFile](#)

[_file.readPrmFile](#)

`_prompt`

(Windows only) (As of 3.1.002.06)

Ask the user for input. Prompt user for OK/Cancel, Yes/No, Text, Date.

Syntax

```
_prompt(type, message [, default | choicelist])
```

Parameters

type is the type of popup Windows dialog to display.

- **OKCancel** - user can press OK or Cancel button.
- **YesNo** - user can press Yes or No button.
- **YesNoCancel** - user can press Yes, No, or Cancel button.
- **String** - user is prompted to enter a line of text.
- **Password** - same as String but text with echo stars instead a actual text entered.
- **Choice** - user can choose from a dropdown list of responses.
- **ChoiceID** - as above, but a tag associated with the selection is returned.
- **Date** - user can enter a calendar date.

The *type* parameter is case-insensitive, so **YesNo**, **yesno** are both accepted.

message is a prompting message to be displayed at the top of the popup dialog. You can also set the dialog title with this, by preceding the message text with the dialog title followed by a vertical bar, for example: "OK to Continue | Do you wish to continue this operation:" will set both the dialog title and the message text.

default is the default value of the input. For **YesNo**, **OKCancel**, and **YesNoCancel** the default value can be set to the same values as are returned by these calls. A value of **1** makes the "Yes" or "OK" button the default, a value of **0** makes the "No" button the default, and a value of **-1** makes the "Cancel" button the default. For type **String** the default parameter will be displayed as the initial value of the input edit box. For **Date** the *default* parameter is an initial date string in YYYYMMDD format. For the **Choice** and **ChoiceID** types, the default parameter is not used, but a default choice can be set in the *choicelist* (see below).

choicelist is a JavaScript structure or array listing the choices the user will select from. It can be a simple array of strings, such as:

```
['choice 1', 'choice 2', 'choice 3']
```

or a JavaScript structure such as

```
{ first:'choice 1', second:'choice 2', ...}
```

The **Choice** type option will always return the displayed string, so "choice 2" for instance. The **ChoiceID** type option will instead return the index or attribute, so in the second case above it would return "second" if the 'choice 2' option is selected. (If you use the simple array choices with the **ChoiceID** option, you'll get **0**, **1**, **2** etc. as the returned values.)

If you wish to specify a choice other than the first one as the default displayed choice, put an asterisk in front of the choice value.

```
{ first:'choice 1', second: '*choice 2', ...}
```

Returned Value

The returned value depends on the type of the prompt call.

YesNo, **OKCancel**, and **YesNoCancel** return a value of **1** is returned for the "Yes" or "OK" option, a value of **0** for the "No" option, and a value of **-1** for the "Cancel" option.

Date returns the selected date as a string formatted as YYYYMMDD.

String and **Choice** return the entered or selected string.

ChoiceID returns the JavaScript associated property name or array index of the selected item, rather than the item itself.

Examples

```
if (!_prompt('yesno', 'Continue?|Do you wish to continue?)) return;
name = _prompt('string', 'Enter your name:');
date = _prompt('date', 'Select a new date:');
date = _prompt('date', 'Select a new date:', '20181225'); // default to xmas 2018
```

See Also

[_file.getSaveFileName](#)

`_resolve`

String substitution

Syntax

`_resolve(text)`

Parameters

text is a text string with all manner of placeholders to be replaced. These include [Auto/Embedded Fields](#), [% variables](#), and [\\$X folder mappings](#).

Returned Value

A new string with substitutions done.

Description

This routine is a script equivalent of the auto/embedded Field substitution available in normal text labels. If used in the context of Merge, i.e. in the script in a Form, where there is a Document DOM available, `_resolve` looks for substrings of the form `[fieldname]` and replaces them with the Field's value from the current document. This function uses the same substitution logic, so if the text between the left and right square brackets is not a known name the text and square brackets are left unchanged in the text string. Unlike `_mergeEmbedded`, in any context (Merge, RunScript, or Job Processing scripts), `_resolve` will also do substitution of the `%` variables (typically date elements) that is done for output file names. See any PRT file. Further, any `$X` folder mapping will be substituted as well. The string is not presumed to be a file name, so no replacement of characters that are invalid for file names is done. Also, backslashes are not converted to forward slashes. By contrast see `_file.resolveName`.

Examples

In a Summary Section page, one might use:

```
report._value = _resolve("The report for branch [branch] is in file: $T/  
[branch]_%.pdf");
```

See Also

[Auto/Embedded Fields](#)

[_auto](#)

[_file.resolveName](#)

[_mergeEmbedded](#)

`_run` (Execute Another Program)

Run a user-specified program, passing user-specified command line parameters.

Syntax

```
_run(program [, args [, logStdout]])
```

Parameters

program is the fully qualified program name. (fully is preferred but sufficiently will do.)

args can be a text string that is the full command line or *args* can be a JavaScript object or array whose properties define the argument list. If *args* is a text string, the string will simply be passed as-is to the called program. If *args* is a JavaScript object, each object property name becomes a DocOrigin-style `-name` command line option name and the property's value becomes its value. E.g. `args.form = "$F/myForm.xatw";`. For third-party applications that don't use the `-name=value` syntax, use a numeric property name, e.g. `args[0] = "3rd party command line part";`.

logStdout (As of 3.1.002.07) A Boolean which specifies whether the `stdout` and `stderr` outputs of the external executable should be captured and added to the calling program's log file.

Returned Value

If the selected program cannot be found or cannot be run (e.g. missing a DLL), the return code is platform-specific. On Windows, `_run` returns a value of **111** to indicate that the task could not be found or executed. If the program is run, the returned value is the exit code provided by the called program.

Description

The `_run` function is used to spawn other programs, both DocOrigin tasks and other executables. Your application will wait until that task has completed before resuming. (See also `_runNoWait`.)

i Note: While you can invoke DocOrigin Merge with the `_run` function, it is strongly recommended that you use the `_merge` function. It has been specifically created to spawn DocOrigin Merge. Similarly, you are strongly advised to use the `_runScript` function rather than using `_run` to launch RunScript. By using those purpose-built functions, the called program gets the same jobID as the calling program. Also, the log file is shared as well as any session file that is being used. Hence information can be passed between the programs by using session settings.

Examples

First Variation

```
var program = $E + "/Merge";
var arglist = "-form=$F/myform.xatw -data=$D/mydata.xml";
var rc = _run(program, arglist);
// Note: better to use var rc = _merge(arglist);
if (rc != 0) _logPrintf("The _run return code was: %d\n", rc);
```

Second Variation

```
var program = $E + "/Merge";           // or "$E/Merge";
var args = {};
```

```
args.form = $F + "/myform.xatw"; // or "$F/myform.xatw";
args.data = $D + "/mydata.xml"; // or "$D/mydata.xml";
var rc = _run(program, args);
// Bote: better to use var rc = _merge(args);
if (rc != 0) _logPrintf("The _run return code was: %d\n", rc);
```

Remember that `$E` is pre-populated to refer to the appropriate program folder (`.../DO/Bin`) for your chosen installation folder. `$F`, and `$D` are presumed to be folder mapping variables that you defined in your `$O/Paths.pm` override file to refer to appropriate folders. See [\\$X String Substitutions](#). Also, the example uses `Merge`, without the `.exe` to make this portable to `*nix` systems. Similarly, the example uses forward slashes (`/`) not backslashes (`\`) for portability reasons. Besides, backslashes would have to be doubled up when used in JavaScript strings.

In the first variation, if the form name had a space in its name, e.g. `Sample Invoice.xatw`, then you would have to quote it as in `'-form="$F/Sample Invoice.xatw'`. It has to be double quotes, not single quotes, and hence the overall quoting around the whole `argList`, in JavaScript, would use single quotes. In the second variation, it is safe to use `$X` path substitutions on all platforms. In the first variation, on Linux systems, if you pass a `$X` it will be incorrectly interpreted as an environment variable reference.

Consider using `_sprintf`:

```
var program = $E + "/Merge"; // or "$E/Merge";
var args = {};
args.form = $F + "/myform.xatw"; // or "$F/myform.xatw";
args.data = $D + "/mydata.xml"; // or "$D/mydata.xml";
var rc = _run(program, args);
// Bote: better to use var rc = _merge(args);
if (rc != 0) _logPrintf("The _run return code was: %d\n", rc);
```

Don't forget to put a slash at the start of the base file name. Do check your return codes.

An "Args" Object or Array

- ✓ It is usually easier to use the `_sprintf` function rather than endless string concatenations.

It's often quite tedious to produce an "args" list string. For running DocOrigin programs, ones that use the `-option=value` notation, it is preferred to use the second variation above. The resulting code is much easier to understand and to maintain. But suppose a command-line option can occur multiple times; what then?

Clearly, it is pointless to use:

```
args.option = "ABC";
args.option = "XYZ";
```

Naturally, the last value alone will win out. A classic example of this need is for when you wish to give `Merge` two `-filter` options. The way to accomplish that is by using numeric property "names" in your `args` variable. When building up the command line from the supplied `args` variable, the code discards any numeric property name and uses just the property value. Here is an example:

```
var args = {};
args.form = ...;
args.data = ...;
args[0] = "-filter filter1.wjs -symbolset cp1252";
args[1] = "-filter filter2.wjs";
var rc = _merge(args);
if (rc != 0) _logPrintf("The _merge return code was: %d\n", rc);
```

Do note that in this format the value side of the assignment includes the option name. It is not necessary to use the exact numbers: 0, 1, 2, ... You can use any numbers you like but do not use the same number multiple times in one args definition, else the latter one will overwrite the former one.

Another method is to use a JavaScript array. Here is an example:

```
var args = [];
args.form = ...;
args.data = ...;
args[0] = "-filter filter1.wjs -symbolset cp1252";
args.push("-filter filter2.wjs");
var rc = _merge(args);
if (rc != 0) _logPrintf("The _merge return code was: %d\n", rc);
```

Notice the use of `[]` which signifies an array, rather than `{}` which signifies an object. JavaScript is very flexible. You can use `args["form"] = ...` or `args.form = ...`, or `args.push("-form ...")`; All of those syntaxes are supported and can be intermixed. The joy of using `push()` is that you don't have to dream up unique array index numbers.

Use a PRM File

Getting all the quoting of command line parts correct can be annoying. One way to make your life simpler is to use a, possibly temporary, PRM file. Of course, this is for only DocOrigin programs. Other programs do not know how to process PRM files. If you were to have or create a PRM file, named say, `myParams.prm`, you could use `_run` as follows:

```
var rc = _run("a DO program", "@myParams.prm");
```

The `@` is OK, but not necessary if it is the first parameter and it ends in PRM. This PRM method may be more comfortable for some users.

Redirecting Program Outputs

Many programs write output to `stdout` and possibly error messages to `stderr`. By using the `logStdout` parameter, with a value of `true`, in your `_run` call, you can cause that `stdout` and `stderr` output to be captured and automatically copied into the calling program's log file. This is very handy for keeping all messages together in one log file. If you do capture that output into your log be aware that it won't also come out to the console.

You may not want to capture that output in your log file but rather redirect that output to some file. The OS standard way to do that is by using the `>` operator. This is OS standard practice. If you create a command line (as opposed to an array of args) you can include the `>` operator (or other operators, e.g. `>>`, `|`) as you see fit. `_run` will feed that to the OS and the OS will do what it normally does.

(As of 3.2.001.05) If you want to use the `mpre` easily maintainable convenience of an `args` array you can do the following:

```
var args = [];
args.push("parameter 1");
args.push("parameter 2");
args.push("parameter ...");
args.push(">");
args.push("myFile.txt");
var rc = _run("myProgram", args);
```

The `>` operator must be provided as a separate array item. The operators that are supported using this array syntax are: `>`, `>>`, `1>`, `2>`, `2>&1`, `|`, and `<`. Indeed the OS supports more, less common, operators. If you have the need to use one of the less common operators, then you must use the command line method of the `_run` options, and not the `args` array/object option.

Debugging

It is hugely beneficial, during development, to log out the command line. Doing a

```
_logf("args: '%s'\n", args);
```

is a great way to do that. The DocOrigin JavaScript extensions support displaying JavaScript objects as `%s`. Also try `%S`.

See Also

- [_merge](#)
- [_runScript](#)
- [_runNoWait](#)
- [Session](#)

`_runNoWait`

(As of 3.2.001.02)

Execute another program and do not wait for the result. Runs the specified program, passing the argument list as command line parameters.

Syntax

```
_runNoWait(program [, args])
```

Parameters

program is the fully qualified program name. (Fully is preferred but sufficiently will do.)

args can be a text string that is the full command line or *args* can be a JavaScript object or array whose properties define the argument list. If *args* is a text string, the string will simply be passed as-is to the called program. If *args* is a JavaScript object, each object property name becomes a DocOrigin-style -name command line option name and the property's value becomes its value. E.g.

```
args.form = "$F/myForm.xatw";
```

For third-party applications that don't use the -name=value syntax, use a numeric property name, e.g.

```
args[0] = "3rd party command line part";
```

Returned Value

Returns zero on success.

Description

It differs from `_run` ([Execute Another Program](#)) in that we do not wait for any result to come back. Fire and forget.

See Also

[_run](#) ([Execute Another Program](#))

`_runScript`

(As of 3.0.003.19)

Call DocOrigin `RunScript`. Runs `RunScript`, passing the argument list as command line parameters.

Syntax

```
_runScript(args)
```

Parameters

args is either a string listing all the `RunScript` command line parameters or a JavaScript object whose properties define the command line parameters. If *args* is a string, the string will be passed as-is to the `RunScript` program. If *args* is a JavaScript object, each object property name is converted to a DocOrigin-style `-name` parameter and the property's value becomes the parameter's value. The property names mirror the command line option names.

Returned Value

The return value from `RunScript`. `RunScript` will return **0** if it runs successfully. If `RunScript` cannot be found or cannot be run the return code is **111**.

Description

Your application will wait until `RunScript` has been completed before resuming. There are a couple of bonuses in choosing to use `_runScript` instead of supplying the necessary parameters to `-run`. First is that you don't have to work out where the `RunScript` executable is. The second is that the `-logfile` specification that was provided to the app that is running your script will automatically be passed along to the `RunScript` app instance that you are launching. Of course, if you specify the `-logfile` option explicitly, your option will be the one in effect.

See [_merge \(Call DocOrigin Merge\)](#) for more details and examples.

Do check your return codes.

See Also

[_run \(Execute Another Program\)](#)

[_merge \(Call DocOrigin Merge\)](#)

_sendmail

Send an email.

Syntax

```
_sendmail(args)
```

Parameters

args is a JavaScript object with specific property values for each argument to `_sendmail`.

Returned Value

0 if successful. Otherwise an error code. Note that if messages are queued and not sent immediately, the return code only reflects the success of adding the message to the send queue, not its success or failure to get sent. That information will then be available in the logfile for the DocOriginSendMailServer task. If the message is sent immediately, the return code may reflect the actual status feedback from the email server. The cURL program has an impressive set of capabilities and options and a long list of return codes. You can view this at [cURL man page](#).

Description

ALERT: Don't let your message get trapped by a Spam Filter

The construction of your email message and the wording you use will affect how likely it is that a customer's spam filter may flag your email as spam. There are a number of websites with suggestions and tips on how to avoid this. Try a search for "how to avoid spam filters".

All parameters to `_sendmail` are defined as properties of a JavaScript object. For example:

```
var args = {};

args.To = "support@docorigin.com";
args.Subject = "Testing of DocOrigin sendmail command";
args.attach = "mytestfile.txt";
args.text = "This is my test message\nIt has two lines.";

rc = _sendmail(args);
if (rc != 0) _message("sendmail error - rc=%d", rc);
```

A "classic" is:

```
args.attach = _printer.getOutputFile("PDF");
```

Most email parameters can also be defaulted in the `DocOriginSendMailServer.prm` file. The parameters marked as (*Mandatory*) must be set either explicitly in the *args* definition or have a default value set in the `DocOriginSendMailServer.prm` file.

Args

The available parameters (*args*) are:

<code>args.<i>attach</i></code>	A list of one or more file names separated by semicolons. These files are attached to the email. <code>_printer.getOutputFile("PDF")</code> is commonly used to refer to the output just produced.
---------------------------------	--

args. bcc	Set a BCC (blind CC) address. This is a string containing one or more valid email addresses separated by semicolons.
args. cc	Set a CC address. This is a string containing one or more valid email addresses separated by semicolons.
args. checkaddress	Validate all email addresses before sending. This setting is typically set in the DocOriginSendMailServer.prm file.
args. cidfolder	A file folder on your computer where embedded html images are located. See HTML Image Embedding below for details. Defaults to the folder containing the HTML text.
args. from	<i>(Mandatory)</i> Set the sender's return address. This address must be present either in the args list or as a default setting in the DocOriginSendMailServer.prm file. Using Microsoft Exchange? See the note in Sending Mail .
args. header	<i>(As of 3.1.002.06)</i> Add an optional email header record to the email. This option can also be set in the DocOriginSendMailServer.prm file. Set this to a null string to override a -header setting in the DocOriginSendMailServer.prm file. If more than one header is required, separate the headers with "\r\n". This feature is useful if you require additional X-type headers in your emails.
args. hold	Prevents emails from actually being sent. If set to true , this parameter prevents the actual transmission of the emails. This option can also be set in the DocOriginSendMailServer.prm file to prevent any transmission of emails while testing the system. Messages that are queued will be placed in the queue but not sent. Note that if the -hold is then removed any queued messages will get sent.
args. keep	Determines whether to save the email that has been sent or to discard it. If set to true , sent email is stored in -mailSentFolder. This setting is typically set in the DocOriginSendMailServer.prm file.
args. html	Provide the message text as HTML-formatted text. This allows font typeface, size, bolding etc. as well as other HTML-style formatting. If the first character of the HTML string is an @ the string is assumed to be a filename. The contents of the file is read and used as the HTML message body. If both args. html and args. text are provided then the email message is sent with a "mimetype" of multipart/alternative which allows mail clients that don't support HTML mail to fallback to displaying the text email only.
args. subject	The subject line of the message.
args. testMail	<i>(As of version 3.1.002.07)</i> Causes all email addresses within the email (To, Cc, Bcc) to be replaced with this supplied email address. This is a quick way to test an email call and avoid sending emails to their intended recipients. This setting is also available on the program command line or the DocOriginSendMailServer.prm file.

args. text	Provide the message text as plain text. If the first character of the text string is an @ the string is assumed to be a filename. The contents of the file is read and used as the text message body. If both args. html and args. text are provided then the email message is sent with a "mimetype" of multipart/alternative which allows mail clients that don't support html mail to fallback to displaying the text email only.
args. to	<i>(Mandatory)</i> Set the To address. This is a string containing one or more valid email addresses separated by semicolons.
args. queue	Queue this message to be sent by DocOriginSendMailServer rather than immediately.

HTML Image Embedding

When you have specified an HTML message, Sendmail does some additional processing to attempt to nicely embed images into the HTML message body. Normally...

```

```

works as expected - fetching the image from your server and displaying it. However, if you encode this as

```

```

the file "yourimage.jpg" from your computer will be embedded into the email as a special attachment and referenced from there. The image is embedded into your HTML message without needing to reference any server. This can be useful for email viewers that might otherwise either inhibit image display or prompt the customer to turn on their display.

The string `src="cid:"` must be exactly like that, the same case, quotes, etc. for this to work. If the args.**html** parameter references a file such as in

```
args.html = "@C:/DocOrigin/User/myhtmlfile.txt";
```

the cid image file is assumed to be in the same folder as the HTML. Otherwise, you can explicitly specify the location of the image file using the **cidfolder** option of `_sendmail()` or by setting the command line `-cidfolder` option.

⚠ **ALERT: Interplay with -PDFCombineDocuments setting**

The default setting of `CombineDocuments` is `Yes` for the PDF driver. That means that Merge keeps the output PDF file open for the entire duration of the job. You won't be able to send that combined document until **End-Of-Job**. If you wish to send individual document emails at **End-Of-Document** then ensure that you set:

```
-PDFCombineDocuments No
```

See Also

[_run \(Execute Another Program\)](#)
[_merge \(Call DocOrigin Merge\)](#)

`_session` (Accessing Session Data)

(As of 3.1.002.10)

DocOrigin maintains an internal mechanism for passing information between various DocOrigin programs that are being spawned internally or explicitly using the DocOrigin scripting functions. This mechanism is called Session. See the [Session](#) page for a general overview.

Functions

<code>_session.set(<i>name</i>, <i>value</i>)</code>	Set Session variable <i>name</i> to a value.
<code>_session.get(<i>name</i> [, <i>raw</i>])</code>	Fetch the value of Session variable <i>name</i> . If <i>raw</i> (defaults to <i>true</i>) is <i>false</i> we try to convert some values to numbers and Booleans.
<code>_session.clear()</code>	Clears all Session data.

(As of version 3.2.001.01) Access a value in a Session variable directly as in:

```
_session.foo = "some value";
var foo = _session.foo;
```

Note, `_session` variables are always stored as strings (see `_session.get(xxx, false)` special case above).

Examples...

```
_logf("session = %S", _session.object());
```

Might return this in the logfile

```
session =
{
  "_LastApp":"Merge",
  "_LastUpdated":"2020/01/30 12:10:41",
  "$OUTPUT":"c:\\temp\\AtwTempFiles\\AutoFieldTest_6.pdf",
  "abc":"ABC",
  "count":3
}
```

Other examples...

```
var o = _session.object();
for (var name in o) _logf("Session[%s] = '%s'\n", name, o[name]);

var lastApp = o._LastApp;
```

See Also

[Session Overview](#)

`_sprintf`

(As of 3.0.004.05)

Format and return string.

Syntax

```
_sprintf(format [, p1 [, p2 ... ]])
```

Parameters

format is a template of the text to be returned. It is similar to a C `sprintf` format string where embedded `%s`, `%d`, and `%f` markers that get replaced by parameters passed to `_sprintf`.

p1, *p2* ... are the optional parameters to be substituted into the *format* string.

Returned Value

The formatted string.

Description

See the `_printf` command for formatting options. This function is identical to the `_printf()` function except that the resulting string is returned instead of printed.

Example

```
str = _sprintf("name=%s, city=%s", "Tony", "Paris");  
// sets str to 'name=Tony, city=Paris'
```

See Also

[fp.fprintf](#)
[_logf](#)
[_message](#)
[_printf](#)
[_tracef](#)

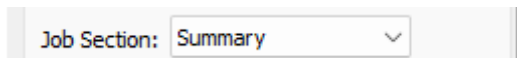
`_summary`

(Merge only)

Create data for Summary document.

Description

`_summary` is a pre-defined JavaScript object that can be used to create a data file that is to be merged with your Panes with the **Job Section "Summary"** as seen in **Object Properties**:



The `_summary` object is an instance of the `XmlFile` Class. See the [XmlFile Class \(Write XML Files\)](#) for the description of the functions that are available for use. The physical XML file that backs the `_summary` object is a temporary file that is created automatically when Merge first starts up. As with all temporary files, it is deleted when the Merge run is complete. During the processing of the form and data you can use the `_summary` object's methods to emit data into that temporary file.

It's likely that you will have been putting data into the file using `_summary` all through the processing, often in **End-of-Document** event scripts. You have one final chance to add data to the file, once all documents have been processed when the **Start-of-Summary** event script(s) are run. Once that event cycle is over, Merge will combine the summary data with the form, using only those Panes from the form where the **Job Section** has been designated as **"Summary"**.

The summary section output is a new document separate from all the Main Section documents. It will always begin on a new page.

See Also

[XmlFile Class \(Write XML Files\)](#)

_system

(Merge only)

_system is a JavaScript object which has a number of methods (functions) and properties related to the active Merge run. These include access to current data, form, script file names as well as active page and document counts.

Functions and Properties

<code>_system._dataFile</code>	The path and filename of the data file being processed.
<code>_system._event</code>	The name of the script event currently being executed.
<code>_system._formFile</code>	The path and filename of the form template being processed.
<code>_system._ndocuments</code>	The total number of documents processed so far. This value may be different from <code>_printer.prt.ndocuments</code> which counts only those documents printed to the specified printer.
<code>_system._nPages</code>	The total number of pages in the current document processed so far. This counts visible, numbered, pages only.
<code>_system._nTotalPages</code>	The total number of pages (all documents) processed so far. This counts visible, numbered, pages only. This object is different from <code>_printer.{prt}.ntotalpages</code> which counts only those pages printed to the specified printer.
<code>_system._scriptFile</code>	The path and filename of the script file being processed (or null).
<code>_system.docPageCount([blIncludeNumberedPages], [blIncludeUnnumberedPages])</code>	Pages in the current document. (As of 3.0.002.06) By default, blIncludeNumberedPages is true , and blIncludeUnnumberedPages is false . Typically, the unnumbered pages are for preamble material, separator pages, or terms and conditions pages, whereas the numbered pages are typically the core' of the document. By default, you get back the count of the numbered pages only.
<code>_system.rename(from, to [, charReplacement])</code>	<p>Rename a data XML tag name. This function may be used at the Start of Job script event to rename data tags to match the form template. from is the name as it appears in the data file. This string is case-insensitive. to is the name in use within the form. This string is case-sensitive and will be used exactly as passed to the method. All occurrences of from are changed, no matter at what level within the data file. To have an effect, this method must be called prior to any data being loaded, that is in the Start of Job event script. All subsequent references to the object must use the new to name.</p> <p>(As of 3.1.001.21) The optional third parameter, charReplacement can be used to indicate character replacement. In that case, each instance of any character in the "from" string will be replaced by the "to" string.</p> <p>Examples:</p> <pre>_system.rename("JF01", "InvoiceDate");</pre> <p>Will rename any item in the data file named "JF01" to "InvoiceDate".</p> <pre>_system.rename("-.", "_", true);</pre> <p>Will replace all dashes and dots with underscores in any data name.</p>

<code>_system.setDecimal(<i>char</i>)</code>	Set the character to be interpreted as the decimal point. This is not part of the <code>_locale</code> number formatting but is used when internally converting data strings to numbers. Locale is for output, this is for interpreting input. It needs to be set very early on, most appropriately in the form object's Start-of-Job event. e.g. <code>_system.setDecimal(",");</code>
<code>_system.totalPageCo unt([<i>bIncludeNumberedPag es</i>], [<i>bIncludeUnnumberedP ages</i>])</code>	Pages in the Merge run as a whole. See <code>_system.docPageCount()</code> above for parameter settings.

`_toBase64`

(As of 3.1.002.10)

Convert a string to Base64 encoding.

Syntax

```
_toBase64(string [, bytesperchar])
```

Parameters

string is the string to be converted. It must contain 8bit characters only (unicode 0-255);

bytesperchar is the coding size of each character in the *string* parameter. The default is **1**, indicating that each string character is to be treated as a single 8bit value when it is converted. This will be mean that no character in the string is expected to have a Unicode character code greater than 255. To convert the full unicode character set, set *bytesperchar* to **2**.

Returned Value

Returns the resulting base64-encoded string.

Example

```

var str = "DocOrigin 2020";
var b64 = _toBase64(str); // "RG9jT3JpZ2luIDIwMjA="
var txt = _fromBase64(b64); // "DocOrigin 2020"

var str2 = "保险单号";
var b64 = _toBase64(str2, 2); // "T92WaVNVU/c="
var txt = _fromBase64(b64, 2); // "保险单号"

```

Note that as an alternative to encoding 2-byte Unicode directly into base64, you could also convert the Unicode string to UTF-8 first. UTF-8 is a more compact representation where character codes below 128 (the ASCII set) are represented by simple single-byte characters, while others are encoded as 2 or 3 byte sequences.

The choice is dependent on your intended usage of the base64 encoded string.

```

var str = "DocOrigin fête 2020";
var utf8 = unescape(encodeURIComponent(str)); // "DocOrigin fÃªte 2020"
var b64 = _toBase64(str); // "RG9jT3JpZ2luIGbDqnRlIDIwMjA="
var u = _fromBase64(b64); // "DocOrigin fÃªte 2020"
var txt = decodeURIComponent(escape(u)); // back to "DocOrigin fête 2020"

```

See Also

[_fromBase64](#)

[_file.toBase64](#)

[_file.fromBase64](#)

`_toDOUnits`

This method converts common units of measure (inches, centimeters) to internal DocOrigin units of measure.

Syntax

```
_toDOUnits(string)
```

Parameters

string is a string value to be converted.

Returned Value

Returns an integer number of internal units (microns).

Description

This routine converts a number such as "1.25in" into internal DocOrigin coordinates (microns). See the [Script Units](#) section for a list of units of measure that may be specified. Invalid strings return a value of **0**.

See Also

[_fromDOUnits](#)
[Script Units](#)

_toXmlString

(As of 3.2.001.06)

Convert a string to XML encoding.

Syntax

`_toXmlString(string)`

Parameters

string is the string to be converted. Standard set of XML symbols that has to be escaped are handled - <>'&.

Returned Value

Returns the resulting XML-encoded string.

Example

```
_toXmlString("<>'\"&"); // " <>'&"
```

See Also

[_fromXmlString](#)

`_tracef`

Format and write a line of text and write to the trace file.

Syntax

```
_tracef(format [, p1 [, p2 ...]])
```

Parameters

format is a template of the text to be written to the file. It is similar to a C format string where embedded `%s`, `%d`, and `%f` markers get replaced by parameters passed to `_tracef`. For a more complete description of the formatting, process see [_printf](#).

p1, *p2*, ... are the optional parameters to be substituted into the *format* string.

Returned Value

None

Description

The trace file must be enabled by specifying the `-trace` command line option. If script tracing is not enabled `_tracef` statements are ignored.

Example

```
_tracef("name=%s, city=%s", "Tony", "Paris"); // prints 'name=Tony, city=Paris'
```

See Also

- [fp.fprintf](#)
- [_logf](#)
- [_message](#)
- [_printf](#)
- [_sprintf](#)

`_xml`

(Merge only)

Convert `-filter` data to XML.

Description

`_xml` is a pre-defined JavaScript object that can be used to convert non-XML data to XML format within Merge. This feature is available only when using a JavaScript script filter to process non-XML data within Merge (see the [JavaScript Filters](#) section for details). Writing data using the `_xml` object creates a temporary XML file that Merge will subsequently use as the "real" data file to merge with a document template. See the [XmlFile Class \(Write XML Files\)](#) description of the functions that are available.

Example

```
_xml.tag("Header");
_xml.Name = "Wayne Hall";
_xml.Phone = "613-555-1515";
_xml.Address = {Addr1:"2525 Circle Crescent",
                Addr2:"Ottawa",
                Addr3:"Canada"};
_xml.endTag();
```

Will create the following XML:

```
<Header>
  <Name>Wayne Hall</Name>
  <Phone>613-555-1515</Phone>
  <Address>
    <Addr1>2525 Circle Crescent</Addr1>
    <Addr2>Ottawa</Addr2>
    <Addr3>Canada</Addr3>
  </Address>
</Header>
```

See Also

[XmlFile Class \(Write XML Files\)](#)

XmlFile Class (Write XML Files)

Overview

XmlFile is a JavaScript class that can be used to open and write XML data to any file. (As of 3.1.001.01) See [XmlInput](#) for the DocOrigin script function for reading XML files.)

You can create an XmlFile object as follows:

```
var oXml = new XmlFile(sFilePath[, bSuppressDefaultXmlTags[, bOpenNow]]);
```

where

sFilePath	(Mandatory) A string representing the output file path.
bSuppressDefaultXmlTags	(As of 3.1.002.02) A Boolean which defines whether to suppress the <XMLData> and <Document> tags. Default is false .
bOpenNow	(As of 3.1.002.02) A Boolean which defines whether to <u>immediately</u> open and create the output file, versus delay until some tag or data output call is made. Default is false .

This will create the oXml object with the specified file as an XML output file. Subsequent function calls or property assignments can be used to specify the XML tags and data to store in the output file. When you are finished you should close the file using the oXml.**close**() function. When the file is closed, all outstanding closing XML tags are automatically added to complete the file. The XmlFile functions ensure that you create properly structured XML. It tracks open sub-structures and ensures the closing </end> tags are inserted as required.

Functions

In the following table, oXml is used as an example of an XmlFile object name, but you can use any name. It would be created with:

```
var oXml = new XmlFile ("somefile.xml", bSuppressDefaultXmlTags, bOpenNow);
```

After creating the object you may use any of the following functions:

oXml. close ()	Close the file. If bOpenNow is false and no output has been written, the output file is not created. All required ending tags are automatically added. Once the file is closed, it is available for opening by any other application.
oXml. getFileName ()	Return the name of the XML file. This returns the name of the file that this object opened. This can be useful when the XmlFile Class has been created by another process such as when a Merge filter script is given the _xml object. In the example above it would return somefile.xml.
oXml. newDocument ()	Begin a new document. This closes off any previous non-empty document with a </Document> close-tag; and then starts a new document with the <Document> open-tag. If bSuppressDefaultXmlTags is true or if there is a still-empty document already open, this function does nothing.
oXml. pi (string)	Insert an XML Processing Instruction. This will insert an XML processing instruction <? string ?> into the output file.

<code>oXml. comment(string)</code>	Add <i>string</i> as an XML comment. <code><!-- string --></code>
<code>oXml.tag(tagname[, attrs])</code>	Begin a new XML group. A new XML group <code><tagname></code> is added to the output file as the start tag of the enclosing element for all the subsequent data up to the next <code>oXml.endTag()</code> call. For every <code>oXml.tag()</code> call there must be a matching <code>oXml.endTag()</code> call. (As of 3.1.002.10) <i>attrs</i> is optional; it is either a "formatted attributes string", e.g. <code>'attr1="value1" attr2="value2"'</code> or an object <code>{attr1:"value1", attr2:"value2", ...}</code> .
<code>oXml.endTag ()</code>	End the current XML group.
<code>oXml.set(tag , value[, attrs])</code>	Add <code><tag>value</tag></code> to the output. (As of 3.1.002.10) <i>attrs</i> were introduced.


Writing Data

You can assign data values or structures to any named property of the `oXml` object. This will cause the data to be inserted inside appropriate XML tags. As an example:

```
oXml.mydata = "This is my data";
oXml.another = "more stuff";
```

Results in XML output of:

```
<mydata>This is my data</mydata>
<another>more stuff</another>
```

 As of 3.1.002.10, there is a special `__attrs__` property available for assigning XML attributes via JavaScript objects. See an example below.

Example

You may also assign entire JavaScript object structures to the `oXml` object as in:

```
oXml.mydata = { __attrs__: { comment: "This is the mydata tag" },
  Detail: { __attrs__: {comment: "This is the Detail tag"},
    Name: 'Bert',
    Addr: '123 Sesame Street'
  }
};
```

Results **in** XML output of:

Results in XML output of:

```

<mydata comment="This is the mydata tag">
  <Detail comment="This is the Detail tag">
    <Name>Bert</Name>
    <Addr>123 Sesame Street</Addr>
  </Detail>
</mydata>

```

These structures can be nested to any depth.

```

var myfile = new XmlFile("myfile.xml");
myfile.tag("Header");
myfile.comment = "This is my address info.";
myfile.Name = "Bert";
myfile.Phone = "555-555-5555";
myfile.Address = {Addr1:"123 Sesame Street",
                  Addr2:"New York",
                  Addr3:"USA"};
myfile.endTag();
myfile.close();

```

Will create the following XML:

```

<?xml version="1.0" encoding="UTF-8"?>
<XMLData>
  <Document>
    <Header>
      <comment>This is my address info.</comment>
      <Name>Bert</Name>
      <Phone>555-555-5555</Phone>
      <Address>
        <Addr1>123 Sesame Street</Addr1>
        <Addr2>New York</Addr2>
        <Addr3>USA</Addr3>
      </Address>
    </Header>
  </Document>
</XMLData>

```

The

```
oXml.tag = {tag1:"value1", tag2:"value2", ... tagn:"valuen"};
```

usage is very powerful. The within-curly-braces syntax is really defining a JavaScript object. Of course, if you happen to have a JavaScript object already available, constructed through whatever prior processing, you can output the elements of that object to XML simply by saying:

```
oXml.tag = objectName;
```

The elements of the object will be iterated over and an XML element will be emitted for each. The group of elements will be encased in the usual way with the specified `<tag>` bookends. This is quite powerful when constructing XML data files based on your own application's objects.

_xml

When processing DocOrigin Merge [filter](#) scripts a special instance of the `xmlFile` object is automatically created. It is called `_xml` and is pre-programmed to output to the temporary file Merge will use as the filtered output data stream. All of the above methods will work with the object named `_xml`.

_summary

Merge also provides a second `XmlFile` object named `_summary`. It can be used to create data for a Merge summary document.

See Also


[_summary](#)

[_xml](#)

XmlInput Class (Read XML Files)

(As of 3.1.001.01)

XmlInput is a JavaScript class that can be used to open and read any XML file. Is meant for simple scripting use, not for creating tree structures that have to be "recursed" to dig out what is in them.

 The variable names `oXI` and `o` are just examples.
Multiple `XmlInput` objects can be alive and independently active at the same time.

Functions

<code>oXI=XmlInput(<i>fileName</i>)</code>	Create an <code>XmlInput</code> object.
<code>o=oXI.next()</code>	Get the next element from the XML file.
<code>o=oXI.next(<i>tag</i>)</code>	Get the next element with the provided tag name.
<code>o=oXI.next([<i>tag1</i>, <i>tag2</i>, ...])</code>	Get the next element with one of the provided tag names.
<code>o=oXI.get()</code>	Get the next <u>leaf data</u> tag.
<code>o=oXI.get(<i>tag</i>)</code>	Get the next leaf tag with the provided tag name.
<code>o=oXI.get([<i>tag1</i>, <i>tag2</i>, ...])</code>	Get the next end tag with one of the provided tag names.
<code>o=oXI.close()</code>	Close the file, free resources.

Create an XmlInput Object

You create an `XmlInput` object via:

```
var oXI = XmlInput(an xml file name);
```

If the XML file does not exist then a JS exception will be thrown. (You should check for that with a try-catch block). Otherwise, it will be a fine JavaScript object.


The bookend to this is: `oXI.close()`;

One Element at a Time

The simplest thing to do is to read the XML file sequentially and see what you get back. You can read the next element of the file with:

```
var o = oXI.next();
```

`oXI.next()` returns another JavaScript object with lots of useful properties. Which properties it has is dependent on what the next element of the XML file was. That could be a "Start Tag", an "End Tag", a piece of data, a comment, or a Processing Instruction (PI). If it comes back with `null` that means that you have hit end-of-file.

 You might say, "Gee, what is this doing for me? Just simple sequential access, what good is that?". If you have ever tried to parse out an XML file in script you will (or should) appreciate how much nit-picky code is required to properly parse an XML file in all its many permitted syntax variations. The underlying C++ code uses the Xerces library to do that gnarly parsing. Because of that you don't even have to think about it. Just accept the elements that come through to you without any fuss required. `<>` & etc. even CDATA -- all handled for free.

The following is a really simple script that I urge you to use to understand what you get by using `oXI.next()`.

```

try {
  var oXI = XmlInput("C:/temp/simple.xml");
  while (true) {
    var o = oXI.next();
    if (o == null) break;
    _printf("\n");
    for (var x in o)
      _printf("o has '%s' as '%s'\n", x, o[x]);
  }
  oXI.close();
} catch (e) {
  _printf("caught error '%s'\n", e);
  return 201;
}

```

The o object that is returned has some subset (as applicable) of the following properties:

Property	Meaning
type	The kind of element that was returned (see below for details).
line	The line number of the XML file that has this element.
column	The column in that line where the element ends.
som	The dotted schema object model notation for the returned element.
startTag	For start (aka open) tags, the tag name <tag>.
endTag	For end (aka close) tags, the tag name </tag>.
leafData	For a leaf node (a node with no children) the data it contains.
attr	For start tags that have attributes, this is an object that contains all the attributes and their values.
comment	The full comment text between <!-- and -->.
piName	The 'target' of a Processing Instruction. E.g. in <?DocOrigin ... ?> it would be DocOrigin.
piValue	The full text after the target name in a Processing Instruction.

type, line, column, and som are always present. The others are present only when applicable.

The following types can occur:

Type	Meaning
1	This is a start tag; may include an 'attr' object property.
2	This is leaf data.
4	This is an end tag.
8	This is a Processing Instruction -- includes both piName, and piValue.
16	This is a comment.
6	That is, 4 + 2, an end tag and the relevant leaf data.
7	That is, 4 + 2 + 1, a start tag, end tag, and leaf data. For <tag/> scenarios.

Note that 2 will never occur on its own.

You may or may not use that type property. Instead, you might use:

```

if (typeof o.leafData != "undefined") {
  // Yay, I have real data for tag o.endTag
  ...
}

```

It's pretty easy, and simple, to race through the XML file, picking off items of interest. If you choose to act on that data immediately or choose to store away the data in arrays, or objects of your own design, for later reference, well, that is up to you. You are in control.

Just the Leaf Nodes

It could be that you do not need all the structural tags, but rather just tags with data in them, i.e. leaf tags. You can skip over all the other stuff and get only the data elements by using (in a loop):

```
o = oXI.get();
```

Remember that not only do you get the start tag, end tag and data, you also get the full SOM expression. You do know what structure surrounds this data if it is of interest to you. You don't need to worry about `typeof .. != "undefined"` because `startTag`, `leafData`, and `endTag` will always be defined after an `oXI.get()`.

Just that Tag

It's quite probable that you know the structure of the XML file that you are reading. Perhaps you simply want to get the `runDate` control element. You can do that by using:

```
o = oXI.get("runDate");
```

(or whatever leaf tag name interests you). We only ever use leaf tag names, not dotted SOM expressions. The full dotted SOM expression is provided back to you, but must not be provided to the `oXI.get()` function. If you misspell the tag name, chances are that it will race along all the way to end-of-file and you will get `null` back.

⚠ CAUTION: The reading of the XML file proceeds in only one direction: forward. If you wish to access several specific tags, do so in the order in which they appear in the file.

Get 'One of These' Tags

Maybe you are not absolutely certain of the tag order or simply don't want to bother to do that research. You can specify an array of tag names to `oXI.get()` and it will return with whichever one it finds first. It does not return an array of objects but rather returns the first one that it finds that matches your list.

Imagine an XML file that contains a snippet like this:

```
<Detail>
  <Item>23</Item>
  <Desc>A fine item, you just have to have</Desc>
  <Price>19.99</Price>
  <Qty>4</Qty>
</Detail>
```

You might choose to use:

```
o = oXI.get(["Item", "Qty", "Price", "Desc", "Detail"]);
```

It will return with whichever one it finds first (next?). Strangely, I added the `"Detail"` tag name to my list. It's not even a leaf tag! Why did I do that? Well, I didn't want to run off into the next `Detail` structure so I put a backstop on my get request. In my script, I would check to see if the end tag `"Detail"` was returned. If it was, I would know that I had exhausted that `Detail` structure. Because of that desire, this form of the `oXI.get()` function returns both leaf nodes and end tags (assuming they are in your interest list).

```
// Process a Detail structure
var tagsOfInterest = ["Item", "Qty", "Price", "Desc", "Detail"];
```

```

var detail = {}; // My detail object
while (true) {
  o = oXI.get(tagsOfInterest);
  if (o == null) break; // Ouch! missing </Detail>!!
  if (o.endTag == "Detail") break; // Finished this Detail structure
  detail[o.endTag] = o.leafData;
}
// Great my detail object is all filled out with .Item, .Qty, .Price, and .Desc

```

Zip to the Details Section

Reminder, you can just read the whole file sequentially. Perhaps you want to quickly skip over a bunch of header matter and get to the `Details` section. In that case, you are after a start tag of `Details`. You can do that with:

```
o = oXI.next("Details");
```

`oXI.next()` has the same variations as `oXI.get()`. That is, you might not supply a parameter and it would return the next tag that it finds. Or you might supply an array of tags that interest you and it will return the next one of those that it finds. In practice, you would probably use the variations of `oXI.next()` augmented by only the no-arguments version of `oXI.get()`. The latter to race through leaf data nodes, and the former to zip along to any named tag. `oXI.get(parms)` provides no advantages over `oXI.next(parms)`.

Attributes

Some people insist on putting attributes in their XML files. If they do, you will get (no choice in the matter) an `attr` property on the start tag element which has those attributes. Let's consider an example where your XML has the following:

```
<foo bar="high" length="long"> ...
```

When your use of `oXI` returns that start tag element it will have an `o.attr` property. In this example you could reference:

```
o.attr.bar and o.attr.length.
```

Or in the unlikely case of your not knowing what attributes to expect, you could use the usual:

```

for (var prop in o.attr) {
  // do something with prop and o.attr[prop]
}

```

Where Am I?

It's possible that you discover a data value that seems in error. You may wish to log a message about that. It's helpful to report the file name and the location of where you detected the issue. To that end you always have available:

```
o.line and o.column.
```

You can also use `oXI.getFileName()` just in case you forgot which file you opened.

Repositioning

While blocking your mind of any thought of performance you can always call:

```
o = oXI.reposition(line, column)
```

If you leave `column` out it is deemed to be 0. If you leave both `line` and `column` out they are both deemed 0, and you effectively are asking to "rewind". If necessary (and it likely will be) the XML parsing operation will start back at the beginning of the file and race along, behind the scenes, until it gets a line and column that is equal to, or the first one greater than, the values that you supplied. It returns the element at that position. Subsequent `oXI` calls will carry on from there. It may be that you remembered some previously returned line and column values and want to return to the scene of the crime.

Document at a Time Access

For programmers and academics, there is a natural desire to skip all this sequential stuff and load a DOM. Then allow you to (saddle you with) traversing a tree recursively to dig out information of interest. Well, hey, if you wish to create such constructs, go for it. Now that all the gnarly syntactical parsing (which is not very rewarding to the soul) is taken care of, bless you Xerxes, you can easily create the ivory tower constructs of your dreams. I have a hunch that you will be more efficient in your forms development using the, essentially sequential, access methods supplied above. And I feel certain that you will have a far greater handle on what you code. (BTW: Do employ functions.)

We could still add document-at-a-time access someday should feedback rooted in practicality be provided that leads that way.

XMLHttpRequest (Send/Receive HTML Pages)

(Windows only)

This class allows the use of the standard XMLHttpRequest functions that allow the posting and receiving of HTML pages.

Usage

To use the XMLHttpRequest class you must first create an instance of the class:

```
var http = new XMLHttpRequest;
```

You can now use the HTTP object to invoke the various functions below.

Functions

http. readyState	<i>(As of 3.2.001.03)</i> Represents the state of the request. Essential for "async" calls. Get data when it is "4" (COMPLETED). Values: 0 (UNINITIALIZED): The object has been created, but not initialized (the open method has not been called). 1 (LOADING): The object has been created, but the send method has not been called. 2 (LOADED): The send method has been called, but the status and headers are not yet available. 3 (INTERACTIVE): Some data has been received. Calling the responseBody and responseText properties at this state to obtain partial results will return an error because status and response headers are not fully available. 4 (COMPLETED): All the data has been received, and the complete data is available in the responseBody and responseText properties.
http. status	<i>(As of 3.2.001.03)</i> Represents the HTTP status code returned by a request.
http. statusText	<i>(As of 3.2.001.03)</i> Represents the HTTP response line status.
http. getAllResponseHeaders()	Get all response header name/value pairs. This routine returns all name/value pairs in a single string. Each pair is delimited by a carriage return/linefeed (CR/LF) sequence.
http. getResponseHeader(name)	Get a single named response header value.
http. open()	Open a link to an HTTP web site.
http. send()	Send the request and receive a response.
http. setRequestHeader()	Add a custom HTTP header.
http. responseText	Get the textual response made by the invoked URL. See the example under http.responseText .

`http.open` - define a link to an http site

Define a link to an http site.

Syntax

```
http.open(method, url[, async[, user[, password]])
```

Parameters

http - the variable name used in a prior call to `new XMLHttpRequest`;

method - "GET" or "POST" (not case sensitive). The mode of communication to the web page.

url - the URL to the web page to be communicated with.

async - true for asynchronous behaviour, false otherwise. For a GET operation, you want it to be false.

user - if website requires authentication then user and password should be provided.

password

Returned Value

Zero if successful.

Description

This routine opens a link to a web page.

See Also

[XMLHttpRequest](#)

http.send - send to server, get response

Send to server, get response

Syntax

```
http.send(data)
```

Parameters

http - is the variable name used in a prior call to new XMLHttpRequest;

data - any text data to be sent to the web URL.

Returned Value

0 if successful.

Description

Send to server, get response.

`http.setRequestHeader` - define a custom request header

Syntax

```
http.setRequestHeader(header, value)
```

Parameters

`http` - is the variable name used in a prior call to `new XMLHttpRequest`;

`header` - string containing name of the header to set.

`value` - string value to set the header.

Returned Value

Zero if successful.

Description

Define a custom request header.

Example

```
var http = new XMLHttpRequest;
http.open("POST", "http://www.docorigin.com/echo.php");
http.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
var sData = "first=one&second=two";
http.send(sData);

var sResult = http.responseText; // what was sent back
_message("result='%s'", sResult);
```

See Also

[XMLHttpRequest](#)

http.responseText - get response

Get response

Syntax

```
http.responseText
```

Parameters

http - is the variable name used in a prior call to new XMLHttpRequest;

Returned Value

The text that was returned from the http request. E.g. an HTML or XML file or just echoed output.

Description

After having created an XMLHttpRequest object, opened an URL, and issued that request (via send), you can get the response by examining the responseText property of the XMLHttpRequest object.

Example

Note that the async parameter of open is false since I want synchronous behavior. I want the get to have been done before I check responseText.

```
var http = new XMLHttpRequest;
http.open("GET", "http://www.docorigin.com/MyIp.php", false);
http.send(null);

var sResult = http.responseText; // what was sent back
_message("result='%s'", sResult);
```

Results in: whatever the invoked, in this case PHP, responded with. In the MyIp.php example case that is something like: Your IP is: nn.nnn.n.nn

See Also

[XMLHttpRequest](#)

DOM Functions

(Merge only)

When scripting inside a form one has available the `_document` DOM, and some other DOMs. For the most part, you will want to access properties (see [DOM Properties Accessing the Document Structures](#) of the various objects in the DOM. In fact, navigation between objects in the DOM, is done via properties (e.g. `_firstChild`, `_nextNode`).

Those navigation properties yield a DOM object. Actually one of the most common DOM objects is the `this` object, which is the object to which your script is attached. Let's refer to those DOM objects generically, as `domObj`. For documentation table of contents and indexing purposes, we will say `domObj.function`, but that really means any DOM object, most notably the `this` object of the moment.

Note that by convention, all properties begin with "_" (underscore). Functions do not. All functions are followed by (possible arguments). Hence a leading underscore is not required to differentiate the function reference from some lower-level DOM object in the DOM tree.

domObj.ancestor

(Merge only)

Determine if an object has an ancestor (not necessarily just an immediate parent) of a given name.

Syntax

```
domObj.ancestor(name)
```

Returned Value

The *domObj* pointer of the ancestor with the supplied name, or `null` if no ancestor with that name exists.

Parameters

name is a string, which is the name of a, considered by you to be a possible, ancestor of this object.

Description

Often a form designer inserts more hierarchy into a form design for various reasons. Script is over 'brittle' if it uses constructs such as `this._parent._parent...` so as to get the object of some nearby ancestor. By using this ancestor function, coding is both easier and more maintainable.

See Also

[domObj.childNamed](#)

domObj.appendInstance

(Merge only)

Dynamically append another instance of an existing, Allow Multiple child Pane. *Prior to 3.1.001.02*, this applied to panes only. *As of 3.1.001.02*, it applies to rows as well.

Syntax

```
domObj.appendInstance()
```

(As of 3.0.003.15) **domObj**.appendInstance([*paneName*])

Returned Value


The **domObj** pointer of the pane that was created. `null` if something went awry.

Parameters

As of 3.0.003.15, a Pane name can be supplied. If supplied then that Pane will be cloned from the Form Template DOM, i.e. not requiring an existing Pane instance in the Document DOM.

Description

For all but some minuscule percentage of the time, the merging of data with a form design will automatically produce the exact number of instances you need. However, we have an existence proof that a need to script the creation of additional Panes does exist. Note that one instance of the Pane must exist*. It can be mandatory and invisible, but it must exist. That Pane is cloned and appended to the current set of the parent's child Panes.

 *As of 3.0.003.15, if a Pane name is provided as a parameter then there is no requirement to have an existing Pane instance in the Document DOM.

Example

Terms and Conditions (T&C) clauses. Having them in your data stream is quite a bit yucky. Having them in your form design is equally yucky. Yet they do vary. And significantly, really significantly, they are not items that are the responsibility of the form designer, nor should the constructor of the data want to embed such, other-department things in the data.

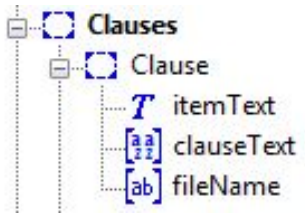
So... imagine a folder that is maintained independently by that 'other-department'. It contains RTF files with the precisely desired wording of each clause. As life marches on, the number of clauses varies. Do let's keep our form designs and data streams immune from that.

One can write a script to enumerate all the RTF files in the designated T&C folder. Now you just need sufficient Panes to put them in.

```
clausePath = $F + "/Clauses";
clauses = clausePath + "/*.rtf";
itemNumber = 0;
_file.findClose(); // Start a fresh file scan
for (fn = _file.findNext(clauses); fn; fn = _file.findNext(clauses)) {
  itemNumber++;
  r = this.Clause.appendInstance(); // Get another pane instance to hold this
  clause
  r.fileName._value = fn; // Hidden field with the filename in it, fwiw
  r.itemText._value = itemNumber + "."; // Number the clauses as we go
  r.clauseText._value = "[@" + fn + "]; // The value just points to the rtf file
  r._visible = true;
```

```
}  
_file.findClose(); // Be kind to mankind, clean up after yourself
```

Form Explorer View



domObj.childNamed

(Merge only)

Determine if an object has an immediate child (not further down descendant) of a given name.

Syntax

```
domObj.childNamed(name)
```

Returned Value

The *domObj* pointer of the child with the supplied name, or `null` if no child with that name exists.

Parameters

name is a string, which is the name of a, considered by you to be a possible, child of this object.

Description

Often, very often, Panes are both not-mandatory and are allowed to occur multiple times. The data will determine how many of those Panes will exist in the `_document` DOM. The Pane may not exist at all. This function allows you to safely ask, the parent Pane (or other containing object type) whether it has any children of name '*name*'.

Critical Usage

You must learn to use this function! Often folks try to use the `_count` property of a possible child Pane to see if it is zero. That will fail. If there are no instances of that child, then trying to refer to a non-existent object's `_count` property is doomed. Not detectible at compile time. Occurs at runtime, with deadly effect. Protect yourself, and your customers -- use `.childNamed("xxxx");`

Example

Consider the classic situation of a "Details" parent Pane, and an Allow Multiple child Pane, "Detail". On the parent "Details" Pane one might say:

```
if (this.childNamed("Detail")) {
  for (var i=0; i < this.Detail._count; i++) {
    ... process each child
  }
}
```

See Also

[domObj.ancestor](#)

domObj.bookmark

(Merge only) (as of 3.1.002.02)

Creates a PDF bookmark for a given object.

Syntax

domObj.bookmark(*key*, *value*)

(As of 3.0.003.15) **domObj**.appendInstance([*paneName*])


Parameters

key specifies the main bookmark category, for example, "CustomerContact".

value specifies a sub-bookmark below the category, for example, "ABC Company".

Both parameters must be strings.

Description

 Note that the PDF driver bookmark feature must be enabled either with your PDF .prt file by setting the <Bookmark>Yes</Bookmark> option, or by using the -PDFBookmark **Yes** command-line option.

Such functions are available now:

```
_document.bookmark(key, value);
```

```
_page.bookmark(key, value);
```

```
domObj.bookmark(key, value);
```

It is also possible to use different kinds of DOM references like `this.bookmark()`, `this._currentPage.bookmark()`, `this.parent.bookmark()` or `_document.main_pane.detail_table.bookmark()`, etc.

Bookmarks to page point to top of the page while bookmarks to an object point to page with vertical offset trying to point directly to the object.

 Note that calling `_page.bookmark()` or `domObject._currentPage.bookmark()` should happen after pagination otherwise it will point at the first instance of logical page.

See Also

[_document.bookmark](#)
[_page \(The Page DOM\)](#)

domObj.clone

(Merge only) (As of 3.1.001.02)

Make an exact copy of a DOM object and add it into the DOM.

Syntax

domObj.clone(*placeAfter*)

Returned Value

The new DOM object.

Parameters

placeAfter is optional (this feature became available in 3.1.001.10). It defines the object that the cloned object should come after. Without this parameter, the cloned object will be in the DOM right after the original object.

Usage

Quite rare. A simple usage is to clone fields and lines and whatnot and then move the cloned objects around or change their attributes as befits the occasion. Another usage is to clone whole pages or series of pages. You can affect copies of part of your document that way.

domObj.deleteObject

(Merge only) (As of 3.0.003.21)

Delete an object and all of its descendants from the DOM tree.

Syntax


```
domObj.deleteObject()
```

Returned Value

True if successful, false otherwise. It will be false if the script would be deleting the object with which the script is associated, or by deleting any ancestor of that object.

Description

The most common thing to do when an object is determined to be not relevant to a document, presumably based on the data of the moment, is to make that object invisible. This goes further by actually deleting the object. Why delete an object? An invisible object still exists and still has an extent which will be taken into account when Merge works out the minimum height to which a containing Pane can be shrunk. If the object is not just invisible but deleted, then that containing Pane may be able to be shrunk to a lesser height. Note that a fresh DOM is created with each new document instance. The object deletion is for one document instance only.

 A script cannot delete the object to which it is attached. E.g. `this.deleteObject();` will always fail. Nor can it delete an object that is an ancestor of itself.

domObj.descendant

(Merge only)

Determine if an object has a descendant object with a given name.

Syntax

```
domObj.descendant(name)
```

Returned Value

The *domObj* pointer of the descendant with the supplied name, or `null` if no descendant with that name exists.

Parameters

name is a string, which is the name of a, considered by you to be a possible, descendant of this object.

Description

The descendant function allows one to write a script that is much easier and more maintainable than a script that uses the `childNamed` function to iterate through the levels of hierarchy to find an object by name.

See Also

[domObj.childNamed](#)

domObj.DOM

(Merge only) (As of 3.1.001.17/21)

Find a DOM object by dotted document object name. Document DOM only, not Data DOM.

Syntax

(As of 3.1.001.21) **domObj**.DOM(**dottedName**)

domObj._DOM(**dottedName**)

Returned Value

The document object that was found or `null` if it was not found.

Parameters

dottedName defines the name of the object which is to be found. This is a literal string or a variable that contains a string. E.g. "Details.Detail.Item".

Usage

This can be a very effective, and safe, way to detect whether an object exists. Scripters often make the mistake of referring to objects in optional panes, causing their script to fail if the referenced pane does not exist.

```
this._value = this.optionalPane.Item._value ; // will fail if optionalPane does not exist
```

Some scripters try to protect themselves by using: `if (this.optionalPane._count > 0)...` but that fails, ending the script if `optionalPane` does not exist. More wary scripters might use the `childNamed(...)` or `descendant(...)` functions and that is good, however, this `DOM(...)` function is hoped to be easier for scripters to use.

Note that DocOrigin's flexibility in dotted names comes to the fore. You do not need to specify every segment of the object's name. DocOrigin will make intelligent inferences based on the object context (the **domObj** in **domObj**.DOM). It checks the siblings of **domObj**, then the descendants of **domObj**, then the descendants of the siblings of **domObj** and continually fans out to find a nearby object that satisfies what was provided in the dotted name argument. Of course, you can supply a full dotted name if you wish. The important thing is that your script will not fail and you can safely check if the result is `null`.

See Also

[domObj.DOMValue](#)

domObj.DOMValue

(Merge only) (As of 3.1.001.21)

Return the `_value` property of a dotted object name, or "" if the named object does not exist. Document DOM only, not Data DOM.

Syntax

(As of 3.1.001.21) `domObj`.DOMValue(*dottedName*)

`domObj`._DOMValue(*dottedName*)

Returned Value

The `_value` property of the document object that was found or "" if the object was not found.

Parameters

dottedName defines the name of the object which is to be found and whose `_value` property is to be returned. This is a literal string or a variable that contains a string. E.g. "Details.Detail.Item".

Usage

Scripters often want to know the value that was provided for a data item. However, frequently these items are inside optional Panes. Direct reference to such objects runs the risk of having the script fail if the referenced optional pane does not exist. By using this function and supplying the object name in dotted notation you can safely get the value of a referenced object without fear of running afoul of optional panes that do not exist. If the object (or any of its ancestors) do not exist the returned value will be "". 99.9% of the time that is the cue that scripters use to decide on some default action. The important thing is that your script will not fail and you can safely use the returned result.

See Also

[domObj.DOM](#)

domObj.fillFromNextPage

⚠️ Deprecated Function

The best way to achieve multi-column output, since 3.0.003.20, is to use the facility built right into Design. No scripting is required. In that scenario, you have a full-width parent pane that has one or more relatively narrow child panes. In the pane object properties of the parent pane you check

Layout in multiple columns.

Having done that, Merge will automatically wrap all of that parent's child Panes, that were instantiated at Data Merge time, into as many columns as can fit in the parent Pane. It will do pagination. It will ensure that the last set of columns are balanced -- i.e. area as much as possible of equal length.

(Merge only)

This is used for two-column output. The *domObj* must be a Group object. This function will fill the Group with the objects that flowed into the next page.

Syntax

```
domObj.fillFromNextPage()
```

Returned Value

Nothing directly, but the *domObj* is updated.

Parameters

The real parameter is the *domObj* itself, which must be a Group object.

Description

This is used to produce two-column, page-long, output. If you persist, good background to this obsolete feature, even if that background is outdated, is in the PaneTalk article [Multi-Column Layout](#).

You design a form as usual except that you make all the panes quite narrow, i.e. only half-or-less the width of the page (as you would imagine if one is going to make it two columns wide). On the right-hand side of the container, you draw a group box that represents where the second column is going to go. So, one has a bunch of flowing, half-width, panes down the left and a tall group box down the right-hand side. The group box must end at the bottom of the container.

The only code is in the Pagination Completed event of the group. It is merely:

```
this.fillFromNextPage();
```

That's it. The narrow panes will flow as usual, in fact they will overflow the page and onto the next page — temporarily that is. Then at Pagination Completed time, `fillFromNextPage()` will pull all the panes on the next page into the area defined by the group box. Subsequently, when the Pagination Completed event processing is done on that next page, it will be detected that the page no longer has any panes, and hence the page will be marked as invisible and not-counted (essentially deleted). So now, two pages have been folded into one page with two columns.

This is useful only when doing two columns on an entire page, not for two columns within some section of a page. It relies on the pagination algorithm being able to do its thing with respect to overflowing panes, inserting headers, etcetera. Whatever panes are instantiated in the next page will be pulled into the right-hand group of the current page. This also means that it is critical that the group's height and the container's height match.

domObj.fillObject

(Merge only)

Fill out an object such that it uses up its full width.

Syntax

```
domObj.fillObject(text)
```

Returned Value

Nothing directly, but the *domObj* is updated.

Parameters

text is a string, most typically the "." string to be appended to the *domObj*'s current value until the *domObj* fills up its entire width.

Description

Typically used for a Table of Contents where one wants dots (.....) to extend from the end of the text through to its corresponding page number. It might be used to fill out a cheque amount with "*".

Example

Imagine a Table of Contents such as:

```

...
2.1 Best Books Ever.....23
2.1.1 Moby Dick.....24
2.1.2 The Island.....26
...

```

It's no easy matter to know how many dots to add when dealing with proportional fonts.

```

// When this refers to an object that holds the text for a line in a Table Of Contents
this.fillObject(".");

```


domObj.getAllTags

(Merge only) (As of 3.1.002.04)


Merge maintains an internal tag table for various settings that control its behavior and for tags defined by the user. This routine can be used to fetch all of those tag settings at once.

Syntax

```
domObj.getAllTags()
```

Background

All objects have a fixed set of properties and those are accessed via standard DOM property access functions. However, some objects can take on either extra properties or have properties that are so rarely set that no specific UI is provided in Design to set them. Instead, Design has a **Format > Object Tags...** menu item that allows a form designer to nominate a tag and assign it a value. This open-ended capability allows many object tags to be set without encumbering the Design UI with "clutter" for tags that are rarely set. In fact, being open-ended, it even allows for third-party tags to be defined for add-on application usage.

 There is also a `getTag` function to allow your script to get a specific previously set tag. That setting may have occurred by script or via the Design UI. See [domObj.setTag](#);

The `getAllTags` function requires no parameters and returns a single JavaScript object which has a property for every tag defined for the **domObj**. If the **domObj** has no tags set, a common occurrence, then the object will have no properties.

Returned Value

A JavaScript object with a property for each tag defined. Usage example:

```
_logf("%E -- For %s", this._fullName);
var tags = this.getAllTags();
for (var tag in tags)
  _logf("..Tag %s = '%s'\n", tag, tags[tag]);
// Of course, you might actually do something, not just print them out!
```

See Also

[domObj.setTag](#)

domObj.getObjectsByName

(As of 3.1.002.06)

Find all objects in the current DOM which match a specified name (or names). Find objects that match a Name.

Syntax

```
domObj.getObjectsByName(name [, name ...] [callback])
```

Parameters

name is either a simple Object Name, or an array of object names to search for.

callback is an optional JavaScript function that will be called for each successful match.

Object names must match exactly in case. If the last character in a *name* parameter is * it is treated as a wildcard. So a search for **Address*** will match objects called 'Address', 'Address1', 'Address2', 'AddressCity' etc.

Note that a simple parameter of '*' will return a list of all objects below the current object.

```
var p=getObjectsByName('*'); //returns all subobjects
```

Returned Value

If a *callback* function is specified, this function returns TRUE. Otherwise, it returns a JavaScript array of pointers to DocOrigin DOM objects. That array is empty if no matches are found.

Background

This routine searches only the current object (typically the object containing this script) as well as all objects within (under) this object. When using the *callback* function, you can cause the search of the DOM to terminate prematurely by having the callback function return TRUE. Otherwise, the *callback* is called for every match in the DOM. This can be useful (for example) if you're only interested in the first match, or already know that there is in fact only one object that will match. This might improve the program's speed.

Example

```
var p=getObjectsByName('UnitPrice', 'Amount');

for(i=0; i<p.length; i++) {
  _logf("%d. name='%s' value='%s'", i, p[i]._name, p[i]._value);
}
```

The above script will generate a listing of all objects called UnitPrice or Amount. If this script is put on a top-level object (Form, Page, Container for instance) then all instances are found. If it occurs on a lower-level Pane it will only find the instances within that Pane.

```
function myfun(p) {
  _logf("%d. name='%s' value='%s'", i, p._name, p._value);
  if (p._value == 0) return true;    // quit when a value of zero is encountered
}

getObjectsByName('UnitPrice', 'Amount', myfun);
```

This second example does the same thing as the first, but via a callback routine. It also terminates the search if an object with a value of zero is encountered.

See Also

[domObj.getObjectsByType](#)

[domObj.getObjectsByTag](#)

[domObj.getObjectsByTagValue](#)

domObj.getObjectsByTag

(As of 3.1.002.06)

Find all objects in the current DOM which match a specified tag name - - find objects matching specific Tag names.

Syntax

```
domObj.getObjectsByTag(tag, [, tag...] [callback])
```

Parameters

tag is either a simple tag name or an array of tag names to search for *domObj*.

callback is an optional JavaScript function that will be called for each successful match.

If the last character in a *tag* parameter is * it is treated as a wildcard. So a search for *BCC** will match objects with a tag name that starts with 'BCC'.

Returned Value

If a *callback* function is specified, this function returns TRUE. Otherwise, it returns a JavaScript array of pointers to DocOrigin DOM objects. That array is empty if no matches are found.

Background

This routine searches only the current object (typically the object containing this script) as well as all objects within (under) this object. When using the *callback* function, you can cause the search of the DOM to terminate prematurely by having the *callback* function return TRUE. Otherwise, the callback is called for every match in the DOM. This can be useful (for example) if you're only interested in the first match, or already know that there is in fact only one object that will match. This might improve the program's speed.

Example

```
var p=getObjectsByTag('BCC*');

for(i=0; i<p.length; i++) {
    _logf("%d. name='%s' value='%s'", i, p[i]._name, p[i]._value);
}
```

The above script will generate a listing of all objects with any tag name starting with BCC. If this script is put on a top-level object (Form, Page, Container for instance) then all instances are found. If it occurs on a lower-level Pane it will only find the instances within that Pane.

```
function myfun(p) {
    _logf("%d. name='%s' value='%s'", i, p._name, p._value);
    if (p._value == 0) return true;    // quit when a value of zero is encountered
}
getObjectsByTag('BCC*', myfun);
```

This second example does the same thing as the first, but via a callback routine. It also terminates the search if an object with a value of zero is encountered.

See Also

[domObj.getObjectsByType](#)
[domObj.getObjectsByName](#)

domObj.getObjectsByTagValue

(Merge only) (As of 3.2.001.03)

Find all objects in the current DOM which match a specified tag name with a specified value.

Syntax

`domObj.getObjectsByTag(tag, value)`

Parameters

tag is a tag name to search for (case insensitive).

value is a tag value to compare with (case insensitive).

Returned Value

It returns a JavaScript array of pointers to DocOrigin DOM objects. That array is empty if no matches are found.

Background

This routine searches only the current object (typically the object containing this script) as well as all objects within (under) this object.

Example

```
var p = getObjectsByTagValue('TagName', 'TagValue');

for(i=0; i<p.length; i++) {
    _logf("%d. name='%s' value='%s'", i, p[i]._name, p[i]._value);
}
```

The above script will generate a listing of all objects with tag name *TagName* and value *TagValue*. If this script is put on a top-level object (Form, Page, Container,) then all instances are found. If it occurs on a lower-level Pane it will only find the instances within that Pane.

See Also

[domObj.getObjectsByTag](#)
[domObj.getObjectsByType](#)
[domObj.getObjectsByName](#)

domObj.getObjectsByType

(Merge only) (As of 3.1.002.06)

Find all objects in the current DOM which match a specified object type (or types). Find all objects that match an object type.

Syntax

```
domObj.getObjectsByType(type, [, type ...] [callback])
```

Parameters

type is either a simple object type, or an array of object types to search for. The following is a list of DocOrigin object types:

Object type names:
"Form", "Page", "Container", "Pane", "Group", "Line", "Rectangle", "Table", "Row", "Cell"
"Field-Text-String", "Field-Text-Number", "Field-Text-Currency", "Field-Text-DateTime", "Field-Text-Date", "Field-Text-Time"
"Field-Checkbox", "Field-Radiobutton", "Field-Image", "Field-Barcode", "Field-Comb"
"Label-Text", "Label-Image", "Label-Barcode"
"Arc", "PolyLine" (dynamically generated by _chart)

callback is an optional JavaScript function that will be called for each successful match.

Returned Value

If a *callback* function is specified, this function returns TRUE. Otherwise, it returns a JavaScript array of pointers to DocOrigin DOM objects. That array is empty if no matches are found.

Background

This routine searches only the current object (typically the object containing this script) as well as all objects within this object. When using the *callback* function, you can cause the search of the DOM to terminate prematurely by having the callback function return TRUE. Otherwise, the *callback* is called for every match in the DOM. This can be useful (for example) if you're only interested in the first match, or already know that there is in fact only one object that will match. This might improve the program's speed.

Example

```
var p=getObjectsByType('Field-Barcode');

for(i=0; i<p.length; i++) {
  _logf("%d. name='%s' value='%s'", i, p[i]._name, p[i]._value);
}
```

The above script will generate a listing of all Fields that are displayed as barcodes. If this script is put on a top-level object (Form, Page, Container,) then all instances are found. If it occurs on a lower-level Pane it will only find the instances within that Pane.

```
function myfun(p) {
  _logf("%d. name='%s' value='%s'", i, p._name, p._value);
  if (p._value == 0) return true;    // quit when a value of zero is encountered
}
```

```
getObjectsByName('Field-Barcode', myfun);
```

This second example does the same thing as the first, but via a callback routine. It also terminates the search if an object with a value of zero is encountered.

See Also

[domObj.getObjectsByTag](#)
[domObj.getObjectsByName](#)

domObj.getPickedObjects

(Merge only) (As of 3.1.002.06)

Find all objects in the current DOM which have been flagged as 'picked'.

Syntax

```
domObj.getPickedObjects()
```

```
domObj._DOM(dottedName)
```

Returned Value

This routine will always return a JavaScript array of pointers to DocOrigin DOM objects. Objects are flagged as 'picked' in DocOrigin Merge either by explicit setting the `_picked` attribute of an object, or by Design when calling an External Tool command. That returned array is empty if no matches are found.

Background

This routine searches only the current object (typically the object containing this script) as well as all objects within this object.

domObj.insertPage

(Merge only) (As of 3.0.002.04)

Insert a static page after the current page.

Syntax

```
domObj.insertPage(pageName [, placeBeforePageObject])
```

The optional *placeBeforePageObject* was added in 3.0.003.27.

Returned Value

Nothing directly, but the DOM is updated with the new page being inserted. As of 3.1.001.18, the *domObj* of the inserted page is returned.

Parameters

pageName is a string that is the name assigned to a Page definition at design time.

placeBeforePageObject is optional and is a Page object (not a string), e.g. `_document.Page3`, not "Page3".

domObj makes the most sense as a Page object. If it is a lower-level object, it really means the Page that the object is on. As of 3.2.001.01, the *domObj* can be the top-level form object, i.e. `_document`. In which case the inserted page will be at the end of the current set of pages, or, if supplied, before *placeBeforePageObject*.

Description

Sometimes one wants to conditionally insert a separator page, or one has a "layout 1" that repeats many times, but you want to insert a Terms and Conditions page as the second page in that sequence. This `insertPage()` function works very well for those scenarios. This function only makes sense in the **Pagination Completed** event. Without the *placeBeforePageObject* option, the inserted page is placed right after the page the *domObj* is / is on. Typically, one would use this function on a Page object and say `this.insertPage("pageName");`

The inserted page will not participate (have participated in) the data merge operation. Hence the inserted page must be designed as a static Page, or at least have all desired Panes as mandatory panes (ergo, effectively static). Global variables and automatic variables can be used. Any script in the inserted page's **Pagination Completed** (or later) events will be executed.

As with any Page, you can choose to page-number it or not number it. In most cases, inserted pages tend to fall into the not-numbered camp, but that's your design choice.

Note that this function does not insert some instance (occurrence) of an already merged page. Instead, it inserts an empty copy of a Page layout as per its design. You are not moving pages around, but inserting a clone of a design-time Page taken from the design file. The Page to be inserted should not be marked as mandatory, and not have any Fields on it that might cause the Page to be emitted by the merge process. Your use of the `insertPage()` function is the only reason this page should come out.

See Also

[domObj.clone](#)

domObj.relayout

(Merge only) (As of 3.1.002.04)

Re-layout a DOM object and all its descendant objects.

Syntax

```
domObj.relayout()
```

Returned Value

None.

Usage

Quite rare. A simple usage is to establish the actual height of a field that contains a reference like `[@xxxx.rtf]`. `relayout` will replace that reference with the contents of the file "xxxx.rtf" and word wrap the resulting text. After that has been done, `_height` will return the actual height of that object.

domObj.reparent

(Merge only)

Make an object in the DOM have a new parent object.

Syntax

```
domObj.reparent(newParent, [xOffset, yOffset])
```

Returned Value

Nothing.

Parameters

newParent defines the object which will be the new parent object for *domObj*. This should be a Page, Container, Pane, or Group. *(As of 3.1.001.15)* it's also acceptable for *newParent* to be zero. That indicates that *domObj* is to simply be removed from the document DOM.

(As of 3.2.001.01)

xOffset is the optional new `_left` for the object once it has its new parent. Defaults to zero.

yOffset is the optional new `_top` for the object once it has its new parent. Defaults to zero.

Usage

Very rare and unusual. This has been used to move objects into different Groups or Panes. Generally, it has been used in some form of "columnarizing". Given the built-in facilities of **Layout in multiple columns**, the usage of this function has dropped. It has been used to move objects off of one page and onto another one. This essentially allowed pages to be merged after pagination had been completed. Clearly, special circumstances were involved. Note that you could easily supply 0 for *xOffset* and *yOffset* and after the fact set `._absLeft` etc. as desired.

domObj.setTag

(Merge only) (previously named setParm, but that name is deprecated)

Merge maintains an internal tag table for various settings that control its behavior. This routine can be used to change these settings - set an internal tag value.

Syntax

```
domObj.setTag(name, value [, driver])
domObj.getTag(name, [, driver])
```

Parameters

name is the tag name to be set.

value is the value to be assigned.

driver is an optional printer configuration file designation used when setting PRT-specific parameters. This is strictly for convenience's sake. You could just as easily have used: *drivername* as the name. For example: "PDFBarcode39Text" as the name is identical to "Barcode39Text", "PDF".

Background

All objects have a fixed set of properties and those are accessed via standard DOM property access functions. However, some objects can take on either extra properties or have properties that are so rarely set that no specific UI is provided in Design to set them. Instead, Design has a **Format > Object Tags...** menu item that allows a form designer to nominate a tag and assign it a value. This open-ended capability allows many object's tags to be set without encumbering the Design UI with "clutter" for tags that are rarely set. In fact, being open-ended, it even allows for third-party tags to be defined for add-on application usage.

These tags participate in the cascading tag philosophy of overriding tags between PRM files, overriding PRM files, command-line options, PRT configuration settings, and now these Design UI-specified or script-specified tag settings. We expect that you will set applicable defaults in your configuration files and for the rare times when you want a different value, you can use setTag to accomplish that setting change on an object-by-object basis. Of course, you can also override configuration settings via Design's **Format > Object Tags...** user interface. The choice is yours.

In practice, override tags are most often used to tweak one or two of the myriad barcode symbology options when you have to deviate from your standard configured settings. setTag can also be used for PDF/UA Accessibility tags. Being able to 'walk the DOM' and use scripting to set accessibility tags is quite powerful. But of course, Design's **Format > Object Tags...** UI can be used as well.

You may invent other uses for tags of your own as well.

Example

```
this.setTag("BarcodeAUSPOSTText", "N");           // No text with this barcode
this.setTag("BarcodeAUSPOSTBarWidth", "0.5mm");    // set the bar width
```

Returned Value

setTag: None.

Old names

At one point these functions were called setParm and getParm. Those names still exist as synonyms for backward compatibility purposes but the preferred names, as of 3.1.001.08 are setTag and getTag. For the technically

curious, you can see the Design UI generated tag names and values for an object in its <parms> . . . </parms> element in the XATW.

See Also

[_parser.parms](#)
[domObj.getTag](#)

domObj.getTag

(Merge only)

The getTag function allows your script to get a previously set tag (see [domObj.setTag](#)). That setting may have occurred by script or via the Design UI.

Syntax

```
domObj.getTag(tagName [, driver])
```

Parameters

tagName is either a literal string or a variable that identifies the sought-after tag.

driver is an optional printer configuration file designation used when setting PRT-specific parameters.

Background

What are the valid settings? Check your PRT files, especially in the barcode section. In Design, when you select an object and use Design's **Format > Object Tags...** menu item you will see a list of any existing tags. You can add new tags if you like. Tags added at Design time are saved in the XATW. Tags added via script during Merge are not -- Merge never writes out the XATW.

The getTag syntax is simply this.getTag(*tagName* [, *driver*]) where *tagName* is either a literal string or a variable that identifies the sought after tag.

As of 3.1.001.03, there is a `_parser.parms(string)` function for scripts that run outside of Merge and are reading the XML of the XATW file. It parses the names and values that were set for the "tags" at Design time or by means external to DocOrigin processes. Generally, this is used for value-added packages that have their own set of tags to manage.

Returned Value

The string value of the chosen tag.

Old names

Before 3.1.001.08 these functions were called `setParm` and `getParm`. Those names still exist as synonyms for backwards compatibility purposes. For the technically curious, you can see the Design UI generated tag names and values for an object in its `<parms> ... </parms>` element in the XATW.

See Also

[_parser.parms](#)

domObj.sum

(Merge only) (As of 3.0.005.07)

Compute the sum of all descendant (child) Fields with a specified name.

Syntax

domObj.sum(*fieldname*)

Returned Value

The sum of all Fields name *fieldname* below the current object.

Parameters

fieldname is a Field name to search for.

domObj.xhtmlToRtf

(Merge only) (As of 3.1.001.21)

Return an RTF string that represents the given XHTML string.

Syntax

```
domObj.xhtmlToRtf([sStr])
```

Returned Value

RTF representation of the given string.

Parameters

sStr is optional. It defines the string of XHTML text to be converted. Without this parameter, the current value of *domObj* is used.

Usage

Very rare.

Examples

When *domObj* is a Field that is currently populated with XHTML markup this script will replace the XHTML with its RTF equivalent.

```
this._value = this.xhtmlToRtf();
```

When another object named field1 contains XHTML markup, this script will replace the *domObj*'s value with the RTF equivalent of field1's value.

```
this._value = this.xhtmlToRtf(field1._value);
```

To set the *domObj*'s value to the RTF equivalent of a literal string that contains XHTML markup.

```
this._value = this.xhtmlToRtf("Here is some <b>bold</b> text");
```


DOM Properties Accessing the Document Structures

In the Merge context one often wants to refer to properties of the objects in the various DOMs: `_document`, `_page`, `_data`. There is quite a large set of available properties to get, and often to update as well. The Script Editor dialog in DocOrigin Design provides you with ready access to these property names (and function names as well) by using a right-click context menu. You don't have to memorize every spelling, but do read through the upcoming lists and establish an understanding of all that is available. Not only do the DOMs themselves begin with "_" (underscore), but all properties do as well. This is to differentiate a DOM Object property such as `_width` from a potential DOM Tree element of, say, `_document.Cartoon.width`.

Property	Description
<code>._absLeft</code> <code>._absRight</code>	These are read-only convenience functions for getting the absolute positions, in microns, of the left and right of the object's virtual minimum enclosing rectangle. See also <code>._left</code> and <code>._right</code> .
<code>._absTop</code> <code>._absBottom</code>	These are convenience functions for getting and setting the absolute positions, in microns, of the top and bottom of the object's virtual minimum enclosing rectangle. (See also <code>.top</code> , <code>.height</code>). Note that these are computed properties not static ones. When one sets <code>_absTop</code> , one actually sets the <code>_top</code> of the object such that if <code>_absTop</code> were recomputed it would come out as desired. Similarly, setting <code>_absBottom</code> actually sets the <code>_height</code> of the object so that its <code>_absBottom</code> would evaluate to the desired value.
<code>._adjustHeight</code>	<i>(As of 3.1.002.06)</i> Return or set the Adjust height flag for a Field, one of "true" or "false".
<code>._adjustWidth</code>	<i>(As of 3.1.002.06)</i> Return or set the Adjust width flag for a Field, one of "true" or "false".
<code>._allowBreakAfter</code>	<i>(As of 3.1.002.06)</i> Return or set the Allow Break After flag for a Pane, one of "true" or "false". Set before pagination.
<code>._allowBreakBefore</code>	<i>(As of 3.1.002.06)</i> Return or set the Allow Break Before flag for a Pane, one of "true" or "false". Set before pagination.
<code>._allowSplit</code>	<i>(As of 3.1.001.10)</i> Return or set the Allow split flag for a Pane, Group, Field or Label, one of "true" or "false".
<code>._allowMultiple</code>	<i>(As of 3.1.001.10)</i> Return or set the Allow multiple flag for a Pane or Row, one of "true" or "false". Set at the Start-of-Job or Start-of-Document event.
<code>._autoFormat</code>	Set to "false" to turn off automatic formatting of Fields using Picture formatting. Note that setting to "true" does nothing. For example: <pre> this._value = Quantity._value * Price._value; if (this._value < 0) { this._value = "VOID"; this._autoFormat = false; } </pre> <p>This code would calculate an extended price and presumably format as currency. If the value was less than zero, the field would instead display VOID.</p>

Property	Description
._background ._backgroundColor	Return or set the background color. See Colors page for options.
._border	Return or set the display of an object's border, one of "true" or "false".
._borderColor	Return or set color of rectangle border. See Colors page for options.
._borderRadius	Return or set the radius for rounded corners. Returned values are in microns. See the Script Units description for setting values.
._bottomInset	Return or set the bottom inset for the object's border. Returned values are in microns. See the Script Units description for setting values.
._bottomMargin	Return or set the bottom margin. Returned values are in microns. See the Script Units description for setting values.
._caption	<i>(As of 3.1.002.06)</i> Return or set the caption field property.
._collate	Return or set the collate paper property. Best used on the Form object.
._color	Return or set the text color. See Colors page for options.
._count	Counts the number of sibling occurrences of a Pane or Row. Includes itself in the count. <u>Caution required</u> , see domObj.childNamed .
._currentContainer	Returns a pointer to this object's Container object.
._currentPage	Returns a pointer to this object's Page object.
._dataSource	Return the Data DOM object whose value was used to populate this Field. This makes sense only on Document Field objects. In other cases, it will return NULL. See the <code>_data</code> section for details on what attributes can be used with a Data DOM object.
._date	<i>(As of 3.2.001.03)</i> Returns the string value of a Field interpreted as a date and formatted as "yyyyMMdd" a string, empty string on failure.
._dateObj	<i>(As of 3.2.001.03)</i> Returns the JavaScript date object initialized by the value of a Field interpreted as a date, null on fail.
._duplex	<i>(As of 3.1.001.10)</i> Return or set a Form or Page's duplex property. Values are "dontChange", "simplex", "top", or "left".
._edge	Return or set the drawing of the object's borders. Values are "all", "none" or any combination of: "top", "bottom", "left", and "right", separated by a space.
._firstChild	Return the first (or only) child object of the current object. For example, if the current object is a Pane, then one could get the first object in that Pane. See also <code>._nextSibling</code> .

Property	Description
._fontBold	Return or set a Field's font bolding status, one of "true" or "false".
._fontItalic	Return or set a Field's font italic status, one of "true" or "false".
._fontSize	Return or set a Field's font point size.
._fontStrikeThrough	<i>(As of 3.1.001.15)</i> Return or set a Field's font strike through status, one of "true" or "false".
._fontTypeface	Return or set a Field's font typeface name.
._fontUnderline	Return or set a Field's underline option, one of "none", "single", "word", or "double".
._fullName	Return the fully qualified name of the object in dotted notation, e.g. <code>._document.SampleInvoice.Page1.CONTAINER1.InvoiceTable</code> . This is most applicable when "walking the DOM" via <code>._nextNode</code> .
._global	<i>(As of 3.0.003.11)</i> Return whether a Field is marked as Global or not. Essentially read-only, since by Data Merged, setting it would have no effect whatsoever.
._height	Return or set the height of an object. Returned values are in microns. See the Script Units description for setting values.
._hjustify ._horzJustify	Return or set the horizontal justification "left", "right", "center", "spread", or "spreadAll".
._hyperlink	Return or set a Field's hyperlink URL. See Hyperlinks for further information on hyperlinks.
._imageMode	Return or set the Image Size/Crop to "Scale", "Clip", "Resize", "Stretch" or "FitWidth".
._imageName	Return or set the Image file name of a Label object.
._initialValue	<i>(As of 3.2.001.01)</i> Return or set a Field's Initial Value . This makes sense only before the End of Merge event.
._input	Return or set a Field's input status, one of "true" or "false". See Fillable Forms for further information and examples.
._inputAmPm	Return or set Input Field AM/PM setting. If "true", use 12hr clock and AM/PM option, otherwise use 24hr clock.
._inputChoice	Return or set Input Field choice list. See Fillable Forms for more information.
._inputClearButton	Return or set Input Field Signature Clear button text. See Fillable Forms for more information.
._inputImage	Return or set Input Field Image. See Fillable Forms for more information.

Property	Description
._inputLabel	Return or set Input Field Label for choice lists, radio buttons etc. See Fillable Forms for more information.
._inputOrder	Return or set Input Field AM/PM setting. If "true", use 12hr clock and AM/PM option, otherwise use 24hr clock.
._inputReadOnly	Return or set Input Field Readonly status, one of "true" or "false". See Fillable Forms for more information.
._inputRequired	Return or set Input Field Required status, one of "true" or "false". See Fillable Forms for more information.
._inputRequiredMessage	Return or set Input Field Required message text. See Fillable Forms for more information.
._inputShortMonth	Return or set Input Field Date month style. Use "true" to display 'Jan' or "false" to display 'January'. See Fillable Forms for more information.
._inputShortYear	Return or set Input Field Date year style. Use "true" to display '09' or "false" to display '2009'. See Fillable Forms for more information.
._inputTray	Return or set the printer Input Tray for the current page.
._inputURL	Return or set Input Field Post destination URL. See Fillable Forms for more information.
._inputYear	Return or set Input Field date year range. See Fillable Forms for more information.
._lastChild	Return the last (or only) child object of the current object. For example, if the current object is a Pane, then one could get the last object in that Pane. See also ._prevSibling .
._left ._top	Return or set left or top position. Returned values are always in microns. See the Script Units description for setting values. The position is relative to top-left of the containing Pane, Cell, or Group. See also ._absLeft and ._absTop .
._leftInset	Return or set the left inset for the object's border. Returned values in microns. See the Script Units description for setting values.
._leftMargin	Return or set the left margin. Returned values in microns. See the Script Units description for setting values.
._lineSpacing	Return or set line spacing for text. Returned values are always in microns. See the Script Units description for setting values.
._lineStyle	Return or set the line style, one of "solid", "dash", "dot", or "dashDot".
._locale	<i>(As of 3.2.001.09)</i> Return the _locale (Format Currency , Number , Date/Time) JavaScript object which is in effect for the current object. If the object's locale is not set then the ancestor objects are traversed to find the inherited locale.

Property	Description
._localeName	(As of 3.1.00.10) Return or set the Locale (e.g. en_US) for the current object. Often this will be null and the ancestor tree will have to be followed to find the inherited locale in effect.
._mandatory	(As of 3.1.002.04) Return or set the Mandatory flag for a Pane, Row or Page, one of "true" or "false". Set at Start-of-Job or Start-of-Document .
._minHeight	Return or set minimum height allowed for this pane. Returned values in microns. See the Script Units description for setting values.
._name	Return or set an object's Name .
._nextNode	Return the next object in the DOM from top to bottom. This is an easy way to "walk the entire DOM". NULL is returned at the end of the DOM.
._nextSibling	Return the next child of a parent. Follows usage of ._firstChild or ._lastChild. NULL is returned if no more children exist in the desired next/previous direction.
._num	Returns the value of a Field as a raw number. This will remove any existing number formatting from a field value such as \$1,234.56 and return the number 1234.56.
._numberOfChildren	Return the number of children that the given object has.
._numberOfPages	Return the total number of pages in the document. This applies only on or after the Pagination Completed event.
._numCopies	Return or set the number of copies to be produced. Best used on the Form object.
._occurrence	Return an object's occurrence number within its enclosing parent pane. The numbers start at 0 the third occurrence of an object would return a value of 2. Typically used with repeating Panes.
._onSplitHeader	(As of 3.1.002.06) Return or set the OnSplit Headers... property for a Pane or Overflow Headers... for a Container. It's a space-separated list of header panes.
._onSplitFooter	(As of 3.1.002.06) Return or set the OnSplit Footers... property for a Pane or Overflow Footers... for a Container. It's a space-separated list of footer panes.
._originalForm	Return a Page's "original form name" for a composite Form created using DocOriginCombineForms.
._outputBin	Return or set the printer Output bin for the current page.
._overflowText	Return any text that overflows from a fixed-size text Field. Not applicable to Single line Fields, or Fields set to Adjust height .
._pageBreak	Return or set page break flag for a Pane.

Property	Description
._pageCount	Return the number of pages in the document. This makes sense only after the Pagination Completed event.
._pageCountForm	<i>(As of 3.1.002.02)</i> Return the number of pages in the form.
._pageNumber	Return the page# of the object document based. Since repeating line items (Panels) typically cause multiple pages to print, the final page number is not set until the OnEndPaginate() event. Prior to this event, all Fields will have the same page number.
._pageNumberForm	<i>(As of 3.1.002.02)</i> Return the page# of the object form based.
._pageOrientation	<i>(As of 3.2.001.01)</i> Return or set a Page's Orientation , one of "Portrait" or "Landscape".
._pagePlacement	<i>(As of 3.1.001.10)</i> Return or set a Page's placement property, one of "any", "anyFromFront" and <i>as of 3.2.001.11</i> , "front", or "back".
._paperSize	<i>(As of 3.2.001.01)</i> Return or set the Paper size , one of "A4", "Letter", etc.
._paperType	Return or set the Paper type for the current page.
._parent	Return the parent object of the current object. This returns NULL when at the top of the DOM's ancestor stack.
._picture	Return or set a Field's picture clause. DocOrigin uses the ICU package for picture formatting. See setting-time-zones .
._presence	Return or set the presence of an object, one of "yes", "no", or "none". "none" indicates that the object takes up no space at design or print time.
._prevSibling	Return the previous child of a parent. Follows usage of ._firstChild, or ._lastChild. NULL is returned if no more children exist in the desired next/previous direction.
._raw	Return the current Field/Label value. For Fields, this is the value before Field formatting has been applied (number, currency, date, and so on). See ._value.
._rightInset	Return or set the right inset for the object's border. Returned values are in microns. See the Script Units description for setting values.
._rightMargin	Return or set the right margin. Returned values are in microns. See the Script Units description for setting values.
._rowCount	Counts the number of child Rows or Panels of a Table or Panel. Table Rows marked as headers or footers are ignored in this count.
._sheetCount	Return the number of sheets of paper required to print the document. This makes sense only after the Pagination Completed event.

Property	Description
._singleLine	(As of 3.1.002.06) Return or set the Single line flag for a field, one of "true" or "false".
._templateObject	Return the Template (Form) DOM object from which the given Document DOM object was created. This makes sense on only Document DOM objects. In other cases, it will return NULL. This is useful for retrieving the value that an attribute was set to at design time. Often <code>_height</code> has a different value in the Document DOM than it has in the Template DOM. For example: <code>var designHeight = this._templateObject._height;</code>
._textWidth	Returns the width of the longest (widest) line in a text block, in microns. See also <code>_overflowText</code> .
._thickness	Return or set the border thickness in microns.
._topInset	Return or set the top inset for the object's border. Returned values are in microns. See the Script Units description for setting values.
._topMargin	Return or set the top margin. Returned values are in microns. See the Script Units description for setting values.
._transparent	Return or set the object's transparency, one of "true" or "false". Set to "true" on an Image to allow what is under the Image to show through in the Image's lighter areas. See <code>TransparencyMask</code> in your configuration file.
._type	Return the type of a document object. See the table of those type values below.
._value	Return or set the current Field/Label value. The returned Field value has formatted text after applying the formatting (number, currency, date, and so on) to the original data. See <code>._raw</code> .
._viewerAttributes	Return or set a Field's viewer attributes. See Fillable Forms for further information and examples.
._visible	Return or set the visibility of the Field or Label, one of "true" or "false".
._vjustify ._vertJustify	Return or set the vertical justification, one of "top", "bottom", or "middle".
._width	Return or set the width of an object. Returned values are in microns. See the Script Units description for setting values.

Object type values returned from `object._type`

"Form", "Page", "Container", "Pane", "Group", "Line", "Rectangle", "Table", "Row", "Cell"

"Field-Text-String", "Field-Text-Number", "Field-Text-Currency",
"Field-Text-DateTime", "Field-Text-Date", "Field-Text-Time"

Object type values returned from *object._type*

"Field-Checkbox", "Field-Radiobutton", "Field-Image", "Field-Barcode", "Field-Comb"
"Label-Text", "Label-Image", "Label-Barcode"
"Arc", "PolyLine" dynamically generated by _chart

Examples

1. The following example determines the sum of an amount_total field by page:

```
// abbreviation for repeating panes
var r = _document.main_pane.detail_table.detail_line;
var sum=0.0; var page=1;

for (var i=0; i < r[0]._count; i++) {
  if (r[i]._pagenumber != page) { // when the page# changes, print info
    _printf("Sum for page %d is %10.2f\n", page, sum);
    sum = 0.0; page++;
  }
  sum = sum + r[i].extended_amount._num;
}
_printf("Sum for page %d is %10.2f\n", page, sum);
```

2. While it is not often that scripting adjusts object locations, here is an example that makes a "trailer" pane into a page"footer". Used in the Pagination Completed event. Imagine that this is on a "totals" pane or something of that nature.

```
// Place this pane at the very bottom of the container
this._absTop = this._currentContainer._absBottom - this._height;
```


Debugging Script

DocOrigin Merge provides a number of options and functions to assist in debugging scripts:

- the `-trace` command line option.

This causes the program to write a detailed trace log of its execution steps as the script is being executed. Note that the trace file is **overwritten** on each run (whereas the log file is appended to). Use this command option as follows:

```
... -trace "C:/DocOrigin/User/test/trace.log" ...
```

- the `_tracef` function call.

This can be used within your script to write information directly to the trace file specified in the `-trace` command line option. If `-trace` is not specified it does nothing. Well, not quite nothing. If you are tracing out values that require work to create (a favorite is `._fullName`, which is fairly expensive to come up with) then that effort/time will be expended even though no trace output will be generated. As of 3.1.002.03, you can use conditional commenting, by prefacing the `_tracef` line with `!!`, to fully remove any processing costs.

- the `_message` function call (Windows only).

This is a variation of the `_message` function that will display a message as a Windows popup message box. It has effect only when the `-debug` command-line option has been specified. That means that you can leave it in during production (where `-debug` is not set!). Well, see the conditional commenting remarks above. That applies here too. By the way, there is a `_dlogf()` function as well. Log output occurs only when `-debug` is specified.

For example:

```
_dmessage("my var='%s'", myvar); // display myvar
```

Techniques

See all Command-Line Options

Sometimes you just want to know what the command line options were and any other information that has been provided to your script. The following is a fine way to see everything in `_job`.

```
if (typeof _job != "undefined") {
  for (var prop in _job)
    _logf("_job.%s = '%s'\n", prop, _job[prop]);
  if (typeof _job.command != "undefined")
    for (var prop in _job.command)
      _logf("_job.command.%s = '%s'\n", prop, _job.command[prop]);
  if (typeof _job.options != "undefined")
    for (var prop in _job.options)
      _logf("_job.options.%s = '%s'\n", prop, _job.options[prop]);
  if (typeof _job.filter != "undefined")
    for (var prop in _job.filter)
      _logf("_job.filter.%s = '%s'\n", prop, _job.filter[prop]);
}
```

In general, it is good to become friends with the standard JavaScript features of:

```
typeof variable
```

```
and for (var prop in object) ... object[prop]
```

Note: As of 3.1.001.24 the function `_job.logf(["context string"])` was added to all of the above logging of the `_job` object. The `context string` is optional. It simply helps to identify where in the script you are dumping out this info.

Also in 3.1.001.24, a similar function, `_cache.logf(["context string"])` was introduced to easily log all the `_cache` settings defined at the time of the call.

i You cannot use `for (var prop in domObject)`. The performance penalty to support that would be far too great. You have to know the property names of those dynamic DOM objects.

Traverse the DOM

Well, suppose you just want to know your whole world, perhaps wanting to see what the names really are as opposed to what you *think* is out there:

```
_logf("\n%E -- The entire document DOM..\n");
for (var r=_document; r; r=r._nextNode) {
  _logf(".. '%s'\n", r._fullName);
}
_logf("\n\n");
```

It's a great habit to put single quotes around strings that you are debugging. You might be surprised how often there is a trailing (or leading) character that you did not expect.

Of course, you can traverse other DOMs than the `_document` DOM, e.g. the `_data dom`, or an individual page's `_page DOM`. And you can start at some level other than the root of the DOM.

Sometimes you may be surprised by what doesn't show up. A bit of reflection can then lead to an 'Ahah' moment.

Note the use of `_fullName`. It's a wonderful thing.

What pages do I have

At `Pagination Completed` time it is often desirable just to 'traverse' through all the pages. This can be true at `Data Merged` time too, but less often, in my experience.

```
_logf("\n%E -- All my pages..\n");
for (var pg=_document._firstChild; pg; pg=pg._nextSibling) {
  _logf(".. '%s'\n", pg._fullName);
}
_logf("\n\n");
```

The *trick* here is that page objects are always the immediate children of the top-level `_document` object. That makes the above loop far faster than using the entire `_nextNode` loop depicted farther up.

Naturally, you might want to look at more than the names of the pages, and you might want to predicate your logging with whether `pg._visible` is `true`. It's all available at your property's fingertips.

Note that `%E` is a special format specifier. It requires no parameter. It reports the name of the event that this script is in, e.g. "Data Merged" or "Pagination Completed". Use it. It might tweak you to the fact that you put your script in the wrong event.

What panes do I have

Suppose you are on a page object and you want to know what panes have been instantiated for this page [instance]. You could use:

```
_logf("\n%E -- All panes of '%s'..\n", this._fullName);
for (var pn=this._firstChild; pn; pn=pn._nextNode) {
  if (pn._type == "Pane")
    _logf(".. '%s'\n", pn._fullName);
  if (pn._type == "Page") break;
}
```

Because panes can be nested, you can't use just `_nextSibling`. Instead, use `_nextNode`. But now you have way too many nodes. Use the `_type` property to select only the type of object that you want. It happens that you will be surprised at all the instances that it reports, or perhaps more so, doesn't report. It's very helpful to know these things.

See Also

[-phase](#)
[_cache \(Scripting Object\)](#)

Colors

DocOrigin scripting defines a number of pre-defined color names that can be used on document objects. You can also specify colors using standard RGB notation.

Setting Colors











Colors may be set by either specifying the RGB value of the color or using one of the pre-defined color names from the table below. RGB values must be specified as a 6-digit hex code with an optional # character preceding it. As of version 3.0.002.07 colors may also be specified as a 3-digit hex code such as #3fe which will be expanded to #30f0e0. (This is compatible with standard browser syntax).



When the color of a DocOrigin object is returned it will always be returned in the 6-digit format, with no leading # character.

```
Address._color = "red";
Item._color = "FF0000"; // equivalent of "red"
Item2._color = "#FF0000"; // alternate notation
a = Item2._color; // returns "FF0000"
```

Note that all color settings - whether pre-defined names or hex codes - must be defined as strings.

Pre-defined Colors

Transparent	FFFFFFF	
Beige	F5F5DC	
Black	000000	
Blue	0000FF	
Brown	A52A2A	
Coral	FF7F50	
Cyan	00FFFF	
DarkGray	A9A9A9	
GreenYellow	ADFF2F	
Gray	808080	
Green	00FF00	

LightBlue	ADD8E6	
LightCyan	E0FFFF	
LightGreen	90EE90	
LightGray	D3D3D3	
LightPink	FFB6C1	
LightYellow	FFFFE0	
Magenta	FF00FF	
Orange	FFA500	
Pink	FFC0CB	
Red	FF0000	
Salmon	FA8072	
Turquoise	40E0D0	
Yellow	FFFF00	
White	FFFFFF	

To set the text background to transparent use:

```
this._backgroundColor = "transparent";
```

See Also

[DOM Properties](#)

Script Units

Several script extensions require the setting of document object positions or sizes. For example:

```
Field1._left = "1.5in";
```

Internally DocOrigin stores all positions and sizes in units of *microns* - 1/1000000 of an inch. When you request the value of an object's position or size you will get back an integer value in microns. However, when you set the value of a position or size you can also specify the number in various convenient units of measure:

```
Field0._left = 1500000;    // 1500000 microns
Field1._left = "1.5in";   // 1500000 microns
Field2._left = '1.5";     // 1500000 microns
Field3._left = "10pt";    // 1389 microns
Field4._left = "1.5cm";   // 590551 microns
Field5._left = "15mm";    // 590551 microns
```

Note that when doing arithmetic on these coordinates that you must use microns. The function `_toDOUnits` can also be used to convert from the above units to microns. The function `_fromDOUnits` can also be used to convert from the microns to the above units.

```
Field2._left += _toDOUnits("1in");    // move 1 inch to the right
Field3._left += 1000000;               // also move 1 inch
_fromDOUnits(2000000, "in");          // returns 2 (inches)
```

See Also

[_toDOUnits](#)

Custom Design Script Menu

Users can create custom options in the Design Script Editor by placing a DesignScriptMenu.json file in the DocOrigin Overrides (\$O) folder. The example below shows some simple, real JavaScript options along with a few "Sample Only" entries. In this example, options will display:

- directly in the top menu
- child menu with seven real JavaScript Samples
- and a second child menu level with 3 fake samples

```

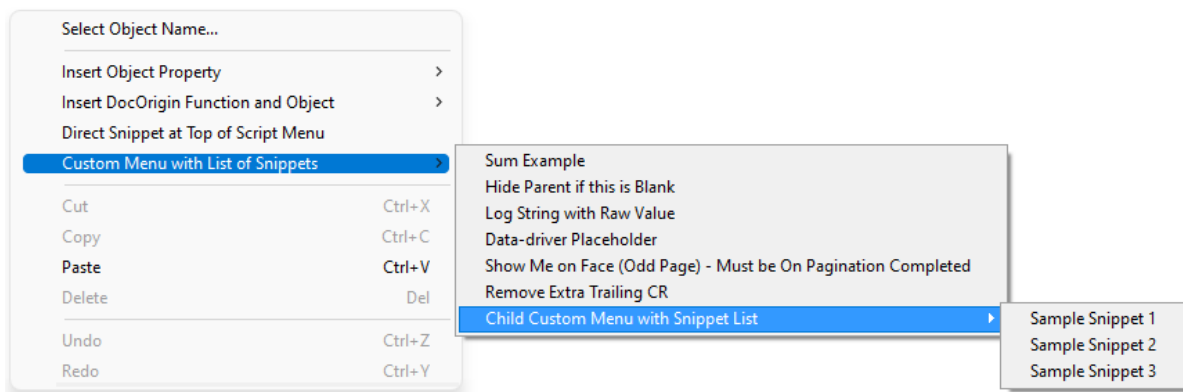
$O/DesignScriptMenu.json

{
  "Direct Snippet at Top of Script Menu" : "insert script goes here",
  "Custom Menu with List of Snippets" :
  {
    "Sum Example" : "this._value = _document.sum('INSERT_FIELD_NAME')",
    "Hide Parent if this is Blank" : "if (this._value == '') this._parent._presence =
'none'",
    "Log String with Raw Value" : "_logf('My message is = %s',this._raw)",
    "Data-driver Placeholder" : "this.setTag('placeholder',
_data.insertQualifiedFieldName._value + '.pdf')",
    "Show Me on Face (Odd Page) - Must be On Pagination Completed" : "if
(this._pageNumber % 2 == 0) this._presence = 'none'",
    "Remove Extra Trailing CR" : "this._value = this._value.replace('\n', '')",
    "Child Custom Menu with Snippet List" :
    {
      "Sample Snippet 1" : "xxx",
      "Sample Snippet 2" : "yyy",
      "Sample Snippet 3" : "zzz"
    }
  }
}

```

As you can see in the screenshot below, two new top-level options are shown, **Direct Snippet at Top of Script Menu** and **Custom Menu with List of Snippets**.

- You can create a set of options with a parent menu by enclosing it in curly brackets.
- The name shown to the user is on the left side of the colon (:) enclosed in double-quotes such as "Sum Example"
- The inserted script is on the right side of the colon (:) enclosed in double-quotes such as "this._value = _document.sum('INSERT_FIELD_NAME')"
- If your script contains quotes, single quotes must be used
- You can include placeholders to help the user such as INSERT_FIELD_NAME
- Each option in a list must be divided by a comma
- Try it! Create a file called DesignScriptMenu.json and place it in your \$O/Overrides folder and paste the content above.




File Naming Conventions

Characters Restricted from File Names

File names in DocOrigin are, by default, restricted to names that do NOT contain any of the following characters

?[]=+<>;',*"

nor any ASCII control characters (0x00 - 0x1F).

 *As of version 3.2.001.01*, the user has control over the application of character restrictions via the command line option `-RestrictedFileNames` **Yes/No**; the default is **Yes**. A value of **No** will allow the restricted characters to be used, but your OS may object and processing failures may occur.

The restricted characters `[]` are interpreted as [auto/embedded field](#) name references. Note that most of those substitutions are available only in Merge and that embedded references are processed before the character restrictions take place. If any of the restricted characters are present in an output file name they are automatically replaced with the underscore character. For example, a file name of `f?r+ed` will be converted to `f_r_ed`. File names supplied for input files (e.g. a form name) will not replace the restricted characters and thus may generate errors if the Operating System rejects the name.

% Placeholders in File Names

File names can contain one or more of the following case-sensitive substitution strings, which are expanded at run time.

```
%a - abbreviated weekday name (for example, "Wed")
%A - full weekday name (for example, "Wednesday")
%b - abbreviated month name (for example, "Mar")
%B - full month name (for example, "March")
%d - replace with the 2 digit current day
%D - replace with the current data file name (no path or extension) (as of 3.0.005.05)
%F or %f - replace with the current form name (no path or extension)
%H - replace with the 2 digit current hour (local time, 24 hour clock)
%I - replace with the 2 digit current hour (local time, 12 hour clock)
%i - in a Folder Monitor context, replace with the Folder Monitor instance name
%j - replace with the 3 digit day-of-year (001-366)
%m - replace with the 2 digit current month
%M - replace with the 2 digit current minute
%p - current locale's AM/PM indicator for 12 hour clock
%P - replace with the current process id (as of 3.0.003.20)
%S - replace with the 2 digit current second
%t - the full path of the temporary output file produced when using the run:: syntax
    (Merge-only) (as of 3.0.004.05)
%T - replace with the path of the DocOrigin temp folder (as of 3.0.003.20)
%u - replace with a short, file safe, unique string (as of 3.0.003.20)
%U - Week of year (2 digit 00-53, starting on Sunday)
%v - last output file path+name+ext (Merge-only) (as of 3.1.002.07)
%w - Weekday as number (0-6), starting on Sunday
%W - Week of year (2 digit 00-53, starting on Monday)
%x - last output name+ext (Merge-only) (as of 3.1.002.07)
%y - replace with the 2 digit current year
%Y - replace with the 4 digit current year
%z - last output file name only (Merge-only) (as of 3.1.002.07) %% - replace with %
```


For example the command line setting:

```
-logfile $L%d_%m_%Y_logfile.txt
```

Would generate a logfile named:

```
C:\DocOrigin\User\Logs\15_03_2012_logfile.txt
```

assuming March 15, 2012 were the current date.

 The above % placeholders may be used in [command line options](#). However, if a file name template string with embedded % placeholders were passed as an argument to a script function, you would not get your desired result. Script functions **do not** automatically expand the % placeholders listed above. The user is expected to exert control over their file names by using [_file.resolveName](#) or [_resolve](#) - which do handle the % placeholder substitutions. The user would pass the [resolved](#) file name to the script function. Script functions **do** automatically resolve **\$X** folder mappings if they are used in an argument that is for a file or directory name.

Folder Monitor Instance Name Substitution

In the Folder Monitor context the special placeholder, **%i** can be used to include the monitor instance name in file name specifications.

If you are not using "Multiple Monitors", but just the "Default Monitor", then the sequence `_%i` will be replaced with nothing (discarded).

E.g.

```
-logfileFormat $L/FM_%i_%U.log  
-queue1 $U/FolderMonitor/%i/Jobs
```

For a monitor instance named AdHoc with DocOrigin installed in the usual location, those would become, in week 23:

```
-logfileFormat C:/DocOrigin/User/Logs/FM_AdHoc_23.log  
-queue1 C:/DocOrigin/User/FolderMonitor/AdHoc/Jobs
```

For the Default monitor, it would be:

```
-logfileFormat C:/DocOrigin/User/Logs/FM_23.log  
-queue1 C:/DocOrigin/User/FolderMonitor//Jobs
```

\$X String Substitutions

Using a parameter file, it is possible to set up some shortcuts or abbreviations for commonly used command strings. Any statement of the form:

```
$X = string or
$X string
```

where **X** is any single uppercase letter, and string is any string of characters.

Subsequently, anywhere that **\$X** appears in a value string, it will be replaced by the value of its associated string.

For example, consider the following two files:

Paths.prm

```
* Paths.prm
* $U User directory
$U $R/User
* $F is the user's forms directory
$F $U/Forms
* $L directory for log files
$L $U/Logs
```

Test.prm

```
@Paths.prm
-logfile $L/%d_%m_%y_logfile.txt
-form $F/DynamicForm.xatw
-data $F/DynamicForm.xml
```

When the command line Merge @Test.prm is executed, each occurrence of **\$L** and **\$F** in Test.prm will be replaced by the path values defined in Paths.prm.

This is recursive. **\$R** is C:/DocOrigin (usually set at install time), hence one can define **\$U** relative to **\$R** and **\$F** relative to **\$U**.

Whenever you run a script within a DocOrigin program these \$ values are also automatically defined in the JavaScript environment. So if you set

```
$Z "this is my string"
```

on the command line (or in a .prm file), then when you run a script, the JavaScript variable named **\$Z** will be pre-defined with the value this is my string. E.g., as if you had hand-coded the following at the start of your JavaScript (.wjs) file:

```
$Z = "this is my string"
```


Note that RunScript scripting is not PHP. RunScript does NOT do text substitution in string literals. For example, **\$F** is indeed, literally "\$F" and not automatically "C:\DocOrigin\User\Forms". But since **\$F** is defined as a JavaScript variable, you could say

```
var myForm = $F + "/MyForm.xatw";
```

Notice that one includes the directory separator character in the concatenation.

Update: Having said all that, if a string such as "\$F/MyForm.xatw" were passed to a DocOrigin script utility, to a file open operation, DocOrigin does now automatically replace the **\$F** with whatever path is defined for it. That applies to any **\$X** variable. But this is for those special file opening use cases only. As said before, it is not replaced

automatically as PHP does, at value assignment time, but rather it is replaced when and if used in a [file open](#) operation. We call this operation HardenPath.

 By the way, within DocOrigin scripting and parameter files, you can always use / as the directory separator. \ works as well but often one has to escape it as in \\. It's easier and more consistent to use /.

See Also

[_resolve](#) and [_file.resolveName](#) for ways to explicitly resolve references to **\$X**-type variables, and more.

External file include

In data item values or label values, you can use a file reference rather than a literal value. You supply this in the format: `[@fileName.ext]`. Since you do not have scripting available right in the data stream, it is mightily convenient that something like `[@$A/myAdImage.jpg]` does work, first hardening the **\$A** to whatever path it is defined to be. Similarly for `[@$U/Clauses/myClause.rtf]`.

Please see also [_file.resolveName](#) as the scripting way to "harden a path".

Startup \$'s

By default the DocOrigin Install provides a set of pre-assigned \$ values in the Default-DocOrigin.prm file and automatically includes them in the main applications. This in turn means any scripts you write can use those settings.

When any DocOrigin executable begins it will immediately set the following internally:

\$B	The initial base working directory - the directory the program is running in
\$E	The path to the executable being run e.g. c:/DocOrigin/DO/bin
\$N	The base name of the executable (no extension); e.g. Merge
\$T	The path for DocOrigin temp files <tmp>/AtwTempFiles. Where <tmp> is the user's TMP env var setting

After that, each DocOrigin executable will automatically load \$E/Default-DocOrigin.prm and follow along its use of @@includes of other .prm files. During this process, it updates the definition of any \$X variable as soon as it runs across it. Of course, you are allowed to, and should, add or update your \$X variables. However, these have proved so convenient that some are used in the code and **their meaning should not be tampered with:**

\$R	The DocOrigin install directory
\$U	The DocOrigin User root folder - \$R/User
\$C	Where the user's printer configuration (.prt) files reside - \$U/Config
\$S	Where scripts reside - up to you, possibly \$U/Overrides, or \$U/Scripts
\$O	Where User override files reside
\$L	Where log files reside - \$U/Logs, by default
\$F	Where form files reside. - Up to you, \$U/Forms seems likely
\$P	Semicolon separated paths used to find files. - Automatically postfixed with \$E

You can update the paths that those point to as long as that path reflects the meaning described above. For example, if you wish to have the message files somewhere other than standard, ok, put them there and update \$M to point to that new location for message files.

Our thought is that you should stick with \$V, \$W, \$X, \$Y, and \$Z. If you need more than that please advise us.

\$P was introduced in 3.1.002.01. It differs from all the other folder mappings in that it is a semicolon-separated list of folders which define a "DO PATH" to be searched. Typically it would be used in a _run("\$P/executable.exe", ...) context. The named file would automatically be looked for in each folder specified in \$P, and if not found in any of them, \$E would also be searched. If you know the executable is in \$E you should use \$E, not \$P. \$P is handy when referencing a third-party executable (e.g. ImageMagick.exe) where its installed location might vary from system to system. While less likely, \$P might be used for any file, e.g. in an _file.fopen("\$P/resource.xxx", "r") context. As of 3.1.002.06 it is possible to use environment variables within \$P, like [PATH].

Special \$'s

(Merge only) There are two special variables. Often auxiliary collateral needed by Merge is located relative to the supplied form file or data file. So...

\$\$F

\$\$F means the folder where the form specified by `-form` is. (or first form if a form list is being used). Note that it is not 'the folder where forms are usually kept', but is rather more specific, referring to the folder where the form is for this run of Merge. If `DocOriginCombineForms` is used, resulting in a temporary form file, that has no bearing, **\$\$F** will be the folder used/implied in the `-form` parameter.

\$\$D

\$\$D means the folder where the data specified by `-data` is. Running filters which create temporary versions of the data file have no bearing, **\$\$D** will be the folder used/implied in the `-data` parameter.

These have no connection with, are not at all intertwined with `$F` or `$D`, whatever they might be. These are two, independent, special variables.

It is possible to say `-data $$F/../../data/xxx.xml`. That is, you can define **\$\$D** by using **\$\$F**.

(FilterEditor only) There is one special variable.

\$\$X

\$\$X means where the `.xfilter` file is. **\$\$X** occurs inside `.xfilter` files. Its use makes `.xfilter` files and their related collateral more portable, e.g. for support requests or sharing with other team members.

Glossary

Terms

Term	Definition
Container	A rectangular area within a page object into which dynamic content in the form of one or more panes gets flowed vertically.
DocOriginSendMailServer	Usually, it is impractical to wait for a connection to a mail server to have a single email sent. Instead one often queues email packets in a folder, and DocOriginSendMailServer is launched, <u>without waiting</u> , to effect the connection to the mail server and send the emails. DocOriginSendMailServer is not truly a server as it will exit as soon as all queued-up emails have been sent. It will be started again once a new email is to be sent.
Data Explorer	This is the panel that can be revealed in the Design UI. You nominate a sample XML file to be depicted in the Data Explorer. After that, you can drag elements or even whole structures onto the design canvas in order to create fields with the appropriate names.
Design canvas	The middle part of the Design UI where you draw objects for your form layout.
Document	The result of Merge processing one set of data, combined with the form. Many documents can be included in a single spool file or a single PDF. Think of a document as one invoice, or one statement.
Dynamic form	A form that is constructed in Design such that the objects may be in either fixed or non-fixed locations on the page(s). The objects or object groups in non-fixed locations are typically intended to allow for multiple occurrences of the associated data in the data stream. When the dynamic form is combined with the data stream in Merge, the content of the data stream, that is, the number of occurrences of each non-fixed object group flows down page(s), generating output that uses as many pages as necessary to satisfy the data. Technically, a dynamic form has at least one Container object and one or more Pane objects.
Data DOM (Document Object Model)	A hierarchical representation of the data loaded from the XML data file.
Document DOM	A hierarchical representation of the document to be "printed". Fields are populated with data from the Data DOM. The Document DOM is the result of combining the Form DOM with the Data DOM.
Filter	This is typically a data preprocessor that massages the input data into suitable XML for merging into a form. Various filters are supplied and custom ones can be deployed as well.
FilterEditor	This is a UI tool for graphically defining the rules for extracting data from a spool report file so as to create structured XML for merging into a form.
FolderMonitor	This facility allows jobs dropped into watched folders to be detected and processed.
FolderMonitor Instance	One can have several FolderMonitors running simultaneously, each monitoring its own watched folders. Each running FolderMonitor is termed an 'instance'.
Form DOM	A hierarchical representation of the form as it was defined in Design.
Form Explorer	This is the panel in the Design UI that is typically on the left-hand side. It depicts the hierarchical structure of the form design. Objects may be selected in the Form Explorer, just as they can be on the design canvas. In the Form Explorer, objects may be dragged up or down in the form structure.

Term	Definition
Fully Qualified Name	The hierarchy of tag names for an element from an XML data file. The fully qualified name is the target data field name preceded by the tag names of all of its ancestors, for example: <code>InvoiceTable.DetailLine.Quantity</code> Note that the XML tag names are case-sensitive, therefore, <code>DetailLine.Quantity</code> is not the same as <code>DetailLine.quantity</code> .
Fully Qualified Name Element Node	One level in the reference hierarchy of a fully qualified name, for example, <code>DetailLine</code> in the example above.
Group	A Group object on a form template is a rectangular area which can contain other text, fields, images, or other graphical objects. It creates another level of structure to the data. Unlike a Pane, it is not dynamic.
Job	The name given to the entire set of data that Merge is processing that may result in one or more documents. Merge executes a single job each time it is run.
Job Name Discovery	This is a script (JavaScript) that is used by FolderMonitor to quickly examine the job's data file and determine the applicable 'job name' (job type).
Job Processing Script	This is a script that is run by FolderMonitor, well, FMTransaction actually, to process the data for a given job. The script typically performs a Merge of the data and a nominated form but may include as many steps as you like.
Job Queue	This is a combination of a watched folder name and a file mask which defines which files are to be detected by FolderMonitor. Many such queues may be defined.
Pagination	After the data has been merged into the document DOM, it's quite likely that containers will be filled with more panes than will fit in the container. The pagination phase then breaks such overflowing containers into multiple pages such that each container is as full as is permitted by the form design's Allow split, Break before, Break After, Force Break rules.
Pane	A rectangular object that may contain a set of one or more nested panes or any combination of: table, field, label, group, line, and box objects. Panes are the only objects that flow down the container from one page to the next in a dynamic document. Tables and fields nested within a pane may be split across pages but for that to happen they must be nested within a pane.
Static form	A form that is constructed in Design such that the objects are in fixed locations on the page(s). The locations are set in Design and cannot be altered by Merge processing. Technically, a static form has no Container objects and no Pane objects.

Acronyms

Acronyms	Definition
DOSMS	DocOriginSendMailServer = for sending queued email
FE	In a Design context: Form Explorer; outside of Design: FilterEditor
FM	FolderMonitor
FVR	Form Verification Report -- produced in Design
JND	Job Name Discovery script used by FolderMonitor
JPS	Job Processing Script used by FolderMonitor
MTX	Metrics -- Font Metrics file extension
PRM	Parameter file extension

Acronyms	Definition
MPRM	Merge parameter file extension -- On Windows, it has an association with Merge.exe
PRT	Printer Configuration file extension
QM	QueueMonitor
WJS	"Wonderful" JavaScript file extension
XATW	Xml All The Way -- the form design extension

Return Codes

DocOrigin applications report return codes, most typically in the logs. The following provides a brief explanation of each.

The return code numbers were changed so as to be compatible with Unix which restricts return codes to the range 0 to 255.

If your script or filter executable provides a return code, we request that it be in the 200 to 250 range.

FilterEditor

RC	Mnemonic	Meaning
0	OK/SUCCESS	All is well.
1	ambiguous	Deliberately not used. Does it mean true/Ok or not-zero so not Ok?
2, -101	ERR_FILTER_ARGLIST	Data Filter argument list is invalid.
3, -102	ERR_FILTER_INPUTFILE	Data Filter input file cannot be opened.
4, -103	ERR_FILTER_OUTPUTFILE	Data Filter output file cannot be created.
5, -104	ERR_FILTER_FORMFILE	Data Filter form file cannot be opened.
6, -105	ERR_FILTER_PROCESS	Document sort has encountered issues with the XML data file.
7, -101	ERR_COMBINEFORMS_ARGLIST	Combine forms argument list is invalid.
8, -904	ERR_SCRIPT_INPUT_FILE	When parsing in JavaScript, the input file was not found.
9, -905	ERR_SCRIPT_OUTPUT_FILE	When parsing in JavaScript, the output file could not be written.
10, -906	ERR_SCRIPT_DATA_CONVERT	Error parsing a comma-delimited file.
11, -907	ERR_SCRIPT_FORMAT	Error parsing a fixed format data file to XML.

SendMail

RC	Mnemonic	Meaning
21, -101	ERR_EMAIL_ARGLIST	DocOriginSendMailServer has invalid arguments.
22, -102	ERR_EMAIL_SCANFOLDER	No -mailscanfolder option has been provided.
23, -103	ERR_EMAIL_FROM	The from address was not supplied or is invalid.
24, -104	ERR_EMAIL_ATTACH	An attachment file was not found.

RC	Mnemonic	Meaning
25, -106	ERR_EMAIL_TO	No To address was supplied, or is invalid.
26, -107	ERR_EMAIL_CC	The Cc address is invalid.
27, -108	ERR_EMAIL_BCC	The Bcc address is invalid.
28, -109	ERR_EMAIL_TEXTOPEN	The @file specified for the text mail body was not found.
29, -113	ERR_EMAIL_SEND	cURL was unable to send the email message.
30, -115	ERR_EMAIL_SMTPHOST	The -smtp host option was not specified.
31, -116	ERR_EMAIL_HTMLOPEN	The @file specified for the HTML mail body was not found.
32, -117	ERR_EMAIL_EMLOPEN	Could not create the email packet file to send.
33, -118	ERR_EMAIL_CURLCMDOPEN	Could not create the cURL command file with which to invoke cURL.
34	ERR_EMAIL_ATTACH_OPEN	An attachment file could not be accessed. Check CombineDocuments.

Merge

RC	Mnemonic	Meaning
41, -201	ERR_MERGE_ARGLIST	Invalid or missing argument list.
42, -202	ERR_MERGE_NOFONTS	The configuration has no fonts defined.
43, -203	ERR_MERGE_CANNOTLOG	Log file undefined or not writable.
44, -204	ERR_MERGE_NOCONFIG	Missing or invalid Configuration file.
45, -205	ERR_MERGE_NOFORMFILE	No form file specified.
46, -206	ERR_MERGE_NODATAFILE	No data file specified.
47, -207	ERR_MERGE_LOADSCRIPT	Error loading the -scriptFile script.
48, -208	ERR_MERGE_FILTER	A filter provided a non-zero return code.
49, -209	ERR_MERGE_FORMLOAD	Could not load the form file.
50, -210	ERR_MERGE_WORDWRAPFORM	Could not word wrap the text in a document.
51, -211	ERR_MERGE_PROCESSCRIPT	Error processing any Merge event script.
52, -212	ERR_MERGE_DATAOPEN	Could not open the data file.
53, -213	ERR_MERGE_DATAPROCESS	Error while processing all the data documents.
54, -214	ERR_MERGE_FORMANDDATA	Error merging the data with the form design.
55, -215	ERR_MERGE_EMBEDDEDFIELDS	Error while embedding [fields] in the document.
56, -216	ERR_MERGE_WORDWRAPDOC	Error word-wrapping the document.
57, -217	ERR_MERGE_PAGINATE	Error paginating the document.
58, -218	ERR_MERGE_PRINTDOCUMENT	Error printing the document.

RC	Mnemonic	Meaning
59, -219	ERR_MERGE_GLOBALS	Error while copying global values to applicable global fields.
60, -220	ERR_MERGE_AUTOFIELDS	Error while populating the automatic fields.
61, -221	ERR_MERGE_TO_TOO_LONG	unused
62, -222	ERR_MERGE_CC_TOO_LONG	unused
63, -223	ERR_MERGE_BCC_TOO_LONG	unused
64, -224	ERR_MERGE_ATTACH_TOO_LONG	unused
65, -225	ERR_MERGE_NO_OUTPUT_DATA	Name not provided for the creation of sample data.
66, -226	ERR_MERGE_CREATE_SAMPLE_DATA	Could not create the sample data.
67, -227	ERR_MERGE_MISSING_PROFILE	Could not load profile (including printer table).
68, -228	ERR_MERGE_PLACEHOLDERS	Failed to process placeholders.

PDFExtract

The negative PDFExtract return codes were in use up to 3.1.001.16

RC	Mnemonic	Meaning
41, -1	ERR_MERGE_ARGLIST	Invalid or missing argument list.
70, -2	ERR_INPUT_OPEN_ERR	An input PDF file could not be opened
71, -3	ERR_OUTPUT_OPEN_ERR	The output PDF could not be opened
72, -4	ERR_PDF_READ_ERR	An input PDF file could not be read
73, -5	ERR_DO_MISSING_ERR	The DocOrigin document index markings are missing
74, -6	ERR_INVALID_PAGELIST	The -page option is incorrectly specified
75, -7	ERR_NO_EMBEDDED_DATA	The input PDF does not have its data embedded
76, -8	ERR_NOT_A_DO_PDF	An input PDF is not a DocOrigin-produced PDF
77, -9	ERR_NO_EXTRACT_FILE	No file name provided for -extractdata
78, -10	ERR_ATTACH_OPEN_ERR	A file to be attached could not be opened

FolderMonitor

RC	Mnemonic	Meaning
81, -701	ERR_NO_DATA_FILE	No data file found to process.
82, -702	ERR_FILE_NOT_FOUND	Data file has disappeared since it was enumerated.
83, -703	ERR_FILE_NOT_OPENED	Even after retries, the job's data file could not be opened.
84, -704	ERR_NO_TRANSACTION	No job name was provided for this job.
85, -705	ERR_RECORD_AS_ERROR	Could not move a failed job to the error folder.
86, -706	ERR_RECORD_AS_PROCESSED	Could not move a processed job to the processed folder.

RC	Mnemonic	Meaning
87, -707	ERR_FOLDERMONITOR_NO_ARGLIST	Folder Monitor was given no arguments at all.
88, -708	ERR_FOLDERMONITOR_ARGLIST	Arguments needed for Folder Monitor are missing.
89, -709	ERR_FOLDERMONITOR_INVALIDARGS	An argument to Folder Monitor is invalid.
90, -710	ERR_NO_QUEUES	The <code>-queue1</code> option has not been specified.
91, -711	ERR_NO_SCRIPT_FOLDER	No <code>-scriptFolder</code> option has been provided.
92, -712	ERR_NO_ERROR_FOLDER	No <code>-errorFolder</code> option has been provided.
93, -713	ERR_INVALID_ERROR_FOLDER	The specified error folder name is invalid.
94, -714	ERR_INVALID_PROCESSED_FOLDER	The specified processed folder name is invalid.
95, -715	ERR_ADDING_QUEUE	Could not add another <code>-queueⁿ</code> to Folder Monitor's list of queues.
96, -716	ERR_INVALID_SCAN_FOLDER	The folder to scan for jobs is invalid.
98	ERR_FAIL_STOP_MONITOR	<i>(as of 3.0.003.20)</i> Have your Job Processing script return this so as to both fail the job and stop the host FolderMonitor instance.
99	ERR_PASS_STOP_MONITOR	<i>(as of 3.0.003.20)</i> Have your Job Processing script return this so as to pass the job but also stop the host FolderMonitor instance before starting on a new job.
100	ERR_KEEP_STOP_MONITOR	<i>(as of 3.0.003.20)</i> Have your Job Processing script return this so as to stop the host FolderMonitor instance before starting on a new job, and keep this job in the queue for when the instance is started again.

Process Spawning

RC	Mnemonic	Meaning
111, -801	ERR_SPAWN_PROCESS	The creation of a process failed.
112	ERR_NOT_LICENSED	Non-Design, non-Merge, tool used without a license.
113	ERR_PIPE_PROCESS	The piping while creation of a process failed.

Scripting

RC	Mnemonic	Meaning
121, -901	ERR_SCRIPT_COMPILE	The JavaScript code did not compile.
122, -902	ERR_SCRIPT_EXECUTE	The JavaScript code had an execution error.
123, -903	ERR_SCRIPT_FILE_NOT_FOUND	The JavaScript file was not found.

Xfilter

RC	Mnemonic	Meaning
-2	MERGETRANSFORM_UCONV	uconv failed.
-3	MERGETRANSFORM_CC	Convert carriage control failed.
-4	MERGETRANSFORM_XFILTER	Error loading .xfilter file.

Reserved for Unix

On Unix systems, process return codes are limited to 8 bits, i.e. 0 to 255. In order to finesse our way around this obstacle we often negate the return code and in Merge at least, coalesce the higher numbered return codes to unused smaller numbers so as to be valid return codes for Unix. It's definitely a problem and one reason why we hope to replace the reporting of return codes with the reporting of meaningful strings.

Now that we have renumbered the return codes that we use, this isn't as much of a problem. However, scripts or whatnot may still give us out-of-Unix-range return codes.

RC	Mnemonic	Meaning
127	missing command	Very often, Unix shells report 127 if they cannot find the program to be executed.
128-140	Unix signal	Unix often reports 128 + the applicable signal number.
251-255	small negatives	One often sees 255, which is because someone returned -1

User

RC	Mnemonic	Meaning
200-250	reserved for you	If your scripts or filters or whatever you run issue a return code, it would be nice if it were in this range.

Latest ReadMe

[Click here for the latest ReadMe](#)

Web Services

The DocOrigin web services allow you to use DocOrigin remotely. Both the REST and SOAP web service types are supported. They are independent units but utilize the same core to provide the same functionality.

Here is a short (but incomplete) list of the most obvious scenarios when it may be useful:

1. Integrate DocOrigin document generation services into your applications.
2. Use DocOrigin from web pages.
3. Use DocOrigin from an OS which DocOrigin doesn't support.

Processing Details

Jobs

The unit of work for a DocOrigin web service (WS) is termed a "job". What that job does is defined by a previously deployed script. Either the delivered default script or a user-customized script may be invoked by the DocOrigin WS call. The WS is almost certainly requested to deliver a data payload, and may also provide additional user-defined options if so desired.

Of course, the WS always returns a result. Certainly, it will return operational 'headers' but most frequently it also returns the primary result of the job, quite typically a PDF file. (But it could be anything.) One call and you get your result delivered to you. WS requests may choose either sync or async mode. The vast majority are done in synchronous fashion. You make the call and the result is returned to you ASAP. However, you can initiate an async job, and make later requests for the status and collateral of the defined job. When the WS is requested to do a job, it creates a new, uniquely named, folder for that job. All resources pertaining to that job are kept in that job's folder.

REST and SOAP web services both provide similar request choices:

1. Run a job, sync or async
2. Get a status/result of a job
3. Delete job data (the entire folder created specifically for the identified job)
4. Get a list of files available for a job (i.e. the files left by the script in the job's folder)
5. Get a specific file for a job (e.g. log.txt)
6. Get a WS version

REST API

The REST API is best seen via the built-in "Swagger" docs. Use the Tomcat manager application and click on your web service to navigate there.

The API includes:

1. Execute a job (sync or async)
2. Delete the job folder
3. Get the job result
4. Get a list of a job's available files
5. Get a selected job file
6. Execute Merge job (*as of v. 1.3.5*)

SOAP API

The SOAP API is best seen via its WSDL. Use the Tomcat manager application and click on your web service. You may use that wsdl for your SOAP client implementation.

The API includes:

1. runJob (sync)
2. submitJob (async)
3. deleteJob
4. getJobResult
5. listJobFiles
6. getJobFile
7. runMerge (sync) (*as of v. 1.3.5*)
8. submitMerge (async) (*as of v. 1.3.5*)

Job Internals

All configuration values mentioned below are taken from the provided [Default-]DocOriginWs.ini file. See [Configuration](#).

For reference, what follows is a sample WS configuration override file: \$0/DocOriginWs.ini

```
* DocOrigin Web Services configuration
TempPath=C:\DocOrigin\User\Temp
KeepJobHours=1
* RunTimeoutSec=60
* JobInfo=jobInfo.properties
Input=D0WebService.input
```

When the web service (WS) receives a request to run a job, it creates a new folder under the path specified by the TempPath configuration value. The name of the created folder is guid-like (e.g. 1fb4e383-fe9f-475e-ad90-45ffc981f5e2) and in fact is the "jobid" assigned to the job.

The WS then places the supplied input (data) file into that folder with the file name specified by the Input configuration value. Note that the configuration names (e.g. Input) are case-sensitive.

(Before v. 1.3.5) The WS then executes DocOrigin's RunScript program providing it a set of command line options:

Option	Command line options that your script receives
-cd	the just created, jobid-named, job folder
-input	the file name of the input data supplied to the WS (not passed if input is not supplied to WS)
-script	\$E/Default-WsRun.wjs (or the scriptName parameter supplied to the WS)
-logfile	log.txt (which will be in the job folder)
-options	[value of the options parameter] (not passed if options is not supplied to the WS)

(As of 1.3.5) The WS then executes DocOrigin's command template (specified by the cmdDefault configuration value). By default it runs RunScript program providing it a set of command line options:

Option	Important command line options that your script receives
-cd	the just created, jobid-named, job folder
-input	the file name of the input data supplied to the WS
-script	\$E/Default-WsRun.wjs (or the scriptName parameter supplied to the WS)
-logfile	the name of the log file (which will be in the job folder)
-options	value of the WS options parameter
-user	the name of the WS user requested the job
-async	boolean value representing sync/async type of call

Note that options is not DocOrigin specific but rather an arbitrary string which your script understands. e.g. key1=value1;key2=value2. The WS itself has no idea what you might put in that options string. It's yours to use as you wish.

What your script does is completely up to you. Typically, it quickly examines the supplied data file contents and determines what processing should be done, for example, which form design file to apply. The script might also

look at the supplied options for guidance on what specific user actions apply to this request. Those options are user-defined, the WS is oblivious to them.

The delivered \$E/Default-WsRun.wjs is a great example to start from, if you want to create a customized script. If you do supply a custom script it must be pre-deployed on the server that is hosting the WS. You cannot send the actual script content along with the WS request. That would be super dangerous. Control your WS-accessible scripts.

Returned Results


The WS is designed in a way such that it can immediately return one result file and several properties. The file to return is defined by the output property described below. Typically the result file is a DocOrigin Merge generated PDF file, but it could be anything. Notice that, with further calls to the WS, you can get other results from your job as well. See the API. Typically a single call with a single result suffices.

While your script does whatever you want, it has to keep in mind these rules:

1. The script has to create a 'communications' file named **jobInfo.properties** (by default) in its job folder. That file name has whatever name you defined in the JobInfo configuration value. That file is to contain a set of *name=value* lines. The content of the file will be read by the WS and returned to the caller. For example, in REST, those 'job info properties' are returned as do.xxx response headers and the nominated output file is returned as the response body. See the info block below.
2. Some names in the **jobInfo.properties** file have a special meaning:

jobInfo.properties critical contents	
output	if set, is used by the WS as the file name of the primary 'result' file which is to be returned to the caller as the response body.
mime	if set, is used as the content-type of the "output". (often application/pdf, or plain/text as applicable.

3. **jobInfo.properties** must be created as the very last operation of your script since the WS decides whether the job is complete by checking for this file's existence.

 jobInfo.properties must be created in job's folder. See "Limitation" topic.

The WS returns three special info fields back to indicate job execution state:

(These are returned as response headers in REST)

WS REST response headers	
do.jobId	the assigned job id (and name of the job's folder)
do.completed	is true if jobInfo.properties exists and can be parsed.
do.exitCode	is the return code of the executed command.

For reference, what follows is a fairly typical set of response headers from a REST WS call. Please draw your attention to the custom headers that begin with "do.".

```

HTTP/1.1 100 Continue

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
do.completed: true
do.jobId: f4469907-d668-4cb0-9a93-75e994fdeb73
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers: do.completed, do.jobId, do.exitCode, do.property.rc,
do.property.message, do.property.output, do.property.mime
do.exitCode: 0
do.property.rc: 0
do.property.message: OK
do.property.output: C:\DocOrigin\User\Output\PurchaseOrder_2016-001.pdf
do.property.mime: application/pdf
Date: Thu, 13 Jun 2019 23:23:14 GMT
Content-Type: application/pdf
Transfer-Encoding: chunked

```

Note, that the `jobId` is available to you, and you will need that if you choose to make more WS API calls regarding this same job. Also note that the WS special `do.jobId`, `do.exitCode`, and `do.completed` headers are differentiated from the user-defined headers by being named only `do.xxx` and not `do.property.xxx`.

For more correlation, the following is the `jobInfo.properties` file contents for the above job:

```

rc=0
message=OK
output=C:\DocOrigin\User\Output\PurchaseOrder_2016-001.pdf
mime=application/pdf

```

Note, that `async` calls return only limited set of data (like `jobId`) because job is not yet completed by the time of WS response. You may ask for job data later using `jobId`.

As of v. 1.3.4, added `do.service.url`, `do.service.base`, `do.service.path` to Web Service response header.

Limitations

WS is designed with some build-in automation regarding job outcome. Like, in synchronous scenario WS automatically returns the generated output back to the client. So WS has to know where and how the output is called, you do not want to break this automation. Also, you do not want one job to spoil the file-system results of the other jobs.

By default your script starts out with its current directory being the 'job folder'. It is not recommended to change the current directory in your script or form for the reasons mentioned above. All work should be done in the job's folder, and `jobInfo.properties` must be created there.

Also, while you are welcome to specify "business-level" options like `-cache`, `-config`, etc., it is not recommended to set "system-level" options like output location, logfile location, etc. for the same reasons.

The only useful exception of these rules we can think of is the case when you want a fire-and-forget (`async`) job sending output directly to some printer. Then it is OK to specify custom output location.

The Default Script

By default, DocOrigin provides the **\$E/Default-WsRun.wjs** script which

1. consumes an XML file
2. tries to determine the applicable form name
3. tries to find that form under `%DO_ROOT%\DO\Samples\`
4. runs DocOrigin Merge to generate a PDF
5. returns that PDF as the request response.

The WS returns additional information. For example, in REST, we return `do.jobId`, `do.completed`, `do.exitCode` plus all other `name=value` pairs from the `jobInfo.properties` file as `do.property.xxx` response headers.

Out-of-the-box, you can try the WS by executing a job and passing `%DO_ROOT%\DO\Samples\Sample_Invoice.xml` as the input parameter. You should receive a generated PDF back.

Direct Merge call

As of v. 1.3.5, new functionality was added - direct Merge call. It is **`/jobs/merge`** for REST and **`run/submitMerge`** for SOAP.

This allows you to avoid any script calls which speeds up output generation and simplifies set up process.

Let's say at client side, having your data file ready you already know that you want a form called **`Sample_invoice.xatw`** and PCL output. So you may call **`/jobs/merge`** REST WS just passing:

- input data in `input` parameter
- say `-form $R/DO/Samples/Sample_invoice.xatw -config Default-LJ4.prt` in options parameter

As a result, a single Merge call will be executed using supplied data file and options, and resulting output returned back to the client.

Note, you should be careful which options you are sending with this call. It is natural that you will want to send something "business-level" like `-form`, `-config`, `-cache`, etc. But do not pass "system-level" options like `-cd`, `-logfile`, etc. See "Limitation" topic.

Calling The WS

How do you 'execute a job', i.e. call/invoke the WS? Well..

There are lots of ways, as many ways as there are computer languages and frameworks. The WS distribution package includes a 'reference implementation' for a Java-based invocation of the WS. You can certainly peruse that (both REST and SOAP) for inspiration (if you need any). See [DEMO Application](#).

One way to see an invocation of the WS 'out in the open' is via a cURL call. Following is a Windows .bat file that uses cURL to invoke the WS, i.e. to execute a job. The point here is not .bat files or cURL but just showing one, nothing up our sleeves, way to execute a job.

The meat of the .bat file is as follows

```
set input=%~1
set server=myws.example.com
set port=8443
curl -X POST -u %myDOWSCreds% -k          ^
  https://%server%:%port%/DORest/1.3/jobs ^
  -F input=@"%input%"                   ^
  -F options="a relevant option string"  ^
  -D wsHeaders.txt                       ^
  -o wsOutput.pdf
```

The presumption in this .bat file snippet is that you have put your DO WS credentials into an environment variable named `myDOWSCreds`. That would be in the format `userId:password`. There are other ways to specify your credentials. Use whichever method you like but you do have to give the WS useful credentials.

Of course, you need to know the name of the server where the WS has been installed.

The port is something you need to get from the installer. They could easily have configured the ports that Tomcat is using. By default, port 80 is used for http: access and port 443 is used for https: access. By some long ago programmer's whim, the non-standard port numbers 8080, and 8443 often appear in use. ...Ask the installer.

The `-k` option is a synonym for `- -insecure`. That option has been shown since during initial "playing" it is not uncommon for the server to be protected by a mere self-signed certificate. Not recommended, but out-of-our-

scope, and we don't want you to endure hassles when you are just getting started. Security is something you need to be aware of and have professionally addressed.

The `-D wsHeaders.txt` is not mandatory, but in the interests of illustration, it's nice to see what headers come back.

The `-o wsOutput.pdf` (note the small o) is needed to receive the primary response result. The explicit `.pdf` extension is a bit *optimistic* since some failure could have occurred along the way and it may not be a `.pdf` that comes back. You should be checking the headers that come back before *ass-u-ming*.

If it applied, you could also supply a `-F scriptName="someCustomPredeployed.wjs"`

Prerequisites


Web services operate using a client - server concept.

The client machine is any machine; it does not have any particular prerequisites other than to be able to invoke a web service. curl is handy for that.

The server machine is where DocOrigin and DocOrigin Web Services are. It is this server machine that we are referring to in the following prerequisites discussion.

The DocOrigin web services run under the JVM (Java Virtual Machine). You need to prepare your DocOrigin server environment to support its operation. You may have these facilities already, but if not, you would need to:


STEP 1 Install Java SE 8.

 **Warning!** Java SE 11 and newer are not supported.

You need only the JRE (Java Runtime Environment), not the full JDK (Java Development Kit). If/When Java is installed you should be able to run `java -version` to see/verify which version you have. You should see something like:

```
java version "1.8.0_261"
Java(TM) SE Runtime Environment (build 1.8.0_261-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.261-b12, mixed mode)
```

STEP 2 Install Apache Tomcat 7 or above.

 **Warning!** Tomcat 10 requires the special deployment of `.war` files. See [Deployment](#).

Please note where your Tomcat installation is located.

In a typical Windows, XAMPP-based install, tomcat is at `C:\xampp\tomcat`

In a typical Windows-based independent tomcat install, tomcat is at `C:\Program Files (x86)\Apache Software Foundation\Tomcat 7.0`

Elsewhere, we refer to the tomcat install location as `[tomcat]`. That location will have child folders named: `bin`, `conf`, and `webapps` (among others).

To see/verify which version of Tomcat you have, run:


`[tomcat]\bin\version.bat` on Windows

`[tomcat]/bin/version.sh` on Linux

STEP 3 Set environment variable `DO_ROOT`.

On Windows, this is set automatically by the DocOrigin install.

On Linux, you can set it for Tomcat by adding `export DO_ROOT=/var/DocOrigin/` to the `[tomcat]/bin/setenv.sh` file and then restart Tomcat. If `setenv.sh` is not there, don't despair, just create it with that one `export...` line.

 Don't include the quotes in the export command. Use the path which points to your DocOrigin installation.


Deployment

Tomcat

In order to use Tomcat's Manager app (like [http\(s\)://yourTomcat/manager/html](http(s)://yourTomcat/manager/html)) to deploy .war files, you need to have configured a Tomcat user with the `manager-gui` role. If you don't have such a Tomcat user already, then add these lines to the `[tomcat]/conf/tomcat-users.xml` file in the section `<tomcat-users>` (note, you can (and should!) change the username and password):

```
<role rolename="manager-gui"/>
<user username="admin" password="admin" roles="manager-gui" />
```

Alternatively .war files can be placed in the `[tomcat]/webapps` directory and Tomcat will automatically pick up and deploy them.

 Warning! At the moment DocOrigin web services don't support web deployment on Tomcat 10 so .war files should be placed in the `[tomcat]/webapps-javaee` directory and Tomcat will automatically convert and copy them to the `webapps` directory.

The DocOrigin web services distribution package is a zip file named based on its release version. It consists of three parts:

1. Servers, which are two WAR files for the REST and SOAP services.
2. Client examples, which are two Java applications using REST and SOAP, with sources.
3. Offline documentation.

DocOrigin web services require users with the `do-rest-user` role (for using REST) and/or the `do-soap-user` role (for using SOAP). These roles need to be configured on the server. The roles can be assigned to either new users or any existing users.

To define a user who can make both soap and rest requests to DocOrigin web services, add the following lines to the `<tomcat-users>` section of the `[tomcat]/conf/tomcat-users.xml` file.

Note that you should use username and password of your choice.

```
<role rolename="do-rest-user"/>
<role rolename="do-soap-user"/>
<user username="user" password="pass" roles="do-rest-user,do-soap-user"/>
```

In production, it's recommended to configure your application server to use `https` for all connections. For configuration details see the reference documentation for your application server.

REST

To install the DocOrigin REST web service, deploy the `server/rest/D0Rest.war` file from the distribution package zip. You can either merely copy it to `[tomcat]/webapps` or you can use the Tomcat web-based manager application available at `http(s)://[host:port]/manager/html`

By default, the REST service will be available using the URL: `http(s)://[host:port]/D0Rest/1.3`

SOAP

To install the DocOrigin SOAP web service, deploy the `server/soap/D0Soap.war` file from the distribution package zip. You can either merely copy it to `[tomcat]/webapps` or use the Tomcat web-based manager application available at `http(s)://[host:port]/manager/html`

By default, the SOAP service will be available using the URL: `http(s)://[host:port]/D0Soap/1.3`

Configuration

Copy the following files from the Web Service package's server/resources

```
Default-WsRun.wjs Default-DocOriginWS.ini Default-WsSecurity.ini Default-WsSecurity.wjs
```

to

```
%DO_ROOT%/DO/Bin.
```

The WsRun script can be overridden at %DO_ROOT%/User/Overrides/WsRun.wjs file.

The Default-DocOriginWS.ini file can be overridden with %DO_ROOT%/User/Overrides/DocOriginWS.ini.

Note, when editing the .ini file you cannot use the usual **\$X** style folder mappings.

The .ini file can define the following properties:

- TempPath root folder to store jobs, defaults to C:\DocOrigin\User\Temp\
- Input file name where input is stored in job's folder defaults to input
- JobInfo file name of communication file, stored in job's folder, defaults to jobInfo.properties
- RunTimeoutSec timeout for sync jobs, job is killed after timeout, defaults to 60
- KeepJobHours approximate time to keep finished jobs folders on file system, those are cleared automatically, defaults to 1
- LogFileName The name of the log file in job folder, defaults to DOWebService.log
- LimitedApi (As of version 1.3.8) Limit the REST API to a single "Execute a job" endpoint


Adjust files if necessary.

Logging

It may happen that you want more or fewer logs produced during web service work. It is configurable.


In order to change the log settings of a deployed app you can edit the [tomcat]\webapps\[DORest or D0Soap]\WEB-INF\classes\logback.xml file and restart the app. Available logging levels are TRACE, DEBUG, INFO, WARN, ERROR, ALL, or OFF.

For more info, see <https://logback.qos.ch/manual/configuration.html>

 Note, logback.xml will be overridden after you redeploy the web service. So it may be a good idea to keep its copy in a safe place if you reconfigured it.

You will be able to see web service logs in stdout files, like [tomcat]\logs\tomcat7-stdout.2019-06-06.log

Script security

 Note, as of version 1.3.5 security is disabled by default. See [Default-]DocOriginWs.ini and cmdDefault.

As of *version 1.3.1*, you can restrict which scripts can be run by each specific Tomcat user.

Security is controlled by the **\$E/Default-WsSecurity.ini** and **\$E/Default-WsSecurity.wjs** files and their usual overrides. On each web service call the WsSecurity.wjs file reads the WsSecurity.ini file and makes the decision of whether to allow or deny access.

By default, all scripts are allowed.

You can add restrictions by putting the WsSecurity.ini file into the **\$U/Overrides** folder and adding lines identifying script names and allowed user names. See the examples in **\$E/Default-WsSecurity.ini**.

If the script decides that access is denied, the http response headers provided by the web service will contain a non-zero `do.exitCode` and the `do.property.message` will be "*access denied*".

More advanced requirements may require overriding **\$E/Default-WsSecurity.wjs** with your own script and implementing security in any preferred way.

DEMO Application

You can find reference implementations for DocOrigin WS REST and SOAP clients under the `client/rest` and `client/soap` folders of the distribution package.

Collateral is provided to build them with Maven but of course, you need Maven to be installed to build it using Maven. Alternatively, you can import `pom.xml` into the Java IDE of your choice.

REST

The REST client reference implementation is located in `client/rest/restClient`.

It's a Maven project with all the configured dependencies that you need to do REST calls, plus it contains an example call to run a WS job.

To build and package the app from the command line, execute `mvn package` in the `client/rest/restClient` folder (the one with the `pom.xml` file in it).

After the build completes, in the target folder, you'll find two jars: `restClient.jar` - which is just the client alone, and `restClientWithDependencies.jar` - which is a runnable uberjar with the client app and all its dependencies.

You can experiment with it by running it via the command line using the supplied `testRest.bat` file. It will read the `testRest.properties` file at runtime and execute the REST request. Be sure to adjust the `testRest.properties` file to match your configuration.

SOAP

The SOAP client reference implementation is located in `client/soap/soapClient`.

It's a Maven project with all the configured dependencies that you need to do SOAP calls and Web Services Description Language (WSDL) for the java generation step, plus an example call to run a WS job.

To build and package the app from the command line, execute `mvn package` in the `client/soap/soapClient` folder (the one with the `pom.xml` file in it).

Once the build is complete, in the target folder you'll find two jars: `soapClient.jar` - with just the client alone, and `soapClientWithDependencies.jar` - a runnable uberjar with the client app and all its dependencies.

You can experiment with it by running it via the command line using the supplied `testSoap.bat` file. It will read the `testSoap.properties` file at runtime (be sure to adjust those properties to match your configuration) and execute the SOAP request.

Test

To test the default installation, copy `Sample_Invoice.xml` from `%DO_ROOT%/DO/Samples` to the folder where `testSoap.bat` is located. After a successful run, you'll notice a newly generated PDF in that same folder.