



Documaker

Documaker Server System Reference

version 11.3

Skywire Software, L.L.C.
3000 Internet Boulevard
Suite 200
Frisco, Texas 75034
www.skywiresoftware.com

Phone:	(U. S.)	972.377.1110
	(EMEA)	+44 (0) 1372 366 200
FAX:	(U. S.)	972.377.1109
	(EMEA)	+44 (0) 1372 366 201
Support:	(U. S.)	866.4SKYWIRE
	(EMEA)	+44 (0) 1372 366 222
		support@skywiresoftware.com

PUBLICATION COPYRIGHT NOTICE

Copyright © 2008 Skywire Software, L.L.C. All rights reserved.

Printed in the United States of America.

This publication contains proprietary information which is the property of Skywire Software or its subsidiaries. This publication may also be protected under the copyright and trade secret laws of other countries.

TRADEMARKS

Skywire® is a registered trademark of Skywire Software, L.L.C.

Docucorp®, its products (Docucreate™, Documaker™, Docupresentment™, Docusave®, Documanager™, Poweroffice®, Docutoolbox™, and Transall™), and its logo are trademarks or registered trademarks of Skywire Software or its subsidiaries.

The Docucorp product modules (Commcommander™, Docuflex®, Documerge®, Docugraph™, Docusolve®, Docuword™, Dynacomp®, DWSD™, DBL™, Freeform®, Grafxc commander™, Imagecreate™, I.R.I.S.™, MARS/NT™, Powermapping™, Printcommander®, Rulecommander™, Shuttle™, VLAM®, Virtual Library Access Method™, Template Technology™, and X/HP™ are trademarks of Skywire Software or its subsidiaries.

Skywire Software (or its subsidiaries) and Mynd Corporation are joint owners of the DAP™ and Document Automation Platform™ product trademarks.

Docuflex is based in part on the work of Jean-loup Gailly and Mark Adler.

Docuflex is based in part on the work of Sam Leffler and Silicon Graphic, Inc.

Copyright © 1988-1997 Sam Leffler.

Copyright © 1991-1997 Silicon Graphics, Inc.

Docuflex is based in part on the work of the Independent JPEG Group.

The Graphic Interchange Format© is the Copyright property of CompuServe Incorporated. GIFSM is a Service Mark property of CompuServe Incorporated.

Docuflex is based in part on the work of Graphics Server Technologies, L.P.

Copyright © 1988-2002 Graphics Server Technologies, L.P.

All other trademarks, registered trademarks, and service marks mentioned within this publication or its associated software are property of their respective owners.

SOFTWARE COPYRIGHT NOTICE AND COPY LIMITATIONS

Your license agreement with Skywire Software or its subsidiaries, authorizes the number of copies that can be made, if any, and the computer systems on which the software may be used. Any duplication or use of any Skywire Software (or its subsidiaries) software in whole or in part, other than as authorized in the license agreement, must be authorized in writing by an officer of Skywire Software or its subsidiaries.

PUBLICATION COPY LIMITATIONS

Licensed users of the Skywire Software (or its subsidiaries) software described in this publication are authorized to make additional hard copies of this publication, for internal use only, as long as the total number of copies does not exceed the total number of seats or licenses of the software purchased, and the licensee or customer complies with the terms and conditions of the License Agreement in effect for the software. Otherwise, no part of this publication may be copied, distributed, transmitted, transcribed, stored in a retrieval system, or translated into any human or computer language, in any form or by any means, electronic, mechanical, manual, or otherwise, without permission in writing by an officer of Skywire Software or its subsidiaries.

DISCLAIMER

The contents of this publication and the computer software it represents are subject to change without notice. Publication of this manual is not a commitment by Skywire Software or its subsidiaries to provide the features described. Neither Skywire Software nor its subsidiaries assume responsibility or liability for errors that may appear herein. Skywire Software and its subsidiaries reserve the right to revise this publication and to make changes in it from time to time without obligation of Skywire Software or its subsidiaries to notify any person or organization of such revision or changes.

The screens and other illustrations in this publication are meant to be representative, not exact duplicates, of those that appear on your monitor or printer.

Contents

Chapter 1, Introduction

- 2 System Overview
- 3 Rules Publishing Solution Overview
- 4 Document Automation Evolution
 - 4 Stage 1 - paper automation
 - 5 Stage 2 - workflow automation
 - 6 Stage 3 - paperless information automation
- 7 Document Automation Goals
- 8 System Benefits

Chapter 2, Understanding the System

- 11 Processing Overview
- 14 Processing Options
- 15 Using Banner Processing
 - 15 Enabling banner processing
 - 15 Specifying banner forms and scripts
 - 17 Banner form processing and multi-file print
 - 18 Processing logic
 - 19 DAL functions
 - 19 Banner processing example
- 21 Using Multi-step Processing
 - 21 Creating Transaction Records
 - 22 File Summary
 - 23 Processing Transactions
 - 24 Output Files for GenPrint
 - 24 Output Files for GenWIP
 - 24 Output Files for GenArc
 - 25 File Summary
 - 27 Creating Print Spool Files
 - 28 File Summary
 - 29 Sending Incomplete Transactions to WIP

	30	File Summary
31		Archiving Transactions
	32	File Summary
	32	Rules Used in Multi-Step Processing
33		Restarting the GenData Program
	33	RULCheckTransaction rule
	34	RestartJob rule
	34	INI options
35		Generating Batch Status Emails
37		Tracking Batch Page Statistics
	37	Recipient Page Statistics
38		Batch Totals Summary File
	38	Accessing totals in GenPrint
	38	INI Options
	39	Sample Log File
40		Default DFD Files
	40	RCBStatsDtIDFD
	42	RCBStatsTotDFD
43		Controlling GenTrn Processing
45		Using Single-step Processing
	45	Creating and Processing Transaction Records
	46	System Settings and Resources
	47	Creating Print Files
	48	File Summary
	49	Using the MultiFilePrint Callback Function
50		Mapping Fields with XPath
51		Running Archive in Single-Step Processing
51		Running WIP in Single-step Processing
52		Rules Used in Single-step Processing
	52	Archive
	52	BatchingByRecipINI
	52	BatchByPageCount
	52	BuildMasterFormList
	53	ConvertWIP
	53	InitArchive
	53	InitConvertWIP
	53	InitPrint
	53	InitSetRecipCache

53	NoGenTrnTransactionProc
53	PageBatchStage1InitTerm
54	PaginateAndPropagate
54	PrintFormset
54	ProcessQueue
54	StandardFieldProc
54	StandardImageProc
54	WriteNAFile
54	WriteOutput
54	WriteRCBWithPageCount
55	Single-step Processing Example
55	Base rules
56	Base form set rules
56	Base image rules
56	Base field rules
57	Using IDS to Run Documaker
58	Writing Unique Data into Recipient Batch Records
59	Optional formatting information
60	Example
62	BANNER.DAL
65	Using Class Recipients
67	Running Documaker Using XML Job Tickets
68	Handling 2-up Printing
68	2-up printing with single-page forms
69	2-up printing with multi-page forms
69	Changing the INI File
70	Creating the TWOUP control group
71	Creating the Added_Fonts control group
71	Changing the Recipient Batch DFD File
72	Rules Used for 2-up Printing
72	AddLine
72	AddTextLabel
72	ForceNoImages
72	GetRCBRec
72	InitMerge
72	InitPageBatchedJob
73	MergeAFP
73	ParseCommentExample
73	PrintData
73	ProcessRecord

	74	Placing the 2-up Rules in the JDT File
75		2-up Processing Example
	75	2upbycnt.bat
	75	2upstep1.ini
	75	2upstep2.ini
	75	2upstep3.ini
76		Running the GenData Program
	76	Step 1 - Using the AFGJOB1.JDT file
	77	Step 2 - Using the AFGJOB2.JDT file
	78	Step 3 - Using the AFGJOB3.JDT file
79		Splitting Recipient Batch Print Streams
	79	Splitting batches by sheet count
	80	Creating PDF output
	80	DAL functions
	81	DeviceName
	81	SetDeviceName
	81	BreakBatch
	82	UniqueString
82		Using DAL to Manipulate File Names
	83	FileDrive
	83	FilePath
	83	FileName
	84	FileExt
	84	FullFileName
85		Assigning Printer Types Per Logical Batch Printer
87		Controlling WIP Field Assignments
90		Generating Email Notifications from GenWIP
	91	Errors
93		Using Multi-mail Processing
	93	Changing the RCPDFDFL.DAT and TRNDFDFL.DAT Files
	94	Setting Up the FSISYS.INI File for Multi-mail Processing
96		Adding and Removing Pages
	96	Using Custom Code
	96	Adding pages
	96	Removing pages
	96	Using DAL Scripts
	96	Adding pages
	97	Removing pages

98	Using IDS
98	Adding pages
98	Removing pages
99	Adding Indexes and Tables of Contents
100	Using Run-Time Options
100	GenData Command Line Options
100	GenPrint Command Line Options
101	GenTrn Command Line Options
101	Debugging Options
102	Noting font IDs of zero
102	Suppressing elapsed runtime messages
103	Grouping Print Batches
104	Controlling Console Logging
104	Logging INI File Names and Options in the TRACE File
105	Listing the Rules Executed
106	Analyzing DAL Performance
108	When Extract Files Exceed The Offset Limits
109	Controlling What is in the MultiFilePrint Log
111	Using INI Built-In Functions
111	~GetEnv
112	~Platform
112	~OS
113	~DALRUN
	~DALVAR
113	~Encrypted
114	~ProcessID
115	~WIPField
115	Accessing WIP Fields
116	Formatting arguments
118	Specifying locales
119	Using the ~Field function
120	Outputting WIP Field Data Onto the XML Tree
121	Using XML Files
121	Handling Overflow
122	Triggering Forms and Sections
123	Using XPath
123	XPath Syntax

123	Axes
124	Symbols
124	Functions
125	Expressions
126	Using the XPath Testing Utility
126	Example 1
127	Example 2
127	Example 3
127	Example 4
127	Example 5
128	Example 6
128	Example 7
128	Example 8
129	Example 9
129	Example 10
130	Example XML File

Chapter 3, Implementing Your System

132	Why Use a Methodology?
133	Phase 1 - Define the requirements
133	Phase 2 - Create the detail forms requirements
133	Phase 3 - Build the Master Resource Library
133	Phase 4 - Install and configure the system
133	Phase 5 - Test the system
133	Phase 6 - Go live
134	Gathering Information
134	Understanding Your Niche
134	Understanding Your Organization
135	Roles and Responsibilities

Chapter 4, Setting Recipients and Copy Counts

138	Concepts
139	Key Files
139	Transaction Trigger Table
139	Trigger Levels

139	Form Set Definition Table
140	Trigger Table Record Format
142	Specifying the Transaction Trigger Table
143	How Transaction Triggering Works
144	Section Level Triggers
146	Form Level Triggers
149	Master and Subordinate Sections
149	Marking Subordinate Sections
150	Marking Master Forms
151	Examples
152	Specifying Copy Counts and Sections
153	FORM.DAT file
153	SETRCPTB.DAT file
153	POL file
154	Using Transaction Codes
154	FORM.DAT file
154	SETRCPTB.DAT file
154	POL file
155	Setting Up Search Mask and Sections
155	FORM.DAT file
155	SETRCPTB.DAT file
156	POL File
157	Using the RECIPIF Rule
157	FORM.DAT file
157	SETRCPTB.DAT file
158	POL file
159	Using Automatic Overflow
159	FORM.DAT file
159	SETRCPTB.DAT file
159	POL file
161	Using Forced Overflow
161	FORM.DAT file
161	SETRCPTB.DAT file
161	POL file
162	Setting Search Masks and Recipients
162	FORM.DAT file
162	SETRCPTB.DAT file
162	POL file

163	Using the Set Recipient Table and Extract Files
164	Formatting Search Masks
164	Spaces
164	Commas
164	Tildes
165	Parentheses
165	Using the OR condition
165	Using the NOT condition
165	Using AND and OR conditions
166	Sorting Forms by Recipient
167	INI files
167	Sort tables
168	Summary

Chapter 5, Working with Fonts

172	Understanding Font Concepts
172	Font Terminology
173	National language terminology
175	How Characters are Represented
175	Bitmap Fonts
175	Scalable Fonts
176	TrueType
176	PostScript
176	How Computers and Printers Use Fonts
178	Using Code Pages
179	ASCII Code Pages
181	EBCDIC Code Pages
182	Character Sets
182	Determining Characters Used in a Printer Font
184	Code Page Names
187	Types of Fonts
187	Using Screen Fonts
187	Font Substitution in Windows
188	Installing Screen Fonts in Windows
188	Using Printer Fonts
188	AFP
188	Coded fonts

188	Code pages
188	Character sets
188	Metacode
189	PCL
189	PostScript Fonts
189	TrueType Fonts
189	Adding Printer Fonts to a Font Cross-reference File
190	Using System Fonts
191	Font Cross-reference Files for Monotype Fonts
191	HPINTL.FXR, HPINTLSM.FXR
191	REL95.FXR, REL95SM.FXR
191	REL102.FXR, REL102SM.FXR
191	REL103.FXR, REL103SM.FXR
193	REL112.FXR
	REL112SM.FXR
194	Using Custom Fonts
195	On AFP printers
196	On Xerox Metacode printers
196	On PCL printers
196	On PostScript printers
197	Using Font Cross-Reference Files
198	How FXR Settings Affect Display and Print Quality
199	Maintaining FXR Files
199	Choosing a Font Cross-reference File
201	International Language Support
201	Using the ANSI Code Page for PC Platforms
202	Using Code Page 37 for EBCDIC Platforms
203	Using International Characters
204	Converting Text Files from one Code Page to Another
205	Setting Up PostScript Fonts
208	Fonts for PDF Files
208	Importing PostScript Symbol Fonts
210	Font Naming Conventions
211	Using Font Manager
211	Starting Font Manager
211	From Image Editor
211	From Docucreate
212	Working with the Font List

212	Selecting Fonts
213	Deselecting Fonts
213	Filtering the List of Fonts
214	Adding Fonts to a Font Set
216	Description tab
219	Dimensions tab
221	Printers tab
225	Other tab
225	Copying Font Information
226	Editing Font Information
227	Converting Fonts
229	Converting fonts from other vendors
229	Working with multiple printers
231	Deleting Fonts
232	Inserting Fonts
234	Inserting bitmap fonts and FXR files
235	Inserting PostScript and TrueType Fonts
238	Choosing Screen Fonts
240	Generating Files using Font Manager
240	Generating an FNT File
241	Generating an XRF File
241	Generating PFM Files from an FXR file
244	Mapping Fonts for File Conversions

Chapter 6, Setting Up Printers

246	AFP Printers
246	AFP INI Options
250	Using defaults for the Module and PrintFunc options
251	Using Documaker shading patterns instead of shaded bitmaps
251	Printing highlight colors
252	Character set and code page font information
252	Outputting character set and code page information
253	Using multiple code pages
256	Using LLE records to link to external documents
258	AFP Printer Resources
258	FormDef
258	Fonts

258	Monotype fonts
258	Overlays
258	Page segments
258	AFP 2-up support
259	AFP Troubleshooting
259	Floating section limitations
259	Objects extending beyond the edges
259	Conflicts between page and form orientation
260	Multi-page FAP limitation
260	Printing rotated variable fields
260	AFP 240 dpi print problems
262	Including Documerge Form-level Comment Records
263	Metacode Printers
263	Required JSL INI Options
264	JDLName
264	JDEName
265	DJDEIden, DJDEOffset, and DJDESkip
265	JDLCode
265	JDLData
265	JDLHost
266	Additional Required INI Options
266	OutMode
266	ImageOpt
266	CompressMode
267	CompileInStream
268	Device
268	RelativeScan
268	Specifying Installable Functions
268	Using defaults for the Module and PrintFunc options
269	Optional INI Options
269	Setting the end of the report
269	Starting new pages
270	Adding an OFFSET command
270	Logging pages
271	Specifying spot color
271	Chart performance and print quality
271	Optimizing Metacode print streams
272	Using a common font list
273	Setting a default paper size
273	Automatically sizing sections
274	Inline graphic performance and print quality

274	Adding color to charts
274	Using named paper trays
274	Specifying the printer model
274	Specifying the resolution
274	Displaying console messages
275	Stapling forms
276	Duplex switching
276	Using VSAM to store resources
276	PrintViewOnly
276	Caching files to improve performance
278	Using the loader
278	Using the Class option
279	Adding user-defined DJDE statements
279	Using third-party software to read Metacode files
279	Specifying the paper stock
281	Using Mobius Metacode Print Streams
282	Metacode Printer Resources
282	Fonts
282	Forms
282	Images
282	Logos
282	Metacode Limitations
282	Xerox images
282	HMI support
282	Changing the paper size on the 4235 printer
283	Xerox forms
283	Metacode Troubleshooting
283	Unexpected color output
283	Unexpected black and white output
284	Highlight color should match the PrinterInk option
284	LOG file orientation
284	Output catching up with the input
284	Printing rotated variables
285	Multi-page sections
286	Operator command, FEED, causes duplex problems
286	Line density errors
287	Output data length validation
287	Using Xerox Forms (FRMs)
288	BARRWRAP
288	Transferring Files from Xerox Format Floppies
289	PCL Printers

289	PCL INI Options
292	Using defaults for the Module and PrintFunc options
292	Using PCL 6
293	Printing Under Windows
294	Using High-Capacity Trays 3 and 4 on HP 5SI Printers
295	Using a staple attachment
296	Overriding Paper Size Commands and Tray Selections
297	Using Simple Color Mode
297	Marking objects to print in color
298	Specifying the highlight color to use
298	Printing on different types of printers
298	Creating Compressed PCL Files
298	Bitmap compression
299	Adding Printer Job Level Comments
299	Adding Data for Imaging Systems
300	Limiting the Embedded PCL Fonts
300	PCL Printer Resources
300	Fonts
300	Overlays
301	PostScript Printers
301	PostScript INI Options
304	Using defaults for the Module and PrintFunc options
304	Avoiding a white outline around letters
304	Printing under Windows
305	Generating PostScript Files on z/OS
305	Creating Smaller PostScript Output
305	Bitmap compression
306	Adding DSC Comments
308	Stapling Forms
310	PostScript Printer Resources
310	Fonts
310	Overlays
310	PostScript Printer Definition (PPD) Files
311	Using the GDI Print Driver
311	How it works
313	GDI Printer Driver INI Options
315	Using defaults for the Module and PrintFunc options
316	Avoiding Problems with FAX Drivers
316	Batch Printing to Files
318	Using Pass-through Printing

320	Creating PDF Files
321	Creating RTF Files
321	Generating separate files
322	Adding or removing frames
322	Creating form fields
323	Setting margins
323	Removing the contents of headers and footers
324	Using the VIPP Print Driver
325	VIPP Resource Files
325	Converting bitmaps into VIPP image files
326	Converting FAP files into VIPP segment files
327	VIPP fonts
328	VIPP font encoding files
329	Managing VIPP Resources
332	VIPP INI Options
335	Setting up folders and projects
336	Overriding the list of libraries for projects
337	Setting up paper trays
338	Adding DSC comments
340	VIPP Limitations
340	Troubleshooting
340	VIPP known problems
342	Emailing a Print File
342	Creating EPTLIB print files for Documaker Workstation
343	Creating EPTLIB print files for Documaker Server
345	Creating PDF print files
345	Overriding attached files
345	Using email aliases
346	Choosing the Paper Size
347	US Standard Sizes
348	ISO Sizes
348	ISO A sizes
349	ISO B sizes
350	ISO C sizes
351	Japanese Standard Sizes
352	Printer Support for Paper Sizes
356	Paper Sizes for AFP Printers
358	Creating Print Streams for Docusave
358	Archiving AFP Print Streams

- 359 Archiving Metacode Print Streams
- 360 Archiving PCL Print Streams
- 360 Using DAL Functions
- 362 Adding TLE Records
- 363 Handling Multiple Paper Trays
 - 363 For PCL printers
 - 363 For PostScript printers
 - 364 For GDI printers
 - 365 For AFP printers
 - 365 For Metacode printers
- 365 Including Tray Selections in a Print Stream Batch

Chapter 7, Setting Up Error Messages and Log Files

- 368 Overview
- 369 Configuring the Message System
 - 369 Enabling and Disabling Messages
 - 370 Logging INI Files and Options Used
 - 370 Clearing Messages
 - 370 Defining the Output Message Files
 - 371 Initializing the Output Message Files
 - 372 Turning Off Date Stamps
 - 372 Controlling the Translation Process
 - 373 DBLib Trace Messages
- 374 Creating Messages
 - 374 Using the RPErrProc and RPLogProc Functions
 - 374 RP Struct
 - 374 Message Types
 - 375 Message Number
 - 375 Assigning numbers to custom messages
 - 376 Using Message Tokens
 - 377 Setting Up Message Text
 - 378 Message examples
 - 378 Undefined tokens
 - 378 Adding a new line
 - 379 Determining where the message originated
- 380 Using the Message Token File

Chapter 8, Archiving and Retrieving Information

384 Terminology

384 Files and tables

384 Commit

384 Rollback

384 GenArc

384 AFEMAIN

384 CARFILE

384 APPIDX

384 TEMPIDX

385 CATALOG

385 RESTART

385 DFD

386 System Scenarios

386 Scenarios for OS/390 (MVS)

386 Scenarios for Windows 32-bit

387 Scenarios for UNIX

388 Archive and Retrieval Features

389 Processing Overview

389 DBASE IV

389 DB2

389 SQL server

389 Oracle

389 Files GenArc Uses

389 Input files

389 Output files

389 How the GenArc Program Works

392 Running GenArc

392 Logging archived transactions

392 Archiving to a database

393 Sorting records in a database

393 Preparing SQL

393 Command Line Options

393 INI

393 JOBID

393 DPASSWD

394 DUSERID

394 OPASSWD

	394	OUSERID
	394	RESTART
	394	SQLID
	394	STOPREC
	395	Using the Restart Option
397		Using GenArc with Documanage
	399	Forcing folder updates
	399	FSIUSER.INI sample
	401	APPIDX.DFD sample
	403	CARFILE.DFD sample
	404	Using the Oracle ODBC Driver
	404	CARFILE DFD
	406	Creating the Database and Tables
	409	Resolving Errors
410		Viewing Archives in Documanage
411		Using Multiple Simultaneous ODBC Connections
413		Using WIP and the Archive Index File
414		Formatting Archive Fields
	414	Converting the case of key fields
	414	Reformatting dates
	415	Storing a constant value
416		Retrieving Archived Forms
	416	Files the Archive Module Uses
	416	Input files
	416	Output files
	416	Using the Archive Module
	417	Retrieval Options
419		Working with Documanage
	420	Using Documanage Data Type Support
	421	Setting Up Automatic Category Overrides
	422	Mapping Documaker Archive Fields to Documanage Properties
	424	Using Next/Retrieve Cursor
	425	Enhanced Documanage Document Extended Properties Support

Appendix A, Setting Up Archive/Retrieval Configurations

434	DB2 Server on OS/390 — Windows Client
-----	---------------------------------------

434	Configuring the Server
434	Getting the DB2 location name and LUNAME
435	Defining the SNA server's APPC LU in VTAM
435	Defining the DB2 Application Major Node in VTAM
435	Setting Up the Windows 2000 Server (Middle Tier)
435	Installing and configuring Microsoft's SNA Server
437	Installing and Configuring Microsoft's SNA Server
438	Configuring SNA Server 4.0 SP3
440	Setting Up DB2 on a Windows 2000 Server
440	Installing DB2 on a Windows 2000 Server
440	Configure the DB2 instance
440	Defining an OS/390 node
441	Defining a system database entry
441	Updating TCP/IP values on the Windows 2000 server
441	Defining a database connection services entry
441	Installing and Configuring DB2 on a Windows 2000 Server
441	Defining an OS/390 system
441	Defining a DB2 instance
441	Defining an OS/390 database
441	Setting Up Universal Database on Windows 2000
441	Installing Universal Database
442	Configuring Universal Database
442	Updating TCP/IP-related Values on a Windows 2000 Server
443	Common DB2 Errors
443	Setting Up Clients
443	Defining a DB2/2000 node
444	Defining a system database entry
444	Updating TCP/IP-related values on a Windows client
444	Setting Up the INI Options for the DB2 Driver
446	DB2 Server on Windows — Windows Client
446	Setting up a DB2 Database on the Server
447	Setting Up a Client for DB2 VERSION 6.1
447	Archiving to a remote DB2 database using an ODBC driver
447	Setting up an ODBC data source
448	Setting up INI options for the ODBC driver
449	Archiving to a Remote DB2 Database Using the Native DB2 Driver
449	Setting up a DB2 database
449	Setting up the INI options for the DB2 driver
451	DB2 Server and Client on Windows
451	Setting Up a DB2 Database

451	Setting up an ODBC data source
451	Setting up INI options for ODBC
453	Archiving to a Local DB2 Database Using the Native DB2 Driver
453	Setting up the DB2 database
453	Setting up the INI options for the DB2 driver
455	SQL Server on Windows — ODBC Client on Windows
455	Setting Up a Client
455	Setting up the INI options for ODBC
457	IDS on Windows —DB2 Archive on z/OS
457	Setting Up the DB2 Archive on z/OS
458	Creating a z/OS Database
458	Updating TCP/IP Values on a Windows 2000 Server

Appendix B, System Files

462	Overview
464	Types of Files
464	BCH files
464	CAR files
464	DAT files
464	DBF files
464	DDT files
465	DFD files
465	Error files
465	Extract files
466	FAP files
466	Initialization files
466	JDT files
466	Log files
466	LOG files
466	MDX files
466	Transaction files
467	Resource Files
467	FSISYS.INI file
467	FSIUSER.INI file
467	FAPCOMP.INI
467	FORM.DAT file

	470	SETRCPTB.DAT file
	472	DFD files
	473	TRNDFDFL.DFD file
	473	RCBDFDFL.DFD file
	473	APPIDX.DFD
	474	.DDT files
	475	.JDT files
	475	Extract files
	477	DFD File Format
	477	Fields Group
	478	Field Description Group
480		Files Created by the GenTrn Program
	480	Transaction files
	480	Error files
	480	Log files
481		Files Created by the GenData Program
	481	NAFILE.DAT file
	481	POLFILE.DAT file
	482	NEWTRN.DAT file
	482	Batch files
	482	MANUAL.BCH file
	482	Error batch
	482	Updated log, error, and message files
483		Files Created by the GenPrint Program
	483	Spool files
	483	Updated log and error files
484		Files Created by the GenWIP Program
	484	WIP.DBF file
	484	WIP.MDX file
	484	oooooooo1.DAT file
	484	oooooooo1.POL file
	484	UNIQUE.DBF file
485		Files Used by the GenArc Program
	485	APPIDX.DBF file
	485	APPIDX.DFD file
	485	ARCHIVE.CAR file
	485	APPIDX.MDX file
	485	APPIDX.DFD file

Glossary

487 00000001.DAT File
487 00000001.POL File
488 AFP
488 ARCHIVE.CAR File
488 ARCHIVE.DBF File
488 ARCHIVE.DFD File
488 Base Product
488 .BCH Files
488 Batch Files
488 .CAR Files
489 Custom Solution
489 DAL
489 .DAT Files
489 .DBF Files
489 DDT Files
490 DESKJET.FXR File
490 .DFD Files
490 Distributed Resource Library
490 Duplex
490 ERRFILE.DAT
490 Error Batch
490 Error Files
491 External Database Editor
491 Extract Files
491 .FAP Files
491 FDB.DBF File
491 fetype
492 Field Database Editor
492 Fixed Data
492 Font Manager
492 Form
492 Form Set
492 Form Set Manager
493 FORM.DAT File
493 FSISYS.INI File

493 FSIUSER.INI File
493 .FXR Files
493 GenArc Program
494 GenData Program
494 GenPrint Program
494 GenTrn Program
494 GenWIP Program
494 Help Editor
495 Image (Section)
495 Image Editor
495 .INI Files
495 INTL.FXR
495 INTLSM.FXR
495 .JDT Files
495 Library Manager
496 Log Files
496 .LOG Files
496 Logo Manager
496 MANUAL.BCH File
496 Master Resource Library
496 Metacode
496 .MDX Files
496 NAFILE.DAT File
497 NEWTRN.DAT File
497 Objects
497 Overflow
497 Page
497 PCL
497 POLFILE.DAT File
497 PostScript
498 Section
498 SETRCPTB.DAT File
498 Simplex
498 System Releases
498 System Patches
498 Table Editor
498 Transaction List

498 .TRN Files
499 TRNDFDFL.DFD File
499 UFSTSM.FXR File
499 UNIQUE.DBF File
499 Variable Data
499 WIP.DBF File
499 WIP.MDX
499 xBase

501 Index

CHAPTER 1

Introduction

Welcome to the Documaker rules-based publishing solution. This product consists of a complete set of tools which provide solutions for all your form and document processing needs. The system includes these major components:

- Documaker Studio (and legacy Docucreate)
- Documaker Workstation
- Documaker Server

This manual serves as a reference to Documaker Server. This chapter discusses the following topics:

- [System Overview on page 2](#)
- [Rules Publishing Solution Overview on page 3](#)
- [Document Automation Evolution on page 4](#)
- [System Benefits on page 8](#)

SYSTEM OVERVIEW

Documaker Server is part of the Rules Publishing Solution, which also includes Documaker Studio, Documaker Workstation, and reusable resource libraries.

Documaker Server uses resources you create using Documaker Studio to process information and forms. This processing includes merging external data onto forms, processing data according to rules you set up, creating print-ready files, archiving data and forms, and, if applicable, sending incomplete forms to Documaker for completion by a user.

Forms can be completed using Documaker when user input is required or, if all of your information can be extracted from external data sources, you can set up Documaker Server to process forms without requiring user input.

Documaker Server can create print-ready files for a variety of printer languages including, AFP, PostScript, PCL, and Xerox Metacode printers. In addition, using Docupresentment, the system can produce output in Adobe Acrobat PDF format.

The following topic discusses the entire Rules Publishing Solution, its purpose, its underlying concepts and how it all works together to provide you with an enterprise-level solution to meet your document creation, processing, and storage needs.

RULES PUBLISHING SOLUTION OVERVIEW

Document automation is the basic concept underlying the system. An understanding of document automation helps you understand the purpose of the Rules Publishing Solution.

Document automation replaces paper documents with electronic media. Generally, document automation is an integrated process within enterprise information systems.

The greatest challenge that document intensive industries face is the efficient processing of forms and documents. Moving toward the era of electronic information means finding workable solutions for the paper-to-electronic media replacement process. New business directions include developing ways to automate document handling processes, which extend beyond simply creating electronic output or print.

Document automation is rapidly becoming an integral part of today's business environment. The Rules Publishing Solution creates a total business solution which lets you automate both paper document processing and electronic document management.

Let's examine document automation outside the Rules Publishing Solution to build a knowledge base applicable to unique platforms. Then we can apply the basic concepts to the Rules Publishing Solution.

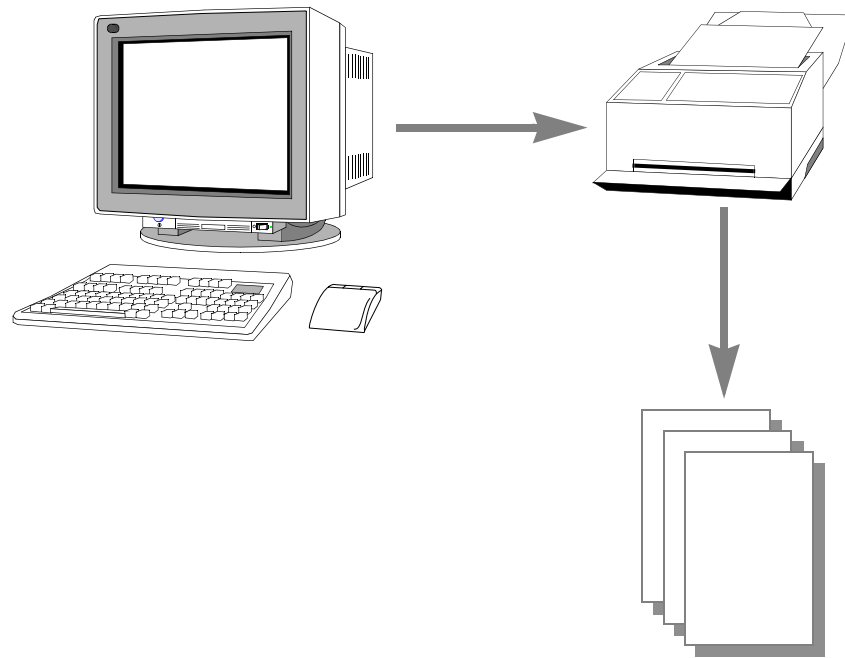
DOCUMENT AUTOMATION EVOLUTION

Through the years, document automation has moved in concert with technological evolution. The technological evolution has progressed from initial ideas and applications about forms processing, to the integrated management of electronic documents. The distinction between merely automating paper production and permanently integrating electronic processing and management is critical to understanding the technological evolution. This table shows the progression of document automation in the current environment.

Stage	Type of Automation	Components
1	Paper Automation	Business correspondence Forms processing Document assembly
2	Workflow Automation	Electronic mail Electronic data interchange Electronic funds transfer Integrated facsimile
3	Paperless Information Automation	Cooperative processing Enterprise indexing Integrated section processing Multimedia

Stage 1 - paper automation

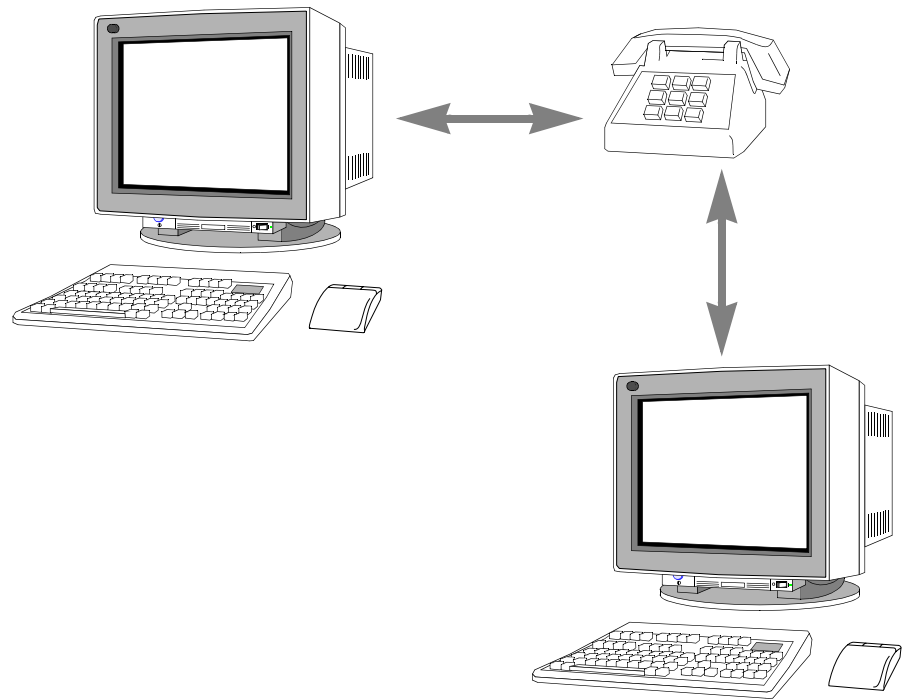
Paper automation, enabled by the advent of computers and laser printers, is the first stage of the document automation evolution. Most people think of the processing and assembly of business correspondence and forms by computers as document automation. While the computer does perform some information processing, this stage of document automation evolution is still very paper intensive. It does not extend to associated automated document workflow and procedures.



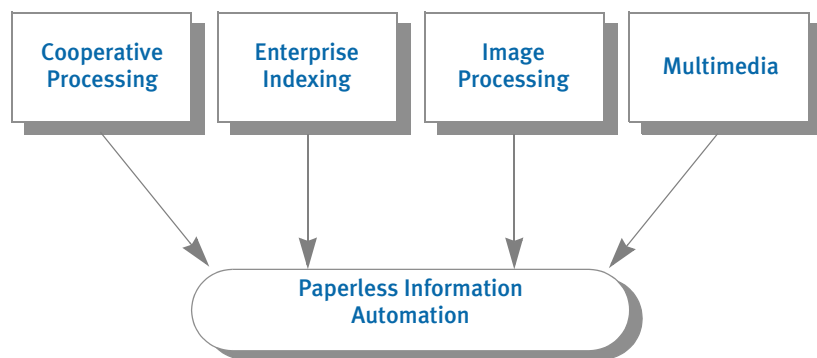
Stage 2 - workflow automation

Workflow automation, enabled by the proliferation of personal computers, communication standards, Local Area Networks (LANs), Wide Area Networks (WANs), and integrated FAX machines, is the second stage in the document automation evolution. Workflow automation goes beyond information processing to the transfer of digitized information across telecommunication lines. It eliminates many manual procedures, often clerical in nature, from the workflow process.

Stage 3 - paperless
information automation



Paperless information automation combines multiple technologies across multiple organizations, enterprises, and government entities. Information elements from various sources are shared and are readily available in flexible electronic formats. Paperless information automation enables you to reuse the information contained in the documents. Electronic documents are much easier to track, maintain, update, route, file, and retrieve.



DOCUMENT AUTOMATION GOALS

Document automation combines many elements of the evolutionary stages previously discussed to accomplish these primary objectives:

- **Eliminate paper**
Paper consumes enormous resources. Document automation decreases the costs associated with paper documents, and decreases the requirements for both long term and short term storage, retrieval, and document distribution.
- **Automate manual procedures**
Automating manual procedures associated with document automation increases efficiency, increases accuracy, and reduces costs. Repetitive and unnecessary procedures are identified and eliminated.
- **Automate system interfaces**
Interfaces which allow exchange of data between automated systems eliminate the need to manually enter data. Automated system interfaces also eliminate the need to supplement automated processes with manual functions. Automated system interfaces reduce errors, increase efficiency, and simplify the workflow.

As you can see, document automation encompasses many different technologies which merge in a variety of ways. In the current business environment, there are many single technologies and partial solutions which mimic document automation at first glance. Keep in mind, a single solution using one technology is not document automation. Document automation involves multiple technologies which help you manage forms and documents, workflow, procedures, and other electronic media, based on the needs and requirements of each individual organization or enterprise.

SYSTEM BENEFITS

The system's cohesive design results in many benefits to the user. The system provides a seamless interface to your existing systems by integrating document automation technology with your current systems, and by offering you a customized computer system with reusable resources. You can select modules to meet your specifications.

The system also provides you with the following advantages in your document automation processing:

- **Functional** - The system's configuration meets a wide variety of document processing needs. The system's expandable architecture utilizes technological innovations to meet changing processing needs.
- **Portable** - The system's architecture allows core processing modules to operate on multiple hardware platforms and in multiple operating environments. This design gives the user control of the system configuration in order to meet individual needs.
- **Modular** - The system's configuration lets you select modules to customize your system. The modular design eases maintenance by segregating functions in independent modules. A change in one module does not necessitate multiple changes throughout the system. This modular design also improves performance by eliminating unnecessary processing.
- **Reusable** - The biggest advantage in using the system is the reusability of resources. Libraries are composed of customizable resource units such as sections (sections) and rules, which can be reused. Reusing resources increases efficiency and promotes consistency throughout your system and product.
- **Easy to use** - System components have a graphical user interface common to all components. The system's seamless system interface provides transparent print and data merge capabilities.

CHAPTER 2

Understanding the System

In Chapter 1, you were introduced to the system as a whole. This chapter provides an overview of Documaker Server.

As you review this chapter you will learn about the programs that make up Documaker Server. Following the overview, you will learn about the files used and created by the system programs in both the multi- and single-step processes.

This chapter contains the following topics:

- [Processing Overview on page 11](#)
- [Processing Options on page 14](#)
- [Using Banner Processing on page 15](#)
- [Using Multi-step Processing on page 21](#)
- [Restarting the GenData Program on page 33](#)
- [Tracking Batch Page Statistics on page 37](#)
- [Generating Batch Status Emails on page 35](#)
- [Controlling GenTrn Processing on page 43](#)
- [Using Single-step Processing on page 45](#)
- [Using IDS to Run Documaker on page 57](#)
- [Writing Unique Data into Recipient Batch Records on page 58](#)
- [Using Class Recipients on page 65](#)
- [Running Documaker Using XML Job Tickets on page 67](#)
- [Handling 2-up Printing on page 68](#)
- [Splitting Recipient Batch Print Streams on page 79](#)

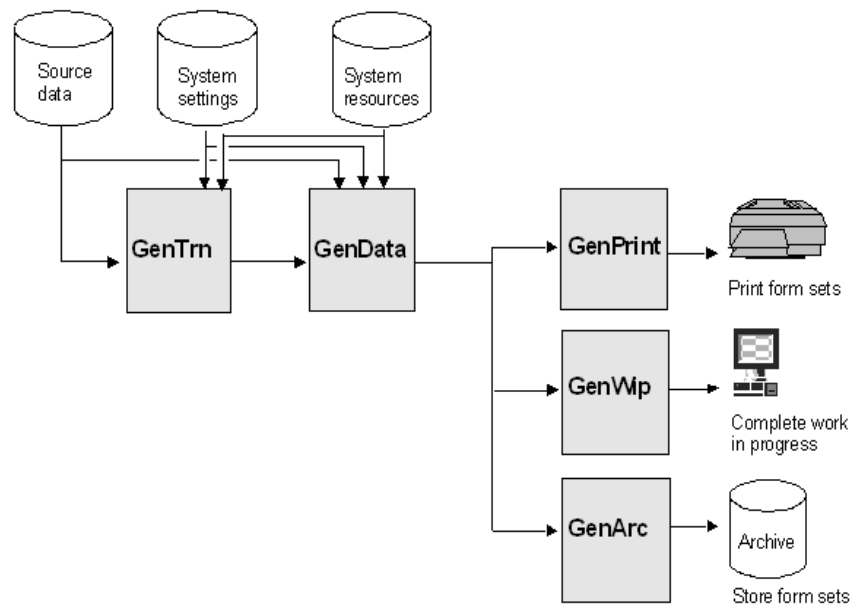
- [Assigning Printer Types Per Logical Batch Printer on page 85](#)
- [Controlling WIP Field Assignments on page 87](#)
- [Generating Email Notifications from GenWIP on page 90](#)
- [Using Multi-mail Processing on page 93](#)
- [Adding and Removing Pages on page 96](#)
- [Adding Indexes and Tables of Contents on page 99](#)
- [Using Run-Time Options on page 100](#)
- [Controlling What is in the MultiFilePrint Log on page 109](#)
- [Using INI Built-In Functions on page 111](#)
- [Outputting WIP Field Data Onto the XML Tree on page 120](#)
- [Using XML Files on page 121](#)
- [Using XPath on page 123](#)

PROCESSING OVERVIEW

Documaker Server is designed to gather source data, process that data by applying rules you define, merge the data onto pre-designed forms, and print the result. In addition, Documaker Server can automatically check for incomplete data and send that data to Documaker for completion. Documaker Server can also automatically archive completed transactions which you can later view as needed.

The following illustration shows a high level view of Documaker Server:

NOTE: This illustration and the other illustrations in this chapter show a typical, workstation-based system flow. Your system may be set up differently. Furthermore, the system can be customized in many ways and can run on a variety of platforms. For instance, if your source data is properly formatted, you can bypass the GenTrn program. Or, you may choose to run the GenTrn, GenData, and GenPrint programs on a host machine and then download the information and use a system utility (FIXOFFS) to prepare it for use by the GenWIP and GenArc programs running on a workstation. You could also run the GenArc program on the host and only run the GenWIP program on a workstation.



This illustration shows the main programs which make up Documaker Server and an overall view of the processing cycle.

- **GenTrn.** The GenTrn program reads source data and uses system settings to create transaction records. The source data is stored in extract files. Depending on the operating system you use, this program has various names such as *GENTNW32.EXE* for 32-bit Windows environments.
- **GenData.** The GenData program takes the transaction records created by the GenTrn program and uses system settings and resources to apply processing rules to those transactions.

The GenData program creates output files the GenPrint program can use. It also creates files with incomplete transactions which the GenWIP program can use. The GenWIP program creates from these files, output you can display and complete using the WIP module of Documaker Workstation.

The output from the GenData program is also used by the GenArc program to archive data. Depending on the operating system you use, this program has various names such as *GENDAW32.EXE* for 32-bit Windows environments.

NOTE: The illustration on the preceding page and this overview discuss the standard or multi-step processing flow of the system. By using specific rules you can have the GenData program execute both the functions of GenTrn and GenPrint. This is called *single-step processing* and can improve performance. To learn more, see [Using Single-step Processing on page 45](#).

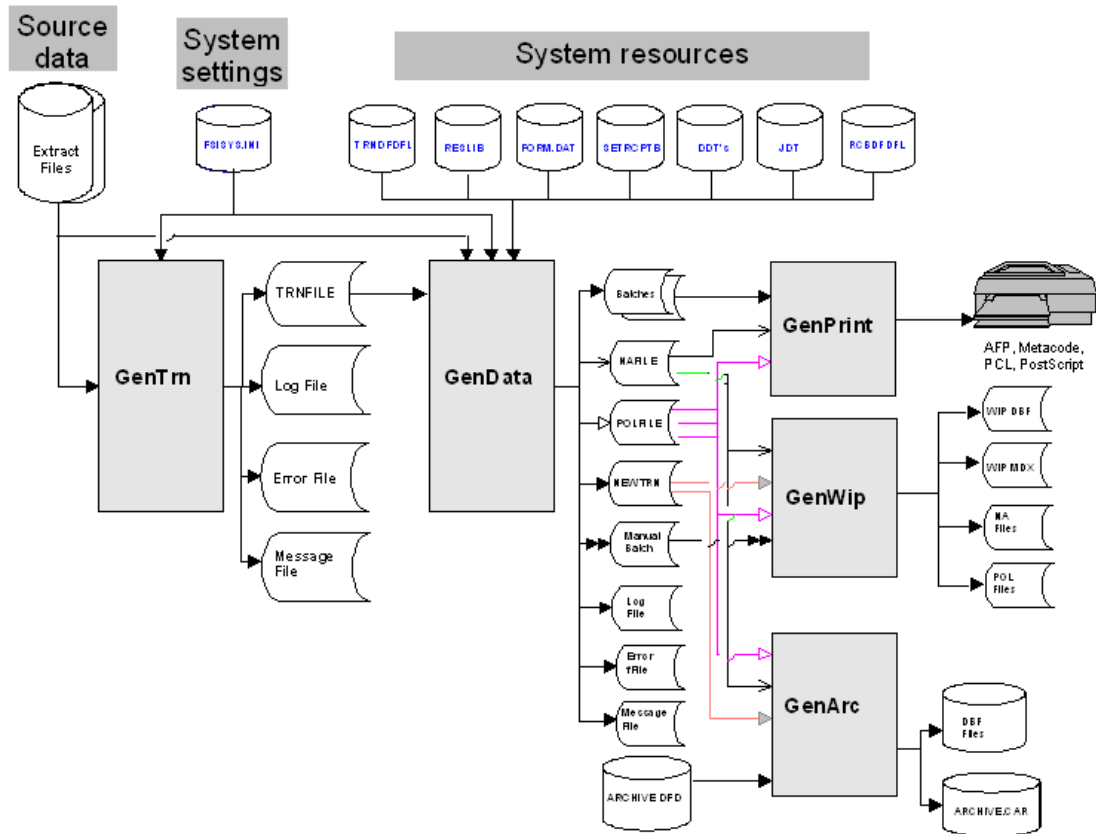
- GenPrint. The GenPrint program takes information produced by the GenData program and creates printer spool files for use with PCL, AFP, Metacode, and PostScript compatible printers. In addition, If you have purchased Docupresentment (IDS), the GenPrint program can also produce a Portable Document File or PDF (Acrobat) output. Depending on the operating system, this program has various names such as *GENPTW32.EXE* for 32-bit Windows environments.
- GenWIP. The GenWIP program receives information about incomplete transactions from the GenData program and processes that information so you can use the WIP module of Documaker to display the form and fill in the missing information. Once completed, you can print, archive, print and archive, delete, or change the status of form sets using Documaker. Depending on the operating system, this program has various names such as *GENWPW32.EXE* for 32-bit Windows environments.

NOTE: When using Documaker Server, a transaction may be placed in WIP for completion by a data entry operator. In these cases, you would first complete the transaction before it is archived.

- GenArc. The GenArc program archives data so you can store the information efficiently and retrieve it quickly. This program receives information from the GenData program. Depending on the operating system, this program has various names such as *GENACW32.EXE* for 32-bit Windows environments.

The previous illustration showed a high level view of Documaker Server which shows you the main programs in the system and its processing cycle. These programs create and use several types of files as they process information. The following illustration shows this processing flow in greater detail, though not every possible system file is included.

Understanding how the information flows from one program to another and which files are used and created is key to understanding Documaker Server. Here you can see all of the files the system uses and creates during its processing cycle.



You can find information about all these files and programs in the Glossary. You can also find examples of certain files in Appendix B, [System Files on page 461](#). Let's first look at the GenTrn program and the files it uses and creates.

NOTE: You can run the GenData and GenPrint programs on z/OS (MVS) using resources retrieved from Documanager (on a Windows server) via Library Manager. For information on setting up the library in Documanager and setting the INI options on z/OS to access this library, refer to the [Documaker Studio User Guide](#). See Using Documanager in Chapter 9, Managing Resources.

PROCESSING OPTIONS

You can run Documaker Server as a multi- or single-step process. Variations of these processes provide additional options such as AFP 2-up printing and multi-mail sorting.

Chapter 2 begins with a general overview of the system. From this point forward, we will review specific processing options. The following topic discusses running the system using the multi-step process. This topic is followed by a discussion of running the system using the single-step processes. The remainder of the chapter provides brief explanations of 2-up and multi-mail printing.

NOTE: To gain a complete understanding of the different features of the multi- and single-step processes, it is important to read through both sections. Certain information that is common to both processes is only described in the multi-step section.

To help determine which option is best suited for a particular need, a brief description of the run-time options and related processes are provided in the table below:

Process	Description
2-Up Printing	Two-up printing is a two-step process which passes input through GenData three (3) times with a different JDT file each pass. This process is similar to the single-step process in that GenData performs the work, but the three passes through GenData actually represent two steps of the multi-step process: processing the transactions and printing the transactions. Two-up printing is AFP printer-specific. For more information, see Handling 2-up Printing on page 68 .
Banner	The system lets you process banners at several points in the processing cycle. Doing this involves using a simplified AFGJOB.JDT file. For more information, see Using Banner Processing on page 15 .
Multi-mail	GenData groups transactions with the same multi-mail code into selected print batches to be sorted and delivered to the same location. For more information, see Using Multi-mail Processing on page 93 .
Multi-step	The system programs, GenTrn, GenData and GenPrint, each perform a set of steps to read data, create output files and print. GenWIP and GenArc are optional programs to complete incomplete transactions and archive data for retrieval. For more information, see Using Multi-step Processing on page 21 .
Restarting the system	You can set up the GenData program to restart itself at a particular transaction if it encounters a failure. For more information, see Restarting the GenData Program on page 33 .
Single-step	To enhance system performance, the steps of the GenTrn, GenData and GenPrint programs are performed in one step by GenData. The GenWIP and GenArc programs function the same as in the multi-step process. For more information, see Using Single-step Processing on page 45 .

USING BANNER PROCESSING

The system includes support for banner processing. Banner processing is supported at these points in the processing cycle:

- Beginning of a batch
 - Before a transaction is processed
 - After a transaction is processed
- End of a batch

Banner processing is optional at each point. Banner processing can optionally include FAP forms processing and DAL script processing.

You specify the FAP forms for banner processing in this manner:

```
;key1;key2;form name;
```

The forms must appear in the FORM.DAT file in DefLib. The associated sections (images) for those forms and must reside in FormLib.

You can set up banner forms and scripts at a global level so they can be used by all print batches. Individual recipient print batches can specify local forms or scripts to override the global forms and scripts.

Keep in mind these limitations:

- This enhancement only affects the GenPrint program. Documaker Workstation has a separate banner handling method, and does not support this method of banner processing.
- Only the standard printer drivers, such as AFP, Metacode, PCL, and Postscript, support batch banner processing. Avoid batch banner processing if you are using another print driver.
- Banner pages are printed at the group level. As a result, this bypasses the custom callback function named in the CallbackFunc option of the Print control group since it is a form set-level callback.

NOTE: Version 10.1 added batch-level banner processing to multi-step mode. Version 10.2 added batch-level banner processing to single-step processing — printing via GenData using the PrintFormset rule.

Enabling banner processing

For performance reasons banner processing is, by default, disabled. You must enable it using one or both of these INI options:

```
< Printer >
  EnableTransBanner = True
  EnableBatchBanner = True
```

Omitting either option disables the associated level of batch banner processing. Once enabled, banner processing is in effect for the entire GenPrint run. You can, however, disable banner processing for individual batches by specifying forms and scripts with blank names.

Specifying banner forms and scripts

You can globally specify forms and scripts for all batches, or locally for specific batches. Use these INI options to specify global batch forms and scripts:

```
< Printer >
```

```
BatchBannerBeginForm    = form name
BatchBannerBeginScript = script name
BatchBannerEndForm      = form name
BatchBannerEndScript    = script name
TransBannerBeginForm    = form name
TransBannerBeginScript = script name
TransBannerEndForm      = form name
TransBannerEndScript    = script name
```

Specify form names as follows:

```
;KEY1;KEY2;Form name;
```

You must have an associated form line in the FORM.DAT file to match the specified form. The sections (FAP files) for the forms are specified in the form lines in the FORM.DAT file. You must include these FAP files in FormLib.

Store the banner forms in a separate and unique banner form group, defined by a combination of *Key1* and *Key2*. You can use the AddForm DAL function in a DAL script to insert additional forms for banner processing. Place these additional forms and sections in the same group as the initial banner form. Each form is printed separately and after all banner forms are printed, the entire banner group is removed from the document set. For these reasons, it is critical that you isolate the banner forms from the rest of the transaction document set by specifying a *Key1/Key2* combination that does not otherwise occur within the document.

The FAP files assigned to the form (on the form line in the FORM.DAT file) must have the recipient *BANNER* with a copy count of at least one. When banner forms are printed, only sections assigned to the recipient *BANNER* with a non-zero copy count are printed.

Specify the DAL script names without a path or extension. For best results, store the DAL scripts in your DAL libraries because they are easier to maintain. The system automatically loads DAL libraries if you include these INI options:

```
< DALLibraries >
LIB = library1
LIB = library2
```

The DAL script libraries or files must reside in DefLib.

You can specify forms and scripts at the recipient batch level to override the global specification. Here is an example of how you do this:

```
< Print_Batches >
  BATCH1 = BATCH1.BCH
  BATCH2 = BATCH2.BCH
< Batch1 >
  BatchBannerBeginForm    = form name
  BatchBannerBeginScript = script name
  BatchBannerEndForm      = form name
  BatchBannerEndScript    = script name
  TransBannerBeginForm    = form name
  TransBannerBeginScript = script name
  TransBannerEndForm      = form name
  TransBannerEndScript    = script name
```

You can specify some, none, or all of the forms and scripts for local override of the default global forms and scripts.

An individual batch can completely or partially disable banner processing if the forms, script names, or both are blank, as shown here:

```
< Batch1 >
  BatchBannerBeginForm    =
  BatchBannerBeginScript =
  BatchBannerEndForm      =
  BatchBannerEndScript   =
  TransBannerBeginForm    =
  TransBannerBeginScript =
  TransBannerEndForm      =
  TransBannerEndScript   =
```

Banner form processing and multi-file print

Use the RetainTransBeginForm option to make pre-transaction transaction banner form processing compatible with multi-file printing. Banner forms print separately from the rest of the document. When using multi-file printing with print drivers such as PDF or RTF, banner forms do not appear in the output file. This options lets the banner form appear in the same print file.

Banner pages are, by design, not considered part of the form set. A pre-transaction banner page is designed to print separately, using data from the form set, but as if it were not physically part of the form set. For that reason, when *printing* to a single-file-per-transaction format such as PDF, RTF, XML, or HTML, and using the MultiFile print callback method to produce separate files, the banner output is not included in the output file.

It is possible to use pre-transaction banner forms as a way of producing a mailer sheet for a form set. This works for true printed output, but if you are producing a PDF file, for example, the banner (mailer page) does not appear within the PDF.

If, however, you use the RetainTransBeginForm option to retain the pre-transaction banner form, the banner process proceeds as before, but the printing of the banner is initially suppressed. The banner page is retained and remains inside the form set, as the first form in the form set. When the form set is processed by the PDF driver to produce the PDF file, the pre-transaction banner form (or mailer sheet) is then included in the resulting PDF file.

Keep in mind however that the document is only temporarily modified during the print step. The banner form is not included with the actual, intelligent form set when it is archived. For instance, if the intelligent document format is used for archiving, the mailer sheet does not appear as part of the form set, and will not print if retrieved from archive. If, however, you archive the PDF output, then the mailer sheet will appear in the PDF file.

You can place the RetainTransBeginForm option in the Printer control group as a global setting or you can place it at the recipient batch level. A setting at the recipient batch level overrides a setting in the Printer control group.

Here is an example of how you could set a global or default setting in the Printer control group and override that setting for a particular recipient batch:

```
< Printer >
  RetainTransBeginForm = Yes
  ... (other applicable options omitted - see the following note)

< Print_Batches >
  Batch1 = BATCH1.BCH
  Batch2 = BATCH2.BCH
```

```
< Batch1 >  
  RetainTransBeginForm = No  
  ... (other applicable options omitted - see the following note)
```

Option	Description
RetainTransBeginForm	Enter Yes if you want the system to include the transaction banner form in the form set. The default is No. If you are using the PDF, RTF, XML, or HTML print driver, this means the banner pages will be included in each transaction's print file.

NOTE: There are additional INI settings required for single- and multi-step processing. For more information about single-step processing, see the discussion of the PrintFormset rule in the [Rules Reference](#).

For more information about multi-step processing, see the discussion of the MultiFilePrint callback function in the [Using the PDF Print Driver](#).

Processing logic

Banner processing functions are part of the base system and are primarily located in GenLib. The GenPrint program, however, first routes the processing to CusLib. This lets you use the exit points in CusLib to create additional customized processing before, after, or in place of, the calls to GenLib routines.

The processing sequence for banner processing (at any level) is as follows:

- 1 If a banner form is specified, it is created in the form set and the FAP files are loaded.
- 2 If a banner DAL script is specified, it is executed.
- 3 For any banner form specified in step 1 or created during step 2, the following steps take place:
 - any variable fields in the banner form that are still empty are updated, first from matching GVM variables, such as fields in the recipient batch record, then from matching DAL variables.
 - the form is printed.
- 4 If there were banner forms to process, after updating the fields and printing the forms, the entire banner form group is removed from the form set.

NOTE: You can suppress the printing of the banner page by using the SuppressBanner DAL function. This is useful when you need to combine several transactions within the same transaction banner pages.

If there are registered *comment record* functions, each banner form in the form group receives its own set of comment records. If the additional forms should not receive their own comment records, add the sections for those forms to the original form—do not add them as separate forms.

DAL functions

You can also use these DAL functions with banner processing. See the [DAL Reference](#) for more information.

- **RecipName.** Returns the name, such as INSURED, AGENT, COMPANY, and so on, of the recipient batch record of the transaction currently being printed.
- **RecipBatch.** Returns the name, such as BATCH1, BATCH2, ERROR, MANUAL, and so on, of the recipient batch file being processed.
- **SuppressBanner.** Suppresses the current banner from printing. You can use this function when you want to combine several transactions inside one set of banner pages, based on a flag that the DAL script checks.

Banner processing example

Assume you have these FAP files in your forms library (FormLib).

- btchbannr
- btctrail
- trnbannr
- trntrail

Here is an excerpt from the FSISYS.INI file:

```
< Printer >
  PrtType = PCL
  EnableTransBanner = TRUE
  EnableBatchBanner = TRUE
  BatchBannerBeginScript = PreBatch
  TransBannerBeginScript = PreTrans
  BatchBannerEndScript = PstBatch
  TransBannerEndScript = PstTrans
  BatchBannerBeginForm = ;BANNER;BATCH;BATCH BANNER;
  BatchBannerEndForm = ;BANNER;BATCH;BATCH TRAILER;
  TransBannerBeginForm = ;BANNER;TRANSACTION;TRANS HEADER;
  TransBannerEndForm = ;BANNER;TRANSACTION;TRANS TRAILER;
< DALLibraries >
  LIB = Banner
```

Here is an excerpt from the FORM.DAT file:

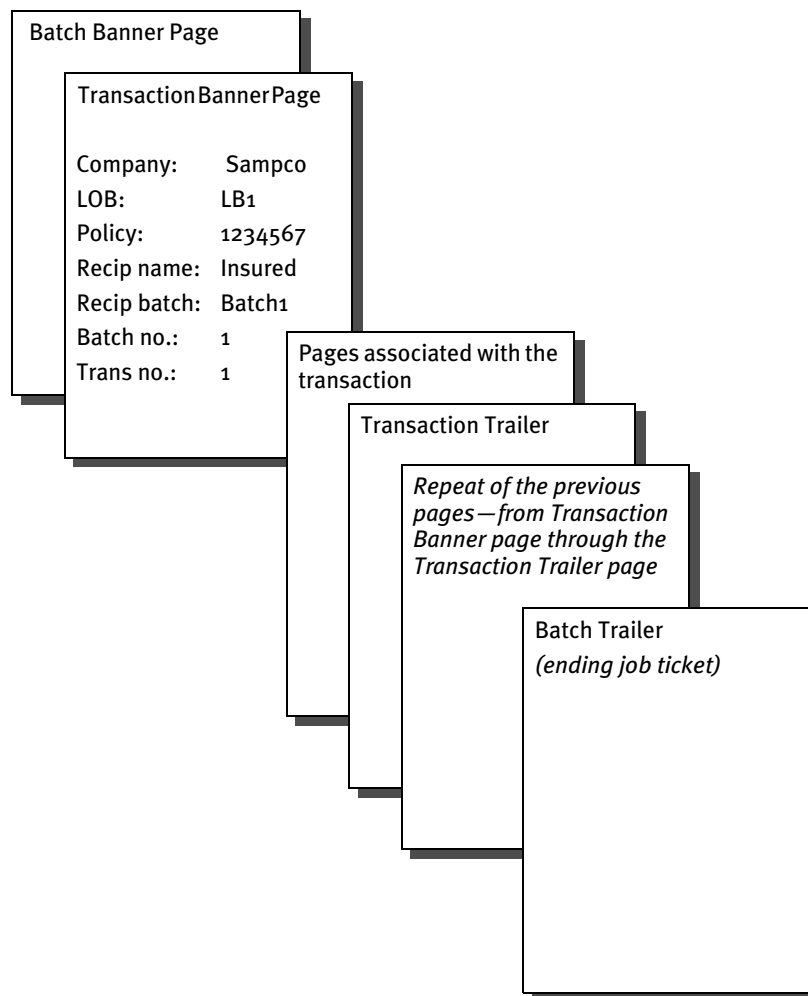
```
;BANNER;BATCH;Batch Banner;Batch Banner (Job\
Ticker);N;;btcbannr|D<BANNER(1)>;
;BANNER;BATCH;Batch Trailer;Batch Trailer (End\
Ticket);N;;btctrail|<BANNER(1)>;
;BANNER;TRANSACTION;Trans Trailer;Transaction Trailer (End\
Ticket);N;;trntrail|D<BANNER(1)>;
;BANNER;TRANSACTION;Trans Header;Transaction Banner\
Page;N;;trnbannr|D<BANNER(1)>;
```

Here is an example of the BANNER.DAL file in DefLib:

```
BeginSub PreBatch
    #batch += 1
    #trans = 0
    rb = RecipBatch()
    rn = RecipName()
EndSub

BeginSub PreTrans
    #trans += 1
    rb = RecipBatch()
    rn = RecipName()
EndSub
```

These additions to the FORM.DAT and FSISYS.INI files plus file additions to the FormLib and DefLib sub-directory would cause the following pages to be added to each batch:



USING MULTI-STEP PROCESSING

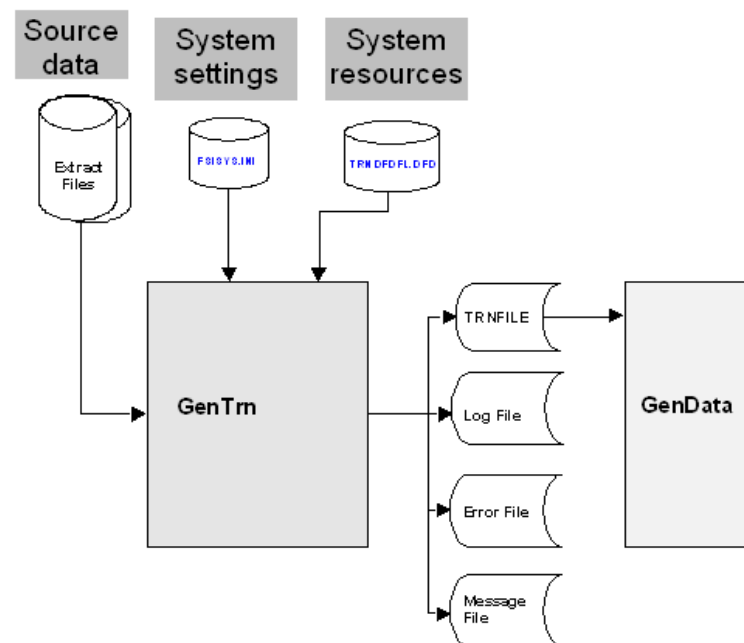
This topic describes the standard, multi-step approach to processing. In a multi-step processing scenario, the system takes these steps:

- Create the transaction records
- Process the transactions
- Create print spool files
- Send incomplete transactions to work-in-progress (WIP)
- Archive transactions

NOTE: Be sure to carefully read this topic even if you are using single-step processing.

CREATING TRANSACTION RECORDS

This illustration shows the files used and created by the GenTrn program as it creates transaction records:



The GenTrn program takes the source data, which is stored in extract files, and creates a list of the transactions, which is stored in the TRNFILE, or transaction file. This transaction list is then used by the GenData program as it processes the transactions.

The GenTrn program uses settings in the FSISYS.INI and TRNDFD.FL.DFD files to determine how to process the transactions. These files provide the GenTrn program with information about the format and structure of the extract file, such as how to determine where each new record starts.

The GenTrn program also produces a log file of its activities, a message file, and an error file which you can use to resolve any errors that occur.

File Summary

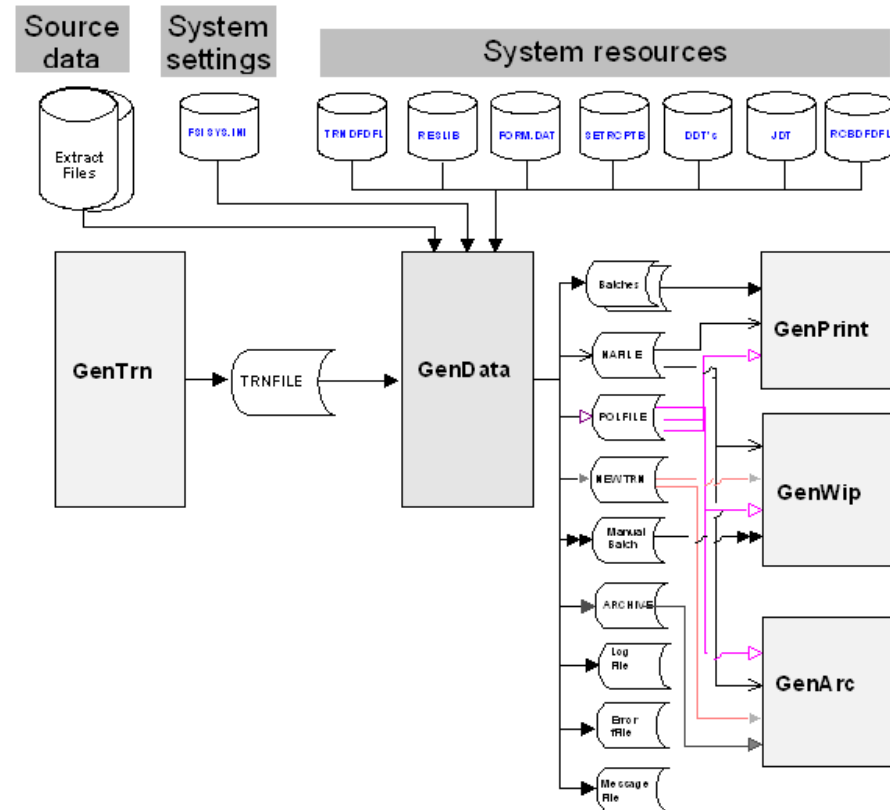
This table summarizes the files used to supply information (input) and the files created by (output) the GenTrn program:

NOTE: You can use the Data control group in the FSISYS.INI file to specify the names and extensions for all other input and all of the output files.

	File Name or Type	Default Extension	File Format	Description
Input	Extract files	.DAT	text	Contains the data you want to process.
	FSISYS	.INI	text	Initialization file which includes system settings.
	TRNDFDFL	.DFD	text	Defines the attributes of the variable fields in the TRNFILE.DAT file.
Output	TRNFILE	.DAT	text	Serves as an index to the individual transactions. Used by the GenData program as it processes the source data in the extract file.
	Log file	.DAT	text	Serves as a processing log for the GenTrn program. The system records the information by transaction.
	Error file	.DAT	text	Notes any errors and warnings encountered by the GenTrn program as it created the TRNFILE.DAT file. The system records the information by transaction.
	Message file	.DAT	text	Contains errors and warnings.

PROCESSING TRANSACTIONS

The following illustration shows the files used and created by the GenData program as it processes transactions:



The GenData program uses the transaction list (TRNFILE) created by the GenTrn program as it processes the source data stored in the extract files. The FSISYS.INI file provides system setting information, such as whether or not it should stop processing if it encounters errors, how to identify key fields in extract files, whether or not it should check the output data size against the defined field length, and so on.

The files listed under *System resources* provide additional information such as:

- How to read the transaction file (TRNDFDFL.DFD)
- The forms, logos, and other resources to use when creating the form sets (RESLIB)
- What forms to use (FORM.DAT)
- Who to send the forms to (SETRCPTB.DAT)
- What processing rules to apply to the data (DDTs)
- What processing rules to apply to this job (JDTs)
- How the batch files are defined (RCBDFDFL.DFD)

NOTE: You can learn more about these files in Appendix B, [System Files on page 461](#).

Output Files for GenPrint

The output files created by the GenData program include three types of files used by the GenPrint program: Batch files, NAFILES, and POLFILES. Batch files list the transactions which should be included in each batch print job. NAFILES store section and variable field information. POLFILES define the form set the GenPrint program should use for each transaction it processes.

Output Files for GenWIP

The GenWIP program also uses the NAFILE and POLFILE to store section and variable field information and to define the form sets. In addition, the GenData program creates manual batch files specifically for the GenWIP program.

The GenData program creates manual batch files if it is unable to complete the processing of a form set. Typically, this occurs if the form set is missing information. The GenWIP program uses this file to create separate transactions which can then be completed manually using the Entry module of Documaker Workstation. The data for the separate transactions are stored in files with the extension *DAT*, such as 00000001.DAT, 00000002.DAT, and so on.

Output Files for GenArc

The GenArc program also uses the NAFILE and POLFILE to store section and variable field information and to define the form sets. In addition, the GenArc program uses the NEWTRN files to tell it where to find data in the NAFILES and which forms to use in the POLFILES.

File Summary

This table summarizes the files used to supply information (input) and the files created by (output) the GenData program:

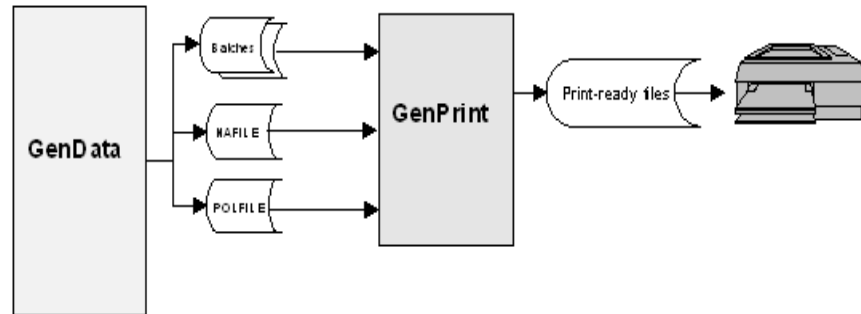
NOTE: You can use the Data control group in the FSISYS.INI file to specify the names and extensions for all other input and all of the output files.

	File name or Type	Default Extension	File Format	Description
Input	Extract files		text	Contains the data you want to process.
	FSISYS	INI	text	Initialization file which includes system settings.
	TRNFILE	DAT	text	Used as an index to the individual transactions stored in the extract file.
	TRNDFDFL	DFD	text	Tells GenData how to read the TRNFILE.
	FORM	DAT	text	Defines the forms in a form set.
	SETRCPTB	DAT	text	Defines the recipients of a form set.
	DDT files	DDT	text	Contains the rules GenData applies to the data.
	JDT files	JDT	text	Contains the rules GenData follows when processing the job.
	RCBDFDFL	DFD	text	Defines the attributes of the variable fields in a batch file.
	Resources	(various)	(various)	Includes logos (.LOG), font cross reference files (.FXR), sections (.FAP), and so on.
Output	Batch files	BCH	text	Indicates which transactions should be included in a given batch job. Used by the GenPrint program.
	NAFILE	DAT	text	Contains section and variable field information. Used by the GenPrint, GenWIP, and GenArc programs.
	POLFILE	DAT	text	Defines the forms to use for each batch. Used by the GenPrint, GenWIP, and GenArc programs.
	NEWTRN	DAT	text	Tells the GenArc program where to find data in the NAFILE and which forms to use in the POLFILE.

File name or Type	Default Extension	File Format	Description
Manual batch files	BCH	text	Created if the form is incomplete. Used by GenWIP to allow an operator to complete the form in the Entry module of Documaker.
Error batch files	.BCH	text	Created if the system spots an error, such as if the system spots an error and the form is marked as host required. In contrast to manual batch files, you cannot correct these errors using the GenWIP program. Instead, you must correct the error in the extract file, change the flag to operator required, or change the FAP file and then process the transaction again.
ARCHIVE	DFD	text	Tells the GenArc program how to store archived data.
Log file	DAT	text	Serves as a processing log. Created by the GenTrn program, the GenData program adds information to this file.
Error file	DAT	text	Notes any errors encountered by the GenData program. Created by the GenTrn program, the GenData program adds information to this file (as do the GenPrint, GenWIP, and GenArc programs).
Message file	.DAT	text	Contains errors and warnings.

CREATING PRINT SPOOL FILES

The following illustration shows the files used and created by the GenPrint program as it creates print-ready files:



The GenPrint program receives batch files from the GenData program which tell it what transactions to print, NAFILEs which tell it what data to print, and POLFILEs which tell it which forms to print.

With this information, the GenPrint program creates print-ready files for AFP, Xerox Metacode, PCL, or PostScript compatible printers. The GenPrint program serves as the print engine for the system.

NOTE: In addition, the GenPrint program can also create PDF (Acrobat) if you have purchased the PDF Print Driver. For more information about this product, contact your sales representative.

File Summary

This table summarizes the files used to supply information (input) and the files created by (output) the GenPrint program:

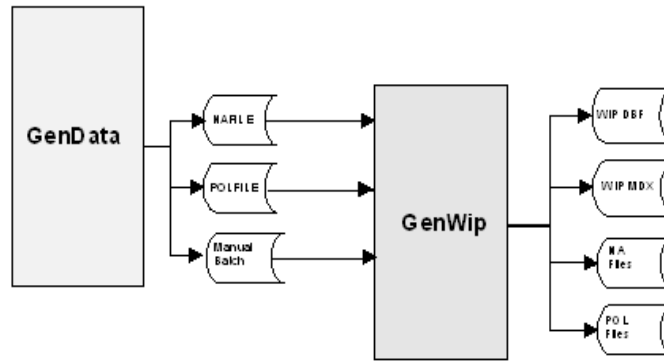
NOTE: You can use the Data control group in the FSISYS.INI file to specify the names and extensions for all other input and all of the output files.

	File name or Type	Default Extension	File Format	Description
Input	Batch files	BCH	text	Indicates which transactions should be printed in a given batch. Used as trigger files by the GenPrint program.
	NAFILE	DAT	text	Contains section and variable field information.
	POLFILE	DAT	text	Defines the forms to use for each batch.
	RCBDFDFL	DFD	text	Defines the attributes of the variable fields in a batch file.
Output	Print-ready files	AFP, PCL, XER, PST, PDF*	AFP, PCL, MetaCode, PostScript, or PDF*	Printer spool files which can be printed on the printer of your choice.

* To produce PDF files, you must also have the PDF Print Driver.

SENDING INCOMPLETE TRANSACTIONS TO WIP

The following illustration shows the files used and created by the GenWIP program as it processes incomplete transactions:



The GenWIP program receives information from the GenData program about incomplete transactions the GenData program found during its processing cycle. With this information, the GenWIP program creates files the WIP module of Documaker can read. Through the WIP module, data entry operators can complete the transactions by entering the missing information.

The manual batch file tells the GenWIP program which transactions are incomplete and should be included in work-in-progress (WIP).

Using the information in the manual batch files, the GenWIP program extracts the information it needs from the NAFILE and POLFILE. With this information, it then creates individual NA and POL files for each incomplete transaction. The GenWIP also creates a WIP.DBF (database) file which contains information about the incomplete transactions. The WIP.MDX file serves as an index to this file. Both the WIP.DBF and WIP.MDX files are used by the WIP module of Documaker Workstation.

File Summary

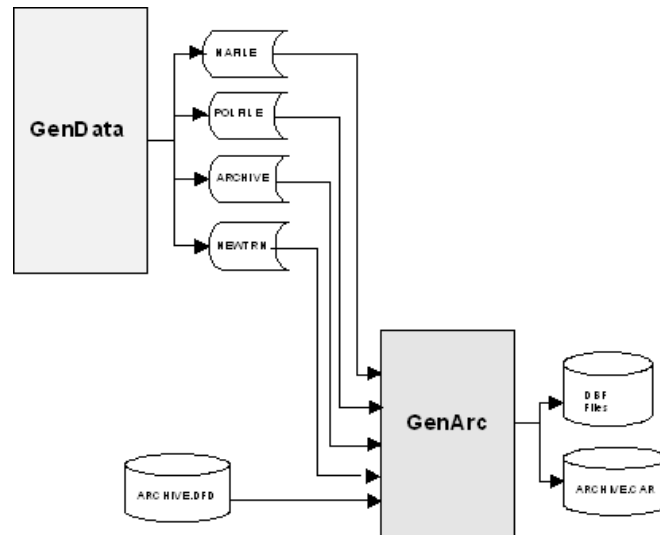
This table summarizes the files used to supply information (input) and the files created by (output) the GenWIP program:

NOTE: You can use the Data control group in the FSISYS.INI file to specify the names and extensions for all other input and all of the output files.

	File Name or Type	Default Extension	File Format	Description
Input	NAFILE	DAT	text	Contains section and variable field information.
	POLFILE	DAT	text	Defines the forms to use for each batch.
	RCBDFDFL	DFD	text	Defines the attributes of the variable fields in the batch files.
	Manual batch	BCH	text	Indicates which transactions should be included.
Output	WIP	DBF		Contains information about the incomplete transactions extracted from the NAFILE and POLFILE.
	WIP	MDX		Serves as an index to the WIP.DBF file.
	NA Files	DAT	text	Contains the data (section and variable field information) for a specific transaction. These files are named numerically and each file has a corresponding POL file.
	POL Files	POL	text	Defines the forms to use for a specific transaction. These files are named numerically and each file has a corresponding NA file.

ARCHIVING TRANSACTIONS

The following illustration shows the files used and created by the GenArc program as it archives completed transactions:



The GenArc program receives information from the GenData program, using many of the same files used by the GenWIP and GenPrint programs, such as the NAFILE and POLFILE. These two files identify the data to archive. The NEWTRN file tells the GenArc program where to find data in the NAFILE, which is created by the GenArc program.

In addition, the GenArc program also uses the ARCHIVE.DFD file which tells it how to store the data.

With this information, the GenArc program creates DBF and CAR files. The DBF files serve as an index to the CAR files, where the archived information is actually stored. You can have multiple CAR files.

File Summary

This table summarizes the files used to supply information (input) and the files created by (output) the GenArc program:

NOTE: You can use the Data control group in the FSISYS.INI file to specify the names and extensions for all other input and all of the output files.

	File Name or Type	Default Extension	File Format	Description
Input	NAFILE	DAT	text	Contains section and variable field information.
	POLFILE	DAT	text	Defines the forms to use for each batch.
	NEWTRN	DAT	text	Tells the GenArc program where to find data in the NAFILE and which forms to use in the POLFILE.
	APPIDX	DFD	text	Tells the GenArc program how to store the data.
Output	DBF files	DBF	text	Serves as an index to the archived data in the CAR files.
	ARCHIVE	CAR	CAR	Contains the archived forms.

RULES USED IN MULTI-STEP PROCESSING

Several rules are used to execute the programs of the multi-step process. For a complete listing and description of these and other rules, see the [Rules Reference](#).

RESTARTING THE GENDATA PROGRAM

You can set up the GenData program to restart itself at a particular transaction if it encounters a failure. To accomplish this, the system uses a restart file. You use INI options to set up the restart file.

NOTE: This feature does not apply if you are using single-step processing.

The restart file stores checkpoint information at specified intervals. If an error is encountered, the program resets itself and then checks each transaction until it isolates the transaction causing the error.

The restart file is removed at the end of a successful run. If the file exists at the start of a GenData run, the system assumes a restart is necessary and will open and read the file. The checkpoint information lets the system set internal pointers and output files in such a way that it can begin at that transaction.

These rules are used to handle restarting the GenData program:

- RULCheckTransaction
- RestartJob

RULCheckTransaction rule

The RULCheckTransaction rule is always the first base form set rule. It saves the EXTRFILE offset, TRNFILE offset, NEWTRN offset, NAFILE offset, POLFILE offset, and batch file offsets into a restart (RSTFILE) file.

These offsets are updated in the post process after a specific number of transactions. You specify the number of transactions using the CheckCount option. You define the Restart file and the and check count in the Restart control group:

```
< Restart >
  RstFile =
  CheckCount =
```

Option	Description
RstFile	Enter the name of the restart file. If you omit this option, the system uses RSTFILE.RST (DD:RSTFILE for MVS) as the file name. The system uses the DataPath option in the Data control group to determine where to create the restart file. The default location is the current working directory.
CheckCount	Enter a number to specify the number of transactions to process before updating the offsets. For instance, if you specify two hundred (200), the system processes two hundred transactions, updates the offsets, processes two hundred more transactions, and so on. The default is 100. You can also use the <code>/cnt</code> command line option with the GenData program to override the CheckCount option. Here is an example: <code>gendaw32 /cnt=10</code>

Here is an example:

```
;RULCheckTransaction;2;Always the first form set rule;
```

RestartJob rule The RestartJob is always the first base rule. This rule opens the restart file (RSTFILE) and resets the EXTRFILE, TRNFILE, NEWTRN, NAFILE, POLFILE, and batch files at the broken transaction if the restart file exists. If the restart file does not exist, the RestartJob rule is skipped.

NOTE: For more information on these rules, see the [Rules Reference](#). You can also set up the GenArc program to restart itself. For more information, see [Using the Restart Option on page 395](#).

Here is an example:

```
;RestartJob;1;Always the first base rule;
```

INI options To use the restart feature, you should also set the following INI options:

```
< GenDataStopOn >
  BaseErrors          = Yes
  TransactionErrors    = Yes
  ImageErrors          = Yes
  FieldErrors         = Yes
```

GENERATING BATCH STATUS EMAILS

You can set up the GenData program to check recipient batches and notify the print operator via email as to when to expect output print files.

You use INI options to have the JobInit1 rule notify batch recipients about batch file information. On Windows, Microsoft mail and the SMTP mail type is supported. On UNIX, only the SMTP mail type is supported.

With the INI settings shown below, the GenData program can...

- Notify a user that a batch is not empty. For example, the GenData program can send email notification if there are transactions in the error or manual batches or both.
- Notify a user that a batch is empty. For example, it can send an email to the print operator telling the operator not to expect a print file for processing.
- The notifications above can be skipped on per batch basis. For example, you can have the GenData program skip batches that do not produce print files or produce files that do not need to be printed.
- For each notification email you can specify a send to address, reply to address, message body, optional attachment, and message subject.
- To each email you can optionally attach a recipient batch file.
- The notification email message can include variable data which comes from GVM variables.

To use this feature, make sure you have your INI files set up as shown here. The new control groups and options appear in **bold** and are documented in the following table.

```
< Print_Batches >
    Batch1 = batch1.bch
    Batch2 = batch2.bch
    Batch2 = batch3.bch
    Manual = manual.bch
    Error = error.bch
< Batch1 >
    Printer = Printer1
    Notify = BchRecip1
...
< BatchNotify:BchRecip1 >
    Empty = Yes
    MailType = MSM
    AttachBatchFile = Yes
    SendTo = John Formaker
    Subject = Batch 1 is empty
    BodyTemplate = email.txt
...
< Mail >
    MailType = MSM
;    MailType = SMTP
< MailType:MSM >
    Module = MSMW32
    MailFunc = MSMMail
    ReplyTo = replyto@docucorp.com
    UserID = test
    SuppressDlg = Yes
    HiddenMsgSupport = Yes
```

```
Name = MS Exchange Settings
Recipient = test@oracle.com
```

Option	Description
Batch1 control group	
Notify	Enter the name of INI control group where the notification options are specified. In the example above, the control group name would be <i>BatchNotify:BchRecip1</i> .
BatchNotify:BchRecip1 control group	
Empty	Enter Yes if you want the system to notify you if this batch is empty or missing. Enter No if you want the system to notify you if the batch is not empty.
MailType	Enter MSM to specify the mail type as Microsoft mail. Enter SMTP to specify the mail type as SMTP. SMTP is the only option for UNIX.
AttachBatchFile	Enter Yes to attach the batch file if it exists and is not empty. Enter No if you do not want the system to attach it.
SendTo	Enter the name of the recipient or his or her email address.
Subject	Enter the text you want the system to place in the email subject line. For instance, you could enter <i>Batch 1 is empty</i> .
BodyTemplate	Here you can specify a template file, such as email.txt, to use when creating an email message. It has format: data for item one <% //test1,%s %> and trailing data

TRACKING BATCH PAGE STATISTICS

The system lets you track job statistics that show you...

- Total pages
- Pages not including copy counts
- Printed sheets
- Sheets by tray (1 through 9)

You can compile these statistics by batch, recipient within each batch, and job totals. You can also have the system write the totals to a recipient detail file, a batch summary file, and the log file. Totals are written to the log file by default.

You can add recipient totals to the recipient batch records by adding the appropriate global variables (GVMs) to the recipient batch file's Data Format Definition (DFD) file. If you create the optional batch summary file, the batch page statistics will be available to the GenPrint program via the batch total GVMs.

RECIPIENT PAGE STATISTICS

These statistics are captured for each recipient batch record written to the batch file:

Statistic	GVM	Description
Recipient	RCB_NAME	The current recipient name
Total Pages	RCB_TOTAL	The total recipient pages including non-print (display only) pages
Total Pages - No Copy	RCB_TOTAL_NC	The total recipient pages not including copy counts. Non-print (display-only) pages are included.
Total Sheets	RCB_SHEETS	The total printed sheets for the transaction (omits display-only pages)
Total Tray 1	RCB_TRAY1	The total printed sheets for Tray 1
Total Tray 2	RCB_TRAY2	The total printed sheets for Tray 2
Total Tray 3	RCB_TRAY3	The total printed sheets for Tray 3
Total Tray 4	RCB_TRAY4	The total printed sheets for Tray 4
Total Tray 5	RCB_TRAY5	The total printed sheets for Tray 5
Total Tray 6	RCB_TRAY6	The total printed sheets for Tray 6
Total Tray 7	RCB_TRAY7	The total printed sheets for Tray 7
Total Tray 8	RCB_TRAY8	The total printed sheets for Tray 8
Total Tray 9	RCB_TRAY9	The total printed sheets for Tray 9

BATCH TOTALS SUMMARY FILE

The system can write a summary record for each recipient within each batch and a total summary record to the optional Batch Totals Summary file. To have the system create this file, include the RCBStatsTot option in the Data control group and specify a file name.

You can modify the summary total file layout using a custom DFD. Specify the name of the custom DFD in the RCBStatsTotDFD option in the Data control group. If you omit the RCBStatsTotDFD option, the default DFD file is used (see [Default DFD Files on page 40](#)).

If there are more than one recipient for a given batch file, a Total record is written. The BATCH_RCB_NAME value is set to **** Total **** for the total file record. If a total record exists, the total record is loaded by the GenPrint program.

Accessing totals in GenPrint

If you set the RCBStats option in the RunMode control group to Yes and RCBStatsTot option in the Data control group has a value, the GenPrint program loads the total values for each batch. These values will then be available as GVM variables.

INI Options

You use the following INI options to record statistics:

```
< RunMode >
RCBStats      =
RCBTotals     =
```

Option	Description
RCBStats	Enter No if you do not want to execute statistics processing. The default is Yes, unless the system is running under IDS. If IDS is running Documaker Server, the default is No.
RCBTotals	Enter No if you do not want the system to write recipient totals to the log file. The default is Yes.

```
< Data >
RCBStatDt1DFD =
RCBStatsTotDFD =
RCBStatsDt1   =
RCBStatsTot   =
```

Option	Description
RCBStatDt1DFD	Enter a name for the RCB Statistics Detail File DFD. The system defaults to an internal DFD entry.
RCBStatsTotDFD	Enter a name for the RCB Statistics Total File DFD. The system defaults to an internal DFD entry.
RCBStatsDt1	Enter the name and path you want assigned to the detail log file. The system will create this file if you include a value for this option.
RCBStatsTot	Enter the name and path you want assigned to the total log file. The system will create this file if you include a value for this option.

SAMPLE LOG FILE

Here is an example of a log file:

```

-----
Batch Page Statistics
-----
Batch(BATCH1):
- Total for Recipient(AGENT) in Batch(BATCH1):
  Pages      :      9
  Pages(nc)  :      9
  Sheets     :      6
  Tray1      :      2
  Tray2      :      2
  Tray3      :      0
  Tray4      :      2
  Tray5      :      0
  Tray6      :      0
  Tray7      :      0
  Tray8      :      0
  Tray9      :      0
- Total for Recipient(COMPANY) in Batch(BATCH1):
  Pages      :     21
  Pages(nc)  :     21
  Sheets     :     16
  Tray1      :      3
  Tray2      :      2
  Tray3      :      9
  Tray4      :      2
  Tray5      :      0
  Tray6      :      0
  Tray7      :      0
  Tray8      :      0
  Tray9      :      0
- Total for Recipient(INSURED) in Batch(BATCH1):
  Pages      :     44
  Pages(nc)  :     44
  Sheets     :     28
  Tray1      :      6
  Tray2      :     11
  Tray3      :      9
  Tray4      :      2
  Tray5      :      0
  Tray6      :      0
  Tray7      :      0
  Tray8      :      0
  Tray9      :      0
- Total for Batch(BATCH1):
  Pages      :     74
  Pages(nc)  :     74
  Sheets     :     50
  Tray1      :     11
  Tray2      :     15
  Tray3      :     18
  Tray4      :      6
  Tray5      :      0

```

```
Tray6      :      0
Tray7      :      0
Tray8      :      0
Tray9      :      0
Job Page Statistics:
Pages      :      74
Pages(nc)  :      74
Sheets     :      50
Tray1      :      11
Tray2      :      15
Tray3      :      18
Tray4      :       6
Tray5      :       0
Tray6      :       0
Tray7      :       0
Tray8      :       0
Tray9      :       0
```

DEFAULT DFD FILES

Here are examples of the DFD files:

RCBStatsDtlDFD

```
< FIELDS >
FIELDNAME = RCB_BATCH
FIELDNAME = RCB_NAME
FIELDNAME = RCB_TRANS
FIELDNAME = RCB_TOTAL
FIELDNAME = RCB_TOTAL_NC
FIELDNAME = RCB_SHEETS
FIELDNAME = RCB_TRAY1
FIELDNAME = RCB_TRAY2
FIELDNAME = RCB_TRAY3
FIELDNAME = RCB_TRAY4
FIELDNAME = RCB_TRAY5
FIELDNAME = RCB_TRAY6
FIELDNAME = RCB_TRAY7
FIELDNAME = RCB_TRAY8
FIELDNAME = RCB_TRAY9
< FIELD:RCB_BATCH >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 21
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
EXT_LENGTH = 20
KEY = Y
REQUIRED = Y
< FIELD: RCB_NAME>
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 21
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
EXT_LENGTH = 20
KEY = Y
REQUIRED = Y
< FIELD:RCB_TRANS >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 31
```

```
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
EXT_LENGTH = 30
KEY = N
REQUIRED = N
< FIELD:RCB_TOTAL >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 11
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 10
  KEY = N
  REQUIRED = N
< FIELD:RCB_TOTAL_NC >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 11
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 10
  KEY = N
  REQUIRED = N
< FIELD:RCB_TRAY1 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 11
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 10
  KEY = N
  REQUIRED = N
< FIELD:RCB_TRAY2 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 11
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 10
  KEY = N
  REQUIRED = N
< FIELD:RCB_TRAY3 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 11
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 10
  KEY = N
  REQUIRED = N
< FIELD:RCB_TRAY4 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 11
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 10
  KEY = N
  REQUIRED = N
< FIELD:RCB_TRAY5 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 11
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 10
  KEY = N
  REQUIRED = N
< FIELD:RCB_TRAY6 >
  INT_TYPE = CHAR_ARRAY
```

```
INT_LENGTH = 11
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
EXT_LENGTH = 10
KEY = N
REQUIRED = N
< FIELD:RCB_TRAY7 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 11
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 10
  KEY = N
  REQUIRED = N
< FIELD:RCB_TRAY8 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 11
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 10
  KEY = N
  REQUIRED = N
< FIELD:RCB_TRAY9 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 11
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 10
  KEY = N
  REQUIRED = N
```

RCBStatsTotDFD

```
< FIELDS >
  FIELDNAME = BATCH_NAME
  FIELDNAME = BATCH_RCB_NAME
  FIELDNAME = BATCH_TOTAL
  FIELDNAME = BATCH_TOTAL_NC
  FIELDNAME = BATCH_SHEETS
  FIELDNAME = BATCH_TRAY1
  FIELDNAME = BATCH_TRAY2
  FIELDNAME = BATCH_TRAY3
  FIELDNAME = BATCH_TRAY4
  FIELDNAME = BATCH_TRAY5
  FIELDNAME = BATCH_TRAY6
  FIELDNAME = BATCH_TRAY7
  FIELDNAME = BATCH_TRAY8
  FIELDNAME = BATCH_TRAY9
```

CONTROLLING GENTRN PROCESSING

Include the following control group and option in the FSISYS.INI file when you want the GenTrn program to continue processing transactions when errors occur. By default, the GenTrn program halts when it encounters an error.

NOTE: This control group and option is typically used if you are using XML extract files and you do not want the GenTrn program to stop every time it encounters an error. For any type of extract file, using this option detects missing Key1 and Key2 information.

Here is an example of the control group and option:

```
< GenTranStopOn >
    TransactionErrors = Parameter1;Parameter2;Parameter3;
```

Parameter	Description
Parameter1	Enter No to turn the GenTranStopOn option off. The default is Yes.
Parameter2	Enter the name of the transaction file. To write out the error transaction, enter the name of the file where you want the extract file records written. If you omit the path, the system uses the DataPath option in the Data control group in the FSISYS.INI file to determine where to locate this file.
Parameter3	The system only looks at this parameter if you entered a file name for Parameter2. Enter Yes to tell the system to append the error transactions accumulated during this processing run to the file created in a prior run. Enter No to tell the system to overwrite any existing file. If Parameter2 exists and you omit this parameter, the system defaults to No. If you enter Yes, you must remove the file when necessary. Keep in mind that over a series of processing runs, this file will expand in size.

Separate the parameters with semicolons (;).

The system records all errors and warnings it encounters during a processing run in the ERRORFILE.DAT file. In addition, it writes the extract file records of the transaction in error to the file you specify in Parameter2. This lets you inspect those transactions and determine the best way to proceed.

Here are some examples. This option:

```
TransactionErrors = No;..\Extracts\ErrorTransaction.dat;No;
```

Is the same as:

```
TransactionErrors = No;..\Extracts\ErrorTransaction.dat;;
```

Both let the GenTrn program continue processing subsequent transactions when errors occur. These options tell the GenTrn program to write the error transaction to a file named ERRORTRANSACTION.DAT, stored in the \Extracts directory.

```
TransactionErrors = No; ErrorTransaction.dat;Yes;
```

This option lets the GenTrn program continue processing subsequent transactions when errors occur. Since the path of the error transaction file was omitted, the system uses the DataPath option in the Data control group in the FSISYS.INI file to find the file so it can append any error transactions to the existing error transaction file.

```
TransactionErrors = No;;;
```

This option lets the GenTrn program continue processing subsequent transactions when errors occur. It does not, however, write out error transactions.

When using this option, you may encounter these errors:

- Problem in loading the XML file. Syntax error.

```
GenTrn
```

```
Transaction Error Report - System timestamp: Mon Dec 16 13:42:27 2002
DM12041: Error: FAP library error: Transaction:<1111111111>,
area:<DXMLoadXMLRecs>
  code1:<48>, code2:<0>
  msg:<XML Parse Error: The 15 chars before error=<    <Key1>Comp1<>,
the 8 chars starting at error=</Key1c>
>>.
DM12041: Error : FAP library error: Transaction:<1111111111>,
area:<DXMLoadXMLRecs>
  code1:<48>, code2:<0>
  msg:<mismatched tag at line 3 column 16>.
DM10293: Error: Error in <BuildTrnRecs>: Unable to
<DXMLoadXMLRecs()>.
  Skip Transaction# <2>.
Warning: the specific info you see may not be the info for the error
transaction. It may be the info on the last complete transaction.

==> Warning count:    0
==> Error   count:    3
```

- No problem in loading the file, however, Key1 is omitted in the transaction.

```
GenTrn
```

```
Transaction Error Report - System timestamp: Fri Dec 13 13:52:13 2002
DM1002: Error: Required INI definition omitted.
Cannot locate INI group <Key1Table> with value = defined.
DM15062: Error in BuildTrnRecs(): Unable to GENGetDocSetNames(pRPS).
Skip Transaction# <3>.

==> Warning count:    0
==> Error   count:    2
```

USING SINGLE-STEP PROCESSING

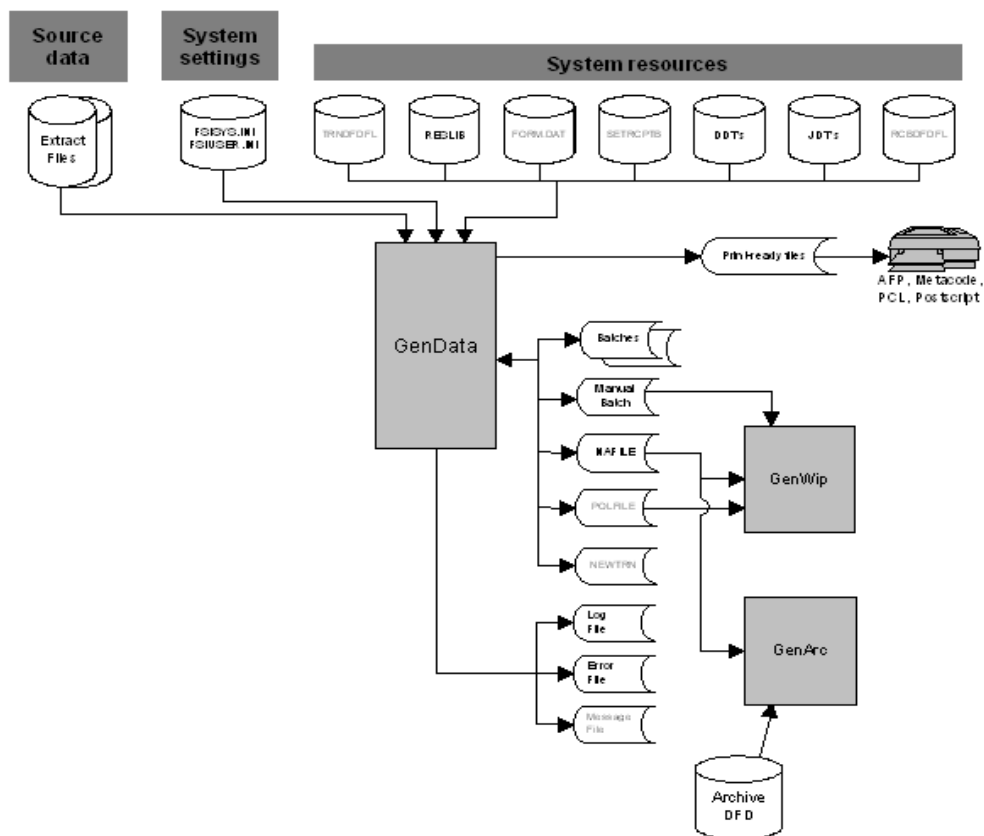
The single-step process improves the performance of your system by combining the functions of GenTrn, GenData and GenPrint into one step performed by GenData. This process is used when no intermediate steps are necessary.

The GenWIP and GenArc options are performed the same as in the multi-step process. See [Sending Incomplete Transactions to WIP on page 29](#) and [Archiving Transactions on page 31](#) for more information on the functions of the GenWIP and GenArc programs.

NOTE: When running in single-step mode, you can only produce a single print stream. For instance, the most common method of print batching is to batch by recipient, in single-step processing, however, you cannot produce separate print streams for each recipient batch.

CREATING AND PROCESSING TRANSACTION RECORDS

In the multi-step process, the GenTrn program creates transaction records that are sent to the GenData program for processing. In the single-step process, the GenData program performs both of these actions in one step.



As shown in the illustration above, the GenData program processes transaction records, originated from the source data, and creates various output files for print, WIP or GenArc. By combining the functions of GenTrn and GenPrint into GenData, you reduce the number of times the system needs to open and close files, thus enhancing the overall performance of your system.

System Settings and Resources

The FSI SYS.INI and the FSI USER.INI file provide system setting information, such as whether or not it should stop processing if it encounters errors, how to identify key fields in extract files, whether or not it should check the output data size against the defined field length, and so on.

The files listed under system resources provide additional information such as:

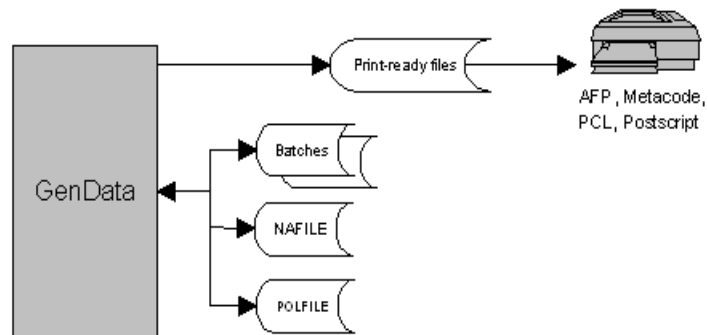
- How to read the transaction file (TRNDFDL.DFD)
- The forms, logos, and other resources to use when creating the form sets (RESLIB)
- What forms to use (FORM.DAT)
- Who to send the forms to (SETRCPTB.DAT)
- What processing rules to apply to the data (DDTs)
- What processing rules to apply to this job (JDTs)
- How the batch files are defined (RCBDFDL.DFD)

NOTE: You can learn more about these files in Appendix B, [System Files on page 461](#).

The advantage of single-step processing is the improvement to performance. The disadvantage is that it is much more difficult to correct errors because the system does not create batch files at the end of each step. These batch files tell you what occurred and help you spot and correct errors.

CREATING PRINT FILES

With the placement of specific rules, you can make the GenData program perform the functions of the GenTrn and GenPrint programs. In other words, when GenData is processing transactions files, it is also producing the print-ready files necessary to print on AFP, Metacode, PCL, or Postscript printers.



As in the multi-step process, the GenData program creates these types of files:

- Batch files - list the transactions which should be included in each batch print job
- NAFILES - store section and variable field information
- POLFILES - define the form set the GenPrint program should use for each transaction it processes

NOTE: When using single-step processing, you should clear all messages before each processing run. For information on how to do this, see [Clearing Messages on page 370](#).

File Summary

This table summarizes the files used to supply information (input) and the files created by (output) the GenData program:

NOTE: You can use the Data control group in the FSISYS.INI file to specify the names and extensions for all other input files and all of the output files.

	File name or Type	Default Extension	File Format	Description
Input	Extract files		text	Contains the data you want to process.
	FSISYS	INI	text	Initialization file which includes system settings.
	TRNDFDFL	DFD	text	Tells GenData how to read and write the TRNFILE.
	FORM	DAT	text	Defines the forms in a form set.
	SETRCPTB	DAT	text	Defines the recipients of a form set.
	DDT files	DDT	text	Contains the rules GenData applies to the data.
	JDT files	JDT	text	Contains the rules GenData follows when processing the job.
	RCBDFDFL	DFD	text	Defines the attributes of the variable fields in a batch file.
	Resources	(various)	(various)	Includes logos (LOG), font cross reference files (FXR), sections (FAP), and so on.
Output	Batch files	BCH	text	Indicates which transactions should be included in a given batch job.
	NAFILE	DAT	text	Contains section and variable field information. Used by the, GenWIP, and GenArc programs.
	POLFILE	DAT	text	Defines the forms to use for each batch. Used by the GenWIP and GenArc programs.
	NEWTRN	DAT	text	Tells the GenArc program where to find data in the NAFILE and which forms to use in the POLFILE.
	Manual batch files	BCH	text	Created if the form is incomplete. Used by GenWIP to allow an operator to complete the form in the Entry module.

File name or Type	Default Extension	File Format	Description
Error batch files	.BCH	text	Created if the system spots an error, such as if the system spots an error and the form is marked as host required. In contrast to manual batch files, you cannot correct these errors using the GenWIP program. Instead, you must correct the error in the extract file, change the flag to operator required, or change the FAP file and then process the transaction again.
ARCHIVE	DFD	text	Tells the GenArc program how to store archived data.
Log file	DAT	text	Serves as a processing log. Created by the GenData program in the single-step process.
Error file	DAT	text	Notes any errors encountered by the GenData program. Created by the GenData program in the single-step process.
Message file	.DAT	text	Intermediate file which contains log and error messages. These messages are then translated and written to either the LOGFILE.DAT or ERRFILE.DAT files.

USING THE MULTIFILEPRINT CALLBACK FUNCTION

The system includes a MultiFilePrint callback function designed for running the GenData program in single-step mode. The log file is either a semicolon delimited text file—the same as the file created by MultiFilePrint—or an XML file.

The layout of the XML file is as follows:

```

-
-
.\data\BATCH1.BCH
SAMPKO
LB1
1234567
T1
INSUREDS COPY
DATA\0rDcP7WxytE82ECp5jexhWXVqkjV840Vw_F-GykT_VMfd.PDF
-
.\data\BATCH2.BCH
SAMPKO
LB1
1234567
T1
COMPANY COPY
DATA\0v317pBdVqHceoRL5hf2xqjJ7WAMxiRVO9U70iFiXIcne.PDF

```

You can use the INI options in the DocSetNames control group to determine which XML elements are created. The values are the same as those written to a recipient batch or transaction file.

The MultiFilePrint callback function should only be used with the PDF, RTF, HTML, and XML print drivers. See also [Controlling What is in the MultiFilePrint Log on page 109](#).

MAPPING FIELDS WITH XPATH

The GenTrn program and the NoGenTrnTransactionProc rule let you use the TRN_Fields control group to map all of your fields with XPath. To let the system know you are using the XML file, set the XMLTrnFields option in the TRN_File control group to Yes and also set the XMLExtract option in the RunMode control group to Yes.

Here is an example:

```
< RunMode >
    XMLExtract = Yes
< TRN_File >
    XMLTrnFields= Yes
< TRN_Fields >
    Company    = !/Forms/Key1
    LOB        = !/Forms/Key2
    PolicyNum   = !/Forms/PolicyNum
    RunDate     = !/Forms/RunDate;DM-4;D4
```

NOTE: Use this format for the Trn_Fields control group options:

(Field in the Transaction DFD File) = XPath;Field Format

Be sure to include the leading exclamation mark (!). This tells the system to use an XML path search but is not part of the actual search routine. Do not specify whether a field is a key. The system does not support multiple (search) keys with the XML implementation.

If you are selectively excluding transactions, in your exclude file, instead of an offset and SearchMask, replace it with the XPath. Here is an example:

```
!/Forms[PolicyType="OLD"]
```

RUNNING ARCHIVE IN SINGLE-STEP PROCESSING

Using rules developed for archiving via Docupresentment, you can run the GenArc program as part of single-step processing.

Use the InitArchive rule to check the INI options in the Trigger2Archive control group, initialize the database, open the APPIDX.DFD and CAR files, and perform other steps to initialize archive.

The Archive rule then unloads the current form set and converts field data for archive using the INI options in the Trigger2Archive control group.

Here is an example:

```
< Base Rules >
;InitArchive;1;;
< Base Form Set Rules >
;Archive;2;;
```

NOTE: For more information on these rules, see the [Rules Reference](#).

RUNNING WIP IN SINGLE-STEP PROCESSING

You can use the InitConvertWIP and ConvertWIP rules to run the GenWIP program in single-step mode.

Use the InitConvertWIP rule to perform the initialization necessary for the ConvertWIP rule.

Use the ConvertWIP rule to see if the current transaction is assigned to the MANUAL.BCH file. If it is, the rule adds the record to WIP and unloads the contents of the POLFILE.DAT and NAFILE.DAT files to new files with unique names.

You can then view these WIP records using Documaker Workstation or the WIP Edit plug-in, which is part of the Docupresentment suite of products.

Here is an example:

```
< Base Rules >
;InitConvertWIP;1;;
< Base Form Set Rules >
;ConvertWIP;2;;
```

NOTE: For more information on these rules, see the [Rules Reference](#).

RULES USED IN SINGLE-STEP PROCESSING

Specific rules are used to combine the execution and functionality of the GenTrn, GenData, and GenPrint programs into a single step. To begin familiarizing yourself with these rules, an alphabetical listing and brief description follows. You can find more information in the Rules Reference.

Archive Use this form set level (level 2) rule after the InitArchive rule to unload the current form set and convert field data for archive using the INI options in the Trigger2Archive control group.

BatchingByRecipINI Use this form set level (level 2) rule to send transactions to a batch you specify based on data in the extract file. To use this rule, you must include the BatchingByRecip control group in your FSISYS.INI file with options similar to those shown below:

```
< BatchingByRecip >
  Batch_Recip_Def = default;"ERROR"
  Batch_Recip_Def = 4,1234567;"BATCH1";INSURED
  Batch_Recip_Def = true;"BATCH2";INSURED
  Batch_Recip_Def = True;"BATCH3";COMPANY | true;"BATCH2";AGENT
```

You must also add the TWOUP control group and CounterTbl option to the FSISYS.INI file.

BatchByPageCount Use this form set level rule to send a transaction to a specific batch based on the number of pages produced by processing the transaction. The batch used is determined by the PageRange option in the Batch control group.

In the example below; transactions that produce 1 to 7 pages are send to Batch1. Transactions that produce 8 to 25 pages are send to Batch2. In addition, you must add the TWOUP control group and CounterTbl option to the FSISYS.INI file.

```
< Batches >
  Batch1    = .\data\Batch1
  Batch2    = .\data\Batch2
  Batch3    = .\data\Batch3
  Error     = .\data\Error
  Manual    = .\data\Manual

< Batch1 >
  Printer   = Batch1_PTR
  ...      .....
  PageRange = 1,7

< Batch2 >
  Printer   = Batch2_PTR
  ...      .....
  PageRange = 8,25
  .....

< TWOUP >
  CounterTbl = .\data\counter.tbl
```

BuildMasterFormList Use this job level rule (level 1) to load the FORM.DAT file into an internal linked list within the GenData program. You must include this rule in the AFGJOB.JDT file because the RunSetRcpTbl rule is dependent on the list this rule creates.

ConvertWIP	Use this form set level (level 2) rule to see if the current transaction is assigned to the MANUAL.BCH file. If it is, the rule adds the record to WIP and unloads the contents of the POLFILE.DAT and NAFILE.DAT files to new files with unique names. You can then view these WIP records using Documaker Workstation or the WIP Edit plug-in, which is part of Docupresentment.
InitArchive	Use this job level (level 1) to check the INI options in the Trigger2Archive control group, initialize the database, open the APPIDX.DFD and CAR files, and perform other steps to initialize archive.
InitConvertWIP	Use this job level (level 1) rule to perform the initialization necessary for the ConvertWIP rule.
InitPrint	Use this job level (level 1) rule to load printer and recipient batch information. This rule sets up PRTLIB data, initializes print options, and loads a table which contains page totals for recipient batch files.
InitSetRecipCache	Use this job level rule (level 1) to set the amount of cache the system uses to store recipient information in memory. With this rule you can tell the system the amount of memory to set aside and use for storing information in the Key1 and Key2 fields, often used to store the company, line of business, and transaction codes. You can use this rule to improve processing performance for complex forms. This rule has no affect on the processing speed for static forms.
<hr/> <p>NOTE: If you omit this rule, the system does not set aside memory for the Key1 and Key2 fields.</p> <hr/>	
NoGenTrnTransactionProc	Use this form set level rule when you use the GenData program by itself to execute the GenTrn and GenData steps. In the single-step processing environment, this rule processes the extract file and creates the information normally created in both the GenTrn and GenData steps. When combined with the InitPrint and PrintFormset rules, it creates the output files normally created during the GenPrint step.
<hr/> <p>NOTE: Do not use this rule if you are running the GenTrn, GenData, and GenPrint programs as separate processes (multi-step processing).</p> <hr/>	
PageBatchStage1InitTerm	<p>Use this job level rule (level 1) to create and populate a list of records which contain page ranges and total page counts for each recipient batch file.</p> <p>This rule is typically used for handling 2-up printing for AFP and compatible printers.</p> <p>This rule creates a list (populated in another rule) to contain the recipient batch records for a multi-mail transaction set. The rule then writes out the recipient records for the final multi-mail transaction set and writes out the total page counts for each recipient batch. You must add the TWOUP control group and CounterTbl option to the FSISYS.INI file, as shown here:</p>

```
< TwoUp >
CounterTbl = .\data\counter.tbl
```

PaginateAndPropagate	Use this form set level (level 2) rule to paginate the form set and merge in or propagate field data.
PrintFormset	Use this form set level (level 2) rule when you run the GenData program by itself to execute GenTrn and GenPrint processes. In the single-step processing environment, this rule, when combined with the InitPrint rule, prints form sets.
<hr/> NOTE: Do not use this rule if you are running the GenTrn, GenData, and GenPrint programs as separate processes (multi-step processing). <hr/>	
ProcessQueue	Use this form set level (level 2) rule to process the queue you specify. This rule loops through the list of functions for the queue you specify and then frees the queue when finished.
StandardFieldProc	This rule is a field level rule (level 4), which you must include in the AFGJOB.JDT file. This rule is used when you are using the performance mode JDT and should be the first field level rule. This rule tells the system to process each field on all of the sections triggered by the SETRCPTB.DAT file. If you use the StandardFieldProc rule in your JDT, you must also include the WriteNAFile rule.
StandardImageProc	This rule is an section (image) level rule (level 3) which you must include in the AFGJOB.JDT file. This rule is used when you are using the performance mode JDT and should be the first section level rule. This rule tells the system to process each section triggered by the SETRCPTB.DAT file.
WriteNAFile	Use this form set level rule (level 2) to append the NAFILE.DAT file data records for the current form set into an existing NAFILE.DAT file. When you use the NoGenTrnTransactionProc rule, which replaces the RULStandardProc rule, you must include the WriteNAFile rule to cause data (records) to be written to the NAFILE during the GenData processing step. In addition, you must also include the WriteOutput rule to cause data (records) to be written to the POLFILE.DAT and NEWTRN.DAT files during the GenData processing step.
WriteOutput	<p>Use this form set level (level 2) rule to append the POLFILE.DAT file data records for the current form set into an existing POLFILE.DAT file.</p> <p>You also use this rule when you are using the GenData program by itself to execute the GenTrn, GenData, and GenPrint processing steps.</p> <p>If you use this rule, do not use the UpdatePOLFile rule.</p>
WriteRCBWithPageCount	Use this form set level rule (level 2) to write page counts for each recipient. This rule is typically used for handling 2-up printing on AFP and compatible printers. To use this rule, you must update the RCBDFDFL.DFD file with the following items:

```
< FIELDS >
  FIELDNAME = CurPage
  FIELDNAME = TotPage
  FIELDNAME = AccumPage
  FIELDNAME = MMFIELD
< FIELD:CurPage >
  INT_TYPE = LONG
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
```



```

EXT_LENGTH = 10
KEY = N
REQUIRED = N
< FIELD:TotPage >
  INT_TYPE = LONG
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 10
  KEY = N
  REQUIRED = N
< FIELD:AccumPage >
  INT_TYPE = LONG
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 10
  KEY = N
  REQUIRED = N
< FIELD:MMFIELD >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 7
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 6
  KEY = N
  REQUIRED = N

```

SINGLE-STEP PROCESSING EXAMPLE

As stated earlier, the single-step process is performed by combining the execution and functionality of the GenTrn, GenData, and GenPrint programs. This is done by placing certain rules into a specialized JDT. The earlier illustration shows the input and output files used by GenData to process transactions and print output files in one step. The following file describes the JDT used to process the job and an example of the rules used to combine the GenTrn, GenData, and GenPrint functions.

To make this happen, the NoGenTrnTransactionProc rule, along with other rules, are placed in the JDT file as seen in the following sample file. An actual sample file can be seen in the RPEX1 sample library.

Base rules

The following base rules are designed for the performance mode.

```

;RULStandardJobProc;1;Always the first job level rule;
;SetErrHdr;1;***:-----;
;SetErrHdr;1;***: BillPrint Data Generation (Base);
;SetErrHdr;1;***:
;SetErrHdr;1;***: Transaction:      ***ACCOUNTNUM***;
;SetErrHdr;1;***: Company Name:    ***Company***;
;SetErrHdr;1;***: Line of Business: ***LOB***;
;SetErrHdr;1;***: Run Date:       ***RunDate***;
;SetErrHdr;1;***:-----;
;JobInit1;;
;CreateGlbVar;1;TXTLst,PVOID;
;CreateGlbVar;1;TblLstH,PVOID;
;InitOvFlw;1;;
;SetOvFlwSym;1;SUBGROUPOVF,SUBGROUP,5;
;BuildMasterFormList;4;
;PageBatchStage1InitTerm;;
;InitSetrecipCache;;

```

The following rule is required to execute GenData and GenPrint as a single step.

```
;InitPrint;;;
```

Base form set rules

The following base form set rules causes GenTrn and GenData to be combined into a single step.

```
;NoGenTrnTransactionProc;;;
;ResetOvFlw;2;;
;BuildFormList;;;
;LoadRcpTbl;;;
;RunSetRcpTbl;;;
```

The following rules are required to execute GenData and GenPrint as a single step.

```
;PrintFormset;;;
;WriteOutput;;;
;WriteNaFile;;;
;WriteRCBWithPageCount;;;
;ProcessQueue;;PostPaginationQueue;
;PaginateAndPropagate;;;
;BatchingByRecipINI;;;
```

Base image rules

The following base image rules apply to every image (section) in this base.

```
;StandardImageProc;3;Always the 1st image level rule;
```

Base field rules

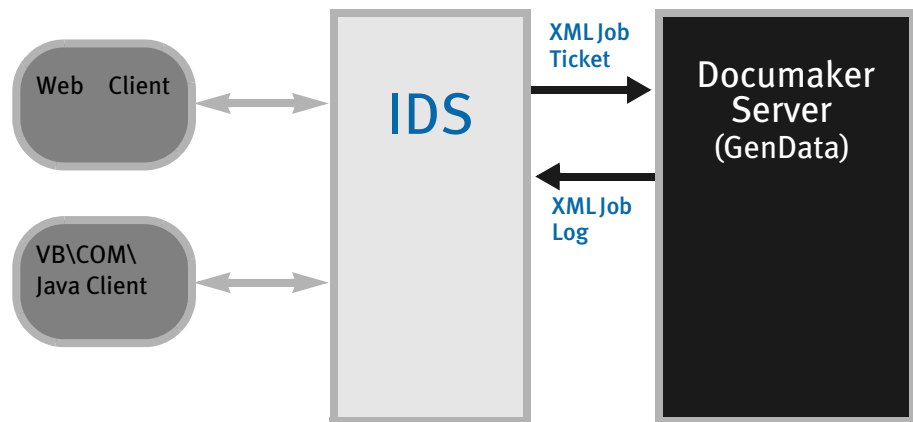
The following base field rules apply to every field in this base.

```
;StandardFieldProc;4;Always the 1st field level rule;
```

USING IDS TO RUN DOCUMAKER

If you have a license for both Documaker and Docupresentment, you can set up the Internet Document Server (IDS) to run Documaker as a subordinate process. Web clients communicate with IDS using queues. IDS communicates with Documaker via XML files called *job tickets* and *job logs*.

This diagram illustrates the process:



IDS can start or stop Documaker Server as needed, without user interaction. One IDS session controls one Documaker process. You can, however, implement multiple IDS sessions and have multiple Documaker Server processes as well.

Keep in mind these limitations:

- You can only run Documaker in single step mode.
- You must run Documaker on Windows 2000 or higher.
- Different resource setups for Documaker are supported, but Documaker processing restarts if resources are changed, eliminating the performance benefits. This should not be a problem because it is unlikely multiple Documaker Server setups will be used with a single IDS implementation. You can, however, experience problems testing a system with multiple setups.
- During processing, some INI options can be changed by the client. Since some Documaker rules use static variables and store INI values in memory, it is possible that a client will be unable to change an INI option if those Documaker rules are used. To handle these situations, you must restart Documaker.

For more information, see the [Internet Document Server Guide](#) and the [SDK Reference](#).

WRITING UNIQUE DATA INTO RECIPIENT BATCH RECORDS

The GenData program lets you add unique data to each recipient batch record before it is written to the recipient batch files. The recipient batch record data and format is defined by the GVM variable definitions in the RCBDFFDL.DAT file.

You can use this capability if you need to add...

- Address information or other field level information to the batch record, which is typically unique for each recipient.
- Recipient information that is not handled by normal field mapping from the transaction DFD to the recipient batch DFD.
- Cumulative or calculated information not available until the document is nearly completed.

Understanding the System

Before this feature was implemented in version 10.2, the recipient batch records were identical except for the recipient code field which contains a unique identifier assigned to a given recipient. If additional recipient data was required, you had to write a custom rule.

Use the options in the RecipMap2GVM control group to set up this capability. Data that can be added to the recipient batch record can be:

- Contents of a variable field on the specified section or form/section
- Constant value
- Data from an existing INI built-in functions, such as ~DALRun
- Data from a custom written INI function

Here is an example of the RecipMap2GVM control group:

```
< RecipMap2GVM >
  Form      =
  Image     =
  Req       =
  Opt       =
```

Option	Description
--------	-------------

Form	(optional) Enter the name of the form.
Image	Enter the name of the section (image). You can also enter a section name root. A section name root is the first part of a name. For instance, <i>MAILER</i> is the root name for sections with names such as <i>MAILER A</i> , <i>MAILER_B</i> , or <i>MAILERS</i> .
Req *	A semicolon delimited string that contains one of the following: <ul style="list-style-type: none">- GVM variable name; variable field name; optional formatting information- GVM variable name; blank character (space); constant value- GVM variable name; INI built-in function

Option	Description
--------	-------------

Opt *	A semicolon delimited string that contains one of the following: <ul style="list-style-type: none"> - GVM variable name; variable field name; optional formatting information - GVM variable name; blank character (space); constant value - GVM variable name; INI built-in function
-------	--

* = Repeat for each GVM variable you are setting up.

Optional formatting information

You can add optional formatting information as a parameter of the Opt INI option. This formatting information is comprised of four items separated by commas.

Item	Description
Input fetypes	D or d = date N or n = number
Input format mask	Date - see the FmtDate rule in the Rules Reference. Number – see the FmtNum rule in the Rules Reference.
Output fetypes	D or d = date N or n = number
Output format mask	Date - see the FmtDate rule in the Rules Reference. Number – see the FmtNum rule in the Rules Reference.

Here are some formatting examples:

d, "1/4", d, "4/4"

This converts an input date, mmddyyyy, into *month name dd, yyyy*, such as February 17, 2006.

n, nCAD, nUSD, "\$zzz,zz9.99"

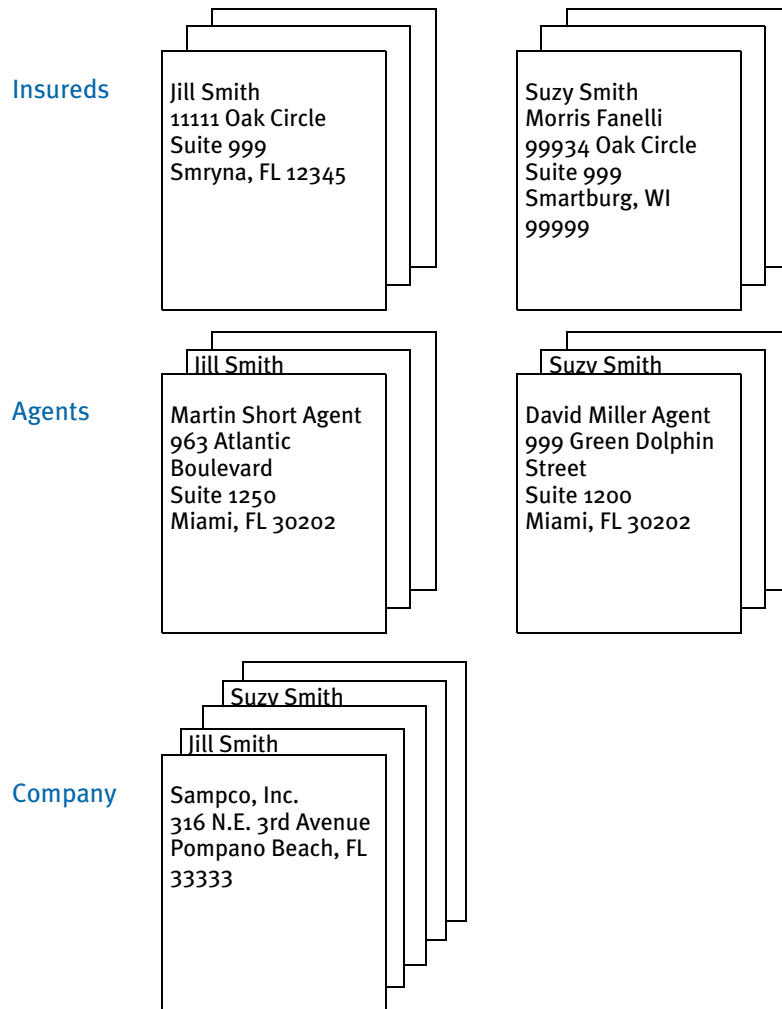
This converts an input numeric value in Canadian French format into a value in United States format.

Keep in mind...

- For the Req option, if the data is missing an error occurs and the transaction is send to the error batch.
- For the Opt option, if the data is missing the system stores an empty string in the GVM variable.
- A RCB GVM variable cannot be restored to its original or default value after it has been changed using this method.
- Any RCB GVM variable not assigned using this method retains the value originally set during the transaction processing.
- Some RCB GVM variables should never be changed using this mapping technique. These include:
 - TRN_Offset
 - NA_Offset

- POL_Offset
- If the section defined in the Image option in the RecipMap2GVM control group does not name a section, the feature is disabled for all transactions.
- If the section defined in the Image option is missing from the form set being processed, the GVM data is not changed. Depending on where the GVM data is mapped, this could mean data from the prior transaction will still be in the GVM variables.
- If there are multiple sections with the same name in the form set, the form specified in the Form option is used to identify the section to use. If the Form option is omitted, the first section found in the current form set is used.
- The system assumes the specified section contains all of the unique data except for a constant value or data gathered from an INI built-in function.
- If more than one recipient is assigned to the section, all recipient batch records receive the same added data.

Example This example creates a mailer cover page for each insured, agent, and/or company recipient per transaction. The cover page is created using banner page processing which occurs during GenPrint processing. Examples of the three different mailer cover pages are as follows.



This example assumes that the:

- Agent and company recipient batch files are sorted (agent number and company name, respectively) before the GenPrint program runs. This sorting allows for the creation of only one mailer cover page per unique agent and company.
- Unique information is contained on the form/section, Dec Page/Q1MDC1.
- The FSIUSER.INI file includes these control groups and options:

```
< RecipMap2GVM >
  Form      = Dec Page
  Image     = Q1MDC1
  Opt       = Name1;Insured Name;
  Opt       = Name2;Insured Name2;
  Opt       = Address1;Address Line1;
  Opt       = Address2;Address Line2;
  Opt       = CityCounty;prtvalue;
  Opt       = AgentName;Agent Name;
  Opt       = AgentID; Agent ID;
  Opt       = OfficeAddress;Office Address;
  Opt       = TownandState;Town And State;
< Printer >
  PrtType           = PCL
  EnableTransBanner = True
  EnableBatchBanner = False
  TransBannerBeginScript= PreTrans
  TransBannerEndScript = PstTrans
  TransBannerBeginForm = ;BANNER;TRANSACTION;TRANS HEADER;
  TransBannerEndForm   = ;BANNER;TRANSACTION;TRANS TRAILER;
< DALLibraries >
  LIB = Banner
```

BANNER.DAL

The DefLib directory contains this DAL script:

```
* This script obtains the required unique data from the recipient
* batch record and stores it on the mailer form.
```

```
BeginSub PreTrans
```

```
blank_gvm = Pad(" ",41," ")
SetGVM("NameA"      ,blank_gvm,, "C",41)
SetGVM("NameB"      ,blank_gvm,, "C",41)
SetGVM("AddressA"   ,blank_gvm,, "C",41)
SetGVM("AddressB"   ,blank_gvm,, "C",41)
SetGVM("CityCounty1",blank_gvm,, "C",41)
If Trim(RecipName()) = "INSURED" Then
  SetGVM("NameA"      ,GVM("Name1")      , , "C",41)
  SetGVM("NameB"      ,GVM("Name2")      , , "C",41)
  SetGVM("AddressA"   ,GVM("Address1")   , , "C",41)
  SetGVM("AddressB"   ,GVM("Address2")   , , "C",41)
  SetGVM("CityCounty1",GVM("CityCounty") , , "C",41)
  GoTo exit:
End
```

```
last_agent_id = last_agent_id
If Trim(RecipName()) = "AGENT" Then
```



```

If last_agent_id != Trim(GVM("AgentID")) Then
  last_agent_id = Trim(GVM("AgentID"))
  SetGVM("NameA"      ,GVM("AgentName")      ,,"C",41)
  SetGVM("NameB"      ,GVM("OfficeAddress")  ,,"C",41)
  SetGVM("AddressA"   ,GVM("TownandState")   ,,"C",41)
  GoTo exit:
Else
  SuppressBanner()
  GoTo exit :
End
End

last_company_name = last_company_name
If Trim(RecipName()) = "COMPANY" Then
  If Trim(GVM("Company")) != last_company_name Then
    last_company_name = Trim(GVM("Company"))
    If Trim(GVM("Company")) = "SAMPCO" Then;
      SetGVM("NameA"      ,"Sampco, Inc."      ,,"C",41)
      SetGVM("NameB"      ,"316 N.E. 3rd Avenue" ,,"C",41)
      SetGVM("AddressA"   ,"Pompano Beach, FL 33333" ,,"C",41)
      GoTo exit:
    ElseIf Trim(GVM("Company")) = "FSI"
      SetGVM("NameA"      ,"FSI Inc."      ,,"C",41)
      SetGVM("NameB"      ,"222 Newbury St." ,,"C",41)
      SetGVM("AddressA"   ,"Northwest City, FL 99999" ,,"C",41)
      GoTo exit:
    End
  Else
    SuppressBanner()
    GoTo exit:
  End
End
exit:
EndSub

BeginSub PstTrans
EndSub

```

The RCBDFDFL.DAT file contains the following GVM variable definitions which are defined in the RecipMap2GVM control group:

- Name1
- Name2
- Address1
- Address2
- CityCounty
- AgentName
- AgentID
- OfficeAddress
- TownAndState

Here are two recipient batch records from this example:

```
SAMPCOLB12234567SCOM1FLT1 B2199802232234567890      0      22560
*****001      3724      452Jill Smith      Morris
11111 Oak Circle      Suite 999      Smyrna,
FL 12345      Martin Short Agent      963 Main Street,
Suite 1250 Miami, FL 30202
```

```
FSI CPP4234567FSIM1WIT1 B3199802234234567890      0      30360
*****001      4667      565Suzy Smith      Morris
99934 Oak Circle      Suite 999      SmartBurg,
WI 99999      David Miller Agent      999 Main Street,
Suite 1200 Miami, FL 30202
```

USING CLASS RECIPIENTS

A *class recipient* identifies a recipient that represents one or more persons or entities. For instance, in an insurance implementation, you might have a policy that has a several recipients declared as an *Additional Interest*. Instead of declaring each as a separate recipient with separate triggering logic, it is more convenient to declare a single recipient name that represents all those of the same type or class. All members of this class receive virtually identical copies of the document.

In this scenario, you do not have to do anything special to declare a class recipient in your form definitions. Merely determine the appropriate title for this class of recipients and define that name as you would a normal recipient that represents a single entity.

If you want all members of the class to receive identical copies of the document, use the trigger for the recipient to assign a copy count to each form or section — where the count equals the number of members in the class.

There are some limitations to using form copy counts to provide recipient copies. For instance, this does not let you print unique information about each member of the class recipient, as would be necessary on a mailer page, for instance.

NOTE: It is possible to handle this using trigger overflow processing to physically trigger multiple copies of each form — one for each member, but a disadvantage of this approach is that each item (form or section) triggered is physically duplicated in the form set and therefore each requires data processing. This means that if there are a large number of these duplicate recipients, the throughput performance of transactions could be affected.

To handle this situation, the RecipMap2GVM feature can write additional batch records for each member of a class recipient. The RecipMap2GVM feature lets you write unique recipient information to each batch record.

With this method, only a minimal amount of additional processing occurs in the form set mapping. Yet, because a separate batch record is written for each member, the system prints a separate copy of the document for each member and you can use the unique information saved in each batch record to provide a unique banner page, such as a mailer, for each member in the print output.

To use the RecipMap2GVM feature, follow these steps:

- 1 Add a section to your form set definition and assign this section the name of your class recipient. Normally, you would also flag this section as *hidden*, since you would not want it to display or print. This purpose of this section is to hold the unique information for each member of the class recipient.
- 2 Define a trigger for the section that uses overflow to generate as many copies of the section as there are members in the data. The idea is to trigger an instance of the section for each member recipient. Be sure to also declare and create the appropriate overflow variable in the AFGJOB.JDT file you will use during data mapping.
- 3 Create the section and add fields that map the data to be written to the batch record for each member. Be sure to use the appropriate overflow variable for this section in your rule mapping definitions. Also remember to assign the appropriate section level rule to increment the overflow symbol after processing each section.

- 4 Set up your RecipMap2GVM INI control group and modify your RCBDFFDL.DFD (Recipient Table DFD) file to include your unique data fields for the recipient batch records. Specify the new section as the section required in the RecipMap2GVM control group and set up each of the fields to map into your RCBDFFDL.DFD file layout.

NOTE: See [Writing Unique Data into Recipient Batch Records on page 58](#) for more information on the RecipMap2GVM control group.

When you run the GenData program, your new section will trigger once for each member recipient. During normal processing, the fields on each section will map (using overflow variables) the unique data for each member. Because you have multiple copies of the section triggered, the RecipMap2GVM feature creates a separate batch record for each instance of the section. Therefore, you receive a separate record representing each individual member of your class recipient.

When the GenPrint program runs, having a separate record for each class recipient in the batch causes that transaction to print once for each member. And by using banner page processing, you can take the unique information written into each batch record and map that information to a mailer page, making the final output unique to each member of the class.

RUNNING DOCUMAKER USING XML JOB TICKETS

You can run Documaker from another application using an XML job ticket. You receive results in an XML job log file.

The layout of these files is the same as those used by IDS for running Documaker. See [Using IDS to Run Documaker on page 57](#) for more information.

The name of the job ticket is passed to the GenData program on the command line as

```
/jticket= parameter
```

The default name is *JOBTICKET.XML*.

To set this up replace the StandardJobProc rule with the TicketJobProc rule. Keep in mind you must run Documaker in single step mode, since only the GenData program is executed. See [Using Single-step Processing on page 45](#) for more information.

You can specify the name of the resulting job log file using this command line parameter:

```
/jlog=
```

The default is *JOBLOG.XML*.

HANDLING 2-UP PRINTING

Two-up printing lets you print two transactions on the same page of single- and multi-page forms. 2-up printing is a two-step process which passes input through GenData three (3) times, using a different JDT file each time.

This process is similar to the single-step process in that GenData performs the work, but the three passes through GenData actually represent two steps of the multi-step process: processing the transactions and printing the transactions.

For more information and to see example JDT files, see [Single-step Processing Example on page 55](#).

NOTE: 2-up printing is only available for AFP printers.

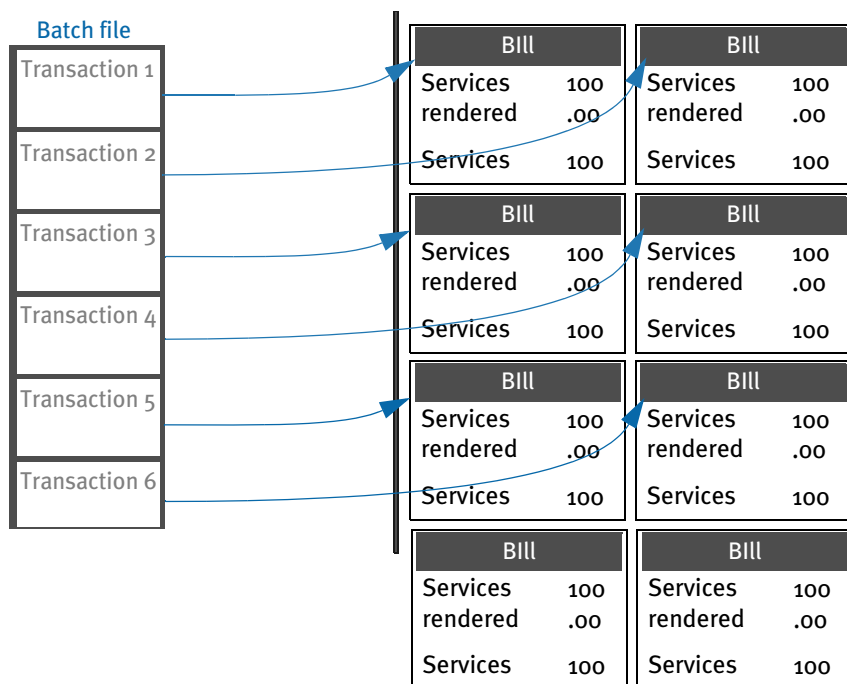
There are several scenarios in which 2-up printing applies:

- 2-up printing with single-page forms
- 2-up printing with multi-page forms

The following illustrations describe these scenarios.

2-up printing with single-page forms

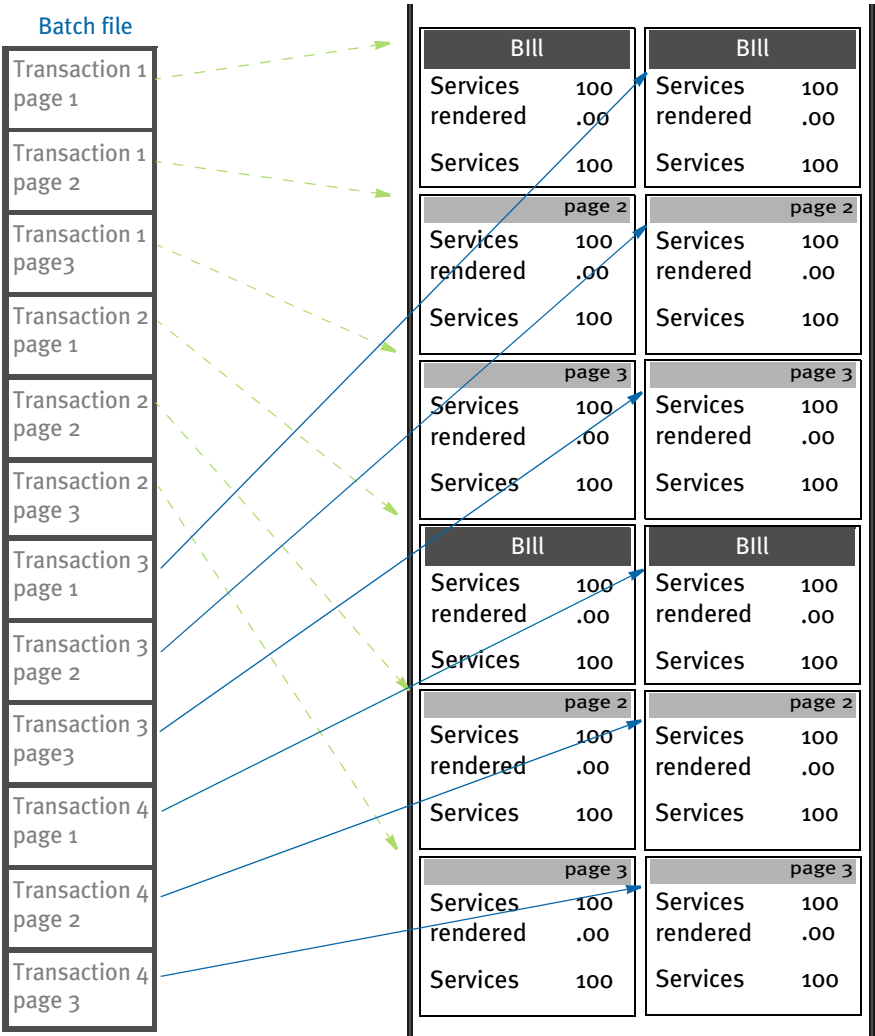
This illustration shows how 2-up printing works when you use single-page forms, such as some types of bills and statements.



In this scenario, the system merges the data for the first transaction onto the form and then prints the form.

2-up printing with multi-page forms

This illustration shows how 2-up printing works when you use multi-page forms.



Changing the INI File

You must make the following changes in your FSISYS.INI file.

NOTE: Changes to the error and manual recipient batch control groups are not necessary.

- You must include a Printer option in the recipient batch control groups for each print file created. These printers must also be defined in the FSISYS.INI file.
- The recipient batch groups must have a FinalPrinter option. This option specifies the printer to use for the final, merged file. This printer must also be defined in the FSISYS.INI file.

- The recipient batch groups must have a PageRange option for page count batching. You specify this option as shown below:

```
PageRange = [min], [max]
```

If you do not specify *min*, the system uses zero (0). If you omit *max*, the system uses (unsigned) -1 (all bits on). The *min* and *max* values are inclusive.

- You can also include in the recipient batch control groups a TwoUpStart option, which can have any of these values (case is irrelevant):
 - L
 - Left
 - R
 - Right

This option specifies whether the merge process should associate the first Printer option with the left or the right side of the page. The system only checks this option when there are multiple Printer options present in the control group. If you omit this option, the file specified in the first Printer option is used for the left side of the page.

Here is an example of a recipient batch control group:

```
< Batch1 >
Printer      = Printer1
Printer      = Printer2
FinalPrinter = Printer3
PageRange    = ,1
TwoUpStart   = R
```

This splits single page transactions evenly between the files specified in the Printer1 and Printer2 control groups. The files specified in the Printer1 and Printer2 control groups will then be merged into the file specified in the Printer3 control group. The file specified in the Printer1 control group is used for the right page.

Creating the TWOUP control group

You must create the TwoUp control group. This control group must contain the CounterTbl option, which specifies the file name for the table that contains recipient batch page counts.

The TwoUp control group can optionally contain the CounterDFD option, which specifies the name of a DFD file. See the [Rules Reference](#) for information about this DFD.

The TwoUp control group can optionally contain the LMargin, LShift, and RShift options. Records on the left page will be shifted to the right by LShift - LMargin, and records on the right page will be shifted to the right by RShift - RMargin. Amounts are in FAP units (2400 per inch). If you omit these options, the system uses these defaults:

```
LMargin = 600
LShift  = 1200
RShift  = 16800
```

```
< TwoUp >
CounterTbl = data\counter.tbl
CounterDFD = deflib\counter.dfd
LMargin    = 300
```



```
LShift = 600
RShift = 15000
```

The first two options define the location of the files shown above.

The LMargin=300 option sets the left margin to 1/4 inch. The LShift=600 option shifts the left page 1/2 inch from the left edge of the paper (1/4 inch beyond the left margin). The RShift=15000 option shifts the right page 6 1/2 inches the left edge of the paper (6 inches from the left margin).

Creating the Added_Fonts control group

You can optionally create the Added_Fonts control group. The options in this group specify additional fonts to add to the AFP output file for text label records which may be added during the merge process. Each option takes the form:

```
FontName = <font name>
```

Here is an example:

```
< Added_Fonts >
  FontName = XOFATIN0
  FontName = XOFAUNN8
```

This tells the system to include the fonts XoFATINo and XoFAUNN8 in the final output file, regardless of whether they are present in the input files.

Changing the Recipient Batch DFD File

The recipient batch DFD file (RCBDFDFL.DAT) must have the following fields with the given types. You can modify the field lengths—just make sure you set the EXT_LENGTH option large enough to represent all of the pages in a multi-mail transaction set. Also make sure you set the INT_LENGTH option larger by one than the EXT_LENGTH option.

Note that the field name is case sensitive. Also, for each of these fields, be sure to add a FIELDNAME= line to the <FIELDS> line in the DFD file.

```
< FIELD:CurPage >
  INT_Type   = CHAR_ARRAY
  INT_Length = 5
  EXT_Type   = CHAR_ARRAY_NO_NULL_TERM
  EXT_Length = 4
  Key        = N
  Required   = N
< FIELD:TotPage >
  INT_Type   = CHAR_ARRAY
  INT_Length = 5
  EXT_Type   = CHAR_ARRAY_NO_NULL_TERM
  EXT_Length = 4
  Key        = N
  Required   = N
< FIELD:AccumPage >
  INT_Type   = LONG
  EXT_Type   = CHAR_ARRAY_NO_NULL_TERM
  EXT_Length = 10
  Key        = N
  Required   = N
```

RULES USED FOR 2-UP PRINTING

The following descriptions will help familiarize you with the rules that are required to perform the 2-up printing process. All of the rules listed in the topic, [Rules Used in Single-step Processing on page 52](#) are required for 2-up printing, plus these additional rules:

NOTE: You can find more information in the [Rules Reference](#).

AddLine	Use this form set level (level 2) rule to add a line record, such as for OMR marks, to the AFP record list built by the MergeAFP rule.
AddTextLabel	Use this form set level (level 2) rule to add a text label record to the AFP record list built by the MergeAFP rule.
ForceNoImages	Use this section (image) level rule (level 3) to return the msgNO_MORE_IMAGES message. This prevents errors if you have no section level rules.
GetRCBRec	<p>Use this form set (level 2) level rule to set the current recipient batch file. This rule initializes the current recipient batch file, if necessary.</p> <p>This rule also sets the first printer for current batch to be the current printer and retrieves the next record from the current recipient batch file.</p>
InitMerge	Use this job level (level 1) rule to create a list of printers, batches, and buffers for the comment (RCB) records. This rule also creates a list to hold AFP records and AFP fonts. After the system finishes running the rule, it deletes everything the rule created.
<hr/> <p>NOTE: The recipient batch files are not used at this stage. The batch list must be created beforehand so the system will know which print files belong together. The <i>skipping batch</i> message is an artifact of the batch file loading process.</p> <hr/>	
InitPageBatchedJob	<p>Use this job level (level 1) rule to open NA and POL files. This rule installs the section level callback function for inserting recipient batch records into the AFP print stream as AFP comment records.</p> <p>When finished, this rule restores the original callback function and closes the NA and POL files.</p>

MergeAFP Use this form set level (level 2) rule to initialize input files. This rule populates the AFP record list, retrieves comment (RCB) records, and terminates the input files.

This rule also initializes output files, and writes out the AFP record list, adding end page and end document records as necessary. The rule then terminates these output files.

ParseCommentExample Use this form set level (level 2) rule to parse comment records into the GVM variable.

PrintData Use this form set (level 2) rule to print the form set. This rule is used for handling 2-up printing on AFP and compatible printers.

NOTE: The section handler installed by the InitPageBatchedJob rule is called during the printing stage. If you want to make any modifications to the recipient batch record, you must do so before this point.

ProcessRecord Use this form set (level 2) rule to switch between print files as necessary when printing 2-up forms on an AFP printer. This rule updates the page count for current print file and loads and merges the form set.

Placing the 2-up Rules in the JDT File

When you use the rules listed at the beginning of this topic to handle 2-up printing, you must place them in the correct places and order in the AFGJOB.JDT file. Use the following table as a guide to where to place these rules. You can insert other rules before, between, or after the 2-up rules—just keep the 2-up rules in the order indicated below with respect to one another.

Stage 1	
Job level	Insert the PageBatchStage1InitTerm rule after the RULStandardJobProc and JobInit1 rules
Form set level	List the form set level rules in this order: WriteOutput CreateRecordList BatchByPageCount PaginateAndPropagate Place these rules after the RULStandardTransactionProc rule and make sure any rule which changes page count appears before these rules.
Stage 2	
Job level	Include these rules in this order: InitPrint InitPageBatchedJob SetErrHdr <i>Do not</i> include the RULStandardJobProc or JobInit1 rules in this stage.
Form set level	Include these rules in this order: GetRCBRec ProcessRecord PrintData <i>Do not</i> include the RULStandardTransactionProc rule in this stage.
Section (image) level	There are no regulations on the order in which you can place rules in this stage. Remember, however, that if there are no section level rules, you must include the ForceNoImages rule to avoid errors.
Stage 3	
Job level	Place the InitMerge rule anywhere after the RULStandardJobProc rule.
Form set level	Make sure the MergeAFP rule is the first rule called. Place rules which add records or determine whether a page pair should be printed after the MergeAFP rule.

Section level	There are no stipulations on the order in which you must place rules in this stage. Remember, however, that if there are no section level rules, you <i>must</i> include the ForceNoImages rule to avoid errors.
---------------	--

2-UP PROCESSING EXAMPLE

As stated earlier, 2-up printing is a two-step process which calls GenData three times with different JDT files. These file excerpts show how to set up your batch and INI files:

2upbycnt.bat

You can set up this batch file as follows:

```
@Echo Off
SetLocal
Echo Y|Del Data\*. * >NUL
GenDaW32.Exe -INI=2upstep1.ini
If Not ErrorLevel 5 GoTo Step1NoError
    Echo "2Up Printing Failed in Step 1."
    GoTo Exit
:Step1NoError
GenDaW32.Exe -INI=2upstep2.ini
If Not ErrorLevel 5 GoTo Step2NoError
    Echo "2Up Printing Failed in Step 2."
    GoTo Exit
:Step2NoError
GenDaW32.Exe -INI=2upstep3.ini
If Not ErrorLevel 5 GoTo Step3NoError
    Echo "2Up Printing Failed in Step 3."
:Step3NoError
EndLocal
:Exit
```

2upstep1.ini

You can set up this INI file as follows:

```
< Data >
    AfgJobFile      = .\Def\AfgJob1.jdt
< Environment >
    FSISYSINI       = .\fsisys.ini
```

2upstep2.ini

You can set up this INI file as follows:

```
< Data >
    AfgJobFile      = .\Def\AfgJob2.jdt
< Environment >
    FSISYSINI       = .\fsisys.ini
```

2upstep3.ini

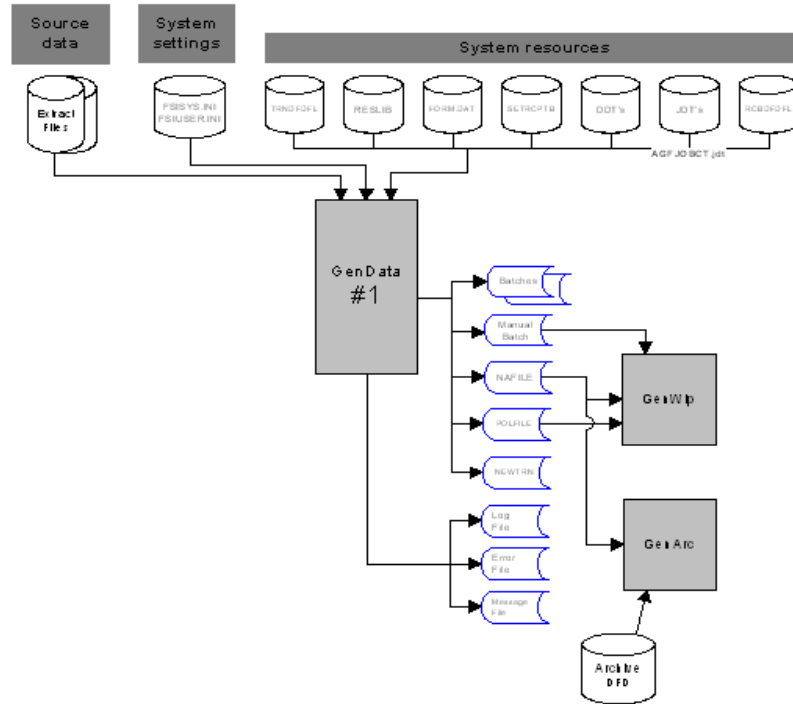
You can set up this INI file as follows:

```
< Data >
    AfgJobFile      = .\Def\AfgJob3.jdt
< Environment >
    FSISYSINI       = .\fsisys.ini
```

RUNNING THE GENDATA PROGRAM

The following pages provide illustrations and an example files for each time the GenData program is run.

Step 1 - Using the AFGJOB1.JDT file



The first execution of GenData uses the AFGJOB1.JDT file with the base and form set rules shown in this example to create output files shown in the illustration.

```

<Base Rules>
;RULStandardJobProc;1;;
;SetErrHdr;1;***:-----;
;SetErrHdr;1;***: BillPrint Data Generation (Base);
;SetErrHdr;1;***:
;SetErrHdr;1;***: Transaction:      ***ACCOUNTNUM***;
;SetErrHdr;1;***: Company Name:    ***Company***;
;SetErrHdr;1;***: Line of Business: ***LOB***;
;SetErrHdr;1;***: Run Date:       ***RunDate***;
;SetErrHdr;1;***:-----;
;
;JobInit1;;
;CreateGlbVar;1;TXTLst,PVOID;
;CreateGlbVar;1;TblLstH,PVOID;
;InitOvFlw;1;;
;SetOvFlwSym;1;SUBGROUPOVF,SUBGROUP,5;
;BuildMasterFormList;4;
;PageBatchStage1InitTerm;;
;InitSetrecipCache;;

<Base Form Set Rules>
;NoGenTrnTransactionProc;;
  
```

```

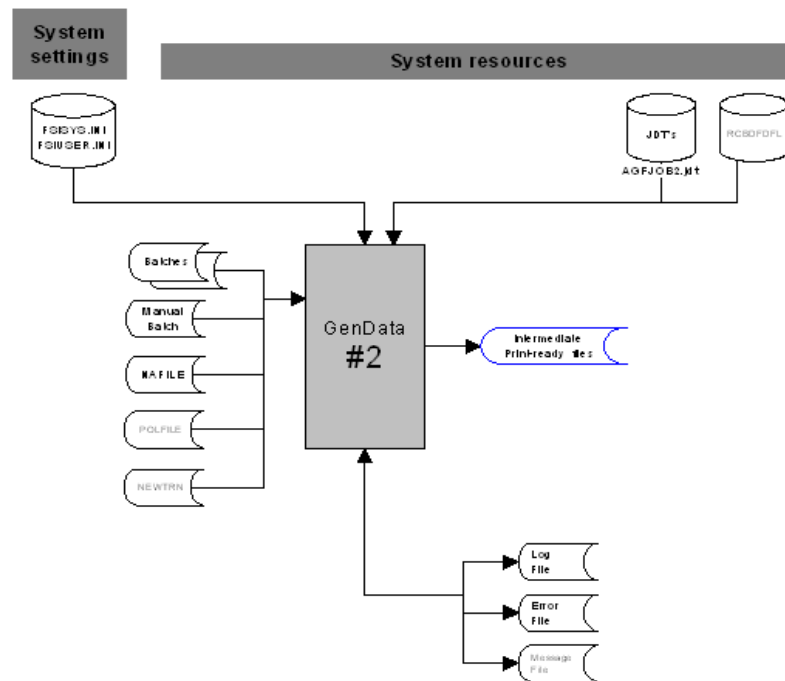
;ResetOvFlw;2;;
;BuildFormList;;;
;LoadRcpTbl;;;
;RunSetRcpTbl;;;
;WriteNaFile;;;
;WriteRCBWithPageCount;;;
;ProcessQueue;;PostPaginationQueue;
;WriteOutput;;;
;CreateRecordList;;
;BatchByPageCount;;
;PaginateAndPropagate;;;

<Base Image Rules>
;StandardImageProc;3;Always the 1st image level rule;

<Base Field Rules>
;StandardFieldProc;4;Always the 1st field level rule;

```

Step 2 - Using the AFGJOB2.JDT file



The second execution of GenData uses the AFGJOB2.JDT file. This JDT file uses the base and form set rules shown in this example to process the intermediate print files.

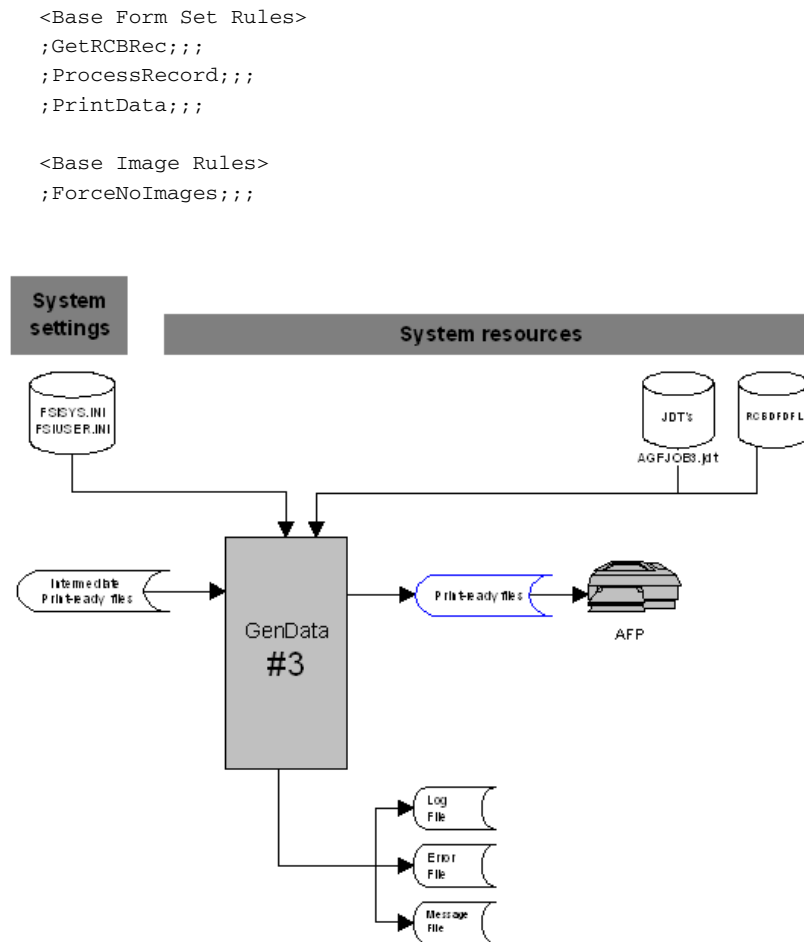
```

<Base Rules>
;InitPrint;;;
;InitPageBatchedJob;;;

;SetErrHdr;1;***:-----
;SetErrHdr;1;***: BillPrint Data Generation (Base) ;
;SetErrHdr;1;***: Company Name: ***Company***;
;SetErrHdr;1;***: SubCompany:***SubCompany***;
;SetErrHdr;1;***: Account #: ***AC-KY-BA***;
;SetErrHdr;1;***:-----

```

Step 3 - Using the
AFGJOB3.JDT file



The third execution of GenData uses the AFGJOB3.JDT file. This JDT file uses base and form set rules shown in this example to merge data intermediate print-ready files into a print-ready file for an AFP printer.

```

<Base Rules>
;RULStandardJobProc;;;
;SetErrHdr;1;***:-----;
;SetErrHdr;1;***: BillPrint Data Generation (Base);
;SetErrHdr;1;***: Company Name:   ***Company***;
;SetErrHdr;1;***: SubCompany:     ***SubCompany***;
;SetErrHdr;1;***: Account #:      ***AC-KY-BA***;
;SetErrHdr;1;***:-----;
;InitMerge;;;

<Base Form Set Rules>
;MergeAFP;;;

<Base Image Rules>
;ForceNoImages;;;

```


SPLITTING RECIPIENT BATCH PRINT STREAMS

The GenPrint program and the PrintFormset rule (when running in single-step mode) are designed to produce one print stream output file for each recipient batch. This print stream output file includes all of the transactions in the recipient batch.

Sometimes, however, you may want to split the print stream output into multiple print stream output files. For instance, you can use this feature to split your batches into files that reflect the amount of paper you can load into your printer at one time.

You can use DAL scripts to set up criteria for splitting the output file to reflect almost any scenario. For example, it can be based on a certain number of transactions, a maximum number of sheets of paper, or on changes in variables in the recipient batch.

NOTE: Some types of print streams require one file per transaction, such as RTF, PDF, and HTML. The typical way of handling this is via the multi-file print callback method, but this feature provides an alternate method which gives you greater control over the naming of the output file.

To do this you use the PrintFormset rule and these DAL functions:

- DeviceName
- SetDeviceName
- BreakBatch
- UniqueString

This rule and these DAL functions let you:

- Split recipient batches into multiple print stream files
- Assign names to those print stream files

For example, here are some things you can do:

Splitting batches by sheet count

You can use these functions to split a batch based on the sheet count during the GenPrint process. Once a batch reaches a certain number of sheets, you can tell the system to:

- Finish processing the current transaction
- End the current print file. (If you are using a post-transaction or post batch banner page, it will print before the file is closed.)
- Repeat this process when the next print file reaches the specified number of sheets

You can use virtually any logic to decide when to break the batch. For instance, to break based on sheet count, use the TotalSheets function to get the number of sheets to maintain a counter across the transactions.

NOTE: Be sure to reset the sheet count variable in the pre-batch banner DAL script.

Here is an example of DAL script logic that might appear in a post-transaction banner DAL script:

```
IF TotalSheets() > 16000
  #COUNTER += 1
  CurFile = DeviceName()
  Drive = FileDrive(CurFile)
  Path = FilePath(CurFile)
  Ext = FileExt(CurFile)
  RecipBatch = RecipBatch()
  NewFile = FullFileName(Drive, Path, RecipBatch & #COUNTER, Ext)
  SetDeviceName(NewFile)
  BreakBatch()
END
```

NOTE: See [Using DAL to Manipulate File Names on page 82](#) for information on using DAL functions to manipulate file names.

Creating PDF output

You can also modify the above script to unconditionally break the batch after each transaction. Assuming you used the SetDeviceName function to assign a proper file name, each recipient printed would receive a separate output file.

This is particularly useful for output types such as PDF, which require a separate file for each transaction.

NOTE: You can also use the multi-file print callback method in GenPrint to get separate files. Similarly, the single-step processing mode currently uses this INI option:

```
< PrintFormset >
  MultiFilePrint = Yes
```

to tell the system to generate separate files for each transaction. Single-step mode automatically generates a unique file name and offers no way to override that name. By using the BreakBatch and SetDeviceName functions, however, you can control the names assigned to the files in single-step mode. To emulate the action of the current code, use the UniqueString function.

DAL functions

Here is a summary of the DAL functions you would use. Keep in mind...

- These print drivers are supported: PCL5, PCL6, PST, MET, AFP, PDF, HTML, and RTF.
- These print drivers *are not* supported: EPT, MDR, and GDI.
- All platforms are supported, but note that while UniqueString is supported on z/OS, z/OS does not support long file names.
- Producing PDF files on z/OS requires a separate license. Contact your sales representative for more information.
- Both multi-step and single-step processing are supported.

The only DAL function actually involved in splitting the print stream is BreakBatch. The others make it easier to implement this functionality. For example, since you need to name the new print stream, you use the SetDeviceName procedure. To find the name of the current device, you use the DeviceName function. If you need to create unique file names, you can use the UniqueString function.

NOTE: While you can call all of these DAL functions in the Rules Processor or Entry, the BreakBatch and SetDeviceName functions are not applicable in Entry since it does not use the batch printing engine. The other functions, DeviceName and UniqueString, are applicable to both Entry and the Rules Processor.

DeviceName

Use this function to return the current output device file name, such as the name of the current print stream output file.

Syntax `DeviceName()`

SetDeviceName

Use this procedure to set a new output device file name which will be used the next time the output device is opened, assuming nothing overrides the name prior to that.

Syntax `SetDeviceName(Device)`

BreakBatch

Use this procedure to tell the Rules Processor to break the output print stream file for the current recipient batch after processing the current recipient, including post transaction banner processing.

Syntax `BreakBatch()`

The procedure is typically called in the transaction banner DAL script. You must use the SetDeviceName function to specify a new device name. Otherwise, the new file has the same name as the old file and overwrites its contents.

After the GenPrint program finishes processing the current transaction, it closes the current output device file. This includes executing any post-batch banner processes. It then continues processing the recipient batch.

If you have assigned a new output device file name using the SetDeviceName function, the system will create and start writing to a new print stream file with that name. The best place to call the BreakBatch function is in the post-transaction banner DAL script.

UniqueString

Use this function to return a 45-character globally unique string.

Syntax

```
UniqueString()
```

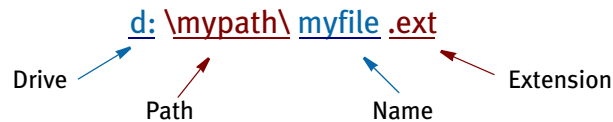
Here is an example:

```
DataPath = GetINIStrString(, "Data", "DataPath")
Drive = FileDrive(DataPath)
Path = FilePath(DataPath)
UniqueID = UniqueString()
Outputname = FullFileName(Drive, Path, UniqueID, ".PDF")
SetDeviceName(Outputname)
```

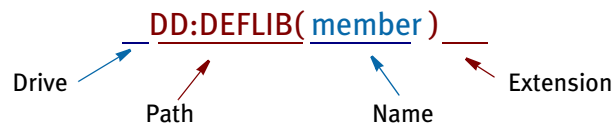
USING DAL TO MANIPULATE FILE NAMES

Since you can use DAL functions to read tables and to set device names for output print stream files, this feature further extends DAL functionality by letting you manipulate file names.

For instance, you can get the components of a file name (drive, path, name, and extension) and combine those into a full file name. For example, for computers running Windows file names look like this:



For computers running OS/390, file names look like this:



In this z/OS example, the drive and extension are omitted, because they are not applicable on z/OS and the parentheses enclosing *member* are part of the path.

To do this you use these DAL functions:

- FileDrive
- FilePath
- FileName
- FileExt
- FullFileName

All platforms are supported and both the Rules Processor and the Entry system are supported.

Each platform will use platform specific logic to extract or assemble the components. For example, UNIX uses forward slashes and OS390 uses DD names or partitioned dataset names for the *path* and member names for *name*.

Here are descriptions of these functions:

FileDrive

Use this function to get the drive component of a file name.

Syntax

```
FileDrive("FullFileName")
```

This function accepts a string containing a fully qualified file name, returns a string that contains the drive component of that file name.

Here is an example:

```
MYDRIVE = FileDrive("d:\mypath\myfile.ext")
```

In this example, MYDRIVE would contain:

"d:"

FilePath

Use this function to get the path component of a file name.

Syntax

```
FilePath("FullFileName")
```

This function accepts a string containing a fully qualified file name, returns a string that contains the path component of that file name.

Here is an example:

```
MYPATH = FilePath("d:\mypath\myfile.ext")
```

In this example, MYPATH would contain:

"\mypath\"

FileName

Use this function to get the name component of a file name.

Syntax

```
FileName("FullFileName")
```

This function accepts a string containing a fully qualified file name, returns a string that contains the name component of that file name.

Here is an example:

```
MYNAME = FileName("d:\mypath\myfile.ext")
```

In this example, MYNAME would contain:

"myfile"

FileExt

Use this function to get the extension component of a file name.

Syntax

```
FileExt("FullFileName")
```

This function accepts a string containing a fully qualified file name, returns a string that contains the extension component of that file name.

Here is an example:

```
MYEXT = FileExt("d:\mypath\myfile.ext")
```

In this example MYEXT would contain:

“.ext”

FullFileName

Use this function to make the full file name.

Syntax

```
FullFileName("Drive", "Path", "Name", "Ext")
```

This function accepts a string containing the drive, path, name, and extension components of a fully qualified file name, assembles them, and returns a string that contains the full file name.

Here is an example:

```
MYFILENAME = FullFileName("d:", "\mypath\", "myfile", ".ext")
```

In this example, MYFILENAME would contain:

“d:\mypath\myfile.ext”

NOTE: If, in this example, *\mypath* had no trailing slash, the FullFileName function would have added it for you.

Here is a z/OS example:

```
FullFileName(, "DD:DEFLIB()", "MEMBER")
```

In this example, the result would be:

DD:DEFLIB(MEMBER)

ASSIGNING PRINTER TYPES PER LOGICAL BATCH PRINTER

Recipient batches often need to be sent to different types of printers. For example, you could have a situation where you want to generate PDF files with one batch, email another batch, and send the rest of the batches to a Metacode printer.

In addition, logical printers may also need different callback functions. For example, one batch might print Metacode and need OMR marks created in a callback function while another batch may need to be split by transaction using the MultiFilePrint callback function.

NOTE: Before version 11.1, the print system only supported one type of printer and only one type of callback per run. You made this assignment using the `PrtType` option in the Printer control group.

You can optionally define for each logical printer a printer type and a callback function. For instance, now the `PrtType` option in the Printer control group defines the default type of printer while the `CallbackFunc` option defines the default callback function you want to use.

Here is an example:

```
< Printer >
  PrtType      = XER ; Default
< Printers >
  Printer      = Printer1
  Printer      = Printer2
< Insured >
  Printer      = Printer1
< Agent >
  Printer      = Printer2
< Printer1 >
  Port         = Output1.XER
< Printer2 >
  Port         = Output2.PDF
  CallbackFunc = RULMultiFilePrint
  PrtType      = PDF
```

When you define a callback function, such as shown below, you are defining the default callback function for *all* defined logical printers:

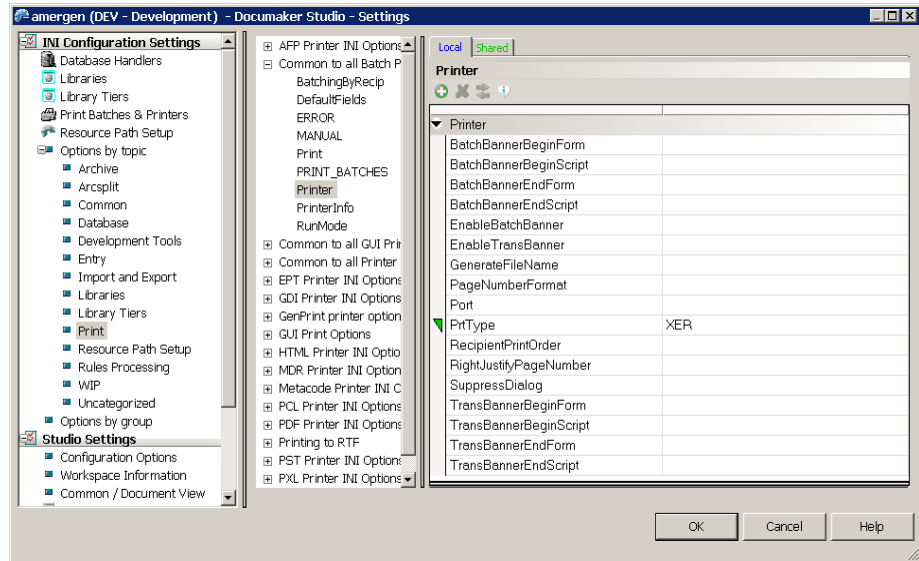
```
< Print >
  CallbackFunc = Mycallback
```

If, however, you do not want a specific logical printer to have a callback function, you can disable the callback for that logical printer by leaving blank the `CallbackFunc` option for that logical printer, as shown here:

```
< MyPrinter >
  CallbackFunc =
```

To disable the default callback, define an empty callback name. Otherwise, the system uses the default callback function.

You can also set these INI options using Documaker Studio's Manage, System, Settings option. Here is an example:



Keep in mind this applies to...

- A batch of transactions. Each transaction within that batch will print to a single type of printer.
- Both single- and multi-step processing of transaction batches.

Single-step processing has limitations as compared to multi-step processing and this feature does not remove those limitations. Single-step processing optimizes the processing of transactions that do not require recipient batching. Single-step processing is, therefore, intended for use with a single input batch of transactions for a single recipient or a single transaction with one or more recipients, such as in real-time processing.

While you can specify multiple printers and associate a different printer per recipient batch, single-step processing can still only process a single recipient batch at a time. Therefore, it is not possible to do the same type of multi-batch processing in single-step as is done in multi-step processing. A given set of transactions can specify a single recipient and you can map that recipient to a different type of printer.

Real-time transaction processing of single transactions may also benefit from this by using the multi-file callback method to split output files, along with necessary logic to create unique file names for each output file. When used in this manner, single-step processing of a single, real-time transaction can call a different driver for each recipient in the transaction.

CONTROLLING WIP FIELD ASSIGNMENTS

You can use options in the Trigger2WIP control group to set almost all of the WIP record fields for each transaction.

NOTE: Do not try to set the ModifyTime, InUse, or the FormSetID fields of the WIP record. The ModifyTime field is assigned by the system when a WIP record is added or updated. If you need to save a date and time for the transaction, store that information in the CreateTime field, using the hextime X format for the destination as shown in one of the examples.

The InUse field is used internally to prevent multiple people from editing the same transaction. Let the system manage this column.

The FormSetID is assigned by the system when a new WIP transaction is created. Let the system handle this.

The Trigger2WIP control group defines which recipient batch (RCB) transaction fields from the manual batch (those kicked to WIP) are mapped to the corresponding WIP transaction record fields.

The options under the Trigger2WIP control group define the mappings as shown here:

The contents of this field... `< Trigger2WIP >` ...is copied into this field

```

RCBField 1 = WIPField 1
RCBField 2 = WIPField 2
...
```

RCBField represents one of the fields defined by the batch transaction record definition (RCBDFDL.DFD). *WIPField* represents a field defined in the WIP database.

NOTE: There may be an external WIP.DFD file that identifies the fields in a WIP record. An external DFD file is not required if you are using the default WIP database layout.

Note, although the normal mapping technique is to name a RCB field on the left side, the left side can name any defined GVM (global variable member). Typically, the only GVMs that exist during GenWIP processing are those defined in the RCB DFD file, but custom applications or single-step WIP systems may have additional GVMs.

The changes in this release support this INI definition and also let you convert data or define a constant value you want to map to a WIP field. For a data conversion, define your INI options as shown here:

```

< Trigger2WIP >
RCBField = WIPField; input format ; output format;
```

The conversion information must appear on the right side of the INI option, after the WIPField name definition. Separate it from the named variable with a semicolon. Here is an example:

```
RunDate = CREATETIME;DD4;X
```

The first semicolon denotes the input format of the data. The second separates the desired output format. In this example, the input format of *DD4* means the source data is a date field in the format *D4*, which is YYYYMMDD.

The output format *X* indicates you want to convert the date value to the internal HEXTIME format used in the WIP CreateTime field.

NOTE: For more information selecting from the pre-defined date formats or defining your own, see the [Rules Reference](#).

Although conversions are often used to change date formats, you can also use them to do additional formatting. The system supports a simple C style sprintf (%s) and constant text formatting, like %s, %10.10s, %-38.38s, and so on. The system does not support any of the other C style formats flags that assume non-text data or asterisk (variable width) designations.

Here is an example:

```
EFFVALUE = APPData; ;(%s%%)
```

Suppose in this example, that EFFVALUE contains the text *10*, the resulting value mapped into the APPData field will read (*10%*).

NOTE: You must use two percent signs (%%) to represent a single percent sign in the output. The system only supports a string %s type format. It does not support numeric data formats of any type.

Normally, the left side of the INI option names a field from the RCB file definition. You can also enter *NULL* as a keyword to mean there is no corresponding RCB data field to associate with the WIP field. This lets you assign the constant data to the WIP field, as shown here:

```
NULL = DESC; ;ABC123 HERE WE GO
```

This example shows how to assign the constant text *ABC123 HERE WE GO* into the DESC field of the WIP record. NULL indicates there is no source variable to associate with this destination field.

You can also use INI built-in functions to provide a constant value to map to the field. For example:

```
NULL = CURRUSER; ;~GETENV USERNAME
```

INI built-in functions are preceded with a tilde (~). This example executes the GETENV built-in INI function, which gets the environment variable USERNAME. If you assume the variable contained the text *TOM*, the WIP variable CURRUSER would be assigned *TOM* after execution of the built-in function.

These options show the defaults used if the Trigger2WIP control group does not override the variables:

```
< AFG2WIP >
  StatusCode = WIP
  RecordType = NEW
  UserID     = DOCUCORP
```

The `StatusCode` option defines which INI option in the `Status_CD` control group to use as the default WIP StatusCD field. Suppose you have the following `Status_CD` control group defined.

```
< Status_CD
    WIP           =W
    Assign        =A
    Quote         =Q
    BatchPrint    =W
    Archive       =AR
    Printed       =P
```

This means a *W* would be assigned to the WIP StatusCD field (usually meaning a normal WIP transaction).

The `RecordType` option defines which INI option to locate in the `Record_Type` control group as the default setting for WIP RecType. Suppose you have these options defined:

```
< Record_Type >
    New          =00
    Assign       =01
    Partial      =02
```

New is the normal default for the AFG2WIP control group and would therefore map *00* into the WIP RecType field.

The `UserID` option defines which user should be assigned the WIP transactions in the `CURRUSER` field. Unless this option is changed or the `CURRUSER` field assigned from the `Trigger2WIP` control group, the system defaults this value to `DOCUCORP`. `DOCUCORP` is one of the default users created in a default user database.

You would normally want to add an option to the AFG2WIP control group to name a valid user in your company, otherwise, users will have to log in as `DOCUCORP` and reassign the WIP to valid users later.

GENERATING EMAIL NOTIFICATIONS FROM GENWIP

You can enable the GenWIP program to send email. The GenWIP program will generate an email message by processing a message body template against variable data in the manual batch. It then sends the message when the document is added to WIP.

NOTE: See also [Emailing a Print File on page 342](#).

Email-specific data can be in the recipient batch read by the GenWIP program or in the INI file. The system checks the recipient batch first. If the field is not present or blank, the system then checks the INI option.

Below is a list of the fields the GenWIP program looks at to get email information. If you want to include other fields, you can use the INI built-in function to accomplish this.

Email is enabled in the GenWIP program when there is both a send-to email address and a subject or message body. The message body is expected to be in a separate file. Email attachment files are also supported and are processed as template files the same as the message body. You use these INI options to enable email processing:

```
< GenWIPEmail >
  EnableEmailNotification=
  MailMessageBody        =
  MailID                  =
  MailSubject              =
  MailAttachment          =
```

Option	Description
EnableEmailNotification	Enter Yes.
MailMessageBody	Enter the path and file name for the email template.
MailID	The email address to send. This is optional if the MAILID is omitted, you can send using this address.
MailSubject	If the MAILSUBJECT is missing or blank, the system will use the text you enter here as the Subject.
MailAttachment	The name of the file to attach.

These field names to go into the RCBDFDFILE:

```
FIELDNAME = MAILID
FIELDNAME = MAILATTACHMENT_IN
FIELDNAME = MAILATTACHMENT_OUT
FIELDNAME = MAILSUBJECT
FIELDNAME = MAILIDFROMADDRESS
FIELDNAME = MAILMESSAGEBODY
```

Group: < FIELD:MAILID >entries:

```
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 51
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
EXT_LENGTH = 50
KEY = N
REQUIRED = Y
```

Group: < FIELD:MAILATTACHMENT_IN > entries:

```

INT_TYPE = CHAR_ARRAY
INT_LENGTH = 129
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
EXT_LENGTH = 128
KEY = N
REQUIRED = Y

```

Group: < FIELD:MAILATTACHMENT_OUT > entries:

```

INT_TYPE = CHAR_ARRAY
INT_LENGTH = 129
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
EXT_LENGTH = 128
KEY = N
REQUIRED = Y

```

Group: < FIELD:MAILMESSAGEBODY > entries:

```

INT_TYPE = CHAR_ARRAY
INT_LENGTH = 129
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
EXT_LENGTH = 128
KEY = N
REQUIRED = Y

```

Group: < FIELD:MAILSUBJECT > entries:

```

INT_TYPE = CHAR_ARRAY
INT_LENGTH = 129
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
EXT_LENGTH = 128
KEY = N
REQUIRED = Yes

```

Errors

Here are the error messages that can appear:

Error	Description
11226	Error in GENCreateEmail(): Unable to get <&Name&> does it exist in rcb dfd file?\n
11227	Error in GENCreateEmail(): Unable to process template check file <&filename&> for valid markup syntax\n
11228	Error in GENCreateEmail(): Unable to open file <&Name&>\n
11229	Error in GENCreateEmail(): Unable to QueryAPI <&apiname&> check for valid path to DLL <&dllname&>\n
11230	Error in GENCreateEmail(): Unable to Logon to email server\n
11231	Error in GENCreateEmail(): Unable to set <&data&> check INI file for valid <&inigroup&> <&inioption&>\n
11232	Error in GENCreateEmail(): Unable to get <&data&> check INI file for <&inigroup&> <&inioption&>\n

Error	Description
11233	Error in GENCreateEmail(): failed to send e-mail <&userid> <&emailaddress>\n

USING MULTI-MAIL PROCESSING

Multi-mail processing groups the transactions with the same multi-mail code into selected print batches based on the number of pages defined in the PageRange INI option. Multi-mail can only be handled as a 2-up process. In the INI example below, all transactions with the same multi mail will be stored in a batch category:

```
batch1-less than five pages
batch2-five to nine pages
batch3-10 or more pages
```

The MM_FIELD option in the TRN_Field control group identifies position, length, type of data and where the multi-mail code is located in the transaction record.

NOTE: The parameter has been named MM_FIELD in the above explanation, however it can be given any name.

The BatchByPageCount rule in the AFGJOB.JDT file identifies the name in the TRN_Field control group, as shown here:

```
BatchByPageCount; ;MMFIELD=MM_FIELD;
```

Changing the RCPDFDFL.DAT and TRNDFDFL.DAT Files

You must make the following changes to the RCPDFDFL.DAT and TRNDFDFL.DAT files for multi-mail processing:

```
< Fields >
:::
FIELD:MMField

< FIELD:MMField >
  INT_Type      = CHAR_ARRAY
  INT_Length    = 7
  EXT_Type      = CHAR_ARRAY_NO_NULL_TERM
  EXT_Length    = 6
  Key           = N
  Required      = N
```

Setting Up the FSISYS.INI File for Multi-mail Processing

Here is an example of how the relevant control groups and options in your FSISYS.INI file should look:

```
< Print_Batches >
  P_Batch1      = .\data\Batch1
  P_Batch2      = .\data\Batch2
  P_Batch3      = .\data\Batch3
  Error         = .\data\Error
  Manual        = .\data\Manual
< P_Batch1 >
  Printer       = Batch1_PTR_1
  Printer       = Batch1_PTR_2
  FinalPrinter  = Batch1_PTR_F
  PageRange     = ,4          (controls which batch is used)
  TwoUpStart    = L
< P_Batch2 >
  Printer       = Batch2_PTR_1
  Printer       = Batch2_PTR_2
  FinalPrinter  = Batch2_PTR_F
  PageRange     = 5,9        (controls which batch is used)
  TwoUpStart    = L
< P_Batch3 >
  Printer       = Batch3_PTR_1
  Printer       = Batch3_PTR_2
  FinalPrinter  = Batch3_PTR_F
  PageRange     = 10,99      (controls which batch is used)
  TwoUpStart    = L
< TRN_FIELDS >
  ...
  MM_Field      = 326,6,N    (defines where the multi-mail code
                             is found in each transaction)
```

The order of the page output on the final print file will produce 2-up printing depending on how many intermediate printer files are specified. The output will look as follows:

```
< P_Batch2 >
  Printer       = Batch2_PTR_1   intermediate printer file
  Printer       = Batch2_PTR_2   intermediate printer file
  FinalPrinter  = Batch2_PTR_F   intermediate printer file
  PageRange     = 5,9
  TwoUpStart    = L

transaction #1 mmcode 111 page 1transaction n mmcode 555 page 1
transaction #1 mmcode 111 page 2transaction n mmcode 555 page 2
transaction #1 mmcode 111 page ntransaction n mmcode 555 page 3
transaction #2 mmcode 126 page 1transaction n mmcode 555 page 4
transaction #2 mmcode 126 page 2transaction n mmcode 555 page 5
transaction #2 mmcode 126 page ntransaction n mmcode 555 page n
transaction #3 mmcode 222 page 1transaction x mmcode 865 page 1
transaction #3 mmcode 222 page 1transaction x mmcode 865 page 2
transaction #3 mmcode 222 page ntransaction x mmcode 865 page n
```


If you define only one printer and a final printer for a batch, the 2-up printing would look as follows:

```
< P_Batch2 >
  Printer           = Batch2_PTR_1
  FinalPrinter      = Batch2_PTR_F
  PageRange         = 5,9
  TwoUpStart        = L

transaction #1 mmcode 111 page 1transaction # 1 mmcode 111 page 2
transaction #1 mmcode 111 page 3transaction # 1 mmcode 111 page 4
transaction #1 mmcode 111 page ntransaction # 2 mmcode 555 page 1
transaction #2 mmcode 555 page 2transaction # 2 mmcode 555 page 3
transaction #2 mmcode 555 page 4transaction # 2 mmcode 555 page n
transaction #3 mmcode 126 page 1transaction # 3 mmcode 126 page 2
transaction #3 mmcode 126 page 2transaction # 3 mmcode 126 page n
transaction #4 mmcode 222 page 1transaction # 4 mmcode 222 page 2
```

ADDING AND REMOVING PAGES

You can add and remove blank pages or a FAP file to a form set. Typically, you would add these pages so each printed page has a front and back.

This lets you change a simplex or mixed plex form set into a fully duplexed form set. For instance, you can use this feature to create PDF files for mixed plex form sets that print in a similar fashion to printers that support mixed plex.

You can access this functionality several ways:

- Using custom code
- Using DAL scripts
- Using Docupresentment rules (version 1.6 and higher)

NOTE: Typically, you use this feature to add blank pages just before the print step. These additional pages are not actually part of the saved document.

If, however, if you added the blank pages before the batch steps that save document information to the NA/POL files, the blank pages would become a permanent part of the document layout.

USING CUSTOM CODE

Adding pages

Use this API to call custom code to add blank pages:

```
DWORD _VMMAPI FAPAddBlankPages(  
    VMMHANDLE objectH,          /* form set or form handle */  
    char FAR * sectionname)     /* if NULL, "Blank Page" */
```

If the section name is NULL, a blank page is created when a dummy page is needed. If the section name is not NULL, the section name is loaded when a dummy page is needed. Omit the path and file extension when you enter *sectionname*.

Removing pages

Use this API to call custom code to remove blank pages:

```
DWORD _VMMAPI FAPDelBlankPages(VMMHANDLE objectH)  
/* formset or form handle */
```

USING DAL SCRIPTS

Adding pages

Use this DAL function to add blank pages:

```
AddBlankPages()  
  
or  
  
AddBlankPages('FAPFile')
```

For example, you can use this function with the banner processing feature. First, specify a DAL script that runs at the start of each transaction. The DAL script calls the `AddBlankPages` function. This tells the system to convert each transaction to a fully duplexed form set with blank pages added as needed.

Here is an example of the INI settings you would need:

```

< Printer >
    EnableTransBanner      = True
    TransBannerBeginScript = PreBatch
< DALLibraries >
    Lib                    = BANNER

```

Here is an example of the BANNER.DAL script:

```

BeginSub PreBatch
AddBlankPages()
EndSub

```

Removing pages

Use this DAL function to remove a page from a form set:

```
DelBlankPages()
```

For example, you can use this function with the banner processing feature. First, specify a DAL script that runs at the start of each transaction. The DAL script calls the DelBlankPages function. This tells the system to remove blank pages from each transaction.

```

< Printer >
    EnableTransBanner      = True
    TransBannerBeginScript = PreBatch
< DALLibraries >
    Lib                    = BANNER

```

Here is an example of the BANNER.DAL script:

```

BeginSub PreBatch
DelBlankPages()
EndSub

```

USING IDS

For more information on the rules listed below see [Using the Documaker Bridge](#).

Adding pages

Use this IDS rule to add blank pages:

```
function = dpros2->DPRAddBlankPages
```

This IDS rule assumes the form set being used has been loaded by the Documaker Bridge into the DSI variable, DPRFORMSET. If you are using this rule with a different bridge, you may need to specify a different DSI variable that contains the form set.

To specify a FAP file to use for the dummy pages, add the name of that FAP file after the form set variable name when you specify the IDS rule. Here is an example:

```
function = dpros2->DPRAddBlankPages,DPRFORMSET,FAPFile
```

Removing pages

Use this IDS rule to remove blank pages:

```
function = dpros2->DPRDelBlankPages
```

This IDS rule assumes that the form set has been loaded by the Documaker Bridge into the DSI variable, DPRFORMSET. If you are using this rule with a different bridge, you may need to specify a different DSI variable.

To specify the FAP file being used for dummy pages, add the FAP file name after the form set variable name when you specify the IDS rule. Omit the path and extension. Here is an example:

```
function = dpros2->DPRAddBlankPages,MTCFORMSET
```

ADDING INDEXES AND TABLES OF CONTENTS

Using Documaker Studio or Image Editor, you can insert tables of contents, lists of figures or indexes to your form sets. This makes it easier for users to navigate through the various forms.

To use this feature, all sections must be loaded *before* the print operation executes. Otherwise, the system will not have all the content available and will not be able to create a complete table of contents, list of figures, or index. Since some print drivers do not force the loading of all sections until necessary, this means you may have to include an additional INI option.

For Documaker Server (GenPrint), you would include this option:

```
< RunMode >  
DownloadFAP = Yes
```

USING RUN-TIME OPTIONS

The system offers several ways you can customize the way it runs. The following topics discuss these options.

GENDATA COMMAND LINE OPTIONS

The GenData program accepts several command line options. Command line options are prefixed with either a backslash (/) or a dash (-). Here is an example:

```
c:\fap\mstrres\rpex1\gendaw32 /ini=my.ini
```

The command line options are explained below:

Option	Description
CNT	Overrides the number of transactions specified in the CheckCount option in the Restart control group. This count specifies the frequency of updating offsets for GenData restart processing.
INI	Tells the program to use the specified FSIUSER.INI file instead of the one in the current directory.
JDT	Tells the program to use the specified AFGJOB.JDT file instead of the one defined in the FSIUSER.INI or FSISYS.INI files.
L	Writes the names of the INI files and options the program uses in the log file.
?	Displays the command line options for the program.

GENPRINT COMMAND LINE OPTIONS

The GenPrint program accepts several command line options. Command line options begin with either a backslash (/) or a dash (-). Here is an example:

```
c:\fap\mstrres\rpex1\genptw32 /ini=my.ini
```

The command line options are explained below:

Option	Description
INI	Tells the program to use the specified FSIUSER.INI file instead of the one in the current directory.
L	Writes the names of the INI files and options the program uses in the log file.
?	Displays the command line options for the program.

GENTrn COMMAND LINE OPTIONS

The GenTrn program accepts several command line options. Command line options are prefixed with either a backslash (/) or a dash (-). Here is an example:

```
c:\fap\mstrres\rpex1\genTnw32 /ini=my.ini
```

The command line options are explained below:

Option	Description
B	Tells the program to build only the transaction file.
F	Tells the program to build only the filter extract file.
FB	Tells the program to build only the filter extract and transaction files.
INI	Tells the program to use the specified FSIUSER.INI file instead of the one in the current directory.
L	Writes the names of the INI files and options the program uses in the log file.
?	Displays the command line options for the program.

DEBUGGING OPTIONS

You can use the following options in the Debug_Switches control group to turn on or off debugging options.

```
< Debug_Switches >
  Debug_If_Rule           = Yes
  Enable_Debug_Options    = Yes
  Show_Debug_Options      = Yes
  LoadListFromTable      = Yes
```

Option	Description
Debug_If_Rule	Set to Yes if you want to turn on the debug options for the IF and DAL rules. The system places the debug data in the LOGFILE.DAT file. Setting this option to Yes slows performance.
Enable_Debug_Options	Set this option to Yes to turn on all debug options.
Show_Debug_Options	Set this option to Yes to make the GEN_DEBUG_DebugSwitchSet function log the state (on or off) of all debug options.
LoadListFromTable	Set this option to Yes to make the Gen_TabUtil_LoadListFromTable function log the contents of any ASCII table it loads.

Noting font IDs of zero

You can use the CheckZeroFontID option to tell the system to display a warning or error message if the field being processed contains a font ID equal to zero (0).

Typically, this means no font was assigned during the mapping. Since the merging of FAP and DDT files in version 11.0, the field definition should be complete at the time of processing. So if you encounter a field with no font ID assigned, it probably means some unusual situation has occurred — like the field was defined via an import method but not actually defined on the FAP file where it resides.

Here is an example of the CheckZeroFontID option:

```
< RunMode>
  CheckZeroFontID =
```

Option	Description
CheckZeroFontID	<p>Enter Yes (or Error) to have the system to issue an error message if it encounters a font ID set to zero (0). If you enter Yes (or Error) and the system encounters a font ID of zero, you get a message similar to this:</p> <pre>DM30046: Error: Field < FLDNAME > on Image < IMGNAME > has Font ID = 0. The field may have been included incorrectly or the FAP has not been updated to include the field's definition.</pre> <p>Enter Warn if you want the system to issue a warning message if it encounters a font ID set to zero. If you enter Warn and the system encounters a font ID of zero, you will get a message similar to this:</p> <pre>DM30046: Warning: Field < FLDNAME > on Image < IMGNAME > has Font ID = 0. The field may have been included incorrectly or the FAP has not been updated to include the field's definition.</pre> <p>In either message, <i>FLDNAME</i> and <i>IMGNAME</i> are reflect the appropriate field name and section (image) name.</p> <p>The default is No, which means nothing is checked and no message is issued.</p>

Suppressing elapsed runtime messages

You can suppress the elapsed runtime message by setting the ElapsedTimeStamp option to No. This turns off the elapsed runtime message for the error, log, and trace files. Here is an example:

```
< Control >
  ElapsedTimeStamp = No
```

Option	Description
ElapsedTimeStamp	Enter No to suppress the elapsed runtime message for the error, log, and trace files. The default is Yes.

NOTE: You can use the existing ErrorFileDateStamp and LogFileDateStamp options to turn off the time stamp in the error and log files. The new ElapsedTimeStamp option controls the elapsed runtime message.

GROUPING PRINT BATCHES

If you want to group all of your print batches (BCH files) in one file, follow these steps:

- 1 Add two options to the FSISYS.INI file. In the RunMode control group, set the AliasPrintBatches option to Yes. In the Data control group, add the BatchTable option. Set this option as shown below:

```
BatchTable = <tablename>
```

If you omit the path, the system uses your entry in the DataPath option of the Data control group.

- 2 Add a key to the RCBDFDFL.DFD file. In the Fields control group, add the following option:

```
FieldName = BatchName
```

Add the option exactly as shown here. Do not substitute the desired batch name, here or in any of the following steps.

- 3 Add a corresponding FIELD:BatchName control group. Note that the lengths you specify in this group must be sufficient to hold the batch name (the option side of the equations in the Print_Batches control group). In the Keys control group, add the following option:

```
Key = BatchName
```

and add a corresponding KEY:BatchName control group, with these options:

```
FieldList = BatchName  
Expression = BatchName
```

If you are using ASCII for the print batch, after you run the GenData program you must sort the batch file using the BatchName field as the key. If you are using xBase or DB2, you should be able to run the GenPrint program without this step.

NOTE: If you are using ASCII for the print batches, be sure to place the BatchName field directly before the NA_Offset field in the RCBDFDFL.DFD file. And when sorting, use the BatchName and NA_Offset fields together as the key.

This will help make sure the print output is identical to that produced with multiple batches. If you are using xBase or DB2, you do not need these additional instructions.

CONTROLLING CONSOLE LOGGING

When processing a large number of transactions, you can see how far along you are without affecting performance by using the LogToConsole option. This option lets you control how often the console is updated with progress information.

Using the LogToConsole option, you specify the number of transactions that should be processed before that information is logged on the console. For instance, if your processing run consisted of 10,000 transactions, you could set the option to log progress on the console after every 1000 transactions are processed. Here is an example:

```
< Control >  
LogToConsole = 1000
```

Option	Description
LogToConsole	<p>Enter the number of transactions you want the system to process before it logs its progress on the console. For instance, enter 1000 to have the system tell you each time it processes 1000 transactions.</p> <p>If you leave this option blank or enter Yes, the system logs the processing of <i>each</i> transaction on the console. If you enter a number, such as 1000, the system will send a log message to the console each time it processes that number of transactions.</p> <p>Keep in mind that logging information to the console affects performance. The more often the system logs information to the console, the greater the affect.</p> <p>Consider how many transactions you will process in the run and use that number to determine appropriate progress benchmarks.</p> <p>If you enter No, the system will not notify you of its progress.</p>

LOGGING INI FILE NAMES AND OPTIONS IN THE TRACE FILE

You can log INI file names and options in the TRACE file during GenTrn, GenData, GenPrint, GenArc, and Documaker Studio processing.

To turn on the logging of INI file names and options, include these INI options:

```
< Debug_Switches >  
Enable_Debug_Options = Yes  
INILib = Yes
```

For the GenTrn, GenData, GenPrint, and GenArc programs, you can include the /L command line parameter to log these file names and options in the TRACE file.

NOTE: Logging the INI file names and options in the TRACE file replaces the writing of the INI file names and options to the LOGFILE, as was done in prior to version 11.2.

LISTING THE RULES EXECUTED

Use the following INI options to tell the system to create a list of the Documaker Server rules executed and the amount of time (in milliseconds) spent for each execution:.

```
< Debug_Switches >
  Enable_Debug_Options = Yes
  BaseRuleTime         = Yes
  FormSetRuleTime      = Yes
  ImageRuleTime        = Yes
  ImageFuncTime        = Yes
  FieldFuncTime        = Yes
```

Option	Description
Enable_Debug_Options	Enter Yes to turn on the logging service.
BaseRuleTime	Enter Yes to report base or job-level (level 1) rules.
FormSetRuleTime	Enter Yes to report form set-level (level 2) rules.
ImageRuleTime	Enter Yes to report image-level (level 3) rules.
ImageFuncTime	Enter Yes to report image functions.
FieldFuncTime	Enter Yes to report field functions.

The rule timings are written to a standard debug trace file. Individual records are tab-delimited with the following fields:

Field	Description
Standard Log Trace info	This field tells you the log entries data, time, and process ID. You can omit this information using the PrintTimeStamp option (see below).
Rule Type	This field provides information like: Base Rule Forward, Base Rule Reverse, and so on.
Time Spent Label	The comment label for the Time Spent field: Time Spent (sec)
Time Spent	The time, in milliseconds, spent executing the rule.
Rule Name	The name of the rule. Image functions use this format: "Image Name". "Rule Name" Field functions use this format: "Image Name". "Field Name". "RuleName"

Turn off the time stamp associated with the rule timing options listed above, set the PrintTimeStamp option to No.

```
< Debug_Switches >
  PrintTimeStamp = No
```

ANALYZING DAL PERFORMANCE

In addition to DAL profile information which includes the time spent per function (DAL subroutine), the system places information into the TRACE file about the total time spent in each function and number of times each function is called.

An example of this information is shown below. This example is from a GenData run which processed 600 transactions. The total processing time was 23 seconds. Only the beginning of the log is shown because of space considerations.

The log is sorted by the cumulative time spent in each script with longest running scripts at the top. The log information appears in the trace file and is written out when the program terminates.

You will find this information appears in the log:

Item	Description
Executed XXX	The number of times script was executed.
Cumulative run time X.XXX	The time in seconds dot milliseconds spent in this script and all scripts/code that was executed from this script.
Compiled or Non-compiled	Whether or not the script was compiled.
Name	The name of the script or the actual script if it was not in an external file.

Some scripts look like they are listed twice, but are not. For instance, in the example below PostTrans_Prod() and PostTrans_Prod actually are the script that had a call to PostTrans_Prod (all it had was "PostTrans_Prod()") and the actual PostTrans_Prod DAL subroutine.

When you analyze the log, keep these things in mind:

- The scripts you need to review are usually the scripts at the top of the log.
- Review any scripts that are executed more times than number of transactions in the run. You can probably modify your implementation so the script is run no more than once per transaction or once per job.
- Review the scripts that run the longest and see if they can be optimized. For example, move assignment of variables outside the loop. Consider parts that can be executed only when needed.
- Typically, scripts that take longer to run or receive a higher number of calls are good candidates for review of either the script itself or the implementation.
- Clock resolution is set at one millisecond. If a script executes in less than one millisecond, the time spent equals zero (0). Scripts that show a high number of calls, even if the time is shown as zero (0), or a relatively small number are good candidates for optimization.

NOTE: The extra logging does affect total time spent executing the program being analyzed and should not be turned on in a production environment or left on when not needed.

```

Executed 600 times Cumulative run time 2.840 Non-compiled Script
PostTrans_Prod()
Executed 600 times Cumulative run time 2.824 Compiled Script
PostTrans_Prod
Executed 600 times Cumulative run time 2.451 Non-compiled Script
PREFILL_VARS()
Executed 600 times Cumulative run time 2.420 Compiled Script
PREFILL_VARS
Executed 600 times Cumulative run time 1.954 Compiled Script
DEFLIB\BarCode.DAL
Executed 534 times Cumulative run time 0.792 Compiled Script
DEFLIB>Delete_Images.DAL
Executed 1150 times Cumulative run time 0.784 Non-compiled Script
CALL("SERVPHONENUM")
Executed 1150 times Cumulative run time 0.737 Compiled Script
DEFLIB\SERVPHONENUM.DAL
Executed 600 times Cumulative run time 0.372 Non-compiled Script
COPYCOUNT()
Executed 1813 times Cumulative run time 0.359 Non-compiled Script
call("INSUREDNAME1")
Executed 1813 times Cumulative run time 0.312 Compiled Script
DEFLIB\INSUREDNAME1.DAL
Executed 600 times Cumulative run time 0.295 Compiled Script
COPYCOUNT
Executed 1180 times Cumulative run time 0.234 Non-compiled Script
call("INSUREDNAME2")
Executed 1200 times Cumulative run time 0.205 Non-compiled Script
call("BROKERNAMELIT")
Executed 1180 times Cumulative run time 0.203 Compiled Script
DEFLIB\INSUREDNAME2.DAL
Executed 567 times Cumulative run time 0.186 Non-compiled Script
Return ((?("POL.NUM.LIT")) & " " & (?("INS.POL.NUM")) &
(?("INS.POL.YREFF")))
Executed 1200 times Cumulative run time 0.186 Non-compiled Script
Call("DMGMERGESETID")
Executed 1137 times Cumulative run time 0.173 Non-compiled Script
call("POLEFFDATE")
Executed 534 times Cumulative run time 0.159 Non-compiled Script
MSGB03A()
Executed 534 times Cumulative run time 0.158 Non-compiled Script
MSGD12A1()
Executed 600 times Cumulative run time 0.158 Non-compiled Script
CALL("SERVADDR1DAL")
Executed 534 times Cumulative run time 0.142 Non-compiled Script
MSGS04A()
Executed 534 times Cumulative run time 0.141 Non-compiled Script
MSGB07B()
Executed 1137 times Cumulative run time 0.139 Non-compiled Script
call("POLEXPDATE")
Executed 534 times Cumulative run time 0.126 Non-compiled Script
MSGS09B()
Executed 1149 times Cumulative run time 0.126 Non-compiled Script
call("DUEDATE")

```

```
Executed 534 times Cumulative run time 0.126 Non-compiled Script
MSGB11A()
Executed 550 times Cumulative run time 0.126 Compiled Script
DEFLIB\UPDATESCANABLE.DAL
Executed 600 times Cumulative run time 0.125 Non-compiled Script
CALL("SERVADDR3DAL")
Executed 534 times Cumulative run time 0.125 Compiled Script
DEFLIB\WITHDRBILLDAY2.DAL
Executed 534 times Cumulative run time 0.125 Non-compiled Script
CALL("WITHDRBILLDAY2")
Executed 534 times Cumulative run time 0.125 Non-compiled Script
MSGM11A()
Executed 534 times Cumulative run time 0.124 Non-compiled Script
MSGD12A3()
Executed 1200 times Cumulative run time 0.124 Compiled Script
DEFLIB\DMGMERGESETID.DAL
Executed 534 times Cumulative run time 0.124 Non-compiled Script
MSGS08A()
Executed 1137 times Cumulative run time 0.123 Compiled Script
DEFLIB\POLEXPDATE.DAL
Executed 534 times Cumulative run time 0.111 Non-compiled Script
MSGC01A()
Executed 534 times Cumulative run time 0.111 Compiled Script MSGB03A
Executed 534 times Cumulative run time 0.110 Non-compiled Script
MSGM07A()
Executed 570 times Cumulative run time 0.110 Non-compiled Script
call("COMPANYNAMELIT")
Executed 534 times Cumulative run time 0.110 Non-compiled Script
MSGD10C()
Executed 534 times Cumulative run time 0.110 Non-compiled Script
MSGM02A()
Executed 600 times Cumulative run time 0.110 Non-compiled Script
CALL("SERVADDR2DAL")
Executed 534 times Cumulative run time 0.110 Compiled Script MSGD12A1
Executed 534 times Cumulative run time 0.110 Non-compiled Script
MSGD10G()
Executed 600 times Cumulative run time 0.109 Non-compiled Script
CALL("DMGTOTALSHEETS")
```

WHEN EXTRACT FILES EXCEED THE OFFSET LIMITS

During GenTrn processing, offsets to individual transactions within the extract file are written to the TRNFILE. A long integer is used to contain these offsets. The long integer can have a value up to about 2,100,000,000 bytes or about 2 GB.

When the extract file offset number inside the TRNFILE is about to exceed the 2GB limit, the GenTrn program gives you the following error message:

```
DM15065: Error in BuildTrnRecs(): Offset for extract file is
approaching 2GB limit.
```

If GenTrn processing is combined into GenData processing (using the NoGenTrnTransactionProc rule in the AFGJOB.JDT file), and this situation is encountered, the GenData program gives you the following error message:

```
DM30049: Error in &lt;RULLoadXtrRecs>(): Offset for extract file is
approaching 2GB limit.
```

CONTROLLING WHAT IS IN THE MULTIFilePRINT LOG

Use the MultiFileLogRecord option to control the content of the log file produced during multi-file printing. For certain print drivers (PDF, RTF, XML, or HTML), you must generate a separate print file for every transaction in a batch.

For this processing mode	You set the
Multi-step processing (GenTrn, GenData, and GenPrint)	CallbackFunc option in the Print control group to MultiFilePrint
Single-step processing (GenData)	MultiFilePrint option in the PrintFormset control group to Yes

During this process, the system creates a log file to keep track of the print files it creates. The MultiFileLogRecord option lets you control the contents of the log file produced.

For multi-step processing using the multi-file callback function, you must change the FSISYS.INI file as shown below:

```
< Print >
  CallbackFunc      = MultiFilePrint
  MultiFileLog      = {log file name and path}
  MultiFileLogRecord = ~DALRUN MyScript.DAL
```

The system first looks for MultiFileLog option in the logical printer control group first, such as Printer1, Printer2, Printer3, and so on. If not found, it then looks for this option in the Print control group.

To control the information written to the MultiFileLog file, specify the name of the DAL script, such as MyScript.DAL, in the MultiFileLogRecord option. The system will then execute this script whenever a new output file needs to be created. If a string is returned, the string is used instead of building the log record as a set of semicolon delimited fields. If an empty string is returned, the current log record format is produced.

NOTE: A linefeed is appended to the string before it is written to the log file.

The DAL script could be as simple as one that returns the string from the DAL function, DeviceName. Here is an example:

```
RETURN( DeviceName() )
```

NOTE: For more information about multi-step processing, see [Using Multi-step Processing on page 21](#) and the discussion of the MultiFilePrint callback function in [Using the PDF Print Driver](#).

In single-step processing (GenData), use the MultiFilePrint option in the PrintFormset control group, as shown here:

```
< PrintFormset >
  MultiFilePrint      = Yes
  LogFileType         =
  LogFile             = {log file name and path}
  MultiFileLogRecord = ~DALRUN MyScript.DAL
  ... (other applicable options omitted - see the following note)
```

The PrintFormset rule checks for the MultiFileLogRecord option and if a string is returned, it uses the string instead of building the log record as a set of semicolon delimited fields. If an empty string is returned, the current log record format is produced.

If you set the LogFileType option to XML, the system generates a log file using XML and ignores the MultiFileLogRecord option.

NOTE: There are additional INI settings required for single- and multi-step processing. For more information about single-step processing, see the discussion of the PrintFormset rule in the [Rules Reference](#).

USING INI BUILT-IN FUNCTIONS

You can use these INI built-in functions when running the system:

- ~GetEnv
- ~Platform
- ~OS
- ~DALRUN
- ~DALVAR
- ~Encrypted
- ~ProcessID
- ~WIPField

In addition, there are several functions you can use to retrieve information from WIP records. See [Accessing WIP Fields on page 115](#) for more information.

~GetEnv

Here are examples which show how you can use the GetEnv function.

```
< MasterResource >
  DefLib = ~Getenv MYDRIVE \mstres\deflib\
```

This INI function recognizes a value that begins with a tilde (~). It then parses out the next word and looks to see if a built-in function has been registered with that name, such as *getenv* in the above example.

Once found, the function is called. It then parses the first word to get the environment variable, such as *MYDRIVE*. Leave a space before and after the environment variable.

Finally, the function puts together the result of the environment data with the remainder of the data line, as in *\mstres\deflib*.

So, if *MYDRIVE=G:\APPS* you would see *G:\APPS\mstres\deflib*.

NOTE: Before executing an application whose INI contains the GetEnv function, you must initialize the operating system environment variables. For Windows 32-bit, you enter on a command line:

```
Set EnvironmentVariable = Value
```

Here are some examples:

```
Set MyDrive=G:\APPS
Set UserID=MVF
```

Be sure to leave a space before and after the environment variable.

For this example, assume the environment contains *USERID=(INITIALS)* and the INI contains:

```
< SignOn >
  UserID = ~GetEnv USERID
```

The logon process picks up your user ID from an environment variable.

This method results in a very generic built-in function that does not assume what the data represents. However, if you were using it to build file names, the environment variables would have to be consistent in terms of whether they contained the final backslash or not. In the above example, *MYDRIVE=G:\APPS* would produce an invalid path because a double backslash would occur.

~Platform

Use the ~Platform function to create multi-platform INI files. The possible return values are: *PC*, and *MVS*. This lets you set up INI control groups and options that work on either a PC or MVS platform. When the system executes this function, it replaces ~Platform with either *PC* or *MVS*, depending on the platform. Here is an example:

```
< Print_Batches >
  P_Batch1   = < Config:~Platform > P_Batch1
  P_Batch2   = < Config:~Platform > P_Batch2
  P_Batch3   = < Config:~Platform > P_Batch3
  Error      = < Config:~Platform > Error
  Manual     = < Config:~Platform > Manual
< CONFIG:PC >
  P_Batch1   = .\data\Batch1
  P_Batch2   = .\data\Batch2
  P_Batch3   = .\data\Batch3
  Error      = .\data\Error
  Manual     = .\data\Manual
< CONFIG:MVS >
  P_Batch1   = DD:Batch1
  P_Batch2   = DD:Batch2
  P_Batch3   = DD:Batch3
  Error      = DD>Error
  Manual     = DD:Manual
```

NOTE: You can also use the File option in the INIFiles control group to load multiple INI files. Place this control group and option in your FSIUSER.INI file. Here is an example:

```
< INIFiles >
  File = PC.INI
  File = MVS.INI
```

You can assign any name as long as you include the *.INI* extension. You can have as many File options as needed. You can customize these files based on the platform you are using.

~OS

Use ~OS function to determine the current operating system environment. The possible return values are: *WIN32*, *HPUX*, *AIX*, *MVS*, *Sun*, and *OS1100*.

Here is an example of the functions usage in the INI file. Be sure to include the space after ~OS.

```
< DBHandler:DB2 >
  BindFile = <DB2:~OS > bindfile =
< DB2:WIN32 >
  BindFile = w32bin\DB2LIB.BND
```

This setup allows for the different bindfiles being specified for different operating systems — compare with the ~Platform function which returns *PC* for *Win32*.

**~DALRUN
~DALVAR**

Use the DALRUN and DALVAR built-in functions to execute DAL scripts or get DAL variable information you can use to complete INI options. For instance, you can use this to map unique recipient information into batch records.

These functions are automatically registered when DAL is initialized. Several programs can initialize DAL, such as the GenData and GenPrint programs, the AFEMAIN program (including RACLIB/RACCO), Documaker Studio, Image Editor, and various utilities such as ARCRET, ARCSPLIT, and DALRUN.

NOTE: If you try to use these functions in systems that do not initialize DAL, an incorrect INI value is returned.

Here is an example:

```
< INIGroup >
  Option1 = ~DALRUN MY.DAL
  Option2 = ~DALVAL XYZ_VAL
```

If the program requests Option1, the script MY.DAL is executed and the resulting option is assigned.

If the program requests Option2, the DAL variable XYZ_VAL is located and its contents are assigned to the INI option.

~Encrypted

Use this built-in function to place encrypted values in an INI file. To get the encrypted value, you can execute the CRYRU utility. Here is an example of how you could use this utility on Windows:

```
cryruw32.exe user1
```

The result would be something like this:

```
Encrypted string (2yz76tCkk0BRiPqLJLG00)
```

You then paste the value (2yz76tCkk0BRiPqLJLG00) into an INI file and use the ~ENCRYPTED INI function, as shown in this example:

```
< SignOn >
  UserID = ~ENCRYPTED 2yz76tCkk0BRiPqLJLG00
```

When Documaker Server or IDS runs and gets the value of the UserID option in the SignOn control group, it will get the real value *USER1*.

NOTE: The encryption method used is proprietary.

Keep in mind these limitations:

- Only Windows and UNIX platforms are supported.
- This feature has nothing to do with secure PDF or PDF encryption.
- Almost any INI option can be encrypted.

~ProcessID

The ProcessID INI built-in function (~ProcessID) provides separate trace files for different instances of Documaker Server/Documaker Bridge. This makes it easier to find performance problems and to separate multiple instances.

Here is an example of how you would set up your INI files in Documaker Server or Documaker Bridge to use the ProcessID built-in INI function:

```
< Data >  
TraceFile = dprtrc~PROCESSID .log
```

Here is an example of an output trace file:

```
1. Tue May 25 21:27:26.489 2006 pid=00003896 SQInstallHandler: Info  
from SQLGetInfo, DBName=<>, DBMS=<Oracle>, DBMS Version=<09.02.0010>  
2. Tue May 25 21:27:26.489 2006 pid=00003896 SQInstallHandler: Info  
from SQLGetInfo, DriverName=<SQORA32.DLL>, DriverVer=<09.02.0000>,  
DriverODBCVer=<03.51>  
3. Tue May 25 21:27:26.677 2006 pid=00003896 SQHandler (LOCATEREC):  
ENTER  
4. Tue May 25 21:27:26.677 2006 pid=00003896 SQBindParamData: calling  
_SQLBindParameter, len = <10>, <JOB_ID> = <DEF_JOB_ID>  
5. Tue May 25 21:27:26.677 2006 pid=00003896 select  
STATUS,JOB_ID,COMM_RECS,LASTREC from SJSRPX1_ORA_RESTART where  
JOB_ID = ?  
6. Tue May 25 21:27:26.693 2006 pid=00003896 SQHandler (LOCATEREC):  
SQLocate returned a row.
```

~WIPField Use this built-in INI function to tell the system to substitute a value in the INI file with a value from the WIP record. This works with either Documaker Workstation (AFEMAIN) or the WIP Edit plug-in.

For example, if you want the UserDict value to equal the value for ORIGUSER in the current WIP record, you would set up the following option:

```
< Spell >
UserDict = ~WIPFIELD ORIGUSER
```

ACCESSING WIP FIELDS

You can access most standard WIP fields using the following built-in INI functions. For instance, if you want to create an export file and a PDF file and have the names for these files be identical except for the extension, you could use these function to create a unique name for a file that does not depend on the current time, but rather on a time that does not change, such as the create or modify time.

Function	Returns the
~Key1	WIP Key1 field
~Key2	WIP Key2 field
~KeyID	WIP KeyID field
~ORIGUSER	Original WIP User ID field (the ID used to create the WIP)
~CREATETIME	WIP Create Time field. You can format this option.
~MODIFYTIME	WIP Modify Time field. You can format this option.
~ORIGFSID	Original WIP form set ID. Keep in mind when routing messages, the original form set ID is not necessarily the same as the current form set ID.
~TRANCODE	WIP Transaction Code field.
~DESC	WIP Description field.
~DATE	The current date value.
~USERID	Currently logged in user ID.
~FIELD	A field value from the form set.

NOTE: You can access all of the WIP fields via DAL using the WIPFld function. And, since DAL can be accessed via the ~DALRUN function (see [page 113](#)), you have another method you can use to get those fields.

The system retrieves the Modify Time and Create Time from the WIP record. You can use the ~DATE function to get the current date value. You can also include a parameter to tell the system to format the date.

Keep in mind that if you are trying to use the value as part of a file name, you should only include characters that are valid in file names.

Here is an example of how to specify a date format:

```
~MODIFYTIME ; %m-%d-%Y;
```

Semicolons (;) begin and end the string that defines the date format. If you omit a semicolon, you get the hexadecimal value of the date for ~MODIFYTIME and ~CREATETIME. For the ~DATE function, you get the format specified by the DateFormat option in the Formats control group. This option defaults to:

```
%m/%d/%y
```

If you include the semicolon, but omit the format information after the semicolon, for ~MODIFYTIME and ~CREATETIME you get the format specified by the DateFormat option in the Formats control group. This option defaults to:

```
%m/%d/%y.
```

Formatting arguments

Format arguments consists of one or more codes. Begin each code with a percent sign (%). Characters that do not begin with a percent sign are copied unchanged to the output buffer.

Any character following a percent sign that is not recognized as a format code is copied to the destination—so you can enter %% to include a percent sign in the resulting output string.

You can choose from these format codes:

Code	Description
%d	Day of month as decimal number (01 - 31)
%H	Hour in 24-hour format (00 - 23)
%I	Hour in 12-hour format (01 - 12)
%m	Month as decimal number (01 - 12)
%M	Minute as decimal number (00 - 59)
%p	Current locale's AM/PM indicator for 12-hour clock
%S	Second as decimal number (00 - 59)
%y	Year without century, as decimal number (00 - 99)
%Y	Year with century, as decimal number
%A	Weekday name, such as Tuesday
%b	Abbreviated month name, such as Mar
%B	Full month name, such as March
%j	Day of year as decimal number (001 - 366)
%w	Weekday as decimal number (0-6, with Sunday as 0)

Code	Description
%@xxx	Specify language locale (where xxx is a 3-letter code that identifies one of the supported languages. For example. A format of %@CAD%A might produce <i>mardi</i> , the French word for Tuesday.

Here are some examples:

This format	Will result in
%m-%d-%Y	01-01-2009
The year is %Y.	The year is 2009.
Born %m/%d/%y at %l:%M %p	Born 01/01/09 at 11:57 PM

Here are some additional format attributes for certain codes:

Code	Description
#	<p>Tells the system to suppress leading zeros for the following format codes. This flag only affects these format codes:</p> <p><code>%#d, %#H, %#I, %#j, %#m, %#M, %#S, %#w</code></p> <p>For example, if <code>%d</code> outputs <code>01</code>, using <code>%#d</code> will produce <code>1</code>. Subsequent codes are not affected unless they also have this flag.</p>
>	<p>Tells the system to uppercase the resulting text. This flag only affects these format codes:</p> <p><code>%>p, %>A, %>b, %>B</code></p> <p>For example, if <code>%A</code> results in <i>Tuesday</i>, using <code>%>A</code> will produce <i>TUESDAY</i>. Subsequent codes are not affected unless they also have this flag.</p>
<	<p>Tells the system to lowercase the resulting text. This flag affects only these codes:</p> <p><code>%<p, %<A, %<b, %<B</code></p> <p>For example, if <code>%b</code> results in <i>Mar</i>, using <code>%<b</code> will produce <i>mar</i>. Subsequent codes are not affected unless they also have this flag.</p>
< >	<p>Tells the system to capitalize the first letter of the resulting text. This flag affects only these codes:</p> <p><code>%<>p, %<>A, %<>b, %<>B</code></p> <p>For example, if <code>%p</code> results in <i>AM</i>, using <code>%<>p</code> will produce <i>Am</i>. Subsequent codes are not affected unless they also have this flag.</p>

Specifying locales

When you use `%@xxx` in the format string, the `xxx` represents a 3-letter code that identifies one of the supported language locales.

Until a locale format code is encountered in the format string, the default locale (typically USD which is US English) is used. Once a locale format code is found, the locale specified remains in effect until another locale code is encountered.

For example, suppose the input date is 03-01-2009. This table shows the output from various formats:

This format	Will result in
<code>"%A, %B %d"</code>	<code>"Monday, March 01"</code>
<code>"%@CAD%A %@CAD%A, %B %d"</code>	<code>"lundi, mars 01"</code>
<code>"%A, %@CAD%B %d"</code>	<code>"Monday, mars 01"</code>
<code>"%@CAD%A, %@USD%B %d"</code>	<code>"lundi, March 01"</code>

Using the ~Field function

The ~Field function lets you use a quoted parameter string to name the specific field to locate within the form set. The definition of the field can name a specific section, form, and group (Key2 or Line of Business), separated by semicolons, that contains the field requested. This lets you make sure you are retrieving a specific field occurrence within the document.

Because object names, like fields, sections, forms, and groups, can sometimes contain spaces or other special characters, you should enclose the entire definition in quotation marks (“”). You cannot quote individual elements of the search.

Here are some examples:

This is a valid definition for the ~Field function:

```
option = ~FIELD "Field;Section;Form;Group"
```

This is *not* a valid definition for the ~Field function:

```
option = ~FIELD "Field";"Section";"Form";"Group"
```

OUTPUTTING WIP FIELD DATA ONTO THE XML TREE

Documaker can export these WIP-related transaction fields onto the XML tree:

Key1	Key2	KeyID
TranCode	StatusCode	Desc
GuidKey	TrnName	LocID
SubLocID	Jurisdictn	QueueID

The XML print driver (print type XMP) includes WIP field data in the output when it is generated from GenData's PrintFormset rule or the GenPrint program. You use the Trigger2WIP control group to map the field information. This WIP field information is included in the resulting XML tree under the DOCSET tag.

NOTE: The transaction batch record is defined by the DFD which is defined via the RCBDFDFL setting. The mapped WIP fields must be defined in the WIP DFD file or the internal WIP definition if an external DFD is not used.

Here is an example of the Trigger2WIP control group set up for field mapping:

```
< Trigger2WIP >
  Company      = Key1
  LOB          = Key2
  PolicyNum    = KeyID
  TransactionType= TranCode
```

The output XML tree should have this format:

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCUMENT TYPE="RPWIP" VERSION="11.2">
  <DOCSET NAME=" ">
    <LIBRARY NAME=" " CONFIG="Batch Processing">Batch
Processing</LIBRARY>
    <ARCEFFECTIVEDATE>20061115</ ARCEFFECTIVEDATE>
    <KEY1 NAME="COMPANY">SAMP&O</KEY1>
    <KEY2 NAME="LOB">LB1</KEY2>
    <KEYID NAME="PolicyNum">1234567</KEYID>
    <TRANCODE NAME="TRANSACTIONTYPE">T1</TRANCODE>
    <STATUSCODE NAME="STATUSCODE" />
    <DESC NAME="DESC" />
    . . .
  </DOCSET>
</DOCUMENT>
```

USING XML FILES

You can use these rules to create an alternative data search method so you can do direct XML mapping within Documaker Server:

Rule	Description
UseXMLExtract	Uses the extract list loaded by the transaction as the source of the XML tree.
XMLFileExtract	Assumes that the extract list contains the name of an external file which is the source of the XML tree.

NOTE: For more information on the new rules, see the [Rules Reference](#).

The extract list and the XML tree are separate. Once the XML tree is loaded, it remains loaded and can be searched by subsequent rules — just like any extract list.

The system supports a mix of these search methods:

- An XDB token reference such as *?TOKEN* looked up in the XDB to get the actual search text
- The legacy Offset,Mask method such as *10,HEADERREC*)
- An XML search text, such as *!/descendant::Item*

In most cases, the XDB token reference will be the preferred method.

An XDB entry can return either a legacy offset/length search mask or an XML search path. XML search masks must begin with an exclamation mark (!). The leading exclamation mark is not actually sent to the search routine.

You can use text movement and formatting rules, like *Move_It*, *MoveNum*, *FmtDate*, and *FmtNumber*, to do simple operations, but keep in mind some of the more complicated options may not work.

For instance, *Move_It* supports a *same record* flag. This does not work in XML searches. Likewise, *Move_Num* supports several binary input data types like BCD and you cannot include those in XML at present.

More complicated rules that have multiple search criteria like *SetAddr*, *SubExtractList*, and *Concat* do not work with XML files.

HANDLING OVERFLOW

The XML search infrastructure has *position* support.

```
/descendant::Forms/child::form[position()=2]/child::field1
```

The 2 in this case indicates you want the second form child. Since you would not want to write the search to work with every explicit number, you must indicate where the overflow variable fits into the equation, as shown here:

```
/descendant::Forms/child::form[position()=****]/child::field1
```

The system first scans the search to see if a replacement is needed for the overflow value. In this case, it would insert the 2 (taken from the overflow variable value) and then do the actual XML search.

You can also handle overflow within overflow by specifying an overflow variable name in the search. For instance, suppose you have multiple cars and each car can have multiple drivers.

```
<car>
<driver>Tom<driver/>
<driver>Tim<driver/>
<car/>
<car>
<driver>Sally<driver/>
<car/>
```

If you had two overflow variables, one working for *car* and one for *driver*, you could create a search like this:

```
/descendant::car[**carvar**]/child::driver[**drivevar**]
```

Where the system gets two overflow variables and insert them into the search text.

TRIGGERING FORMS AND SECTIONS

You can do simple triggering based upon the existence of a node. For example, this

```
/child::car
```

would trigger a form if *car* is a child of the root node. Referring back to the earlier example, you could make it trigger two of the same forms because there are two cars.

The system supports value matching. So you can do the following:

```
/child::car[child::driver="Tom"]
```

Or, you can use the RecipIf rule to trigger a section with custom rule parameters, as shown in this example:

```
A=(!/child::car/child::driver 1,7)::if
(A='Tom')::return("^1^")::end::;
```

If there is such a value in that element in the XML file, the section would trigger. For this to work, define the offset of the variable attribute as 1 and the length of the data you want to compare.

You can also use XML search strings such as these:

This string	Finds
!descendant::PolicyNumber	The PolicyNumber value
!descendant::Forms/child::Form	All forms

USING XPATH

XML path locator (XPath) complies with the standard syntax specifications (W3C standards) found in the XML Path Language, but differs in some regards because it was developed to support the Rules Processor. Because this version of XPath has some limitations, you should check the syntax using the XPATHW32 utility.

XPATH SYNTAX

Here are examples of the valid axes, function calls, signs, and operators to help you understand and use the XPath syntax.

Axes

You have these axes:

Name	Used to locate the
ancestor	Ancestors of the current context node
ancestor-or-self	Ancestors of the current context node and itself
parent	Parents of the current context node
descendant	Descendants of the current context node
descendant-or-self	Descendants of the current context node and itself
attribute	Attributes of the current context node
child	Children of the current context node
following-sibling	Following siblings of the current context node
following	Context nodes that follow the current node
preceding-sibling	Preceding siblings of the current context node
preceding	Context nodes that precede the current node
self	Self context node

When used, an axis is always followed by a context node name separated by two colons (:). For example, the syntax *descendant::para* locates all para descendants of the current context node.

Symbols

You can use these calculation operators:

= != < > + -

Where !=, <, >, + can be used as calculation operators in function position(), such as, *[position()=2]*, *[position()!=2]*, *[3+i]*, *[position()<5]*, and so on. The equals sign (=) is also used for evaluations such as *@Name='Auto'*.

You can use these symbols in a valid XPath:

/ // * :: [] @

Where the pair of brackets ([]) enclose a condition for evaluation, the at symbol (@) is an abbreviation of the attribute, the asterisk (*) is used for a wild card search, and others are used in a valid XPath, as shown below.

Functions

You can use these functions:

Function	Returns
concat(string, string, string...)	The concatenation of the strings
last()	The last element in the selection
name()	The name of the selected elements
node()	The node names
position()	The position of selected elements
text()	The text of selected elements
string(object)	The string from the context node
xml()	The output buffer containing all descendents of the specified element

Expressions

You can use abbreviated syntax with XPath. Here are the valid expressions:

Abbreviated syntax	Full syntax
*	child::*
para	child::para
chapter/para	child::chapter/child::para
para[1]	child::para[position()=1]
/chapter/para[last()]	/child::chapter/child::para[position()=last()]
text()	child::text()
node()	child::node()
para[@type]	child::para[attribute::type]
para[@type="warning"]	child::para[attribute::type="warning"]
para[@type="warning"][(2+i)]	child::para[attribute::type="warning"][(position()#2+i)]
chapter[title]	child::chapter[child::title]
chapter[title='Introduction']	child::chapter[child::title="Introduction"]
doc//para	child::doc/descendant-or-self::node()/child::para
@*	attribute::*
@type	attribute::type
[@name='warning']	[attribute::name='warning']
//para	/descendant-or-self::node()/child::para
.	self::node()
./para	self::node/descendant-or-self::node()/child::para
..	parent::node()
../chapter	parent::node()/child::chapter
../@type	parent::node()/attribute::type

USING THE XPATH TESTING UTILITY

Here is the syntax of the XPATHW32 testing utility:

```
xpathw32 /f= xml file /e=starting node /x= search path
```

The `/e` parameter specifies the node where the search of the XPath starts. You can omit this parameter if you want the search to start from the beginning. A pair of double quotes is required to enclose the search mask. Here is an example:

```
xpathw32 /f="d:\test\test.xml" /x="Forms/Form/Car[@Name='Car1']/text()"
```

This example searches the node *Car* with the attribute *Name*="Car1". It then retrieves its text and returns a text string similar to this one:

```
Text string = Car 1 is Toyota
```

These examples illustrate some search paths most frequently used in Documaker RP applications. Run the testing tool yourself for the answer.

Example 1 These examples search for a list of nodes with or without conditions. Keep in mind a condition is always placed within brackets, as shown here: *[condition]*.

This	Returns
Forms/Form/Car	A list of the Car nodes
Forms/Form/Car[@*][position() < 3]	The first two nodes in the Car node list
Forms/Form/Car[@Name][position() > 1]	A list of the Car nodes above the first element
Forms/Form/Car[text()][position() != 2]	A list of the Car nodes, excluding the second one
Forms/Form/Car[Model]	A list of Car nodes that have a child named Model
Forms/Form/Car/node()	A list of children nodes under the Car nodes
Forms/Form/Car/Coverage[1]	A list of first child Coverage under the Car nodes
Forms/Form/Car[@Name='Car1']/Coverage	A list of nodes Coverage under Car1

Example 2 These examples search for the path for a single element:

This	Produces
Forms/Form/Car[@*][1]	The first node of the Car list with any attributes
Forms/Form/Car[@Name][last()]	The last node of the Car list with the attribute Name
Forms/Form/Car[@Name='Car1']	The Car node with attribute name Car1
Forms/Form/Car[Model='Toyota']	The Car node with a child Model that has a text string of Toyota.
Forms/Form/Car[Model='Nissan']/Coverage[3]	The third child node of Coverage under the parent node Car that has a child named Model with a text string of Nissan

Example 3 These examples search for a list of attributes:

This	Produces
Forms/Form/Car[Model='Nissan']/@*	A list of attributes of the Car node that have a Child node named Model with a value of Nissan
Forms/Form/Car/@Name	A list of the attribute Name that has a parent node of Car

Example 4 These examples search for a single attribute:

This	Produces
Forms/Form/Car[Model='Honda']/@*[1]	The first attribute of the Car node that has a child named Model with a value of Honda
Forms/Form/Car[Model='Honda']/@Name	The attribute Name of the Car node that has a child named Model with a value of Honda
Forms/Form/Car[1]/@Name	The attribute Name of first Car node

Example 5 These examples search for a list of text strings:

This	Produces
Forms/Form/Car/text()	A list of text strings of Car nodes
Forms/Form/Car[Model]/text()	A list of text strings of Car nodes which have children named Model

Example 6 These examples search for a single text string:

This	Produces
<code>Forms/Form/Car[Model='Toyota']/text()</code>	The text string of the Car node which has a child name Model with a value of Toyota
<code>Forms/Form/Car[Model='Honda']/parent/text()</code>	The text string of the node Form which has a child named Car that, in turn, has a child named Model with a value of Honda

NOTE: There are three types of returned lists: elements, attributes, and text. When a list includes only one element, the structure returns a single element instead of a list.

Example 7 These examples search for the name of elements:

This	Returns
<code>//*[name()='Car']</code>	"Car" nodes
<code>Forms/Form/*[name()='Car'][2]/text()</code>	A text string of second "Car" nodes

Example 8 These examples concatenate text strings:

This	Returns
<code>concat('Car1', 'and', 'Car2')</code>	A string "Car1 and Car2"
<code>concat(//Car[@Name='Car1'], 'and', //Car[@Name='Car3'], 'are imported cars.')</code>	A string "Toyata and Nissan are imported cars."

Example 9 These examples search for strings:

This	Returns
<code>string(' 12345')</code>	The string " 12345"
<code>string(//Car[2]/*[1])</code>	The string of the first child of the second Car node

Example 10 This examples returns a buffer that contains all descendants of the specified element:

This	Produces
<code>xpathw32 /f=cars.xml /x="//Car[2]/xml()</code>	<pre><Car Name=" Car2">Car 2 is Honda <Model>Honda</Model> <Coverage>Cover 4</Coverage> <Coverage>Cover 5</Coverage> <Coverage>Cover 6</Coverage> </Car></pre>

Note that the XPath must point to a single element, such as `Car[2]` in the example.

EXAMPLE XML FILE

Here is an example XML file (TEST.XML):

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XML Spy v4.2 U (http://
www.xmlspy.com)-->
<Forms>
  <Form>
    <Car Name=" Car1">Car 1 is Toyota
      <Model>Toyota</Model>
      <Coverage>Cover 1</Coverage>
      <Coverage>Cover 2</Coverage>
      <Coverage>Cover 3</Coverage>
    </Car>
    <Car Name=" Car2">Car 2 is Honda
      <Model>Honda</Model>
      <Coverage>Cover 4</Coverage>
      <Coverage>Cover 5</Coverage>
      <Coverage>Cover 6</Coverage>
    </Car>
    <Car Name="Car3">Car 3 is Nissan
      <Model>Nissan</Model>
      <Coverage>Cover 7</Coverage>
      <Coverage>Cover 8</Coverage>
      <Coverage>Cover 9</Coverage>
    </Car>
  </Form>
</Forms>
```

CHAPTER 3

Implementing Your System

This chapter provides an overview of how a system is implemented. Although implementations are typically handled by the Professional Services Group and each implementation differs, you can make your implementation run more smoothly by understanding the procedures and methodologies outlined here.

In general terms, a system implementation is a set of structured procedures and processes our Business Analysts follow to design, develop, and set up a customized system for a particular client.

This chapter discusses...

- [Why Use a Methodology? on page 132](#)
- [Gathering Information on page 134](#)
- [Roles and Responsibilities on page 135](#)

WHY USE A METHODOLOGY?

When each system implementation is so unique and so configurable, why use a methodology?

Because, a methodology allows for consistent handling of each specific implementation. Consistency promotes efficiency. The smoother and more efficient a system implementation is, the more satisfied you will be. Furthermore, it will be easier to maintain and, if necessary, easier to modify the implemented system should your needs change.

The system Implementation methodology is followed for each implementation project. The methodology is designed to allow for project flexibility to accommodate the various system customizations.

The System Implementation Methodology is comprised of these phases:

Phase 1 - Define Requirements

Phase 2 - Create Detail Forms Requirements

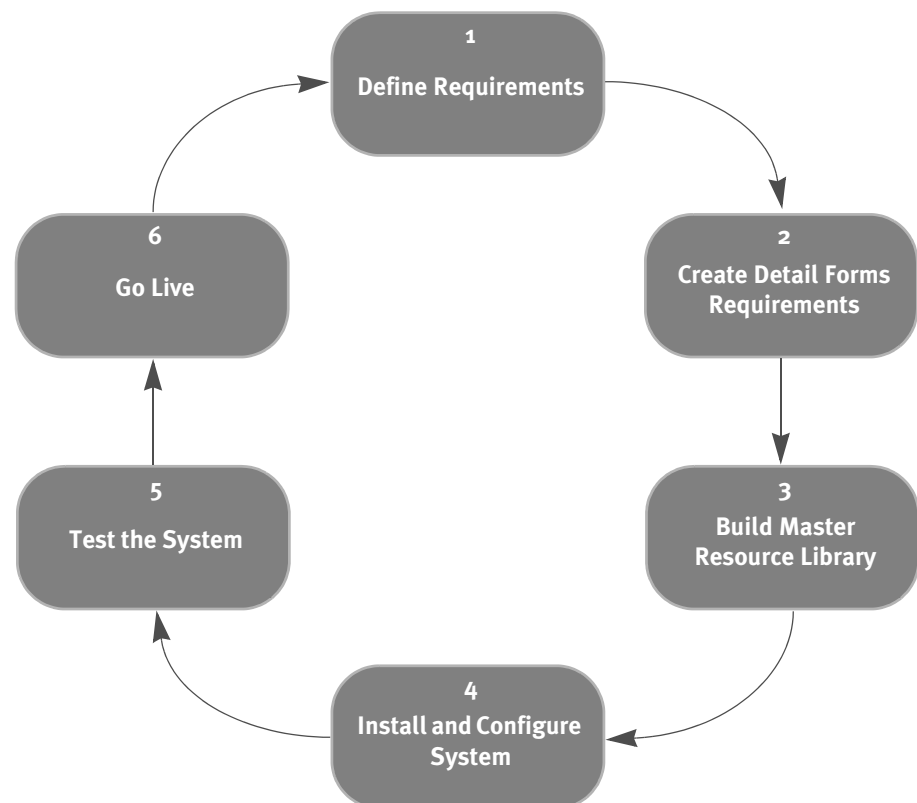
Phase 3 - Build the Master Resource Library

Phase 4 - Install and Configure the System

Phase 5 - Test the System

Phase 6 - Go Live

The methodology phases are cyclical. After completing Phase 6, Phase 1 begins again, to continually evaluate the system and to incorporate product maintenance.



Because each system implementation is different, the time frame for completing each phase varies. Here is a summary of the phases and the related tasks:

Phase 1 - Define the requirements

Defining the requirements is the planning and definition phase of an implementation. In this phase, your processing needs are defined. Your input is very important in accurately identifying your needs.

The primary output of this phase is the Requirements Definition Documentation. This document includes the project scope and schedule, information regarding the technical and functional areas targeted for document automation, and the steps outlining how the implementation will proceed.

Phase 2 - Create the detail forms requirements

Creating the detail forms requirements involves specifying all forms to be converted from paper to electronic forms, and determining how to automate the transfer of data to the forms. Determining how to automate data transfer includes defining how the data will be mapped, defining the data transfers from the source file to the forms, and the form data format. This process requires mapping data in hierarchical succession: form set, form, section, fields, field attributes, and field sequencing and navigation logic.

Documaker Studio or the legacy Docucreate tools are often used during this phase. You can learn more about these tools in the Documaker Studio User Guide or the Docucreate User Guide.

Phase 3 - Build the Master Resource Library

Building the Master Resource Library involves organizing and setting up the resources to be used by your system. Here a complete library of reusable resources is set up. Your users will select from these resources to complete their work activities.

A resource library is divided into these libraries: Section Library, Variable Data Dictionary Library, and Rules Library. Each of these libraries contains files that store different resource components. Depending on your system configuration and location, you may have separate Distributed Resource Libraries, as a subset of the Master Resource Library.

In addition to setting up the resources, this phase involves configuring forms sets, the rules used for processing forms, and the system initialization files that determine how your system operates. During this phase the base system is customized to meet your needs. Customization can range from changing system functions to changing the system interface.

Phase 4 - Install and configure the system

During this phase, the various system modules are installed. After installing the components, you test various aspects and functions of the system, such as printing and archiving, using test scenarios and sample data. Adjustments are made if required to the configuration files. If available, you should use real data for these initial tests.

Phase 5 - Test the system

In Phase 5, system testing begins. Detailed test matrices are created, which are used to test the entire system using real data. A test matrix is a listing of the functions, conditions, and exceptions of the system you want to test. It's important to have plenty of real data you can use for testing purposes during this phase.

Phase 6 - Go live

In Phase 6, the system is now ready for full production. The support personnel assigned to the project will assist you with start up procedures and training.

GATHERING INFORMATION

At the beginning of any implementation, it is important to gather as much relevant information as possible. This information helps ensure requirements are correctly defined, future goals are taken into consideration, and the solution meets your needs exactly.

UNDERSTANDING YOUR NICHE

Understanding your current and future industry positioning is integral in successfully implementing a customized system. The system must suit your needs now, and expand as your company grows. Knowing where you expect to take the company in the future is important for defining a system.

The implemented system must be set up so it can grow as your company grows. The system must also serve the your current needs. To define your current and future needs, you will be asked questions about the your company's goals, industry trends, and company projections, such as:

- Do you expect a significant growth in revenue over the next five years? What is your vision for the future?
- Do you expect to experience a reduction or increase in number of employees?
- Do you envision growth into other related or non-related industries?
- How far has the company grown (or downsized) in the past few years? Can you detect industry trends based on past revenues, and financial status?

One of the greatest benefits of a system is its flexibility. Determining where you are and where you expect to be in the future helps to make sure your system solves your business problems today and tomorrow.

UNDERSTANDING YOUR ORGANIZATION

Understanding your organization is also important in fulfilling your needs. It helps to understand the chain of command, and the responsibilities associated with each role in your organization. To gather information about your organization, you will be asked questions such as:

- Have you had previous experience with document automation? How would you describe that experience positive?
- How many data entry operators do you have, and who and where are they?
- What percentage of total time do employees at each level spend on the system?
- Is there a specific organizational hierarchy or chain of command within the company?
- What is your corporate culture? Is there a discreet division of labor at all levels, or is there cross-training and information sharing?

You may also have documentation about your company, future company directions, system flows and workflows, and other information which is important in mapping an implementation strategy. This background information is important in defining the best solution for your company.

ROLES AND RESPONSIBILITIES

There are many people involved in a system implementation project. A system implementation project team is comprised of both Documaker Professional Services personnel and personnel from your company. The team's goal is to provide a seamless integrated solution for the your document automation needs.

You are an integral member of the system implementation team. With your knowledge of your business needs, you can often be the navigator or guide during the implementation process.

Documaker Professional Services personnel include:

BUSINESS ANALYST. Throughout the project the Business Analyst is responsible for coordinating the project, creating the phase deliverables, and keeping apprised of the status of all processes and subprocesses within the project.

PROJECT MANAGER. The project manager is involved in initial project analysis and planning, and sizing of the system component development process. The project manager is also responsible for creating the project schedule.

SYSTEM DEVELOPERS. The developers are primarily responsible for coding the system components. Additionally, the programmers may provide analysis, and planning input during the initial phases. Professional Services personnel are involved in customization projects.

CHAPTER 4

Setting Recipients and Copy Counts

This chapter describes how you can specify recipients for the individual forms that comprise your form sets and how you can specify the number of copies each recipient will receive.

In this chapter you will find information about:

- [Concepts on page 138](#)
- [Key Files on page 139](#)
- [Trigger Table Record Format on page 140](#)
- [Specifying the Transaction Trigger Table on page 142](#)
- [How Transaction Triggering Works on page 143](#)
- [Form Level Triggers on page 146](#)
- [Master and Subordinate Sections on page 149](#)
- [Examples on page 151](#)
- [Summary on page 168](#)

CONCEPTS

In a manual form system, a data entry operator selects the forms that make up a document set. Some forms may be mandatory and are always included. Others are optional and must be specified by the operator.

The operator chooses forms by examining the data at hand and considering certain conditions pertaining to that data. For instance, if the operator is creating insurance policies, he or she would have to know:

- What company is this for?
- What line of business?
- What type of transaction is this?
- Does the agent need a copy?
- How many copies?
- What about the home office copy?

And so on. The answer to each question affects the makeup of the document set you will assemble.

Documaker Server automates the tasks and selection decisions that an operator makes. The set of forms to be printed, and the recipients of those forms, are selected by executing a series of business rules that test the supplied data to see if certain conditions are met.

As matching conditions are found in the data for a transaction, a form set can be constructed, form by form, with all the proper recipients designated. This is the first step in the assembly of a document set. Later, once the set of forms has been determined, other business rules for each form and variable field can be executed to begin to construct the output data, field by field, within each form.

NOTE: Docucreate includes the Form Set Manager, a tool you can use to set recipients and copy counts. This chapter explains how the underlying files and settings work. You can change these settings either by changing the files in a text editor or by using the Form Set Manager. You can find more information about the Form Set Manager in the Docucreate User Guide and in the tool's on-line Help.

KEY FILES

Here is a discussion of the key files which the system uses to determine who gets what form and how many copies it should print. You'll also find information about important concepts, such as form and section (image) level triggers.

TRANSACTION TRIGGER TABLE

The transaction trigger table (also known as the SETRECIP table, or SETRCPTB.DAT file) is a text file used by Documaker Server to define the conditions under which certain forms are included in form sets, and which recipients are to receive the forms. Each record in the transaction trigger table defines a triggering condition for a form or section and is referred to as a trigger record, or, more simply, a trigger.

Trigger Levels

There are two *levels* of trigger records: *form level triggers* which trigger forms, and *section level triggers* which trigger sections within a form. section level triggers are optional, since some forms automatically include all necessary sections. Also, form level triggers can be optional, since a form can also be triggered by an section level trigger.

NOTE: sections are defined by FAP files and are maintained using Documaker Studio or Image Editor. A section may be an entire page, or a page segment. Forms can be made up of many pages, each containing one or more sections.

FORM SET DEFINITION TABLE

The transaction trigger table works with the form set definition table (also known as the FORM.DAT file) to define the required form set. Together they define many complex inter-relationships and rules, and a number of powerful options by which forms and sections can be triggered, and recipients defined.

In this chapter we will discuss the...

- Purpose of the transaction trigger table.
- Record layout of the transaction trigger table.
- Runtime setup options for the transaction trigger table.
- Rules under which the transaction trigger table program logic operates.

In addition, this chapter discusses several scenarios to illustrate many of the options and variations used to trigger forms and sections.

TRIGGER TABLE RECORD FORMAT

The transaction trigger table is a semi-colon delimited text file. Each record in the table defines a form level or section level trigger condition. Each record contains the following fields:

;GroupName1 (Company)
;GroupName2 (Line of Business)
;Form name
;Image name
;Transaction codes
;Recipient list
;Search mask 1 (Counter)
;Overflow field 1 (Occurrence flag)
;Overflow field 2 (Records per overflow image)
;Overflow field 3 (Records per first image)
;Recipient Copy count
;Search mask 2 (True/False)
;Custom rule;

NOTE: Semicolons are required as field separators, or placeholders. When values are omitted from optional fields, one or more consecutive semicolons may appear.

The table describes each field.

Field	Description
GroupName1	Matches the GroupName1 field in the form set definition table. In an insurance industry application, this would typically contain the Company code. <Key1Table> in the FSISYS.INI file.
GroupName2	Matches the GroupName2 field in the form set definition table. In an insurance industry application, this would typically contain the “line of business” code. <Key2Table> in the FSISYS.INI file.
Form name	The name of the form, as specified in the form set definition table. Note: Form names are descriptive, and do not correlate to any physical file name.
Image name	The name of a section (image) within a form, as specified in the form set definition table. This name also correlates to a physical section file (.FAP file) and often to a Data Definition Table file (.DDT file). Note: An section level trigger record requires an entry in this key field; a form level trigger record must omit any value in this field.

Field	Description
Transaction codes	<p>By including one or more transaction codes in this field, a form is triggered only if the extract file record includes that transaction code.</p> <p>If no transaction code value is mapped from the extract data for a transaction, the system considers all triggers eligible, regardless of whether they specify a transaction code list.</p> <p>Conversely, if a transaction code value is mapped from the data, the system only considers those triggers that have the same value to be eligible for evaluation.</p>
Recipient list	Lets you optionally specify certain recipients.
Search mask 1 (Counter)	Defines the criteria to determine when a form belongs in a form set (or a section within a form). The criteria lets Documaker Server get specific data from the extract file. One form (or section) is added for every occurrence of the Search Mask per Transaction when the overflow flag is set.
Occurrence (overflow) Flag	<p>Indicates the need to calculate overflow conditions. Enter zero (0) for no overflow or 1 for overflow.</p> <p>Also used for Master and Subordinate form and section level flags. You can enter:</p> <p>M=master (used on form level triggers)</p> <p>S=subordinate (used on section level triggers)</p> <p>F=tells the system to override any previous copy count settings and use the copy count settings in this trigger file (used on form level triggers). In essence, this flag tells the system that if this form is already triggered, don't trigger it again—just modify the previously triggered copy.</p>
Records per overflow image	(Used by overflow) Specifies the number of records matching the Counter Search Mask that will fit on the specified overflow form.
Records per first image	(Used by overflow) Specifies the number of records matching the Counter Search Mask that will fit on a specific form before overflowing to a new form.
Recipient copy count	Specifies the number of copies a recipient receives.
Search Mask 2 (True/False)	Similar to Search Mask 1, but only one form will be triggered, regardless of how many occurrences of the condition exists.
Custom Rule	Available field for use with custom rules or search masks. Most common custom rule is RECIPIF.

SPECIFYING THE TRANSACTION TRIGGER TABLE

You specify the file name of the transaction trigger table (also known as the SETRECIP table) in the FSISYS.INI file. For example:

```
< Data >
    SetRcpTb = SETRCPTB.DAT
< MasterResource >
    FormsetTrigger = SETRCPTB.DAT
```

The form set definition table is also specified in the FSISYS.INI file, in the following control group:

```
< MasterResource >
    FormDef = FORM.DAT
```

There are two form set level rules that relate to the transaction trigger table in the AFGJOB.JDT file:

```
<Base Form Set Rules>
    ;LoadRcpTbl;2;;
    ;RunSetRcpTbl;2;;
```

The LoadRcpTbl rule loads the entries from the SETRCPTB.DAT file for the current GroupName1, GroupName2, and Transaction code. The RunSetRcpTbl rule runs all entries in the transaction trigger table that pertain to the current GroupName1, GroupName2, and Transaction code to generate the form set for the current transaction.

For more information on these and other rules, see the [Rules Reference](#).

HOW TRANSACTION TRIGGERING WORKS

The transaction trigger table works with the extract file, TRN file (usually TRNFILE), and the form set definition file (usually FORM.DAT). The TRNFILE contains a record for each transaction passed to Documaker Server.

The record format for the TRNFILE varies by implementation; the format is specified by a DFD (Data Format Definition) file. Each TRNFILE record contains a series of offsets used when processing the transaction.

Offsets in a TRNFILE record define the location where:

- The transaction begins in the extract file
- Data for the transaction is stored in the NAFILE
- The form set for the transaction is stored in the POLFILE
- The TRN record itself begins (this offset is stored in the BCH file, so the entire TRNFILE is not needed)

The form set definition file (FORM.DAT) defines the organization of sections within forms and the organization of forms within form sets. The FORM.DAT is a semi-colon delimited file; its format includes information about...

- Company
- Line of business
- Forms (form options)
- Sections (section options)
- Recipients
- Recipient section copy counts

The recipient table, also known as the transaction trigger table (usually SETRCPTB.DAT), defines when to include a particular form section or recipient of a form section in a form set. The recipient table contains information necessary to determine if a condition exists to include a form. Conditions may be defined by a combination of transaction types and search masks for the extract file as defined above.

Three of the first five transaction trigger fields (GroupName1, GroupName2, and Transaction Code) must match some records within the extract file in order for the trigger conditions to be evaluated. For example, if there are no records with the transaction code specified in the trigger, that trigger will be skipped. If extract records exist that match these three fields, the remaining fields of that trigger are evaluated.

It is not required to use all of the available fields in a transaction trigger record, but if it is necessary to use multiple search masks and/or a custom rule, the following logic applies when evaluating whether to trigger that form or section.

SECTION LEVEL TRIGGERS

Here are some examples of how the system evaluates triggers:

With these settings:

Copy Count	Search Mask 1	Search Mask 2	The result is:
o	T	T	Turn off
o	T	F	Do nothing
o	F	F	Do nothing
o	F	T	Turn off
Non o	T	T	Turn on
Non o	T	F	Turn off
Non o	F	F	Turn off
Non o	F	T	Turn off

The system evaluates search mask 2 first. When this evaluation is performed, the system also takes the copy count into consideration.

If the copy count is zero (o):

- If search mask 2 is true, evaluate search mask 1. If search mask 1 is true, turn on the section based on the copy count (for instance, if the copy count is zero (o), then turn on nothing). If false, turn off the section.
- If search mask 2 is false, then do nothing.

If the copy count is not zero:

- If search mask 2 is true, then evaluate search mask 1. If search mask 1 is true, turn on the section based on the copy count (for instance, if the copy count is zero (o), then turn on nothing). If false, turn off the section.
- If search mask 2 is false, turn off the section.

With these settings:

Copy Count	Search Mask 1	Custom Rule	The result is
o	T	T	Turn off
o	T	F	Turn off
o	F	F	Turn off
o	F	T	Turn off

Non o	T	T	Turn On
Non o	T	F	Turn off
Non o	F	F	Turn off
Non o	F	T	Turn on

When search mask 1 and custom rule are specified, the system uses the custom rule only. When the custom rule is evaluated:

- If true, turn on the section based on the copy count (for instance, if the copy count is zero (o), then turn on nothing)
- If false, do not turn on the section.

With these settings:

Copy Count	Search Mask 2	Custom Rule	The result is
o	T	T	Turn off
o	T	F	Turn off
o	F	F	Do nothing
o	F	T	Do nothing
Non o	T	T	Turn on
Non o	T	F	Turn off
Non o	F	F	Turn off
Non o	F	T	Turn off

The system evaluates search mask 2 first. When this evaluation is performed, the system also takes the copy count into consideration.

If the copy count is zero (o):

- If search mask 2 is True, evaluate the custom rule. If the custom rule is True, turn on the section based on the copy count (for instance, if the copy count is zero (o), then turn on nothing). If false, turn off the section.
- If search mask 2 is false, then do nothing. The custom rule will be ignored. Leave the section as is.

If the copy count is not zero:

- If search mask 2 is true, then evaluate the custom rule. If the custom rule is true, turn on the section based on the copy count (for instance, if the copy count is zero (o), then turn on nothing). If false, turn off the section.
- If search mask 2 is false, turn off the section.

FORM LEVEL TRIGGERS

Here are some examples:

With these settings:

Copy Count	Search Mask 1	Search Mask 2	The result is:
o	T	T	Turn off
o	T	F	Turn off
o	F	F	Turn off
o	F	T	Turn off
Non o	T	T	Turn on
Non o	T	F	Turn off
Non o	F	F	Turn off
Non o	F	T	Turn off

At the form level, search mask 2 is evaluated first. It is unlike the section level in that the copy count is not considered.

If search mask 2 is true, search mask 1 is evaluated:

- If true, trigger the form based on the copy count (for instance, if the copy count is zero (o), then turn on nothing)
- If false, do not trigger the form.

With these settings:

Copy Count	Search Mask 1	Custom Rule	The result is:
o	T	T	Turn off
o	T	F	Turn off
o	F	F	Turn off
o	F	T	Turn off
Non o	T	T	Turn on
Non o	T	F	Turn off
Non o	F	F	Turn off
Non o	F	T	Turn on

When search mask 1 and custom rule are specified, the system uses the custom rule only. When the custom rule is evaluated:

- If true, trigger the form based on the copy count (for instance, if the copy count is zero (0), then turn on nothing)
- If false, do not trigger the form.

With these settings:

Copy Count	Search Mask 2	Custom Rule	The result is:
0	T	T	Turn off
0	T	F	Turn off
0	F	F	Turn off
0	F	T	Turn off
Non 0	T	T	Turn on
Non 0	T	F	Turn off
Non 0	F	F	Turn off
Non 0	F	T	Turn off

At the form level, search mask 2 is evaluated first. It is unlike the section level in that the copy count is not considered. If search mask 2 is true, the custom rule is evaluated:

- If true, trigger the form based on the copy count (for instance, if the copy count is zero (0), then turn on nothing)
- If false, do not trigger the form.

If search mask 2 is false, do not trigger the form.

When a transaction trigger table entry is evaluated to be true or false, the effect varies depending on the type of trigger. The following table explains the effects of form and section level triggers:

Logic	Form Level Trigger	Section Level Trigger
True	<p>Turns on all sections in the form for selected recipients with the copy count specified in the copy count field in the transaction trigger table entry.</p> <p>Turns on sections in the form for non-selected recipients only if those sections have a copy count of at least 1 in the form set definition table.</p>	<p>Turns on the specified section for the selected recipients with the copy count specified in the copy count field in the transaction trigger table entry.</p> <p>Turns on other sections in the form with the same recipients with a copy count of at least 1 in the form set definition table.</p>
False	<p>Does not turn on any images for any recipients.</p>	<p>Turns off the specified image by setting the copy count to zero (0) for the selected recipients or does nothing.</p>

MASTER AND SUBORDINATE SECTIONS

The set recipient table contains both form and section (image) level triggers to handle cases of conditional sections on forms. There are two flag options you can use in the set recipient table (SETRECIP) for transaction triggering. These two flags, *S* and *M*, are used to regulate the evaluation of section level triggers and are placed in the Occurrence (overflow) flag field of form or section level triggers.

NOTE: When you are using master and subordinate triggering, keep in mind you cannot evaluate multiple form level triggers. The system limits you to a single form level trigger for a given group of sections. You can repeat the same sections for another form level trigger.

MARKING SUBORDINATE SECTIONS

The *S* flag, called the subordinate flag, identifies the section as subordinate to the parent or master form. The subordinate flag is enabled when you place an uppercase *S* in the Occurrence flag field (which is the 8th semi-colon delimited field of each table entry), and may be separated from the overflow flag (0 or 1) by a comma. As long as there is an uppercase *S* character in the flags field, the section will be treated as a subordinate. The *S* flag makes the section level trigger dependent on the successful triggering of its parent form by the form level trigger for that form. If the parent form was not triggered on its own account, such as if it was added because of an underlying non-subordinate section being triggered, then all subordinate sections triggers are still ignored.

The intended use of this flag is to eliminate redundant conditional logic at both the section and form level, as well as to maintain a hierarchy of form and section with respect to the inclusion of these entities into a form set. A subordinate section cannot cause the inclusion of the parent form because if the form was not triggered then the subordinate section triggers are never processed. The use of subordinate sections lends itself largely to situations where you want to trigger a form based on some condition, and then conditionally add sections to that form.

If the form was not triggered then all underlying section triggers can be ignored, which eliminates unnecessary processing. The subordinate flag also eliminates processing the same conditional logic over and over again since the logic is only performed once at the form level.

Subordinate sections are subordinate to the master (or parent) form level trigger being true or false, and not actually to the form being triggered. Therefore, it is probably not a good idea to mix subordinate and non-subordinate sections under the same parent form. If the form was triggered by a non-subordinate section, and not by its own conditional, then all subordinate sections for that parent form will still be ignored, despite the fact the form was triggered.

MARKING MASTER FORMS

The master form flag, uppercase *M*, works in a similar manner but on the form level. The *M* flag is used only with form level triggers and is ignored if used with a section level trigger. The *M* flag is used to signify a master form level trigger, causing all of the section level triggers beneath the master form level trigger to be treated as if they were subordinate section level triggers.

When you use the *M* flag with a form level trigger, it does not matter whether the underlying section level triggers have the *S* flag—they will all be treated as if they did. In effect, if the logic in a master form level trigger fails, the form does not trigger and all of the form's section level triggers are ignored. The next section illustrates transaction triggering logic through specific examples.

EXAMPLES

The transaction trigger table works with the form set definition table. The transaction trigger table is usually named SETRCPTB.DAT and the form set definition table is usually named FORM.DAT.

The FORM.DAT file defines which sections make up a form. There are many possible combinations of sections that can constitute a form. A form can be comprised of a single section or multiple sections. The FORM.DAT file also specifies which recipients get which sections. It is possible to have a single form that is composed of four sections, three of which are constant for all recipients, and one section that varies depending on recipient.

Recipient and copy count information contained in the FORM.DAT is also included in the SETRCPTB.DAT transaction trigger table, so it is important to understand how these two tables work together. Designing the two tables independently can often cause undesired results because one table is overriding the other in a manner that the user did not anticipate. But if the two tables are designed to work together, many complex forms with conditional sections and copy counts can be implemented.

In this topic, numerous examples of form set definition files and transaction trigger tables are shown to illustrate some basic relationships between the form set definition table file and the transaction trigger table file.

In each example, the FORM.DAT and SETRCPTB.DAT tables are shown along with the resulting POL file generated by the GenData program. The POL file shows the final form sets created by the GenData program and is used as an input file by the GenPrint program (along with the NA file) to generate printed output.

You will find examples which discuss:

- [Specifying Copy Counts and Sections on page 152](#)
- [Using Transaction Codes on page 154](#)
- [Setting Up Search Mask and Sections on page 155](#)
- [Using the RECIPIF Rule on page 157](#)
- [Using Automatic Overflow on page 159](#)
- [Using Forced Overflow on page 161](#)
- [Setting Search Masks and Recipients on page 162](#)
- [Using the Set Recipient Table and Extract Files on page 163](#)
- [Formatting Search Masks on page 164](#)
- [Sorting Forms by Recipient on page 166](#)

SPECIFYING COPY COUNTS AND SECTIONS

One of the fields that is shared by both the transaction trigger table and the form set definition table is the copy count. The copy count specifies the number of copies of an section to be printed for a given recipient.

In the FORM.DAT file, there can be multiple copy counts—one for each recipient for each section that makes up a form. However, in the SETRCPTB.DAT file, there is only one copy count field for each entry. A single SETRCPTB.DAT entry can reference multiple recipients however, so that one copy count field can be applied to more than one recipient.

NOTE: You can also use GVM or DAL variables to set the copy count for a recipient. For more information see the [Docucreate User Guide](#) or the [Documaker Studio User Guide](#).

The copy count is a typical interaction between the FORM.DAT and the SETRCPTB.DAT. In this example, note from the FORM.DAT that the form DECPAGE is made up of the sections PRUNAME, COMDEC1, COMDEC2, and COMDEC3. The other form in the FORM.DAT is VARFLD, which is made up of one section VARFIELD.

All the sections that make up DECPAGE and VARFLD have individual copy counts associated with each recipient. Note that the sections COMDEC2 and VARFIELD have their copy counts set to zero (o) for each recipient. This means that the default copy counts for these sections is zero (o), and if these forms are included in a form set, these sections will not print for any of the listed recipients unless their copy counts are changed by the SETRCPTB.DAT table.

Now looking at the SETRCPTB.DAT file, the first entry causes the form DECPAGE to be loaded, provided the search mask criteria is true (which it is in this case). This first entry is known as a form level trigger because the section name field has been left blank. While the first SETRCPTB.DAT entry references only INSURED and AGENT in the recipient list field, the form is also triggered for COMPANY as well because COMPANY is listed in the FORM.DAT with a copy count of 1 for all sections that make up DEC PAGE except COMDEC2. COMDEC2 is included in DEC PAGE for recipients INSURED and AGENT because they are in the form level SETRCPTB.DAT entry recipients list field.

The second SETRCPTB.DAT line is a section level entry, referencing the section COMDEC2 in the form DECPAGE. The purpose of this section level entry is to set the copy count of the section COMDEC2 (which defaults to zero (o) in the FORM.DAT) so that it will be included in or excluded from the DEC PAGE if the conditions in its SETRCPTB.DAT entry are true (more on this in Example 3).

In this example, COMDEC2 has already been included for INSURED and AGENT by the previous form level entry. If the conditions of this section level entry are true, the section COMDEC2 will be included for recipient AGENT with a copy count of 1 (which in this case is redundant since the previous form level entry already did this). However, since the section level entry conditions are false, the copy count of COMDEC2 for AGENT is set to zero (o). Looking at the POL file, COMDEC 2 only printed for INSURED, because the copy count for AGENT was set to zero (o).

The final three SETRCPTB.DAT entries are all form level entries for VARFLD. Note that in the FORM.DAT, VARFLD, which is composed of one section, VARFIELD has two recipients, INSURED and COMPANY, both of which have copy counts of zero (0). The three SETRCPTB.DAT entries for VARFLD each reference a different recipient in the recipient list field and assign them copy counts. COMPANY gets 1 copy, INSURED gets 2 copies, and AGENT gets 3 copies. However, looking at the POL file, VARFLD printed once for COMPANY and twice for INSURED, but it did not print at all for AGENT. This is because, even though AGENT was included in the SETRCPTB.DAT entry, AGENT was never an original recipient for VARFLD in the FORM.DAT.

FORM.DAT file

```
;SAMPCO;LB1;DEC PAGE;;R;;PRUNAME|D<INSURED(1),COMPANY(1),AGENT(1)>/
COMDEC1|DS<INSURED(1),COMPANY(1),AGENT(1)>/
COMDEC2|DS<INSURED(0),COMPANY(0),AGENT(0)>/
COMDEC3|DS<INSURED(1),COMPANY(1),AGENT(1)>;

;SAMPCO;LB1;VARFLD;NEW FORM;RD;;VARFIELD|D<INSURED(0),COMPANY(0)>;
```

SETRCPTB.DAT file

```
;SAMPCO;LB1;DEC
PAGE;;T1;INSURED,AGENT;11,HEADERREC,96,~0;0;0;1;;;
;SAMPCO;LB1;DEC
PAGE;COMDEC2;T1;AGENT;11,HEADERREC,11,SPCIALREC,25,Special;0;0;0;1;
;;;

;SAMPCO;LB1;VARFLD;;T1;COMPANY;11,HEADERREC,96,~0;0;0;0;1;;;
;SAMPCO;LB1;VARFLD;;T1;INSURED;11,HEADERREC,96,~0;0;0;0;2;;;
;SAMPCO;LB1;VARFLD;;T1;AGENT;11,HEADERREC,96,~0;0;0;0;3;;;

```

POL file

```
;SAMPCO;LB1;DECPAGE;;R;;PRUNAME|D<INSURED,COMPANY,AGENT>/
COMDEC1|DS<INSURED,COMPANY,AGENT>/COMDEC2|DS<INSURED,>/
COMDEC3|DS<INSURED,COMPANY,AGENT>;

;SAMPCO;LB1;VARFLD;NEW FORM;RD;;VARFIELD|DN<COMPANY>;

;SAMPCO;LB1;VARFLD;NEW FORM;RD;;VARFIELD|DN<INSURED(2),>;

\ENDDOCSET\ 1234567

;SAMPCO;LB1;DECPAGE;;R;;PRUNAME|D<INSURED,COMPANY,AGENT>/
COMDEC1|DS<INSURED,COMPANY,AGENT>/COMDEC2|DS<INSURED,>/
COMDEC3|DS<INSURED,COMPANY,AGENT>;

;SAMPCO;LB1;VARFLD;NEW FORM;RD;;VARFIELD|DN<COMPANY>;

;SAMPCO;LB1;VARFLD;NEW FORM;RD;;VARFIELD|DN<INSURED(2),>;

\ENDDOCSET\ 3234567

;SAMPCO;LB1;DECPAGE;;R;;PRUNAME|D<INSURED,COMPANY,AGENT>/
COMDEC1|DS<INSURED,COMPANY,AGENT>/COMDEC2|DS<INSURED,>/
COMDEC3|DS<INSURED,COMPANY,AGENT>;

;SAMPCO;LB1;VARFLD;NEW FORM;RD;;VARFIELD|DN<COMPANY>;

;SAMPCO;LB1;VARFLD;NEW FORM;RD;;VARFIELD|DN<INSURED(2),>;

\ENDDOCSET\ 5234567
```

USING TRANSACTION CODES

In this example, the same environment as in the first example, [Specifying Copy Counts and Sections](#), is used. In this case, however, the second entry in the SETRCPTB.DAT has been slightly modified. The transaction code field has been changed from T1 to T2 to illustrate that not having the proper transaction code will cause that entry to be skipped.

In this example, the SETRCPTB.DAT section level entry that references COMDEC2 is not being evaluated because the transaction code field does not match the data contained in the extract file. The result of skipping this entry is, unlike the previous example, where COMDEC2 did not print for AGENT, in this example COMDEC2 prints for both AGENT and INSURED.

FORM.DAT file

```
;SAMPCO;LB1;DEC PAGE;;R;;PRUNAME|D<INSURED(1),COMPANY(1),AGENT(1)>/
COMDEC1|DS<INSURED(1),COMPANY(1),AGENT(1)>/
COMDEC2|DS<INSURED(0),COMPANY(0),AGENT(0)>/
COMDEC3|DS<INSURED(1),COMPANY(1),AGENT(1)>;
;SAMPCO;LB1;VARFLD;NEW FORM;RD;;VARFIELD|D<INSURED(0),COMPANY(0)>;
```

SETRCPTB.DAT file

```
;SAMPCO;LB1;DEC
PAGE;;T1;INSURED,AGENT;11,HEADERREC,96,~0;0;0;0;1;;;
;SAMPCO;LB1;DEC
PAGE;COMDEC2;T2;AGENT;11,HEADERREC,11,SPCIALREC,25,Special;0;0;0;1;
;;;
;SAMPCO;LB1;VARFLD;;T1;COMPANY;11,HEADERREC,96,~0;0;0;0;1;;;
;SAMPCO;LB1;VARFLD;;T1;INSURED;11,HEADERREC,96,~0;0;0;0;2;;;
;SAMPCO;LB1;VARFLD;;T1;AGENT;11,HEADERREC,96,~0;0;0;0;3;;;
```

POL file

```
;SAMPCO;LB1;DEC PAGE;;R;;PRUNAME|D<INSURED,COMPANY,AGENT>/
COMDEC1|DS<INSURED,COMPANY,AGENT>/COMDEC2|DS<INSURED,AGENT>/
COMDEC3|DS<INSURED,COMPANY,AGENT>;
;SAMPCO;LB1;VARFLD;NEW FORM;RD;;VARFIELD|DN<COMPANY>;
;SAMPCO;LB1;VARFLD;NEW FORM;RD;;VARFIELD|DN<INSURED(2),>;
\ENDDOCSET\ 1234567
;SAMPCO;LB1;DEC PAGE;;R;;PRUNAME|D<INSURED,COMPANY,AGENT>/
COMDEC1|DS<INSURED,COMPANY,AGENT>/COMDEC2|DS<INSURED,AGENT>/
COMDEC3|DS<INSURED,COMPANY,AGENT>;
;SAMPCO;LB1;VARFLD;NEW FORM;RD;;VARFIELD|DN<COMPANY>;
;SAMPCO;LB1;VARFLD;NEW FORM;RD;;VARFIELD|DN<INSURED(2),>;
\ENDDOCSET\ 3234567
;SAMPCO;LB1;DEC PAGE;;R;;PRUNAME|D<INSURED,COMPANY,AGENT>/
COMDEC1|DS<INSURED,COMPANY,AGENT>/COMDEC2|DS<INSURED,AGENT>/
COMDEC3|DS<INSURED,COMPANY,AGENT>;
;SAMPCO;LB1;VARFLD;NEW FORM;RD;;VARFIELD|DN<COMPANY>;
;SAMPCO;LB1;VARFLD;NEW FORM;RD;;VARFIELD|DN<INSURED(2),>;
\ENDDOCSET\ 5234567
```

SETTING UP SEARCH MASK AND SECTIONS

There are two search mask fields in the SETRCPTB.DAT table structure. The first search mask is known as the *counter search mask* because it works with the overflow counters that immediately follow it in the transaction trigger table format, provided that the overflow flag is set.

The second search mask is known as the *true/false search mask*. Both search masks can be used to set conditions to evaluate whether a set recipient entry should be executed. In this example, the second SETRCBTP.DAT entry that references COMDEC2 has a multiple condition counter search mask.

NOTE: If you want the system to stop searching after it finds the first match, use the true/false search mask instead of the counter search mask. If you place the search mask in the counter search mask field, the system finds the first match and then looks for multiple occurrences.

The first entry in the SETRCPTB.DAT table causes the form DEC PAGE to be triggered for recipients INSURED and AGENT. All sections that make up DEC PAGE and have INSURED and/or AGENT as recipients (from the FORM.DAT file) are triggered with a copy count of 1 for each recipient. The second SETRCPTB.DAT entry is a section level entry that references COMDEC2.

The search mask in this entry will obviously fail because the first condition looks for HEADERREC at offset 11 and the second condition also looks at offset 11, but for SPICALREC. Both conditions cannot be true at the same time, so the search mask fails. The result of this section level search mask failing is to set the copy count for the recipients in the recipient list field, in this case AGENT, to zero (0).

Were the search mask true, AGENT would have been set to a copy count of 1 (which would be no change, since AGENT already had a copy count of 1 for COMDEC2).

Looking at the POL file, COMDEC2 was printed only for INSURED because the copy count of COMDEC2 for AGENT was set to zero (0) when the section level entry in the SETRCPTB.DAT file failed.

FORM.DAT file

```
;SAMP CO;LB1;DEC PAGE;R;PRUNAME|D<INSURED(1),COMPANY(1),AGENT(1)>/
COMDEC1|DS<INSURED(1),COMPANY(1),AGENT(1)>/
COMDEC2|DS<INSURED(0),COMPANY(0),AGENT(0)>/
COMDEC3|DS<INSURED(1),COMPANY(1),AGENT(1)>;
```

```
;SAMP CO;LB1;VARFLD;NEW FORM;RD;;VARFIELD|D<INSURED(0),COMPANY(0)>;
```

SETRCPTB.DAT file

```
;SAMP CO;LB1;DEC
PAGE;;T1;INSURED,AGENT;11,HEADERREC,96,~0;0;0;0;1;;;
```

```
;SAMP CO;LB1;DEC
PAGE;COMDEC2;T1;AGENT;11,HEADERREC,11,SPICALREC,25,Special;0;0;0;1;
;;;
```

```
;SAMP CO;LB1;VARFLD;;T1;COMPANY;11,HEADERREC,96,~0;0;0;0;1;;;
```

```
;SAMP CO;LB1;VARFLD;;T1;INSURED;11,HEADERREC,96,~0;0;0;0;2;;;
```

```
;SAMP CO;LB1;VARFLD;;T1;AGENT;11,HEADERREC,96,~0;0;0;0;3;;;
```

POL File

```
;SAMPCO;LB1;DECPAGE;;R;;PRUNAME|D<INSURED,COMPANY,AGENT>/  
COMDEC1|DS<INSURED,COMPANY,AGENT>/COMDEC2|DS<INSURED,>/  
COMDEC3|DS<INSURED,COMPANY,AGENT>;  
  
;SAMPCO;LB1;VARFLD;NEW FORM;RD;;VARFIELD|DN<COMPANY>;  
  
;SAMPCO;LB1;VARFLD;NEW FORM;RD;;VARFIELD|DN<INSURED(2),>;  
  
\ENDDOCSET\ 1234567  
  
;SAMPCO;LB1;DECPAGE;;R;;PRUNAME|D<INSURED,COMPANY,AGENT>/  
COMDEC1|DS<INSURED,COMPANY,AGENT>/COMDEC2|DS<INSURED,>/  
COMDEC3|DS<INSURED,COMPANY,AGENT>;  
  
;SAMPCO;LB1;VARFLD;NEW FORM;RD;;VARFIELD|DN<COMPANY>;  
  
;SAMPCO;LB1;VARFLD;NEW FORM;RD;;VARFIELD|DN<INSURED(2),>;  
  
\ENDDOCSET\ 3234567  
  
;SAMPCO;LB1;DECPAGE;;R;;PRUNAME|D<INSURED,COMPANY,AGENT>/  
COMDEC1|DS<INSURED,COMPANY,AGENT>/COMDEC2|DS<INSURED,>/  
COMDEC3|DS<INSURED,COMPANY,AGENT>;  
  
;SAMPCO;LB1;VARFLD;NEW FORM;RD;;VARFIELD|DN<COMPANY>;  
  
;SAMPCO;LB1;VARFLD;NEW FORM;RD;;VARFIELD|DN<INSURED(2),>;  
  
\ENDDOCSET\ 5234567
```

USING THE RECIPIF RULE

The RECIPIF rule is the primary rule used in the custom rule field. There are other rules which have been written for specific implementations that have been used in this field, but the RECIPIF rule is a part of base. The RECIPIF rule allows for customized search mask evaluations.

In this example, the RECIPIF rule is being used to evaluate two different conditions:

- does '1995' exist beginning at offset 51 in records with HEADERREC beginning at offset 11
- does 'T1' exist at offset 45 in records with FRMLSTREC beginning at offset 11

Looking at the entry in the SETRCPTB.DAT, notice that there are no search masks - only the RECIPIF rule is being used. Following the Search Mask 2 field, the rule name appears, and the rule itself appears in the following field. Each element of the rule is separated by double colons (::).

The first RECIPIF statements assign variables to the search criteria. In this case, A is assigned to the information appearing in the four characters beginning at offset 51 in records with HEADERREC beginning at offset 11. And B is assigned to the information appearing in the two characters beginning at offset 45 in records with FRMLSTREC beginning at offset 11.

The next RECIPIF statement sets up the evaluation logic for the rule. What should A equal? What should B equal? Should both conditions be true, or just one? In this case, A should be '1995' and B should be 'T1', and both need to be those values for the rule to be evaluated as true. An OR condition could have been used, which would have been true if either A or B matched their desired values.

The next RECIPIF statements set the return values. In this case, if A='1995' and B='T1', then a '1' is returned (note that the boolean '1' is enclosed both in quotes and carats, such as "^1^"). If those conditions are not met, then return a Boolean zero (0). These return values can be reversed to return a zero (0) when the RECIPIF criteria is true and a one (1) when false, should the need arise in a particular implementation. The last RECIPIF entry is the END statement. Here is an example of the RECIPIF rule syntax:

```
;recipif;var1={offset,value offset,length}::var2={offset,value
offset,length} ::if((var1='var1value') boolean
(var2='var2value'))::return("^#^")::else::return("^0^")::end::;
```

NOTE: There is a space between offset,value and offset,length .

FORM.DAT file

```
;SAMPCO;LB1;DEC PAGE;;R;;PRUNAME|D<INSURED(0),COMPANY(0),AGENT(0)>/
COMDEC1|DS<INSURED(0),COMPANY(0),AGENT(0)>/
COMDEC2|DS<INSURED(0),COMPANY(0),AGENT(0)>/
COMDEC3|DS<INSURED(0),COMPANY(0),AGENT(0)>;
```

SETRCPTB.DAT file

```
;SAMPCO;LB1;DEC
PAGE;;INSURED,AGENT;;0;0;0;1;;recipif;A={11,HEADERREC
51,4}::B={11,FRMLSTREC 45,2}::if((A='1995') AND
(B='T1'))::return("^1^")::else::return("^0^")::end::;
```

POL file

```
;SAMPCO;LB1;DEC PAGE;;R;;PRUNAME|D<INSURED,AGENT>/  
COMDEC1|DS<INSURED,AGENT>/COMDEC2|DS<INSURED,AGENT>/  
COMDEC3|DS<INSURED,AGENT>;
```

```
\ENDDOCSET\ 1234567
```

```
;SAMPCO;LB1;DEC PAGE;;R;;PRUNAME|D<INSURED,AGENT>/  
COMDEC1|DS<INSURED,AGENT>/COMDEC2|DS<INSURED,AGENT>/  
COMDEC3|DS<INSURED,AGENT>;
```

```
\ENDDOCSET\ 2234567
```


USING AUTOMATIC OVERFLOW

In some cases, there is information on a form that will repeat an unknown number of times. For example, an auto insurance policy may contain a form that lists the vehicles owned by the insured. The number of vehicles will vary from one insured to another, so there is no way to know in advance how many lines will be needed on a form to list the vehicles. Overflow exists to handle these situations.

There are two types of overflow in the transaction trigger table, forced and automatic. In this example, automatic overflow is used. In automatic overflow, the system automatically determines how many entries exist and inserts them in the form.

Looking at the SETRCPTB.DAT, there is only one section level entry, referencing the section cgdcbd. Looking at the FORM.DAT, section cgdcbd has a default copy count of zero (0), while all the other sections have a default copy count of one (1) for all recipients. So, triggering the section cgdcbd will trigger the remaining sections that make up the form CGDEC.

The SETRCPTB.DAT entry has a simple counter search mask and has the overflow field (occurrence flag) set. The next two overflow-related fields are set to zero (0), so we know that this is an automatic overflow situation.

When this SETRCPTB.DAT entry is executed, it will keep track of the number of records that exist in the extract file that meet this criteria and automatically insert that number of cgdcbd sections into the form CGDEC. Looking at the POL file in this example, many cgdcbd sections were inserted into the form to reflect the number of entries in the extract file that met the specified transaction trigger search criteria.

FORM.DAT file

```
;FSI;GL;CGDEC;General Liability
Declarations;RD;;cgdctp|FDSOX<INSURED(1),COMPANY(1)>/
cgdcdb|RDS<INSURED(0),COMPANY(0)>/
cgdcbt|RDS<INSURED(1),COMPANY(1)>/
cgdcft|RDSOY<INSURED(1),COMPANY(1)>;
```

SETRCPTB.DAT file

```
;FSI;GL;CGDEC;cgdcdb;T1;INSURED,COMPANY;11,CLSSCDREC;1;0;0;1;;;;
```

POL file

```
;FSI;GL;CGDEC;General Liability
Declarations;RD;;cgdctp|FDSOX<INSURED,COMPANY>/
cgdcdb|RDS<INSURED,COMPANY>/cgdcdb|RDS<INSURED,COMPANY>/
cgdcdb|RDS<INSURED,COMPANY>/cgdcdb|RDS<INSURED,COMPANY>/
cgdcdb|RDS<INSURED,COMPANY>/cgdcdb|RDS<INSURED,COMPANY>/cgdcdb|\
RDS<INSURED,COMPANY>/cgdcdb|RDS<INSURED,COMPANY>/
cgdcdb|RDS<INSURED,COMPANY>/cgdcdb|RDS<INSURED,COMPANY>/
cgdcdb|RDS<INSURED,COMPANY>/cgdcdb|RDS<INSURED,COMPANY>/
cgdcdb|RDS<INSURED,COMPANY>/cgdcdb|RD\
S<INSURED,COMPANY>/cgdcdb|RDS<INSURED,COMPANY>/
cgdcdb|RDS<INSURED,COMPANY>/cgdcdb|RDS<INSURED,COMPANY>/
cgdcdb|RDS<INSURED,COMPANY>/cgdcdb|RDS<INSURED,COMPANY>/
cgdcdb|RDS<INSURED,COMPANY>/cgdcdb|RDS<INSURED,COMPANY>/
cgdcdb|RDS<INSURED,COMPANY>/cgdcdb|RDS<
INSURED,COMPANY>/cgdcdb|RDS<INSURED,COMPANY>/
cgdcdb|RDS<INSURED,COMPANY>/cgdcdb|RDS<INSURED,COMPANY>/
cgdcdb|RDS<INSURED,COMPANY>/cgdcdb|RDS<INSURED,COMPANY>/
cgdcft|RDSOY<INSURED,COMPANY>/cgdctp|FDSOX<INSURED,COMPANY>/
cgdcdb|RDS<INSURED,COMPANY>/cgdcdb|RD\
```

```
S<INSURED, COMPANY>/cgdcbb|RDS<INSURED, COMPANY>/  
cgdcbb|RDS<INSURED, COMPANY>/cgdcbb|RDS<INSURED, COMPANY>/  
cgdcbb|RDS<INSURED, COMPANY>/cgdcbb|RDS<INSURED, COMPANY>/  
cgdcbb|RDS<INSURED, COMPANY>/cgdcbb|RDS<INSURED, COMPANY>/  
cgdcbb|RDS<INSURED, COMPANY>/cgdcbb|RDS<  
  
INSURED, COMPANY>/cgdcbb|RDS<INSURED, COMPANY>/  
cgdcbb|RDS<INSURED, COMPANY>/cgdcft|RDSOY<INSURED, COMPANY>;  
  
\ENDDOCSET\ 5234567
```

USING FORCED OVERFLOW

In this example, forced overflow is used. Forced overflow differs from automatic overflow in that there are a set number of overflow entries that can be placed on a given form.

For example, if a form is designed to list all the vehicles owned by an insured, the form designer might have a section that has room to list up to two vehicles. For insureds with two or less vehicles, only that one section is needed. However, for insureds with more than two vehicles, the designer has a separate add-on section to list the remaining vehicles. Forced overflow is used in situations such as this.

In this example, there are two sections in the FORM.DAT that make up the form FCP DEC. The first section, FCPDEC, is the main section, and the second section, FCPDEC2, is the overflow section. Both sections have copy counts of zero (0), allowing the SETRCPTB.DAT entries to control the copy counts.

The first SETRCPTB.DAT entry triggers the form for all recipients (in this case INSURED), leaving the copy counts set to zero (0). The next entry sets FCPDEC's copy count to 1 if the search mask is true. The final SETRCPTB.DAT entry is the forced overflow entry. The same search criteria is used, but the overflow (occurrence) flag is set.

The next two overflow fields specify how many entries are to be split among the two sections. The records per overflow section (6 in this example), specifies how many records will fit on the FCPDEC2 overflow section. The next field, records per first section, specifies how many records will fit on the primary section FCPDEC (2 in this example). So, FCPDEC2 will only be triggered if the search mask criteria is true and there are more than 2 occurrences of this record type.

Looking at the POL file, FCPDEC2 was triggered twice, so there must have been at least 9 overflow records. The first two went on the first section FCPDEC, the next six on the first FCPDEC2 section, and the remaining on the second FCPDEC2 section.

FORM.DAT file

```
;FSI;CPP;FCP DEC;FCPDEC OVERFLOW;RO;;FCPDEC|D<INSURED(0)>/
FCPDEC2|D<INSURED(0)>;
```

SETRCPTB.DAT file

```
;FSI;CPP;FCP DEC;;T1;INSURED;11,PREMLCREC;0;0;0;0;;;
;FSI;CPP;FCP DEC;FCPDEC;T1;INSURED;11,PREMLCREC;0;0;0;1;;;

;FSI;CPP;FCP DEC;FCPDEC2;T1;INSURED;11,PREMLCREC;1;6;2;1;;;
```

POL file

```
;FSI;CPP;FCP DEC;FCPDEC OVERFLOW;RO;;FCPDEC|D<INSURED>/
FCPDEC2|D<INSURED>/FCPDEC2|D<INSURED>;

\ENDDOCSET\ 4234557
```

SETTING SEARCH MASKS AND RECIPIENTS

In this example, two transaction trigger table concepts are illustrated. First, notice that there are two search masks in the SETRCPTB.DAT entries. Both the counter and true/false search masks are being used. Also, in the recipient selection from the SETRCPTB.DAT is used.

The FORM.DAT consists of a single form, OP654, made up of a single section, addr. section addr is defined for three recipients, INSURED, COMPANY, and AGENT, all with default copy counts of zero (0). In the SETRCPTB.DAT, there are two form level entries. In the first entry, we are looking for '1995' at offset 51 in records with HEADERREC at offset 11 and 0 at position 20 in records with FRMLSTREC at offset 11.

If both of these conditions are true, OP654 is triggered for INSURED with a copy count of 1. In the second entry, the same conditions apply for AGENT, with the exception of looking for '1996' in the counter search mask (rather than '1995').

Notice in the POL file that form OP654 was triggered for INSURED only, indicating that the second SETRCPTB.DAT entry failed. The second entry failed because '1996' did not appear at offset 51 in records with HEADERREC at offset 11. This example illustrates that the two search masks work with a logical AND condition, since the true/false search mask is true in both entries.

This example also illustrates letting the SETRCPTB.DAT control the copy counts for a form. When the section OP654 was triggered for INSURED in the first entry, it was triggered for all recipients. Since the default copy count for all recipients is zero (0), and only INSURED was set to a copy count of 1 in the SETRCPTB.DAT entry, OP654 was only printed for INSURED.

FORM.DAT file

```
;SAMPCO;LB2;OP654;First  
Letter;RD;;addr|DS<INSURED(0),COMPANY(0),AGENT(0)>;
```

SETRCPTB.DAT file

```
;SAMPCO;LB2;OP654;;T1;INSURED;11,HEADERREC,51,1995;0;0;0;1;11;FRMLSTREC,20,0;
```

```
;SAMPCO;LB2;OP654;;T1;AGENT;11,HEADERREC,51,1996;0;0;0;1;11,FRMLSTREC,20,0;
```

POL file

```
;SAMPCO;LB2;OP654;First Letter;RD;;addr|DS<INSURED,>;
```

```
\ENDDOCSET\ 6SAMPCO
```

```
;SAMPCO;LB2;OP654;First Letter;RD;;addr|DS<INSURED,>;
```

```
\ENDDOCSET\ 8SAMPCO
```

USING THE SET RECIPIENT TABLE AND EXTRACT FILES

Here are some hints on how to best use the set recipient table (SETRCPTB.DAT) and extract files:

- Fewer triggers equals better performance. Each trigger is like a condition statement for the system to evaluate. The more conditions the system has to evaluate, the slower the processing cycle.
- Use the master (M) and subordinate (S) flags to avoid repetition.

The set recipient table contains both form and section level triggers to handle cases of conditional sections on forms. A section level trigger can be used to trigger a form. This is beneficial in situations where a conditional section can trigger header and footer sections. If, however, you use it improperly, you will add redundant conditional logic at both section and form level—which slows performance.

There are two flags (S and M) which you can use to control the evaluation of section level triggers and to maintain a hierarchy of form and section with respect to the inclusion of these entities into a form set. The S flag, called the subordinate flag, identifies the section as subordinate to the parent or master form level trigger. If the form is not triggered, all underlying section triggers can be ignored, which eliminates unnecessary processing. The subordinate flag also eliminates processing the same conditional logic over and over again since the logic is only performed once at the form level.

The master form flag (M) works in a similar manner but at the form level. When you use the M flag with a form level trigger, it does not matter whether the underlying section level triggers have an S flag—all will be treated as if they did. If the logic in a master form level trigger fails, the form does not trigger and all of the form's section level triggers are ignored.

- Limit your use of the RecipIf rule.

The RecipIf rule is just like the IF rule except it is used in the SETRCPTB.DAT file. The more conditions the system has to evaluate, the slower the processing cycle. Avoiding the RecipIf rule often depends on the structure of the extract file.

The ideal situation is to trigger a form or section based on one search criteria. If you want to trigger a form or section based on more than one search criteria, you may need to use the RecipIf rule. The more conditions you have, the more complicated the RecipIf rule will be. If the system has to look for a value in a given range of data instead of at an exact location, you have to add a long and complicated recipif. There is a price to pay for flexibility and it's paid in performance.

- Structure the data in your extract file to be read in the order that it will be processed. This improves performance since the system will find the next piece of data to process faster.

FORMATTING SEARCH MASKS

Here are some tips to keep in mind when formatting a search mask.

Spaces

- You cannot have a space in any part of the search mask after the comma following an offset unless you intend to search for that space in the extract file. For example,

`"10, DATA"`

is not the same as

`"10, DATA"`

In the second mask, the space is considered part of the search string.

- You cannot have spaces following *DATA* that you do not want to include in the search. For example,

`"10, DATA, 20, DATA"`

is not the same as

`"10, DATA , 20, DATA"`

In the latter, the space following the first word *DATA* is considered part of the search text.

- You can have space following the numerical offset value. For example, `"10 ,DATA"` is interpreted the same as `"10,DATA"`.

Commas

You cannot search for data which contains a comma. For instance, you cannot have a search mask of

`"10, A, B"`

where you expect to find

`"A, B"`

in your extract row.

You can, however, write the search mask to exclude every other possible character that might occur between *A* at offset 10 and *B* at offset 12. For instance, you could create this search mask:

`"10, A, 12, B, 11, ~+, 11, ~="`

assuming that the only other possible combinations are *A+B* and *A=B*.

Tildes

The tilde (~) represents a logical NOT of the search operation. The tilde must immediately follow the comma—but remember that any space after the comma is considered part of the search text.

For example, a search mask of

`"10, ~DATA"`

is only true if *"DATA"* does not occur starting at offset 10.

To search for text that begins with a tilde, include two tildes in a sequence. For example, `"10, ~~DATA"` tells the system to search for *"~DATA"* beginning at offset 10.

If, however, the tilde is not the first character in the search text, you do not duplicate the character. For instance, `"10, DATA~"` is all you have to enter to find *"DATA~"* starting at offset 10.

Parentheses

There is no way to search for text that begins with an open parenthesis. For instance, if you use a search mask like

```
"10, (, 20, DATA"
```

assuming that the open paren character would be at offset 10, you will not get the results you want.

Using the OR condition

The OR condition is defined as OFFSET,(DATA,DATA,DATA). You *must* include a comma between the offset value and the open parenthesis. In addition, you *cannot* include spaces between the comma and open parenthesis or the calculation will be mishandled.

You can have any number of search text items inside the parenthesis as long as they are separated by commas. Having only one search text inside the parenthesis is no different than not using the OR condition. For example, "10,DATA" is the same as "10,(DATA)" and "10,DATA,20,(MORE)" is the same as "10,DATA,20,MORE".

Using the NOT condition

You cannot use the tilde (NOT conditions) with OR condition data in any fashion. It cannot be used outside the parentheses, as shown here

```
OFFSET, ~ (MORE, DATA)
```

nor can you include it inside the parentheses, as shown here

```
OFFSET, (~MORE, DATA)
```

The NOT condition is not supported with the OR search criteria.

Using AND and OR conditions

You can include a mix of AND and OR conditions, but the result is an AND operation. In other words, each individual search mask operation must evaluate to TRUE before the result is assumed TRUE. Here is an example:

```
10, DATA, 20, (MORE, DATA),
```

This statement will only be TRUE when "DATA" occurs starting at offset 10 *and* "MORE" or "DATA" occur at offset 20.

Here are some additional examples:

```
10, (MORE), 10, (DATA)
```

will never be TRUE since the text at offset 10 cannot be both "MORE" and "DATA".

```
10, (MORE, DATA), 10 (SOME, DATA)
```

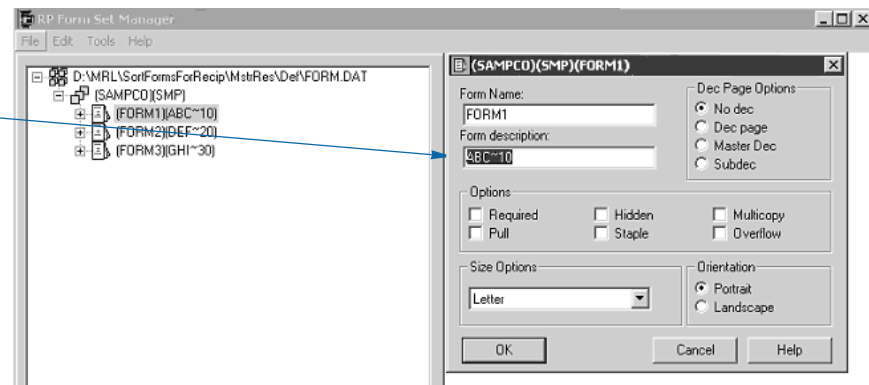
will only be TRUE when "DATA" occurs at offset 10. If the word "SOME" or "MORE" occurs at offset 10, the other part of the condition would return FALSE and the result of the entire statement would be FALSE. So, you can rewrite this statement simply as "10,DATA".

SORTING FORMS BY RECIPIENT

Use the SortFormsForRecip callback function to sort forms in a different order, depending on recipient. This function reads the given sort table and sorts the forms by recipient. A form identifier called a Document Type Number (DTN) tells the system how to sort the forms. The DTN resides in the form description of the FORM.DAT file and begins with a tilde (~).

Here is an example of how you can use Form Set Manager to specify a DTN in the FORM.DAT file.

The Form Description reads:
ABC~10.
The DTN is 10.



Keep in mind:

- This feature does not support running with the MultiFilePrint callback function.
- Use the DTN to identify the category of the form and to specify the assembly order of the form.
- Form sets with identical DTNs are sorted and printed in the order that they are triggered.
- When running in single-step mode, to preserve the order of the original forms being triggered and the NA data being written, these rules must be set in this order in the AFGJOB.JDT file:

```
;PrintFormset;;  
;WriteOutput;;  
;WriteNaFile;;
```

Otherwise, the POLFILE.DAT and NAFILE.DAT files will be out of sync.

- If a form should print for a particular recipient and it is omitted from the sort table, the system warns you. For example, suppose Form1 with a DTN of 10 should be printed for RECIPIENT1 but this form was not specified in the sort table. Here is an example of the warning you would see in the error file:

```
Warning: Document <FORM1>, Description <One~10>  
Recipient <RECIPIENT1> has no matching recipient codes in sort table.
```

Although these error messages do not stop the processing, the result will not be sorted correctly.

INI files Here is how you set up your INI file:

```
< Print >
  CallBackFunc= SortFormsForRecip
< Sort_Forms >
  TableName = ..\MstrRes\Table\sort.tbl
```

This tells the system to use a sort table called SORT.TBL.

Keep in mind, when using the SortFormsForRecip rule on UNIX platforms, you have to enter the extract path with *forward* slashes, as shown here:

```
< Sort_Forms>
  TableName = /mstrres/table/sort.tbl
```

Sort tables Here is an example of a sort table called SORT.TBL:

```
;*,10,20,30;
;CUSTOMER;10,30,20;
;AGENT,OFFICE;20,30,10;
```

The first line in the sort table defines the default sort order for all recipients not defined in the sort table. The second and third lines are sort records. You set up a sort record for each different sort order.

To set up a sort record, begin with a semicolon (;), followed by the recipient names separated with commas (,). End the list of recipients with a semicolon (;). Here is an example:

```
;Recipient1,Recipient2,Recipient3;
```

Next, and on the same line, list the DTNs associated with the form sets. Separate the DTNs with commas (,) and end the list with a semicolon (;). Here is an example of a sort record:

```
;Recipient1,Recipient2,Recipient3;10,20,30;
```

Based on the form sets and the SORT.TBL file shown above, here is an excerpt from the resulting POLFILE.DAT file:

```
;SAMPCO;SMP;FORM1;One~10;R;; ImageA|D<CUSTOMER,AGENT,OFFICE>;
;SAMPCO;SMP;FORM1.1;Two~10;R;; IMAGEA2|D<CUSTOMER,AGENT,OFFICE>;
;SAMPCO;SMP;FORM2;Three~20;R;; IMAGEB|DS<CUSTOMER,AGENT,OFFICE>;
;SAMPCO;SMP;FORM3;Four~30;R;; IMAGEC|D<CUSTOMER,AGENT,OFFICE>;
\ENDDOCSET\ 1234567890
```

The print file for CUSTOMER will be in this order:

```
;SAMPCO;SMP;FORM1;One~10;R;; ImageA
;SAMPCO;SMP;FORM1.1;Two~10;R;; IMAGEA2
;SAMPCO;SMP;FORM3;Four~30;R;; IMAGEC
;SAMPCO;SMP;FORM2;Three~20;R;; IMAGEB
```

The print file for AGENT and OFFICE will be in this order:

```
;SAMPCO;SMP;FORM2;Three~20;R;; IMAGEB
;SAMPCO;SMP;FORM3;Four~30;R;; IMAGEC
;SAMPCO;SMP;FORM1;One~10;R;; ImageA
;SAMPCO;SMP;FORM1.1;Two~10;R;; IMAGEA2
```

SUMMARY

This chapter explains the major principles illustrated in the previous examples and reviews the triggering logic used by the transaction trigger table. Keep in mind that the transaction trigger table cannot be viewed in isolation; it works with the form set definition table, and both must be examined to predict triggering behavior. The form set definition table defines the default recipients and copy counts for form sections. The transaction trigger table may override some or all of the form set definition table settings.

In the case of the copy count, the form set definition table defines a default copy count for each recipient of each form section. A transaction trigger table entry defines a copy count for one or more recipients. This transaction trigger table copy count may be the same or different from that already defined in the form set definition table. When evaluated, a transaction trigger table entry's copy count will override the one already defined for those recipients in the form set definition table for that form section.

A similar relationship exists between the form set definition table and the transaction trigger table for recipients. The form set definition table defines the default recipients for a form section. The transaction trigger table can be used to change the copy count for those recipients. And if a transaction trigger table entry sets the copy count to zero (0) for a particular recipient, it has the effect of removing that form section for that recipient. Keep in mind that a recipient may not be included in a transaction trigger entry unless that recipient has already been included for that form section in the form set definition table.

For a transaction trigger table entry to be evaluated, three of the first five transaction trigger fields (GroupName 1, GroupName 2, and Transaction Code) must match some records within the extract file. For example, if there are no records with the transaction code specified in the trigger, that trigger will be skipped. If extract records exist that match these three fields, the remaining fields of that trigger are evaluated. A blank transaction code field is treated as a wildcard, accepting any transaction code for the trigger.

Of the two transaction trigger table search masks, the true/false mask is evaluated first. Once an extract file record has been found that meets the true/false search mask criteria, the counter search mask is evaluated next, if one is present. The counter and true/false search masks work the same way when the overflow flag is not set. But when the overflow flag is set, the counter search mask criteria search does not stop at the first matching extract file record - the system will continue to search for all matching extract file records.

When the system evaluates the counter or true/false mask, the system searches through all the records in the extract file for the specified transaction. If any of the transactions match the search criteria, the condition is considered true. If there are multiple records with the same search criteria, the system will evaluate all of them. If any of these records match the search criteria, the trigger condition is considered true.

For example, if Search Mask 2 is specified as 11,SPECIAL,20,5 and there are two records containing SPECIAL at offset 11, the first one an A at offset 20 and the second one with a 5 at offset 20, the system will evaluate both records and finding the second meets the search criteria, the trigger condition is considered true. The system will stop searching once a True condition is found, except in overflow situations. For overflow situations, the system will not stop searching. Rather, it will keep searching and counting the number of True conditions. The system will then trigger the number of sections or forms based on that count.

When the custom rule RECIPIF is evaluated, the search is different than that used for Search Masks 1 and 2 in that when the system only evaluates the first found record which matches the search criteria. For example, if the custom rule is specified as follows:

```
;Recipif;A={11,SPECIAL
51,4}::if(A='1995')::return("^1^")::else::return("^0^")::end::
```

There are two records in the extract file containing SPECIAL at offset 11. The first one has 1994 at offset 51, and the other has 1995 at offset 51. When the system stops searching once it finds the first record which matches the search criteria. In this case, it evaluates the record contains 1994 and determines that the trigger condition is false.

When the overflow flag is set, the next two transaction trigger table entry fields, records per overflow section and records per first section, are examined. If both of these fields are set to zero (0), the system will automatically handle the overflow. If these fields are used, they specify how many entries are to be split among the two sections. The records per overflow section specifies how many records will fit on the overflow section. The next field, records per first section, specifies how many records will fit on the primary section.

At a minimum, a transaction trigger table entry must contain a GroupName1 value, a GroupName2 value, a Form Name value, and a Copy Count value. A section level trigger must also contain a section Name value. At a minimum, the three overflow fields must be set to zero (0). A blank Transaction Code field acts as a wildcard, accepting any transaction code. A blank Recipient List field will default to the recipients named in the form set definition table. And the two Search Mask fields and the Custom Rule field may be used as needed to produce the desired triggering results.

CHAPTER 5

Working with Fonts

A font is a collection of letters, symbols, and numbers which share a particular design. The system provides a tool, called Font Manager, which lets you organize sets of fonts for section creation and printing needs.

NOTE: Both Documaker Studio and Docucreate include a Font Manager.

This chapter provides general information on font concepts and types and moves into the specifics of setting up fonts and using Font Manager.

Topics included are as follows:

- [Understanding Font Concepts on page 172](#)
- [Using Code Pages on page 178](#)
- [Types of Fonts on page 187](#)
- [Using System Fonts on page 190](#)
- [Using Font Cross-Reference Files on page 197](#)
- [International Language Support on page 201](#)
- [Setting Up PostScript Fonts on page 205](#)
- [Font Naming Conventions on page 210](#)
- [Using Font Manager on page 211](#)
- [Generating Files using Font Manager on page 240](#)
- [Mapping Fonts for File Conversions on page 244](#)

NOTE: The system also includes several utilities you can use to work with fonts. These utilities are mentioned where appropriate throughout this chapter and are discussed in detail in the [Docutoolbox Reference](#).

UNDERSTANDING FONT CONCEPTS

FONT TERMINOLOGY

The following is a glossary of some common typographic terms you may encounter when working with fonts.

Typography is the art and technique of selecting and arranging type styles, point sizes, line lengths, line spacing, character spacing, and word spacing for typeset applications.

A *typeface* is a unique design of upper- and lower-case characters, numerals, and special symbols. Times-Roman, Arial-Italic, Courier-Bold are examples of typefaces.

A *font* is the implementation, for a specific device, of one typeface. A font contains a group of characters (letters, numbers, punctuation, and so on) which have a specific form and size. As you can see below, a Courier font is one which is designed to look like it was produced by a typewriter.

Courier fonts look like text produced by a typewriter.

A *font family* is family of related font typefaces. Times-Roman, Times-Bold, Times-Italic, and Times-BoldItalic are typefaces which belong to the Times font family.

Font size refers to the vertical point size of a font, where a point is about 1/72 of an inch.

There are several other terms used to describe the characteristics of a font, including:

- Ascender
- Baseline
- Descender

The *ascender* is the portion of a lowercase character that extends above its main body, as in the vertical stem of the character *b*.

_____ ascender
bcxy

The *baseline* is an imaginary line upon which the characters in a line of type rest.

bcxy — — — baseline

The *descender* is the portion of a lowercase character that extends below the baseline, as in *y* or *g*.

bcxy_____ descender

Kerning is the process of decreasing space between two characters for improved readability, such as tucking a lowercase *o* under an uppercase *T*. A variation of kerning, called *tracking*, involves decreasing the amount of space between all characters by a specified percentage.

Leading is the amount of vertical space between lines of text. Leading (pronounced *led-ding*) is measured from baseline to baseline. On old hot-type printing presses, this was done by inserting strips of lead between the cast type.

Fonts are measured in *points*. A point is a typographical unit of measure which equals about 1/72 of an inch. For example, this is a **16 point font** while the rest of the line uses a 10 point font.

A *pica* is another typographical unit of measurement equal to 12 points. There are about 6 picas in one inch.

A *twip* is yet another typographical unit of measurement equal to 1/20th of a point. There are 1440 twips to one inch, 567 twips to one centimeter.

Pitch refers to the amount of horizontal space used for each character of fixed-width fonts. This is often specified in characters-per-inch (CPI). Typically, 10-pitch equals 12 point, 12-pitch equals 10 point, and 15-pitch equals 8 point type, but some fonts use other equivalencies.

Sans serif means without serifs and refers to a character (or typeface) that lacks serifs, such as Arial or Helvetica.

A *serif* is an ornamental aspect of a character. A serif typeface is one whose characters contain serifs (such as Times Roman or Courier).

Spacing can either be fixed or proportional. In a fixed font, such as Courier, every character occupies the same amount of space. In a proportional font, such as Arial or Times, characters have different widths.

Stroke weight refers to the heaviness of the stroke for a specific font. This is usually indicated in font names by including words such as Light, Regular, Book, Demi, Heavy, Black, and Extra Bold.

The *style* of a font is whether it is plain, bold, or italic.

National language terminology

Here are some additional terms you may encounter when working with fonts and supporting international languages.

National character handling is dependent on both the language used, and on the country. In many cases, the language is used only in one country (such as Japanese in Japan). In other cases, there is a national variant of the language (such as Canadian French).

A *code page* is a table which defines the mapping in a computer of each of these characters to a unique hexadecimal number, called a code point. There are three families of code pages: EBCDIC, ASCII, and ISO.

A *character set* defines which characters must be supported for a specific language.

Single byte character sets (SBCS) are character sets which can be defined using a single byte code point (code points range from 0 to 255). Most languages can be defined using an SBCS.

Double byte character sets (DBCS) are character sets which contain so many characters that they require two bytes to define the valid code point range. Languages which require a DBCS are Japanese (Kanji), Korean, and Chinese (both Traditional and Simplified). For example, the Kanji character set uses approximately 6,700 characters out of a total of 65,000 valid code points provided by a DBCS code page.

Multiple byte character sets (MBCS) use both single and double byte code points. This is also referred as a *combined code page*. For example, the combined Japanese code page 932 consists of a SBCS code page 897 and a DBCS code page 301. These code pages use the Shift JIS encoding defined by the Japanese Industry Standard Association, and contains Kanji, Hiragana, and Katakana characters.

Unicode is a character coding system designed to support the interchange, processing, and display of the written texts of the diverse languages of the modern world. In its current version (3.2), the Unicode standard contains over 95,000 distinct coded characters derived from dozens of supported scripts. These characters cover the principal written languages of the Americas, Europe, the Middle East, Africa, India, Asia, and Pacifica. Support for Unicode is growing among operating systems, such as Windows XP, and programming languages, such as Java.

NOTE: Beginning in version 10.2, the system includes support for Unicode. Specific information on how to use Unicode is available in a separate document, entitled *Using Unicode*. To receive this document, contact Documaker Support.

Bi-directional (BIDI) languages or *Extended SBCS* languages are languages which display text in a right-to-left manner and numbers in a left-to-right manner. Hebrew and Arabic are BIDI languages.

ANSI is an acronym for the American National Standards Institute. The Windows ANSI character set is based on code page ISO 8859-x plus additional characters based on an ANSI draft standard.

ASCII is an acronym for the American Standard Code for Information Interchange. ASCII is a 7-bit code that is a US national variant of ISO 646.

Program Integrated Information (PII) includes all text in messages, menus, and reports which is displayed to the user. To provide national language support, all PII text must be isolated for easy translation.

Enabled is a term used to indicate an application that has been altered to handle input, display, and editing of double byte languages (such as Japanese) and bi-directional languages (such as Arabic).

Translated is a term used to indicate an application which has been enabled and has had its Program Integrated Information translated into the national language. A translated application must also support various country settings, such as time, date, currency, and sorting.

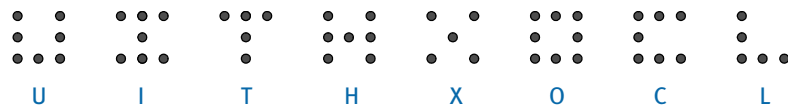
AFM is an extension used with Adobe® PostScript® font files. It stands for Adobe Font Metrics. AFM files are text files that describe a PostScript font.

HOW CHARACTERS ARE REPRESENTED

Fonts can use different methods of internally representing characters. Two categories of representing characters in fonts are known as bitmap fonts and scalable fonts.

Bitmap Fonts

Bitmap fonts describe each character as a pattern of black dots. Bitmap fonts were originally used for printer and screen devices because these devices were only capable of drawing dots. Below is crude representation of how the certain letters could be drawn as a series of dots in a 3x3 grid.



Essentially, this is what happens when a character is drawn to the screen or printed on paper. Fortunately, screen and printer fonts use a whole lot more dots per inch so that the distance between the dots becomes nearly invisible to the naked eye. By the way, this is also the reason why printed text looks better than text on the screen. Printed text often uses 300 or 600 dots per inch while your screen's resolution might be 96 dots per inch.

A different font file is required for each point size and different font files are required for different device resolutions (VGA vs. Super-VGA monitors, 300 dpi vs. 600 dpi printers).

Bitmap fonts are used primarily by printers. Bitmap fonts used by printers cannot be used for displaying text on screens because there are different internal formats and different resolutions. Printers which use bitmap fonts include HP® laser printers, IBM® AFP printers, and Xerox® Metacode printers.

Scalable Fonts

A scalable font can be scaled to any size needed. Characters of scalable fonts are internally represented as outlines (a series of straight lines and curves). These outlines can be scaled to allow characters to be rendered at different resolutions and point sizes. For example, the letter *O* may be represented as outer and inner circular lines whose interior is filled.

Outlines



Final Character



Two types of scalable fonts are TrueType and PostScript fonts.

TrueType TrueType was designed and developed by Apple Computer and Microsoft for use on the Macintosh computer and PCs running Microsoft Windows. TrueType provides a number of advantages over bitmap fonts. TrueType is WYSIWYG (what you see is what you get). The same font can be used with printers and video displays. Typically, TrueType font files have a file extension of TTF.

PostScript PostScript fonts were designed and developed by Adobe Systems Incorporated. PostScript fonts are a special implementation of a PostScript language program. PostScript fonts are scalable fonts. PostScript fonts describe each character as a series of straight-line and curved-line segments. These segments (also known as an outline) along with a flexible coordinate structure allow PostScript fonts to be scaled easily and used on different devices (video monitors and printers). PostScript printers support the PostScript language and fonts. There are several types of PostScript fonts:

- **PostScript Type 1**
When someone refers to a PostScript font, this is the type of font most often referred to. Typically, Type 1 font files have a file extension of PFB.
- **PostScript Type 3**
A Type 3 font is one whose behavior is determined entirely by the PostScript language procedures built into the font. These fonts are typically larger files than Type 1 fonts and do not take advantage of special algorithms built into the PostScript interpreter for rendering characters. This usually results in inferior output at small sizes and low resolution.
- **PostScript Type 0**
A Type 0 (zero) font is a composite font program that can contain several thousand characters, accessed by multi-byte codes. They can be used for non-Roman scripts, such as Japanese kanji.
- **PostScript Multiple Master**
Multiple master font programs are an extension of the Type 1 font format. Multiple master font programs contain a wide variety of typeface variations, such as multiple weights, character widths, and so on.

HOW COMPUTERS AND PRINTERS USE FONTS

What happens to make the letter *A* show up on the screen or print on a printer?

The key to remember is that computers and printers are not very smart. They really don't know anything about letters or punctuation characters.

When you press the letter *A* on the keyboard, the keyboard sends a number to computer. On a PC, this number is usually 65 for the letter *A*. The computer uses this number to produce the letter *A*. For simplicity, let's assume you have a bitmap screen font.

As stated before, bitmap fonts describe each character as a pattern of black dots. Let's assume these patterns are stored in the font as a series of slots where slot 0 is followed by slot 1 which is followed by slot 2, and so on. For the number 65 (letter *A*), the computer simply draws the pattern of dots stored in slot 65. When the bitmap is drawn on the screen, we see what looks like the letter *A*.

If you print the letter *A* with a bitmap font, the concept is essentially the same. The printer receives the number 65 and prints the series of dots stored in slot 65 of the printer font.

The numbers which the computer uses to represent characters are called *code points*.

USING CODE PAGES

A code page is a table which defines the mapping in a computer of each of these characters to a unique hexadecimal number, called a code point. There are three families of code pages: EBCDIC, ASCII, and ISO.

A code page is a table that defines how the characters in a language or group of languages are encoded. A specific value is given to each character in the code page. For example, in code page 850 the letter ñ (lowercase) is encoded as hex A4 (decimal 164), and the letter Ñ (uppercase) is encoded as hex A5 (decimal 165). Of particular interest are these code pages:

- Code Page 850

Code page 850 is also called the Latin-1, multilingual code page. This code page supports the alphabetic characters of the Latin-1-based languages.

- Code Page 437

Code page 437 is the standard personal computer code page. The lower 128 characters are based on the 7-bit ASCII code. The upper 128 characters contain characters from several European languages (including part of the Greek alphabet) and various graphic characters. However, some of the accented characters, such as those used in the Nordic countries, are not represented. The missing characters are available in other code pages (code page 850 will usually contain the desired characters). It contains characters required by 13 languages used in approximately 40 countries.

- Code page 1004

Code page 1004 is the equivalent of the Windows ANSI code page. It contains more international characters than the multilingual code page 850. This character set contains all characters necessary to type all major (West) European languages. This encoding is also the preferred encoding on the Internet.

ISO 8859-x character sets use code points 128 through 255 to represent national characters, while the characters in the 32 to 127 range are those used in the US-ASCII (ISO 646) character set. Thus, ASCII text is a proper subset of all ISO 8859-X character sets.

The code points 128 through 159 are typically used as extended control characters, and are not used for encoding characters. These characters are not currently used to specify anything. This character set is also used by AmigaDOS, Windows, VMS (DEC MCS is practically equivalent to ISO 8859-1) and (practically all) UNIX implementations. MS-DOS normally uses a different character set and is not compatible with this character set.

ASCII Code Pages

ASCII is an acronym for the American Standard Code for Information Interchange. ASCII code pages are used on the PC platform. Code points below 32 for ASCII code pages are considered control characters for internal uses. These code points are usually not displayable characters. Code points from 32 to 127 are usually the same in ASCII code pages and are used for English letters, numbers, and punctuation.

Where ASCII code pages differ is in the characters assigned to code points 128-255. Code points 128-255 are used for international characters, math symbols, and so on. The characters for these code points vary in other code pages.

The characters used in code points below 128 use the English letters, numbers, and punctuation commonly found in ASCII code pages. The upper 128 code points are used for characters from several European languages (including part of the Greek alphabet) and various graphic characters. However, some of the accented characters, such as those used in the Nordic countries, are not represented.

Code page 437 is known as the standard personal computer code page. These characters were originally used in the original IBM PC. This code page is still used today in U.S. English versions of DOS and Windows. The primary code page used for these platforms is also known as the OEM code page.

Code page 850 is also called the multilingual code page. This code page supports many of the characters of the Latin-based alphabet.

The following table shows code page 850. To determine the code point associated with a character, use the numbers in the first row and column in the following table. For example, the letter *A* has a code point of 65 ($64 + 1$) and the space character has a code point of 32 ($32 + 0$).

Code Page 850

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
16																
32		!	“	#	\$	%	&	‘	()	*	+	,	-	.	/
48	o	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
128	Ç	ü	é	â	ä	à	å	ç	ê	ë	è	ï	î	ì	Ä	Å
144	É	æ	Æ	ô	ö	ò	û	ù	ÿ	Ö	Ü	ø	£	Ø	×	f

160	á	í	ó	ú	ñ	Ñ	ä	ö	¿	®	¬	½	¼	ì	«	»
176						Á	Â	À	©					¢	¥	
192							ã	Ã								¤
208	ð	Ð	Ê	Ë	È		Í	Î	Ï					Ì	Î	
224	Ó	ß	Ô	Ò	õ	Õ	µ	þ	Þ	Ú	Û	Ü	Ý	Ý	-	´
240	—	±		¾	¶	§	÷	,	°	™	•	¹	²	³		

There are many more ASCII code pages which are targeted for a specific country and or language. For example, code page 863 is used for Canadian French.

Code page 1004 is the IBM equivalent of the Windows ANSI code page. It contains more international characters than the multilingual code page 850. It contains characters required by 13 languages used in approximately 40 countries. Windows uses the ANSI code page to support most of the languages used in the Western Hemisphere and Western Europe. Keystrokes are translated by Windows from the primary (OEM) code page into the ANSI code page.

The following page shows the Windows ANSI code page. To determine the code point associated with a character, use the numbers in the first row and column in the following table. For example, the letter A has a code point of 65 (64 + 1) and the space character has a code point of 32 (32 + 0).

Code Page 1004 (ANSI Code Page)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
16																
32		!	“	#	\$	%	&	‘	()	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
128	€		,	f	„	…	†	ç	^	‰	Š	‹	Œ		Ž	
144		‘	’	“	”	•	—	—	~	™	š	›	œ		ž	ÿ
160		ì	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯

176	°	±	²	³	´	μ	¶	·	,	¹	º	»	¼	½	¾	¿
192	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
208	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
224	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

EBCDIC CODE PAGES

EBCDIC is an acronym for the Extended Binary Coded Decimal Interchange Code. EBCDIC code pages are used on mainframe (z/OS) and mini computers (AS400). There are many EBCDIC code pages. EBCDIC code pages usually share the same code points for English letters, numbers, and punctuation characters. However, EBCDIC code pages use different code points than ASCII code pages for the same English letters, numbers, and punctuation characters. Code points below 64 for EBCDIC code pages are considered control characters for internal uses. These code points are usually not displayable characters.

Code page 37 is an EBCDIC code page used on many z/OS and AS400 systems. Although the code points are completely different, code page 37 shares most of the same characters as code page 1004 (ANSI). The characters associated with code points 128-159 in the ANSI code page are not defined in code page 37.

NOTE: The system uses some undefined code points (below 64) in code page 37 to try represent these characters. For maximum portability, avoid using code points 128-159 of the ANSI code page when composing forms.

The following page shows a table of code page 37. To determine the code point associated with a character, use the numbers in the first row and column in the following table. For example, the letter *A* has a code point of 193 (192 + 1) and the space character has a code point of 64 (64 + 0).

Code Page 37

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
16																
32																
48																
64			â	ä	à	á	ã	å	ç	ñ	¢	.	<	(+	

80	&	é	ê	ë	è	í	î	ï	ì	ß	!	\$	*)	;	¬
96	-	/	Â	Ã	Ä	Á	Ã	Å	Ç	Ñ		,	%	_	>	?
112	ø	É	Ê	Ë	È	Í	Î	Ï	Ì	˘	:	#	@	'	=	“
128	Ø	a	b	c	d	e	f	g	h	l	«	»	ð	ý	þ	±
144	°	j	k	l	m	n	o	p	q	r	ª	º	æ	,	Æ	¤
160	µ	~	s	t	u	v	w	x	y	z	ı	ı	Đ	Ý	Þ	®
176	^	£	¥	•	©	§	¶	¼	½	¾	[]	-	..	ˆ	×
192	{	A	B	C	D	E	F	G	H	I	-	ô	ö	ò	ó	õ
208	}	J	K	L	M	N	O	P	Q	R	¹	û	ü	ù	ú	ÿ
224	\	÷	S	T	U	V	W	X	Y	Z	²	Ô	Ö	Ò	Ó	Õ
240	o	1	2	3	4	5	6	7	8	9	³	Û	Ü	Ù	Ú	

CHARACTER SETS

You may have noticed that the largest code point shown in the earlier code page tables is 255 (240 + 15). The reason for this is that 255 is the largest value which can fit into a byte of memory. Code pages like this are said to have a single byte character set (SBCS). Some Asian languages, like Japanese and Chinese, contain so many characters that they must be represented by a double byte character set (DBCS) or a multiple byte character set (MBCS).

NOTE: Prior to version 10.2, the system only supported SBCS code pages. Version 10.2 added support for many additional languages using Unicode.

Determining Characters Used in a Printer Font

The simplest way to determine what characters are contained in a printer font is to print a FAP file which contains all possible code points. The system includes a FAP file you can use for this purpose. This FAP file looks very similar to the code page tables shown earlier in this chapter.

To print this FAP file in Image Editor, follow the steps below:

- 1 Start Image Editor.
- 2 Choose File, Open and select the Q1CDPG.FAP file, which is located in the RPEX1\FORMS directory.
- 3 Choose Tools, Font Manager and highlight the font 11016 (this FAP file only uses one font ID, 11016).

- 4 Click the Edit button. The Font Properties window appears. Click the Printers tab.
- 5 Change the AFP, PCL, or Xerox font file name to the font file name you want to test and click Ok. Then click Close to exit Font Manager and save your changes.
- 6 Select File, Print. Print the FAP file using the printer driver which corresponds to the printer font you are testing.

NOTE: Be sure to download fonts if you are testing a PCL font.

CODE PAGE NAMES

One confusing thing about code pages is that different organizations have different names for the same code pages. IBM, Microsoft, and the International Organization for Standardization (ISO) all use different names for essentially the same code page. You may hear a code page referred to by its IBM, Microsoft, or ISO name. For example, the ANSI code page is the same as IBM code page 1004, Microsoft code page 1252, and ISO code page 8859-1.

The following table shows a list of commonly used code pages. For more information, see these books:

- Developing International Software, Second Edition - Microsoft Press
- National Language Design Guide Volume 2 - IBM

Language	Country	Code Pages		
		Windows	OEM	z/OS
U.S. English	USA	1252 (ANSI)	437, 850	037

Western Hemisphere and Western Europe SBCS Code Pages

U.K. English	UK	1252 (ANSI)	850, 437	?
Brazilian Portuguese	Brazil	1252 (ANSI)	850, 437	?
Canadian French	Canada	1252 (ANSI)	850, 863	?
Danish	Denmark	1252 (ANSI)	850	?
Dutch	Netherlands	1252 (ANSI)	850, 437	?
Finnish	Finland	1252 (ANSI)	850, 437	?
French	France	1252 (ANSI)	850, 437	?
German	Germany	1252 (ANSI)	850, 437	?
Italian	Italy	1252 (ANSI)	850, 437	?
Norwegian	Norway	1252 (ANSI)	850	?
Portuguese	Portugal	1252 (ANSI)	850, 860	?
Spanish	Spain	1252 (ANSI)	850, 437	?
Swedish	Sweden	1252 (ANSI)	850, 437	?

Eastern Europe SBCS Code Pages

Russian	Russia	1251 (Cyrillic)	866, 850	?
---------	--------	-----------------	----------	---

Language	Country	Code Pages		
		Windows	OEM	z/OS
Bosnian	Bosnia	?	852, 850	?
Croatian	Croatia	1250 (Eastern Europe)	852, 850	?
Czech	Czech	1250 (E.E.)	852, 850	?
Estonian	Estonia	1257 (Baltic)	?	?
Greek	Greece	1253 (Greek)	?	?
Hungarian	Hungary	1250 (E.E.)	852, 850	?
Latvian	Latvia	1257 (Baltic)	?	?
Lithuanian	Lithuania	1257 (Baltic)	?	?
Polish	Poland	1250 (E.E.)	852, 850	?
Romanian	Romania	1250 (E.E.)	852, 850	?
Serbian-Latin	Serbia	1250 (E.E.)	852, 850	?
Slovak	Slovak	1250 (E.E.)	852, 850	?
Slovenian	Slovenia	1250 (E.E.)	852, 850	?
Turkish	Turkey	1254 (Turkish)	857, 850	?

Extended SBCS Code Pages

Arabic	Arabic speaking	1256 (Arabic)	864, 850, 437	?
Hebrew	Israel	1255 (Hebrew)	862, 850, 437	?
Thai	Thailand	874	874, 437	?

Asian DBCS Code Pages

Japanese	Japan	932	932, 942, 437, 850	?
Korean	Korea	949	949, 850, 437	?
Simplified Chinese	PRC, Singapore	936	1381, 437, 850	?

Language	Country	Code Pages		
		Windows	OEM	z/OS
Traditional Chinese	Taiwan, Hong Kong	950	938, 948, 437, 850 950, 437, 850	? ?

TYPES OF FONTS

The system uses screen and printer fonts for displaying and printing text on forms. The Family field in the FXR contains the name of the screen font to use for displaying text under Windows.

The Font File fields in the FXR contain the names of the printer fonts to use when printing text. The FXR file provides attributes of the fonts and cross references the various font file names and parameters for different printers. The FXR does not contain any printer or screen fonts, only information about printer and screen fonts. FXR files are referred to in this section but are discussed in detail in the section, [Using Font Cross-Reference Files on page 197](#).

USING SCREEN FONTS

Font Substitution in Windows

If the system cannot find a matching screen font using the information in the FXR, it will attempt to substitute a different Windows font. For Windows, the system will automatically try to substitute the following fonts for these missing fonts:

If this font is missing...	The system will substitute this font...
Courier	Courier New
Helv	Arial
Letter Gothic	Courier New
MICR	Courier New (fixed pitch) or Arial (proportional)
OCR A	Courier New (fixed pitch) or Arial (proportional)
OCR B	Courier New (fixed pitch) or Arial (proportional)
Times	Times New Roman
Times Roman	Times New Roman
Tms Rms	Times New Roman
Univers	Arial

Separate INI file control groups are used for Windows 3.1x (16-bit) and Windows 32-bit platforms for defining substitute font names. These control groups are named WINDOWSUBS and WINDOW32SUBS, respectively. Here is an example of the WINDOW32SUBS control group, which shows the defaults settings:

```
< Window32Subs >
  Univers      = Arial
  Helv         = Arial
  Letter Gothic = Courier New
  Courier       = Courier New
  Tms Rms      = Times New Roman
```

```
Times Roman    = Times New Roman
Times          = Times New Roman
```

In this example, the system substitutes the native Windows 32-bit font, Times New Roman, if the Times family font is not found. Likewise, it substitutes Courier New for Letter Gothic and Arial for Univers. If you do not have a font installed which matches the original or substituted fonts, a default font will be used instead (usually Courier).

Installing Screen Fonts in Windows

To avoid these font substitutions, you can install fonts into Windows using the Fonts folder (usually located in the Control Panel). After opening the Fonts folder, select the File, Install New Font option. The Add Fonts window appears and asks for the drive and directory in which the new TrueType font files are located. When you finish selecting the fonts you want to install, click Ok to install them.

For the system to correctly match the fonts installed under Windows, the family and face name must be spelled exactly the same as they appear on the Names tab of the Properties window for the font. Use FXR settings for FAP height, FAP width, and so on, to customize the display of a font. For information on choosing one of these Windows display fonts to match a printer font, see [Choosing Screen Fonts on page 238](#).

USING PRINTER FONTS

The system supports printer fonts for AFP, Xerox Metacode, PCL, and PostScript printers. Here is some background information you should know about each of these print platforms.

AFP

AFP fonts are designed solely for IBM's AFP printers. In AFP terminology, a font is described by three components:

- Coded fonts** A coded font file contains references to specific character set and specific code page. Coded font files always begin with the letter *X*, such as *XoDATIN8*.
- Code pages** In IBM AFP terminology, a code page file maps code points to an AFP character name in a character set file. Code page files always begin with the letter *T*, such as *T1DOCo37*.
- Character sets** A character set file contains the bitmap image of each character in the character set. Character set files always begin with the letter *C*, such as *CoFATIN8.240* or *CoFATIN8.300*. The character set file name extension (240 or 300) indicates whether the bitmap images are drawn at 240 or 300 dots per inch. Each character is given a eight letter AFP character name. For example, the letter *A* has an AFP character name of *LA020000*.

Metacode

Metacode fonts are designed solely for Xerox Metacode printers. Metacode fonts are bitmap fonts. Typically, Metacode font files have a file extension of *FNT*, such as *FXTIN8.FNT*. Characters are accessed by code points.

PCL

PCL is the Printer Control Language developed by Hewlett Packard for its LaserJet (and compatible) printers. PCL bitmap fonts are used by the system. PCL bitmap fonts can have any file name extension. The system provides PCL fonts with an extension of PCL, such as FPTIN8.PCL. Like Metacode fonts, PCL characters are accessed by code points.

PostScript Fonts

PostScript fonts were designed and developed by Adobe Systems Incorporated. PostScript fonts are actually a special implementation of a PostScript language program. PostScript fonts are scalable fonts and there are several types of PostScript fonts, PostScript Type 1 fonts are most common and are the only type supported by the system. Typically, Type 1 font files have a file extension of PFB, such as COURIER.PFB.

Each character in a PostScript font has a PostScript character name. When used as a screen font, the operating system associates code points in a code page with the appropriate PostScript character names.

NOTE: The system uses the CODEPAGE.INI file to associate code points with the appropriate PostScript characters.

TrueType Fonts

TrueType is a scalable font designed and developed by Apple Computer and Microsoft for use on the Macintosh computer and on PCs running Microsoft Windows. TrueType is WYSIWYG (what you see is what you get). The same font can be used with printers and video displays. Typically, TrueType font files have a file extension of TTF.

Adding Printer Fonts to a Font Cross-reference File

Fonts are added to an FXR file using the system's Font Manager. You can insert TrueType, PCL, AFP, Xerox Metacode, certain FormMaker II files, and other FXR files into a font cross-reference file. Font Manager is discussed in [Using Font Manager on page 211](#). For specific instructions on inserting a font into an FXR file, see [Inserting Fonts on page 232](#).

USING SYSTEM FONTS

Oracle Insurance has licensed for use and distribution with the system the following Postscript and TrueType fonts from Monotype Imaging, Inc. (formerly Agfa):

- Albany
- Arial Black
- Arial Narrow
- Courier
- Letter Gothic
- Times
- Univers
- Univers Condensed
- DocuDings
- MICR
- OCRA
- OCRB
- ZIPCODE

Albany (an Arial clone), Arial Narrow, Arial Black, and DocuDings (a Wingdings clone) are clones of commonly-used Windows fonts. The fonts are similar in appearance to the corresponding Windows fonts and have the same character width attributes. In addition, you can now use PCL, PostScript, AFP, and Metacode versions of these fonts for printing.

NOTE: Although DocuDings is very similar to Wingdings, there are some differences. For instance, code point 255 in Wingdings is the flying Windows symbol (✈). The DocuDings font displays a blank space for code point 255. The other code points (characters) are very similar in appearance but are not exact duplicates to the Wingdings font.

The Monotype font sets include the Euro character (€).

From these fonts, we have created fonts to use with AFP, PCL, and Xerox printers. These fonts let you print nearly identical forms on any supported printer. We use the following file naming convention for AFP, PCL, and Xerox printer fonts:

F T F₁ S P

For example, a 10 point bold Courier Xerox font would be named *FXCOBo.FNT*.

F	Standard Documaker system font
T	Printer type where A = AFP, P = PCL, X = Xerox 0 degree, 9 = Xerox 90 degree, 1 = Xerox 180 degree, 2 = Xerox 270 degree
F ₁	Two-character family name where AB = Albany, AL = Arial Black, AN = Arial Narrow, CO = Courier, HV = Helvetica, LG = Letter Gothic, TI = Times, UN = Univers, UC = Univers Condensed, DD = DocuDings, MI = MICR, OA = OCRA, OB = OCRB, ZP = ZIP code
S	Style where B = Bold, I = Italic, O = Bold Italic, N = Normal/Medium
P	Point size where 1 - 9 = point sizes 1-9 and 0 = point size 10 A - Z = point sizes 11-36

Font Cross-reference Files for Monotype Fonts

HPINTL.FXR,
HPINTLSM.FXR

These FXRs provide support for Hewlett Packard (PCL) internal fonts using ANSI code page character sets instead using Monotype-based PCL downloadable fonts. The HPINTLSM.FXR file is a subset of the font information contained in the HPINTL.FXR file—*SM* indicates *small*.

REL95.FXR, REL95SM.FXR

Use these FXRs if you intend to print on an AFP printer using Monotype fonts. These FXRs specify new Monotype AFP coded fonts which use a new code page file. The system uses code page 37 for EBCDIC platforms. These AFP fonts are based on this standard. The REL95SM.FXR file is a subset of the font information contained in the REL95.FXR file—*SM* indicates *small*.

REL102.FXR,
REL102SM.FXR

These FXRs are similar to the REL95 FXRs but also include these fonts: Univers Condensed, MICR, OCRA, and OCRB.

REL103.FXR,
REL103SM.FXR

These FXRs are similar to the REL102 FXRs but also include these fonts: Albany, Arial Black, Arial Narrow, and DocuDings. Be aware that the REL103SM.FXR file does not include DocuDings or all of the point sizes of the Albany group (including bold and italic), the Arial Narrow group (including bold and italic), and the Arial Black group (including italic).

You can identify these fonts via their names. For example *18o10* indicates a 10-point Albany font. The initial *1* indicates Monotype, the *8* indicates Albany, the *o* indicates normal type, and *10* is the point size.

Arial Black fonts are indicated with a nine (9) and Arial Narrow fonts are indicated with a zero (0). DocuDings are indicated with a 34. You can find detailed information on font naming conventions in the Working with Fonts chapter of the [Docucreate User Guide](#).

Below are the PostScript and TrueType fonts included in REL103SM.FXR:

PostScript Font	PostScript Font Name
ALBB____.PFB	Albany-Bold
ALBBI____.PFB	Albany-BoldItalic
ALBIT____.PFB	Albany-Italic
ALBR____.PFB	Albany-Regular
AN____.PFB	ArialNarrowMT
ANB____.PFB	ArialNarrowMT-Bold
ANBI____.PFB	ArialNarrowMT-BoldItalic
ANI____.PFB	ArialNarrowMT-Italic
ARBLI____.PFB	ArialMT-BlackItalic
ARIBL____.PFB	ArialMT-Black
DOCUD____.PFB	DocuDings

TrueType Font	TrueType Font Name
ALB.TTF	Albany AMT
ALBB.TTF	Albany AMT Bold
ALBBI.TTF	Albany AMT Bold Italic
ALBI.TTF	Albany AMT Italic
ARBL.TTF	Arial Black
ARBLIT.TTF	Arial Black Italic
ARIALN.TTF	Arial Narrow
ARIALNB.TTF	Arial Narrow Bold
ARIALNBI.TTF	Arial Narrow Bold Italic
ARIALNI.TTF	Arial Narrow Italic
DOCUDING.TTF	DocuDings

REL112.FXR
REL112SM.FXR

These files differ from the REL103.FXR and REL103SM.FXR files in that...

- The PDF417 fonts were added into the base FXR file.
- Character widths were corrected for font records previously created by importing TrueType fonts.
- Font heights were corrected for the Times fonts so Windows will select the correct screen font.

USING CUSTOM FONTS

To the system, custom fonts are simply fonts which are not based on the ANSI code page. This means that the font contains characters which have different code points or which do not exist in the ANSI code page. If you cannot use the system's Monotype fonts (or at least ANSI code page based fonts), you will need to consider these possible issues:

- Viewing Forms

Viewing forms may be the first problem since the characters in the original printer font do not match the characters used in displaying text on the screen. This problem will be seen during forms composition. This will also be a problem if the you have licensed the Entry or Archive Retrieval modules. Keyboard entry becomes a training issue as well. Under Windows, you must use 4-digit ALT key sequences to prevent code point translation.

If possible, you should convert any custom fonts to TrueType fonts for Windows and install the fonts into your operating system. If the font cross-reference file is properly modified to specify these screen fonts, the system will display your forms correctly. However, these characters may not display properly in Docucreate and Documaker Workstation.

NOTE: The Xerox Font Center will convert a Xerox Metacode font into a PostScript or TrueType font for a fee. They may convert AFP fonts as well. You can reach them at 1-800-445-3668.

- PDF Incompatibility

In addition to the Entry and Archive module problem, PDF or Acrobat files created for Internet archive retrieval use the ANSI code page for displaying forms. Therefore, archived forms based on custom fonts may not display correctly when retrieved through Docupresentment.

- **Printing Forms**

Another problem concerns using custom fonts on multiple (ASCII and EBCDIC) platforms. The system performs ASCII/EBCDIC translation based on the assumption that the ASCII code page is the ANSI code page and that the EBCDIC code page is code page 37. The system also assumes that PCL, PostScript, and Metacode printers use ASCII (hence ANSI) fonts. The system assumes AFP printers use EBCDIC fonts. The following table shows when the system will translate text (from FAP files) and variable data (from extract files) when printed under different platforms and printers.

Platform / Printer	ASCII (Windows 32-bit) ASCII FAP files and Extract data	EBCDIC (z/OS, AS400) EBCDIC FAP files and Extract data
AFP	ASCII to EBCDIC translation	No translation
PCL	No translation	EBCDIC to ASCII translation
PostScript	No translation	EBCDIC to ASCII translation
Xerox Metacode	No translation	EBCDIC to ASCII translation

On AFP printers

On a PC, text will be translated when printing to an AFP printer. Therefore, the code points used in text or variable data on forms are very important. After these code points are translated to the EBCDIC (code page 37), they must match the code points associated with the desired characters in the AFP code page which will be used.

On EBCDIC platforms, such as z/OS, AS400, text is assumed to be EBCDIC and will not be translated when you print to an AFP printer. The key to correct printing is to make sure the text (FAP files) and variable data (extract files) use the code points associated with the desired characters in the AFP code page you will use. Since FAP files are created as ASCII files on a PC, they will need to be transferred to the EBCDIC platform. Since you are using custom fonts, it is quite likely the file transfer software will not perform the proper code point translation. In this case, you may need to upload the files without translation and use the CPCNV utility to translate the files. This may require defining a special code page in the CODEPAGE.INI file for the CPCNV utility to use to do the proper translation.

See [Determining Characters Used in a Printer Font on page 182](#) for help in determining how code points will be associated with font characters.

**On Xerox Metacode
printers**

On a PC, text (code points) will not be translated when printing to a Metacode printer.

On EBCDIC platforms (z/OS, AS400), text is assumed to be EBCDIC and will be translated to ASCII (ANSI code page) when printing to a Metacode printer. Therefore, the EBCDIC code points used in text or variable data on forms are very important. Since the FAP files are ASCII files created on a PC, they will need to be transferred to the EBCDIC platform. Since you are using custom fonts, it is quite likely that the file transfer software will not perform the proper code point translation. In this case, you may need to upload the files without translation and use the CPCNV utility to translate the files. This may require defining a special code page in the CODEPAGE.INI file for the CPCNV utility to use to do the proper translation.

See [Determining Characters Used in a Printer Font on page 182](#) for help in determining how code points will be associated with font characters.

On PCL printers

On a PC, text (code points) will not be translated when printing to a PCL printer. On EBCDIC platforms (z/OS, AS400), PCL print is not currently supported.

On PostScript printers

On a PC, text (code points) will not be translated when printing to a PostScript printer. On EBCDIC platforms (z/OS, AS400), PostScript print is not currently supported.

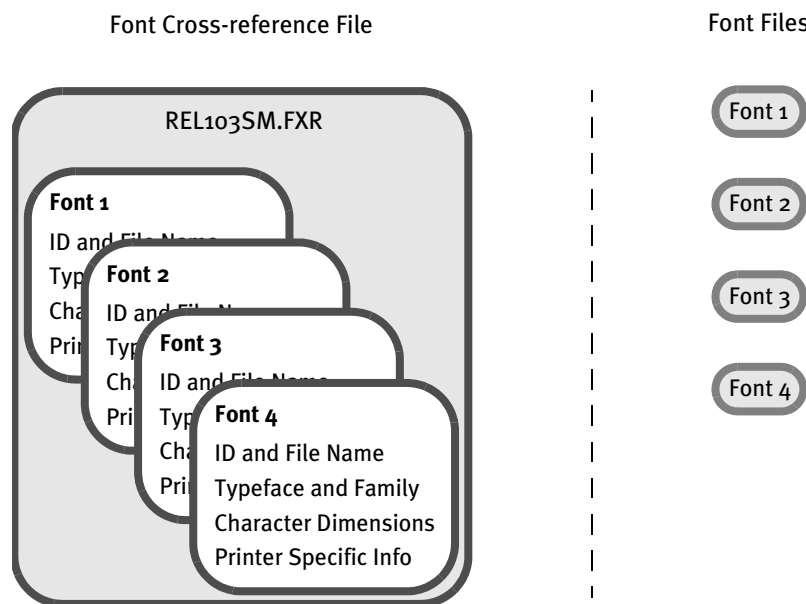
USING FONT CROSS-REFERENCE FILES

The font cross-reference file lets you organize the fonts you use for display and printing. The FXR provides the system with all the necessary font information. It does not contain the actual font files; rather, it contains information about the font attributes. Font attribute information includes formatting styles (bold, italic, and so on), point size (10 point, 14 point, and so on), and font stroke weight (heavy, light, and so on).

Understanding the System

Storing the cross-reference information separately from the physical fonts affords greater flexibility in printer and font usage. You can convert virtually any font for your individual printer environment, provided you obtain appropriate license agreements for the fonts.

Let's examine the organization of the font cross-reference file and the font files. The illustration below depicts a font cross-reference file named REL103SM.FXR. This file contains a single font set. It includes all the crucial information for each font in the font set. The actual font files are physically separate from the font cross-reference file.



As shown above, the font files are distinct from the font cross-reference file. When you work with the font cross-reference file you affect the stored font information. You do not affect the separate and independent font files. The number of available fonts is limited only by your needs and storage space. If you keep this organizational structure in mind you can easily work with the font cross-reference file.

The font cross-reference file provides the names of your independent font files, but it is more than a simple listing of fonts. The file contains crucial font attribute information along with information specific to your printer types. The printer information is crucial because sections are compiled based on your printer environment.

The font cross-reference file ends in the extension *FXR* (for font cross-reference). The system includes these font cross-reference files:

FAP\MSTRRES\FMRES\DEFLIB\HPINTL.FXR

FAP\MSTRRES\FMRES\DEFLIB\HPINTLSM.FXR

FAP\MSTRRES\FMRES\DEFLIB\REL103.FXR

FAP\MSTRRES\FMRES\DEFLIB\REL103SM.FXR

REL103SM.FXR - References Times (Roman), Courier, Univers and Univers Condensed fonts for PostScript, AFP, Metacode, and PCL printers. This FXR file is pre-installed in your system.

Additional PostScript fonts are also included in the REL103.FXR file. This FXR file references standard and supplemental PostScript fonts and all font attributes. You can use the supplemental installation disks to add fonts to your font set, and use the REL103.FXR file as your font reference file, as your system's disk space allows.

Keep in mind these points concerning the FXR file:

- Contains one font set

The font set is the specific group of fonts you choose to include in your font cross-reference file. Each font cross-reference file contains a single font set. You assign each font cross-reference file and font set a unique name. For example, you might organize a font set for creating and printing accounting forms in a font cross-reference file called ACCOUNT.FXR.
- Contains information on multiple fonts

A font set contains numerous fonts. For example, a font set might contain Times New Roman fonts and Gothic fonts of multiple point sizes with bold and italic attributes. A second font set might contain Courier fonts and Helvetica fonts, also of multiple point sizes with bold, italic and regular attributes.
- Independent of your font files

The font cross-reference file works with the printer and window font files. Remember that the font files are separate files from the font cross-reference file.

HOW FXR SETTINGS AFFECT DISPLAY AND PRINT QUALITY

Certain attributes in the FXR file affect how the system displays text. For example, when the system displays text, it uses scalable font technology which exists in Adobe Type 1 Postscript fonts and TrueType fonts. All versions of Windows support TrueType fonts. Windows 2000 also supports PostScript fonts.

These fonts are selected via the family name specified in our FXR, and scaled according to point size, height and width parameters in the FXR. The fonts are spaced according to the character widths specified in the FXR.

Once the font is selected, then it can be zoomed in and out, or additionally scaled as required. Bitmap fonts do not have this scaling ability, which is why scalable fonts are used for display purposes, rather than bitmap fonts.

This means that when the system displays text on the screen, it attempts to mirror how it will look on paper. To achieve the same look on the screen as on paper, the parameters in the FXR are critical. The more accurate the FXR is, the more likely the display will mirror the printed document. The printed document is the standard for the screen display.

Since the system includes Monotype TrueType and PostScript fonts which match its printer fonts, if you install these Monotype fonts on a Windows system, what you see on your screen will more closely match what you print out. The keys are to closely match the printer's fonts and to have the best possible information in the FXR file.

Creating a font cross-reference file is usually done by importing a printer font file using Font Manager. Since the font cross-reference file is a representation of information contained in the printer font file, modification of its fields usually does not affect the printed output. However, modifying these FXR fields can improve the system's ability to display forms. See [Choosing Screen Fonts on page 238](#) for help in selecting a screen font similar to the printer font.

MAINTAINING FXR FILES

Use Font Manager to maintain FXR files. You can start this tool in Documaker Studio using the Manage, System, Fonts option. You can start this tool from Docucreate (choose Resources, Fonts) or from Image Editor (choose Tools, Font Manager). Font Manager makes it easy to insert, edit, copy or delete font information in the FXR file. Instructions for using Font Manager can be found in [Using Font Manager on page 211](#).

Choosing a Font Cross-reference File

During library setup, you must choose either REL103.FXR or REL103SM.FXR as the font cross-reference file for an AFP printer. You should also specify the PCL download font file named REL103SM.FNT in the FntFile option of the Resource Library window.

If you have older versions of the AFP coded font and code page files installed in PSF or PSF/2, you can use these versions to print to the same AFP printer. If you do not keep the older AFP coded font and code page files installed, you must recompile AFP page overlays for the current version using REL103.FXR or REL103SM.FXR.

Understanding the System

This example shows you how the HPINTL.FXR and HPINTLSM.FXR files use PCL escape sequences in the Setup Data field (on the Font Properties window) to use internal fonts on a PCL printer. If you use Font Manager to edit a font in the HPINTL(sm).FXR file, you will see the PCL escape sequence in this field.

For example, if you look at the Setup Data field for font ID 11036 (Times Roman Normal 36 point), you will see:

```
~(19U~(s1p36vosob5T
```

Where Represents

~	an escape character which must always start a PCL escape sequence
(19U	the primary symbol set or code page (Windows 3.1 Latin 1 in this case)
~	the start of a second PCL escape sequence
(s1p	the spacing of the font (proportional in this case)
36v	the height of the font in point size (36 point in this case)
os	the style of the font (normal in this case)

Where Represents

ob	the stroke weight of the font (medium in this case)
5T	the typeface family of the font (Times Roman in this case)

There are other values you can use for each of these sequences. For example, the character or symbol set values used in HPINTL.FXR are:

19U for Windows 3.1 Latin 1

This symbol set matches the Windows ANSI code page and IBM code page 1004. You can find a list of character set values in the HP manual entitled, *PCL 5 Comparison Guide*.

Spacing values are (*s1p* for proportional fonts and (*sop* for fixed pitch fonts.

- Point size values are placed before the *v*
- Font styles are *os* for normal, *1s* for italic
- Font stroke weights are *ob* for medium, *3b* for bold

The typeface family values used in HPINTL.FXR are:

- *5T* for Times Roman
 - *3T* for Courier
 - *6T* for Letter Gothic
 - *52T* for Univers
-

INTERNATIONAL LANGUAGE SUPPORT

Our goal for international language support is to support the languages you are most likely to need. At the present, we consider these languages to be those used in the Western Hemisphere and Western Europe.

If you need support for Far Eastern languages like Chinese, Japanese, or Korean or if you need support for Eastern European languages, you must use version 10.2 or higher. Contact Support to receive a copy of the document, *Using Unicode*, for more information.

USING THE ANSI CODE PAGE FOR PC PLATFORMS

The Windows operating environment supports languages in these countries via a code page known as the *ANSI code page*. Windows supports different keyboard mappings for these countries by translating the key codes into ANSI code points. Therefore, even though a keystroke for an international character generates different *key codes* on English, Spanish, and French keyboards, a Windows application receives the same *ANSI code point*.

Understanding the System

We adopted these standards:

- The ANSI code page is the standard code page for all data files. The text contained in FAP files is stored using the ANSI code page.
- The ANSI code page is the standard for the Monotype fonts included with the system.

See [Using International Characters on page 203](#) for more information.

By adopting these standards, you receive these benefits:

- Support for 13 languages used in approximately 40 countries
- Improved platform resource compatibility (Windows, UNIX, and z/OS).
- You only need one set of Monotype fonts—no need to create separate fonts for each language
- Improved support of other Windows products, such as dictionaries, databases, and so on.

The ANSI code page is used by the World Wide Web and UNIX computers, as well as the Windows operating environment.

There are a few drawbacks to this approach. For instance, although all international alphabetic characters in the Latin character set are supported in the ANSI code page, certain symbols available in other code pages are not supported. These symbols include mathematical, scientific, and line drawing symbols. Greek, Cyrillic, and Asian characters are not supported either. And, in some cases, data files may have to be converted to ANSI code page characters.

USING CODE PAGE 37 FOR EBCDIC PLATFORMS

To support international languages on EBCDIC platforms, such as z/OS and AS400, we use EBCDIC code page 37 as the standard EBCDIC code page. Code page 37 is the native code page for many z/OS and AS400 systems. By using code page 37, you receive these benefits:

- Code page 37 supports languages used in Europe and North and South America, such as French, Spanish, Italian, German, Portuguese, and Danish.
- This reduces or eliminates the need to convert extract files containing international characters on z/OS and AS400 platforms.
- This helps reduce or eliminate the need to convert resources before uploading to EBCDIC platforms from Windows.
- Using code page 37 for EBCDIC platforms creates compatibility problems with resources created in earlier versions. This only affects resources created in an earlier version which contain international or desktop publishing characters.
- All characters defined in code page 37 are also contained in code page 1004, the standard ASCII code page. There are, however, characters in code page 1004 which are not in code page 37—mainly desktop publishing characters from code point 128 to 159. To support these characters, we use undefined code points in code page 37 (code points below 64). For maximum portability, *avoid* using characters not defined in code page 37.

AFP print output and resource files normally use EBCDIC characters. The other supported printers, such as Metacode, PCL, and PostScript, normally use ASCII characters.

NOTE: The current AFP code page file is called T1DOC037, the AFP code page for prior versions was called T100ASC4. The current AFP coded font files are called XoDA????FNT, the AFP coded font files for prior releases were called XoFA????FNT. The AFP character set files are unchanged and can be used by all versions.

USING INTERNATIONAL CHARACTERS

One method for entering international characters is to install a country/language specific version of Windows. These language-specific versions of Windows map characters from the keyboard differently so that it is easier to enter characters common to that language. In the simplest case, a single keystroke will generate an international character.

For example, if you have a Canadian French version of Windows, pressing the slash character (/) on a U.S. keyboard produces an e-acute letter (é). Many international characters require a two-character keystroke combination. Again using the Canadian French keyboard setup, you must press the left square bracket ([) followed by the letter e to generate an e-circumflex letter (ê).

Having to install a special version of Windows would be difficult for those in the U.S. who are trying to compose forms with French characters. Fortunately, there is a simpler solution.

Using the numeric keypad on the right side of your keyboard, you can hold down the ALT key and enter a three-digit number to enter an international character. For example, if your primary (OEM) code page is 437 or 850, you can enter the letter ñ (lowercase) by pressing the ALT key while you type 164 on the numeric keypad. When you release the ALT key, the code point 164 will be generated by the keyboard, which Windows will display as the letter ñ.

NOTE: If you look at the code page 1004 table you will see that on the ANSI code page code point 164 is not the letter ñ. So why is the letter ñ being displayed? Windows recognizes that a code point of 164 has been generated by the keyboard and it is associated with the OEM code page (437 or 850). For this code page, code point 164 maps to the letter ñ. In Windows, the code point from the keyboard is translated from 164 to 241. A Windows program will actually receive a keystroke code point of 241 instead and that code point will display as the desired letter ñ.

You can also use the numeric keypad to enter ANSI code points directly. Using the numeric keypad on the right side of your keyboard, you can press the ALT key and type a four-digit number to key in an international character. For example, you can enter in the letter ñ by pressing the ALT key and typing 0241 on the numeric keypad. Entering a four-digit number beginning with a zero tells Windows you are entering a code point for the ANSI code page. Therefore, Windows does not need to translate the code point and passes the keystroke code point directly to the Windows application.

By standardizing on the ANSI code page, a document containing several languages can be read and written by a number of people from different countries. The keystroke code point translation lets Windows support many OEM code pages and keyboard settings.

NOTE: You can use any Windows text editor, such as Notepad, to edit resource files since Windows also uses the ANSI code page.

CONVERTING TEXT FILES FROM ONE CODE PAGE TO ANOTHER

There are two situations where you may need to convert text files from one code page to another.

- If the customer's data (extract) file is not in the ANSI code page and the file contains international characters, you will need to convert the customer data file to use the ANSI code page.
- If you need to upload system resource files, such as FAP, INI, and menu resource (MEN.RES) files, which contain international characters to an EBCDIC platform, such as z/OS or AS400, and the file transfer software cannot convert ANSI code page file to EBCDIC code page 37.

To convert a file from one code page to another, you can use the CPCNV code page conversion utility. For more information, see the [Docutoolbox Reference](#).

SETTING UP POSTSCRIPT FONTS

The system includes a standard font set with PostScript fonts. These fonts reside in the FAP\MSTRRES\FMRES\DEFLIB\ directory with the sample forms included with Documaker Studio and Docucreate. We devised naming conventions for the bitmap printer fonts that are created from the PostScript fonts supported by the system. PostScript fonts are easily converted to Xerox, AFP, and PCL formats.

NOTE: When you create bitmap printer fonts from PostScript fonts, follow the naming convention outlined in the table below. This will make it easier to track and identify those fonts.

A standard font has a six-character name. Each character indicates a specific piece of data that describes the font. For example, you may take a PostScript font such as Times (Roman), Bold (**TIB____.PFB**), convert the font to Metacode format, and change the name to the standard FSI bitmap font name (**FXTIOM**). The font name characters designate the following:

Character	Definition
1	Converted PostScript fonts always begin with the letter F, indicating a system supported font.
2	Indicates the printer platform associated with the converted font: X = Xerox, A = AFP, P = PCL
3 and 4	Indicate the font family, such as Times Roman, Courier, and so on. AB = Albany AL = Arial Black AN = Arial Narrow CO = Courier DD = DocuDings UC = Univers Condensed LG = Letter Gothic MI = MICR TI = Times (Roman) OA = OCRA UN = Univers(al) OB = OCRB ZP = ZIP code
5	Indicates the style of the font: N = Normal (no attributes), B = Bold, I = Italic, O = Bold, Italic
6	Indicates the point size of the font. Use numbers 1 through 9 for point sizes 1 through 9. o (zero) = 10 point A = 11 point B = 12 point C = 13 point--through-- Z = 36 point

This table lists PostScript fonts and their file names. The list shows the font names before you create and name the fonts using the conventions in the previous table. Point sizes are omitted in the names below. Use the table on the previous page to determine the remaining font file name value for each corresponding font size.

Font	File Name
Albany	ALBR____.PFB
Arial Black	ARIBL____.PFB
Arial Narrow	AN____.PFB
Courier	CO____.PFB
Courier Bold	COB____.PFB
Courier Bold Italic	COBI____.PFB
Courier Italic	COI____.PFB
DocuDings	DOCUD____.PFB
Letter Gothic	LG____.PFB
Letter Gothic Bold	LGB____.PFB
Letter Gothic Bold Italic	LGBSL____.PFB
Letter Gothic Italic	LGSL____.PFB
MICR MT	MICR____.PFB
OCRA MT	OCRA____.PFB
OCRB MT	OCRBMT____.PFB
Times Roman	TIR____.PFB
Times Roman Bold	TIB____.PFB
Times Roman Bold Italic	TIBI____.PFB
Times Roman Italic	TII____.PFB
Univers	UNM____.PFB
Univers Bold	UNB____.PFB
Univers Bold Italic	UNBI____.PFB
Univers Italic	UNMI____.PFB
Univers-Condensed Bold	UNCB____.PFB
Univers-Condensed Medium	UNCM____.PFB

Font	File Name
Univers-Condensed Medium Italic	UNCM1____.PFB
ZIPcode Barcode-Regular	ZIPCODE_.PFB

Remember that PostScript fonts are scaleable. You complete font file name by adding the point size values when you convert the font. Here is an example:

CSBD____.PFB = CS Bookman Bold (any point size)

NOTE: AFM files are Adobe Font Metrics files which describe a PostScript font. These files are used when you install PostScript fonts using Adobe Type Manager.

PostScript fonts reference code pages to define window and print characters. In turn, the code page maps to specific characters in the character set. The PostScript fonts included with Documaker Studio and Docucreate reference code page 1004, W1 and are shown here:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
16																
32		!	"	#	\$	%	&	¢	()	*	+	,	-	.	/
48	o	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
128	€		,		„	º	†	‡	^	‰	Š	‹	Œ		Ž	
144		‘	’	“	”	•	—	—	~	™	š	›	œ		ž	ÿ
160		ı	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
176	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
192	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
208	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
224	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Bitmap fonts are a specific set of symbols or characters. The maximum number of characters a set of bitmap fonts can reference is 256. Scaleable fonts, such as PostScript fonts, may have more than 256 characters, but only 256 can be used at one time. The system's font structure is designed to use the standard code page 1004, W1. Code pages are predefined in your system, and reside in the CODEPAGE.INI file in your DEFLIB directory. The path is FAP\MSTRRES\FMRES\DEFLIB.

The characters in the code page include foreign language characters and mathematical function characters. When you convert PostScript fonts using Font Manager, you always select this code page (1004). You may, however, notice that the PostScript fonts themselves support multiple code pages.

NOTE: If you want to use the internal printer fonts and you will print international characters, your printer must have the character or symbol set named Windows Latin 1 (also known as ANSI code page) on your printer. Be aware that not all PCL printers support this character set.

FONTS FOR PDF FILES

When you are creating PDF files, keep in mind that the following PostScript fonts are included with Adobe Acrobat Reader and do not have to be embedded.

Fixed Pitch Fonts	Proportional Fonts
Courier	Helvetica
Courier-Bold	Helvetica-Bold
Courier-Oblique	Helvetica-Oblique
Courier-BoldOblique	Helvetica-BoldOblique
	Times-Roman
	Times-Bold
	Times-Italic
	Times-BoldItalic
	Symbol
	ZapfDingbats

Importing PostScript Symbol Fonts

You can select a code page when importing PostScript symbol fonts, such as Euro Sans and ITD Zapf Dingbats, which contain characters that do not adhere to a standard Windows code page.

In Font Manager (For both Documaker Studio and the legacy tools), select 9999,WD as the code page when importing these types of PostScript fonts.

NOTE: For normal fonts, you should continue to select *1004,W1* as the code page.

If you import a PostScript font using code page 1004,W1 and the system produces a font record with only a few non-zero character widths or produces an internal error, try using code page 9999,WD to import the font.

For instance, importing Euro Sans and ITC Zapf Dingbats using code page 1004,W1 produces a font record where only the space and hard space characters (code points 32 and 160) contain non-zero character widths. Importing the same fonts using code page 9999,WD produces a font record with non-zero character widths for virtually every code point from 32 to 255.

When you use the PS2PCL utility to convert PostScript symbol fonts to PCL, specify the symbol set by setting the /S parameter to *WD*. This tells the utility that these PostScript fonts that contain characters that do not adhere to a standard Windows code page.

NOTE: When converting normal text fonts with the PS2PCL utility, continue to set the /S parameter to *W1*.

FONT NAMING CONVENTIONS

When adding fonts to a font set, or when installing new fonts, you must give each font a unique ID. Use this 5-digit naming convention:

The...	Indicates...
First digit	the font provider: 1= Monotype 2= Adobe
Second digit The standard FXR file (REL103SM) defines only Times (Roman), Courier, and Univers. If you add other fonts to your FXR, use these font code naming conventions. (DocuDings is included in 3)	the font type or font family: 1 = Times (Roman) 2 = Courier 3 = OCRA, OCRB, MICR, and ZIPcode* 5 = Letter Gothic 6 = Univers 7 = Univers Condensed 8=Albany 9=Arial Black 0=Arial Narrow
Third digit	the font attributes 0= normal 1= bold 2= <i>italic</i> 3= bold, italic
Fourth and fifth digits	the point size of the font, such as 09 point, 12 point, and so on.

* The OCRA, OCRB, MICR, and ZIPcode fonts do not have bold or italic styles so the second and third digits identify these fonts: 30 = OCRA, 31 = OCRB, 32 = MICR, 33 = ZIPcode, and 34 = DocuDings.

For example, 11010 indicates Times (Roman) Regular 10 point, 11214 indicates Times (Roman) Italic 14 point, and 16110 indicates Universal Bold 10 point.

NOTE: You may only use a font ID from 00001 to 32767 and the font ID must be numeric not alphanumeric.

USING FONT MANAGER

Font Manager provides an easy method for working with your font sets.

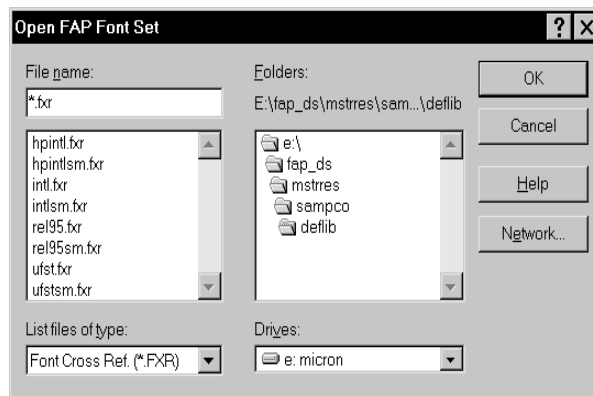
STARTING FONT MANAGER

From Image Editor

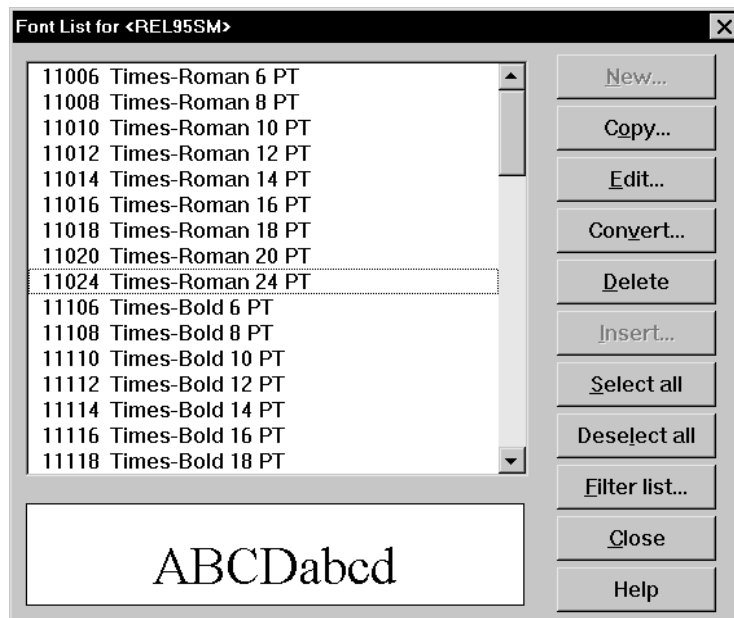
Select Tools, Font Manager. The Font List appears. This window lists the fonts in the font cross-reference file designated for the current library during library setup.

From Docucreate

- 1 Select the Resources menu, then select Fonts; or, click the Resources icon, then click the Fonts icon on your desktop. The Open FAP Font Set window appears, listing all the FXR files in the current library.



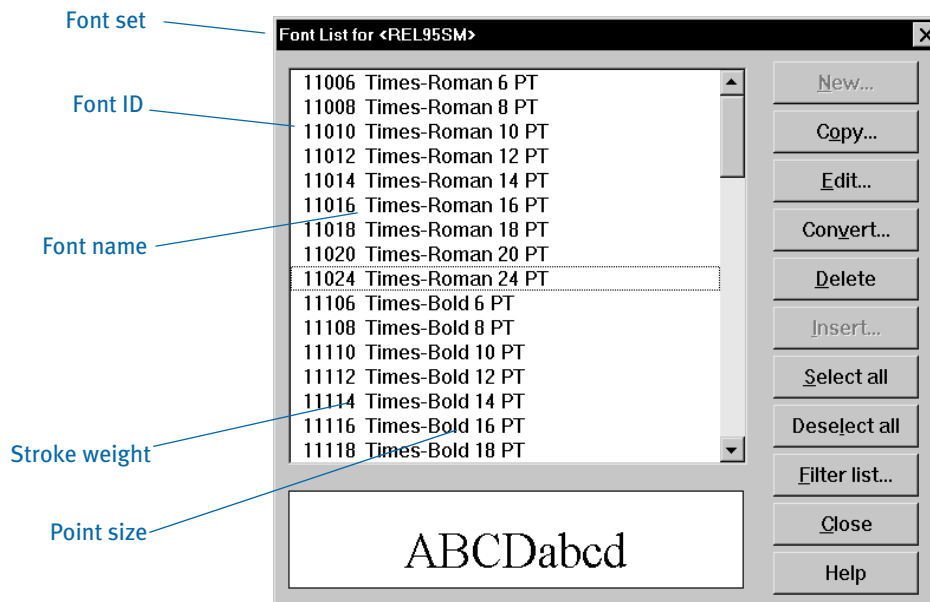
- 2 Highlight the FXR file you want to work with and click Ok. The Font List for the selected FXR appears.



WORKING WITH THE FONT LIST

The Font List gives you important information about your font set. The name of the current font set is displayed in the window's title bar. The Font List provides the names, ID numbers, stroke weights, and point sizes of all the fonts currently in the font set. You can view your fonts in the bottom of the Font List window.

To open the Font List from Image Editor, choose Tools, Font Manager. The Font List window appears:



This window shows you fonts contained in the FXR file, as defined in your resource library setup.

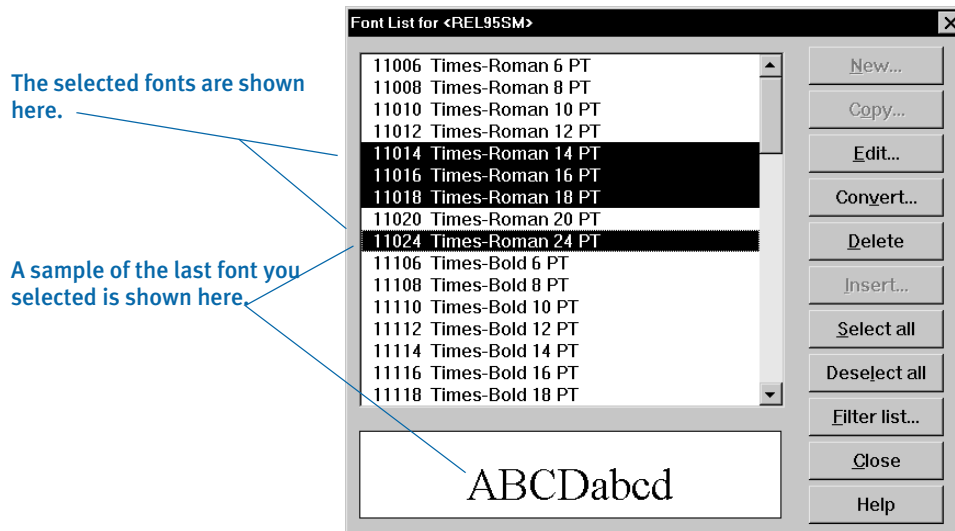
In addition to viewing the fonts, the Font List window gives you additional options for organizing the font set. From the Font List window you insert, copy, delete, edit, and convert fonts. When you select New, Copy or Edit, the Font List opens the Font Properties window.

NOTE: The Description tab of the Font Properties window is the source of the individual font information provided in the Font List.

Selecting Fonts

The Font List lets you select individual, multiple, or all of the fonts in your font list for editing purposes.

- 1 To select a font, click on the desired font to highlight it. The font you select displays in reverse video. Also, a sample of the font displays in the bottom window.



- 2 Continue highlighting fonts in this manner. You can select as many fonts as you want. When you select multiple fonts, your last font selection displays at the bottom of the window.
- 3 To select all fonts in the font list, click Select All.

Deselecting Fonts

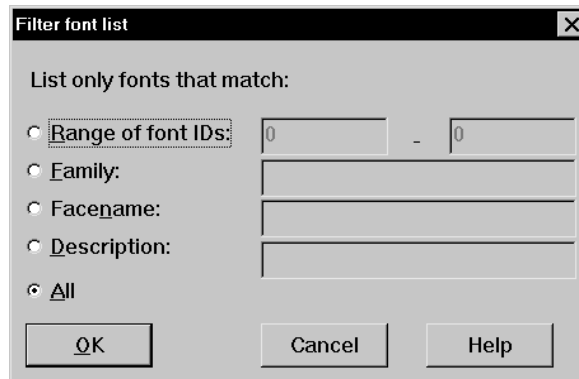
If, after selecting a font, you change your mind, just click it again to deselect it.

You can also use the Deselect All button to deselect all the selected fonts on the Font List. This is particularly useful if you need to activate the New or Insert buttons. The New and Insert buttons are not active if a font is selected. The Deselect All button prevents you from having to scroll through the list of fonts to find a highlighted or selected font near the bottom of the list.

Filtering the List of Fonts

The Filter List option lets you select fonts from the list based on specific criteria. The more focused your font selection scope, the more efficiently you can work with the fonts.

- 1 From the Font List window, click Filter List. The system displays the Filter Font List window.



- 2 Select one option to define your font selection filter criteria. See the following table for a detailed description of the Filter Font List options and related entry descriptions.

To filter by	Then
--------------	------

Range of font IDs	In the entry fields, type the first and last font ID in the font range you want to display. The font IDs you enter in the fields are included in the range. For example, if you enter IDs 16010-16016, those two fonts, plus all fonts in between the range IDs appear in the Font list.
Family	Type the font family name you want to display, such as Universal or Times. The system displays all fonts in the font family.
Face name	Type the face name or font family name you want to. Font Manager does not distinguish between font family and facename. Type the same data in both the Family field, and the Facename field.
Description	Type the font description as shown in the Font Properties window. Here are some examples: Universal Medium or Courier Bold. The selection list displays all point sizes for that font.
All	Click All to see all fonts in the FXR file.

- 3 Click Ok in the Filter Font List window. The font list displays the selected fonts based on your filter criteria.

Adding Fonts to a Font Set

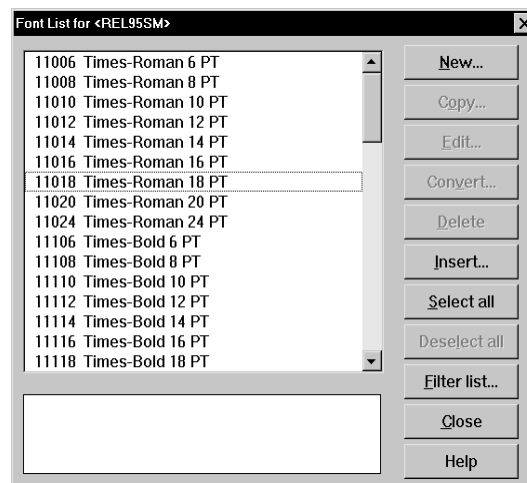
Use the New button to add new fonts to the font set. The New button does not create a font; rather, it just lets you add the font to the font set. If you add a font supplied by another vendor, the vendor provides information about the font name and characteristics.

NOTE: Use New to manually add fonts to your font set. Use Insert to cross-reference your collateral font information, and to add a font if you also have the printer font file).

When you add a font, you must enter font description and dimension information, whereas the Insert option automatically adds this information based on the type of font you insert. You seldom use the New option unless you decide to add new fonts to your font set.

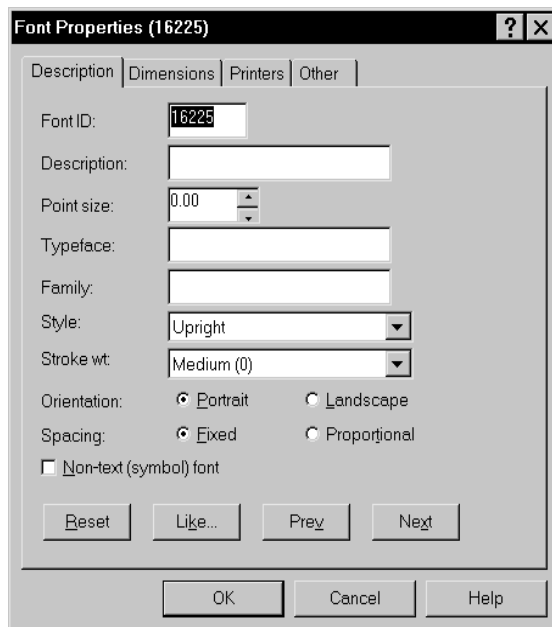
To add a font to the current font set, the font must exist in your master resource library font directory. From the Font Properties window, you specify the description, dimensions and printer specific information for every font.

- 1 Start Font Manager by selecting the Font Manager icon from the system folder, or by selecting Font Manager from the Image Editor Tools menu.
- 2 Select the appropriate font set. The Font List appears.



- 3 Click New. The Font Properties window appears.

NOTE: You cannot select the New button unless there are no fonts selected. Click Deselect All to activate the New button.



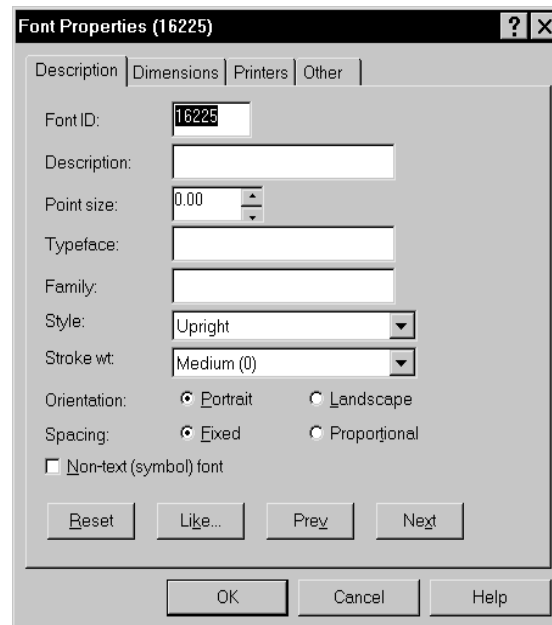
These options are available from each tab of the Font Properties window:

Option	Description
--------	-------------

Reset	Reset reverts your most recent (unsaved) changes to their prior status. Your changes are reset for that particular page without having to exit the property window.
Like	Like lets you select a Windows display font that most closely resembles an available printer font. See Choosing Screen Fonts on page 238 for information on this option.
Prev	Prev displays the previous Prev font in the font list, and related information. If you select Prev while the first font is displayed, you scroll to the last font in the list.
Next	Next displays the next font in the font list, and related information. If you select Next while the last font is displayed, you scroll to the first font in the list.

Description tab

The Description tab contains general description information about the font. You specify the typeface, family and other necessary information required for each font.



Here is a discussion of the fields on this tab:

Field	Description
Font ID	When a font is imported into the FXR, it is assigned a unique font ID. The system uses font IDs to track font usage. In addition, some printers require that you refer to a font by its ID number instead of its name. This value is generated when the printer font is imported into the FXR but may be changed if needed. If you change this field, you may need to change the Setup Data field for PCL Print to use the same font ID.
Description	Used in the font selection window in Font Manager.
Point Size	A point is a typographical vertical measurement, 72 points are equal to approximately 1 inch. Point size is used similarly in PostScript printing. It does not affect PCL, AFP, or Metacode printing.
Typeface	A specific member of a typeface family, such as Times-Roman or Times-Bold. The typeface name helps determine the screen font to use for display purposes.
Family	A group of typefaces that share basic design characteristics and encompass many size and style variations such as Courier or Times. The family name helps determine the screen font used for displaying text when running under Windows.
Style	Upright or Italic The style helps determine the screen font used for displaying text when running under Windows.

Field	Description
Stroke Wt.	The lightness or darkness of the printed typeface, -7 = lightest, 0 = medium, 7 = darkest. The stroke weight helps determine the screen font used for displaying text when running under Windows.
Orientation	Portrait or Landscape. This field is not currently used.
Spacing	Fixed or Proportional The spacing helps determine the screen font used for displaying text when running under Windows.

Remember, the Description tab provides information about the font file, and should match the characteristics of the font. If you enter information which does not match the font file, the font window display changes but the actual font characteristics in the font file and the font print characteristics *do not* change. Only when you change the PostScript font description do your changes affect both the window and print fonts since you use a single font file for both window display and print.

To enter font description information follow these steps:

- 1 Enter the font ID in the Font ID field. The ID must be a unique one to five digit ID (between 00001 and 32767). You must assign a unique ID to every font in the font set. Notice that the title of the window displays the current font ID. The system creates default IDs when you insert a font, based on the next available ID. See [Font Naming Conventions on page 210](#) for help in assigning a font ID.
- 2 Enter a description in the Description field. The description is the information that the form designer sees when selecting a font from a font list. The description should inform the form designer about the font type selection.
- 3 Select the desired point size in the Point Size field. The point size is the height of the font. One point is equal to approximately 1/72 inch.
- 4 Type a typeface name in the Typeface field. Font Manager selects fonts using the typeface name.

NOTE: Valid entries include the standard window display fonts of Courier, Helvetica, and Times (Roman). Type these entries as **Courier**, **Helv**, or **Tms Rmn**. To use a different font type, install the font from your operating system disks, or from your font provider disk. The font name is a valid entry after you install it.

- 5 Type a family name in the Family field. A family name gives the broadest category for the font, and is used to group fonts that are variations of a single design. Family names include Courier, Times (Roman), and Helvetica.

NOTE: The family name, style, and stroke weight fields control which screen font is displayed under Windows.

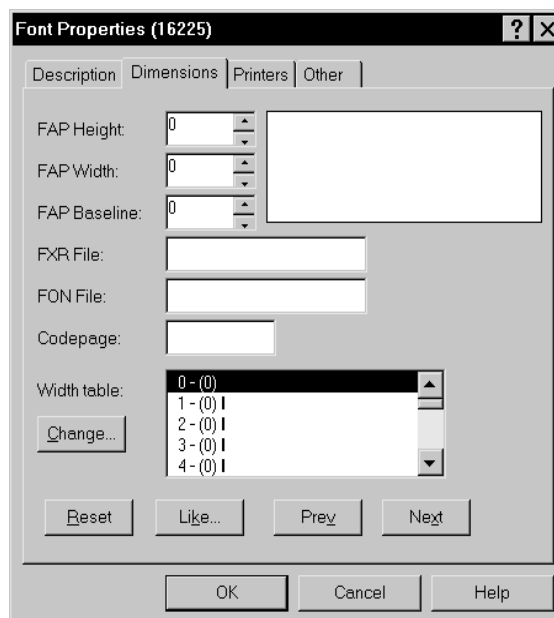
- 6 Select the style in the Style field. Style is either upright or italic. Click the arrow to the right of the field to see the available selections.
- 7 Select the stroke weight in the Stroke Wt field. Stroke weights range from Light to Lightest (-1 through -7), Medium (0) and Bold to Boldest (+1 through +7). Click the arrow to the right of the field to see the available font weight selections and compare the thickness of the font characters.
- 8 Select either Portrait or Landscape Orientation.
- 9 Select either Fixed or Proportional Spacing. Fixed spacing means all the characters have the same width. Proportional spacing means the character widths vary according to the character design.
- 10 Select the Non-Text (Symbol) font field if the font you define is not alphanumeric. This selection lets you read non-text fonts such as MICR fonts. MICR fonts are symbol fonts such as those appearing on the bottom of your bank checks. The MICR font format allows scanners to read the data and translate symbols into alphanumeric characters. Here is an example:



NOTE: Remember, changes made in these steps change only the window display of bitmap fonts. These changes do not alter the way the font prints.

Dimensions tab

The Dimensions tab lets you enter dimensions for the font. You identify the exact character cell dimensions for the font, including height, width, and baseline. When you use Copy or Insert, these dimensions are filled in for you from the font file.



Here is a discussion of the fields on the Dimensions tab:

Field	Description
FAP Height	largest font character height (in FAP units, 2400 dots/inch) The font height affects the size of text displayed when running under Windows.
FAP Width	largest font character width (in FAP units) The font width affects the width of text displayed when running under Windows.
FAP Baseline	largest font character base line (in FAP units) The baseline is measured from the top of the largest character to the imaginary line that the character appears to rest on. The font baseline affects the positioning of text displayed when running under Windows.
FXR File	In the DOS/PC environment, the font file name has the extension <i>FNT</i> . This field is not currently used.
FON File	In the DOS/PC environment, the font width table file name has the extension <i>FON</i> . This field is not currently used.
Code Page	Under Windows, the system uses the ANSI code page. Normally, this field is set to 1004 or is left blank.
Width Table	The width table is calculated from the printer font file and is used to determine character spacing when displaying text. Fixed pitch fonts use the same width for each character. The width is measured in FAP units (2400 to an inch).

Remember, the Dimensions tab provides information about the font file. Information on this window should match the characteristics in the font file. If you enter dimension information which does not match the actual font file, only the bitmap font window display changes. The actual dimensions of the font in the font file do not change, nor do print dimensions of the font change.

- 1 To enter font dimensions information, click Dimensions. Enter the height in the FAP Height field.

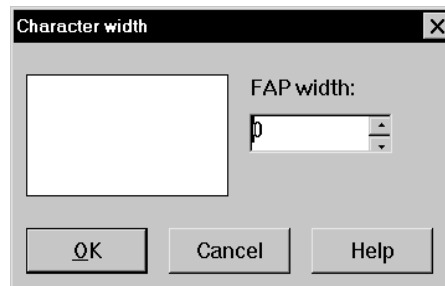
NOTE: Enter character height in FAP units. A FAP unit is equal to 1/2400 of an inch. Therefore, if the character height is one inch, you enter 2400.

- 2 Enter the width in the FAP Width field.
- 3 Enter the baseline in the FAP Baseline field. The baseline is the bottom position of the characters (excluding the tail), in relation to the lines of text above and below it.

Baseline position

Baseline Tail

- 4 Enter the name of the FXR file in the FXR File field. The FXR file is the file used by the font.
- 5 Enter an FON file in the FON File field. The FON file is a window font used by an earlier version of the system called *FormMaker II*. The FON file is the font width table. This table defines the character widths and character sets for each bitmap font. Use this option only for bitmap fonts.
- 6 If you want to change the window display width of a character, highlight the character in the Width Table and click Change. This window appears:



- 7 Select a new width for the character and click Ok. You return to the Font Properties window. Click Ok in the Font Properties window. You return to the Font List window.

NOTE: Remember, changes made in these steps change only the window display of bitmap fonts. These changes do not alter the way the font prints.

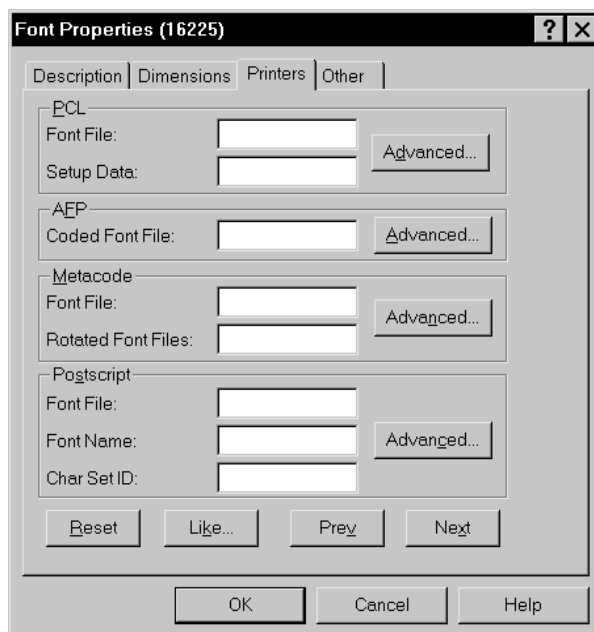
Printers tab

The Printers tab lets you enter printer-specific information for PCL, AFP, Metacode, and PostScript printers. The Other tab lets you enter the same kind of information for printers other than those mentioned above. Here are short descriptions of PCL, AFP, Metacode, and PostScript:

Printer	Description
PCL	Print Control Language (PCL), designed and developed by Hewlett Packard, incorporates commands in compact escape sequence codes that are embedded in the print data stream.
AFP	Advanced Function Printing (AFP), developed by IBM for its Print Services Facility (PSF), is a print server language that generates data streams of objects. The data streams merge with print controls and system commands, to generate IPDS (Intelligent Printer Data Stream). The IPDS is then sent to an IPDS-compatible printer.

Printer	Description
Metacode	Metacode, developed by Xerox, is the native language of its Centralized Printing Systems. Metacode contains code that defines printing functions to the hardware. Metacode lets you position data via page addressing and to specify multiple fonts and data orientations.
PostScript	PostScript, developed by Adobe Systems, is a page description language that translates or describes a document from a computer composition system to a raster output printing system. PostScript describes pages at a high level as a series of abstract graphic objects.

- 1 To enter printer-specific information for PCL, AFP, Metacode, or Postscript printers, click Printers.



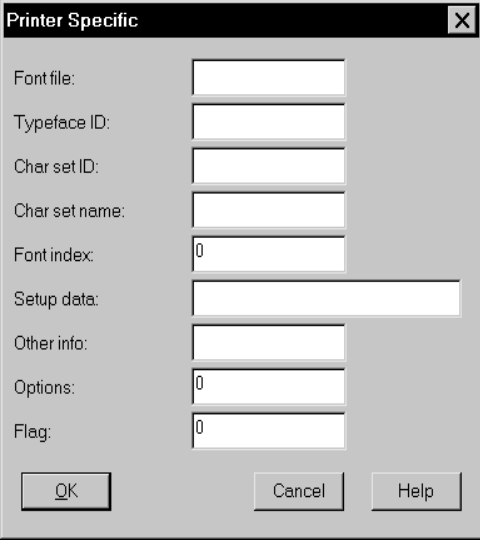
- 2 You only need to make entries into the fields for the printer you use.

Type	Field	Enter the...
PCL		

Type	Field	Enter the...
	Font File	Font file name, including the .PCL extension. For printing to PCL printers, this PCL bitmap font must be located in the FONTLIB master resource directory so that it may be downloaded to the printer if requested. For printing to PostScript printers, this PostScript Type 1 font must be located in the FONTLIB master resource directory so that it may be downloaded to the printer if requested. For printing to AFP printers, this file name must be the name of an AFP coded font file installed on the printer. For printing to Metacode printers, this file name must be the name of a Xerox font installed on the printer.
	Setup Data	Actual PCL printer sequence required to select a font. Normally, the setup data must appear in this format: ~(11018X where 11018 is the Font ID. The font ID must match the font ID you defined on the Description tab. The X must be uppercase. Your setup data may differ if you are using internal printer fonts. Check your printer manual and the online Help for the proper setup data sequence for internal printer fonts.
AFP		
	Coded Font File	Font file name. AFP font file names do not have an extension.
Metacode		
	Font File	Font file name (limited to six characters for Metacode printers). No extension.
	Rotated Font Files	File names or 90, 180 and 220 degree fonts separated by semicolons, such as <i>FNT90;FNT180;FNT270</i>
Postscript		

Type	Field	Enter the...												
	Font Name	Font file name, including the .PFB extension.												
	Font Name	Name of the font, such as Times-Roman. Font Manager fills this field when you insert a PostScript font. The font name also appears in the font .AFM file. All type 1 PostScript fonts require two files for each font family name: .AFM and .PFB. For more specific information about these files, talk with your support representative.												
	Char Set ID	<p>A character set (also known as a symbol set) identifies the set of symbols provided by the font.</p> <p>Some printers require that you refer to a character set by its ID number instead of its name.</p> <p>Used by PostScript printing to build an internal code page. Use W1 for the ANSI code page. This value should match the character set ID specified in the CODEPAGE.INI file. For instance, if you enter 1004 as the code page on the Printers tab, enter W1 here.</p> <table><tr><th>Codepage</th><th>Char Set ID</th></tr><tr><td>1004</td><td>W1</td></tr><tr><td>863</td><td>CF</td></tr><tr><td>850</td><td>PM</td></tr><tr><td>437</td><td>PC</td></tr><tr><td>37</td><td>Z1</td></tr></table>	Codepage	Char Set ID	1004	W1	863	CF	850	PM	437	PC	37	Z1
Codepage	Char Set ID													
1004	W1													
863	CF													
850	PM													
437	PC													
37	Z1													

- Beside each type of printer listed on the Printers tab is an Advanced button you can use to enter additional information. Click the Advanced button to display the Printer Specific window.



The 'Printer Specific' dialog box contains the following fields and buttons:

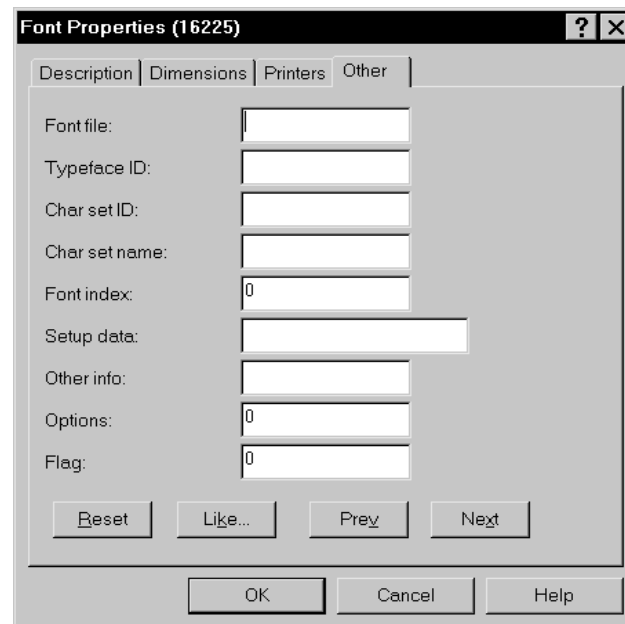
- Font file: [Text Field]
- Typeface ID: [Text Field]
- Char set ID: [Text Field]
- Char set name: [Text Field]
- Font index: [Text Field with value 0]
- Setup data: [Text Field]
- Other info: [Text Field]
- Options: [Text Field with value 0]
- Flag: [Text Field with value 0]
- Buttons: OK, Cancel, Help

- Enter the Font File and Setup Data information. Most of the other fields (Typeface ID, Char Set Name, Font Index, Other, Options, and Flag) are not used in the base system. The Char Set ID field is used for PostScript fonts. Click Ok.

NOTE: The HPINTL.FXR and HPINTLSM.FXR files use PCL escape sequences in the Setup Data field to use internal fonts on a PCL printer. If you use Font Manager to edit a font in the HPINTL(sm).FXR file, you will see the PCL escape sequence in this field.

Other tab

The Other tab can be used for custom implementations of printers other than PCL, AFP, Metacode, or PostScript. You do not need to enter information in these fields.



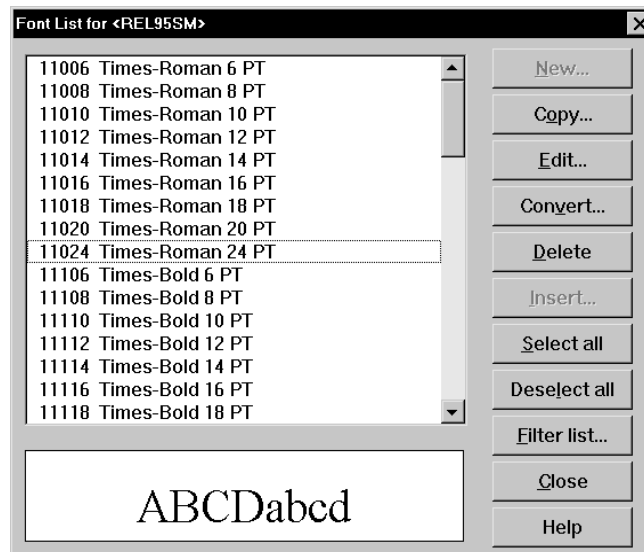
Copying Font Information

The Copy button lets you copy a font and assign a new font identification number or other necessary information. Copy duplicates a font that is in your current font set. You can then open the Font Properties window and change the ID number or other descriptive information associated with the copied font.

NOTE: Image Editor gives you a quick way to change all the fonts in a section (FAP file). For example, suppose you have a section designed with Helvetica Regular 10 point (font ID 16010) and you want to change all text using this font to Times (Roman) 10 point (font ID 17010). You can quickly do this by selecting the Edit, Font Change option.

To copy a font, follow these steps:

- 1 Start Font Manager by selecting Font Manager from the system folder, or by selecting Font Manager from the Image Editor Tools menu.
- 2 Select the appropriate font set. The Font List appears.

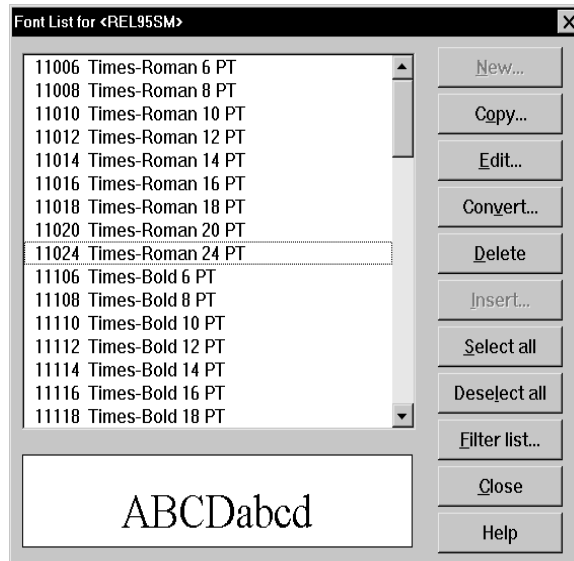


- 3 Click on the font you want to copy to highlight it and click Copy. The Font Properties window appears.
- 4 The system automatically assigns the next number in the list. To change the number to a different font ID, click on the newly added font, then click the Edit button.
- 5 Edit any other font information as necessary. When finished, click Ok.

Editing Font Information

The Edit button gives you a way to edit the information about fonts and printers.

- 1 Start Font Manager by selecting Font Manager from the system folder, or by selecting Font Manager from the Image Editor Tools menu.
- 2 Select the appropriate font set. The Font List appears.



- 3 Highlight the font you want to edit information for and click Edit. The Font Properties window appears.
- 4 Edit any font information as necessary and click Ok.

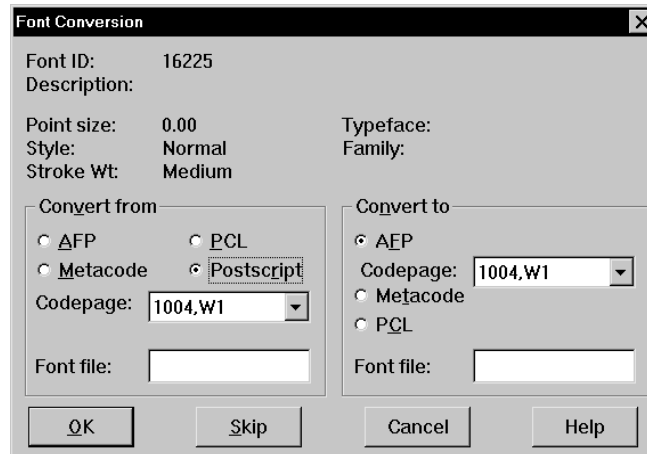
Converting Fonts

Converting fonts lets you modify fonts so you can use the same font for different types of printers. If you want to use the fonts on AFP, PCL, and Xerox printers, you can convert the standard fonts that are included with the system. The standard fonts are PostScript fonts supported only by PostScript printers in pre-converted format. You can also convert fonts for your different printers.

Before you convert fonts, make sure the font files reside in your master resource library. PostScript fonts are loaded in the FAP\MSTRRES\FMRES\DEFLIB directory of your system. If you want to convert other fonts, make sure you load the font files into your font directory. Check with your system supervisor for assistance.

Follow these steps to convert a font:

- 1 From the Font List window, select the font you want to convert, and click Convert. The Font Conversion window appears.



- 2 The window shows information about the current font, such as font ID, description, point size, and so on. This information also appears on the Description tab of the Font Properties window.
- 3 In the Convert From field, click the option associated with the font type you want to convert.
- 4 If you are converting a PostScript font, enter 1004 or an appropriate code page in the Codepage field.

NOTE: If a font file appears in the Convert To field, the font may already be converted. Click Skip to display the next font in the list. By selecting Skip or Cancel, you will not overwrite the converted font.

- 5 In the Convert From field, accept or type the font file name in the Font File field. Remember that you use the pre-converted font file name.
- 6 In the Convert To field, click the option for the font type to which you want to convert. If you convert to an AFP font, enter 1004 in the Codepage field.

NOTE: If you are converting to an AFP font, your font file name has to have an extension of 300. Also files entered in convert from and convert to should have a file extension.

Note too that the system adds character set and code page names when importing AFP fonts into a new FXR file. To add this information to an existing FXR file, you must re-create the FXR file.

- 7 In the Convert To field, enter the new font file name in the Font File field.
- 8 Click Ok to convert the font, or click Skip to stop the conversion. When you select Ok, the system creates a copy of the font in your font directory, and retains a copy of the original pre-converted font.

NOTE: If you selected multiple fonts to convert, information pertaining to the next font displays when you select Skip.

Converting fonts from other vendors

If you purchase additional fonts from a third party vendor, you must convert the fonts for use on your printers. You must also install the fonts onto your computer and printers. Copying the new, converted fonts onto your hard drive lets the system display them accurately on screen.

NOTE: Your resource library setup tells Image Editor where your font files and your cross-reference files reside.

There are several legal issues to be aware of before converting fonts from other vendors. Font vendors generally copyright the fonts they create. You can legally convert fonts only if the font vendor grants permission.

The converted font is bound by the same copyright restrictions that apply to the original font. For example, if your license does not permit you to use the font on more than one computer at a time, then you are not permitted to use the converted font on more than one computer at a time. In addition, it may be a copyright violation to copy converted fonts to other platforms running on the same computer.

Working with multiple printers

If you are working in a multi-printer environment, your combination of printers and specific printing needs dictates your options and entries. You may also need to run the font conversion utility to convert fonts to various printer platforms. You should convert the fonts you want to use for each printer, then define the font on the appropriate printer Font Properties window.

The system includes several font cross-reference files. The REL103SM.FXR file contains all information for Courier, Univers, and Times (Roman) fonts for PCL, AFP, Metacode and PostScript printers.

The DESKJET.FXR provides font information for HP Deskjet printers. Unless you use only Deskjet printers at your company, your system uses the REL103SM.FXR file. The following tables list the font file names used in the standard font cross-reference file.

If you are working in a multiple printer environment, follow these steps:

- 1 Load the font files into the system.
- 2 If the fonts are from a specific printer, copy the font files to the master resource library where you store your fonts, such as FAP\MSTRES\FMRES\DEFLIB\.
- 3 Select the font to convert; then select Convert. The Font Conversion window appears.
- 4 Convert the font to a specific printer platform, following the conversion steps outlined in [Converting Fonts on page 227](#).
- 5 Copy the converted font to your printer.
- 6 Compare the font names with the printer font names.

Repeat these steps for each font you select for use by multiple printers.

Description	PCL font file	Metacode font file	AFP font file
Courier			
7 pt	COM_7_X.SFP	COM7X	Xo42107C
8 pt	COM_8_X.SFP	COM8X	Xo42108C
10 pt	COM_10X.SFP	COM10X	Xo421001
12 pt	COM_12X.SFP	COM12X	Xo4210BC
Courier Bold			
7 pt	COB_7_X.SFP	COB7X	Xo44107C
8 pt	COB_8_X.SFP	COB8X	Xo44108C
10 pt	COB_10X.SFP	COB10X	Xo44100C
12 pt	COB_12X.SFP	COB12X	Xo4410BC
Courier Italic			
7 pt	COO_7_X.SFP	COO7X	Xo43107C
8 pt	COO_8_X.SFP	COO8X	Xo43108C
10 pt	COO_10X.SFP	COO10X	Xo43100C
12 pt	COO_12X.SFP	COO12X	Xo4310BC
Courier Bold Italic			
7 pt	COBO_7_X.SFP	COBO7X	Xo45107C
8 pt	COBO_8_X.SFP	COBO8X	Xo45108C
10 pt	COBO_10X.SFP	COBO10X	Xo45100C
Helvetica			
6 pt	HV_6_X.SFP	HV6X	XoH2106
7 pt	HV_7_X.SFP	HV7X	XoH2107C
12 pt	HV_12X.SFP	HV12X	XoH210BC
14 pt	HV_14X.SFP	HV14X	XoH210DC
16 pt	HV_16X.SFP	HV16X	XoH210FC
24 pt	HV_24X.SFP	HV24X	XoH210NC
Helvetica Bold			
6 pt	HVB_6_X.SFP	HVB6X	XoH4106C

Description	PCL font file	Metacode font file	AFP font file
12 pt	HVB__12X.SFP	HVB12X	XoH410BC
14 pt	HVB__14X.SFP	HVB14X	XoH410DC
16 pt	HVB__16X.SFP	HVB16X	XoH410FC
18 pt	HVB__18X.SFP	HVB18X	XoH410HC
Helvetica Italic			
6 pt	HVO__6_X.SFP	HVO6X	XoH3106C
12 pt	HVO__12X.SFP	HVO12X	XoH310BC
14 pt	HVO__14X.SFP	HVO14X	XoH310DC
16 pt	HVO__16X.SFP	HVO16X	XoH310FC
Helvetica Bold Italic			
6 pt	HVBO_6_X.SFP	HVBO6X	XoH5106C
12 pt	HVBO_12X.SFP	HVBO12X	XoH510BC
14 pt	HVBO_14X.SFP	HVBO14X	XoH510DC
16 pt	HVBO_16X.SFP	HVBO16X	XoH510FC
Helvetica Symbol			
Symbol 10 pt.	SY___10X.SFP	SY10X	Not available

Here is an example which shows how to convert and add a single font (Times (Roman), Normal, 18 point) from the PostScript font set, for an AFP, PCL, and Metacode printer.

Let's assume this is the font name (found in the FAP font directory): (TIR____.PFB).

- 1 Insert the PostScript font into your FXR file.
- 2 Open the Font Conversion utility within the Font List window. Then convert the font to AFP, PCL, and Metacode format in this sequence. See [Converting Fonts on page 227](#) for specific conversion steps. Refer to each respective printer manual for font file naming convention guidelines.
- 3 Copy the appropriate converted font file to the associated printer.

Deleting Fonts

The Delete option lets you exclude a font from your font set. If you delete a font, it is not deleted from the system. The font is simply no longer available in the font set.

NOTE: If you compose a section with a font which is later deleted from the font set, you must revise the section. You can verify that all the fonts within a section are valid fonts in the current font set by using the Font Check tool, available from the Image Editor Tool menu.

Follow these steps to delete a font:

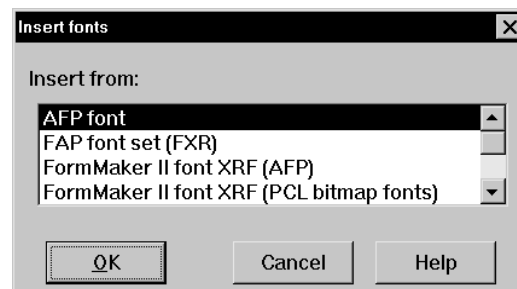
- 1 From the Font List, highlight the font or fonts you want to delete.
- 2 Click Delete. A confirmation message appears.
- 3 Click Yes to confirm or No to cancel.

Inserting Fonts

Use Insert to quickly add font information to your font set files. Insert lets you add multiple font information or individual font information to the font set or FXR file. If necessary, you can edit font information to change the font ID, typeface, family, and printer specific information. (Use New to manually add fonts to your font set; use Insert to automatically add font information.) You can also use Insert to add fonts to your font set. Using Insert is quicker and more efficient since you do not have to manually enter the specific characteristics of each font as you do with the New option.

To insert fonts, click Insert in the Font List window to display the Insert Fonts window.

NOTE: Insert is active only if no font is selected. If the Insert button is not active, click the Deselect All button to make sure no fonts are selected. This will activate the Insert button.

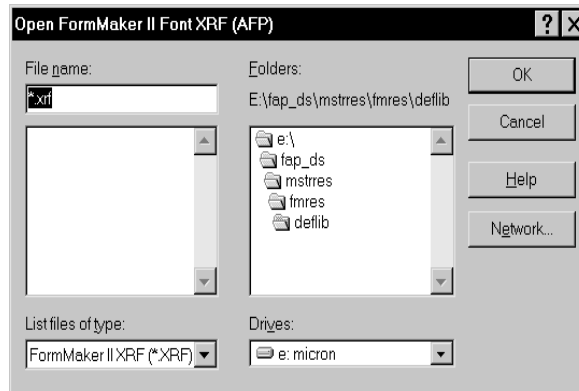


- 1 Click the font type or the fonts you want to insert. You can insert multiple fonts. The following is a brief description of the several font types that are available:

Font Type	Description
AFP font	This option lets you select an AFP coded file to import. AFP character set files begin with the letter <i>X</i> , such as <i>XoDATIN8.FNT</i> . A coded font file contains references to a specific character set and a specific code page. The corresponding character set and code page files must be in the same directory as the coded font file to import it. The font information imported from the AFP font is assigned a font ID which is one greater than the largest font ID contained in the font cross-reference file (FXR).
FAP font set (FXR)	Inserts font information stored in another system font set. Cross-reference files usually end with an FXR file extension. Once you select an import FXR file, you will see a list of the fonts referenced in the file. You can import any or all of the fonts contained in the file.
FormMaker II font XRF* (AFP)	Inserts a FormMaker II cross-reference file and the associated AFP bitmap fonts into the cross-reference file (FXR) you are currently using.
FormMaker II font XRF* (PCL bitmap fonts)	Inserts a FormMaker II PCL cross-reference file and the associated PCL bitmap fonts into the cross-reference file (FXR) you are currently using. You retain the original FormMaker II PCL cross-reference file.
FormMaker II screen font def.* (FON)	Inserts a single FormMaker II window font. Use this option only when a printer font is not available.
PCL bitmap font	Inserts a PCL bitmap font. The font information imported from the PCL font will be assigned a font ID which is one greater than the largest font ID contained in the font cross-reference file (FXR).
PostScript font	Inserts a PostScript scaleable font.
TrueType font	Inserts a TrueType scaleable font.
Xerox Metacode fonts	Inserts a Xerox bitmap font. The font information imported from the Metacode font will be assigned a font ID starting at one greater than the largest font ID contained in the font cross-reference file (FXR).

*These font file formats are from FormMaker II DOS System.

- 2 Click Ok. Depending on the fonts or font type you selected, either an open fonts or load font window will appear:



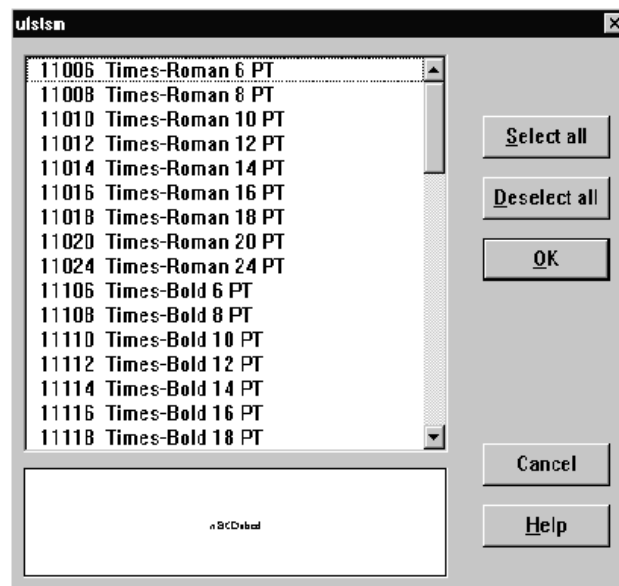
- 3 Select the file you want to open or load. If the file is in a different directory or folder, use the Drives and Folders fields to find the file. Click Ok.

NOTE: There is a Windows Network button available in this window; use it to map network drives.

- 4 To continue with inserting bitmap fonts and FXR files, follow the instructions below. To finish inserting PostScript and TrueType fonts, see [Inserting PostScript and TrueType Fonts on page 235](#).

Inserting bitmap fonts and FXR files

Once you open a FormMaker II XRF file (AFP or PCL) or an FXR file, the Font List window appears.



Here is a summary of your options:

To	Click
Choose all fonts in the FXR or XRF file	Select All
Deselect all fonts currently selected	Deselect All
Choose specific fonts in the FXR or XRF file	The specific fonts
Insert selected fonts into your font set	Ok

Keep in mind...

- If you select a font ID already present in your font cross-reference file, the system will give you the option of replacing the existing font, or assigning the imported font an ID at the end of the font list.
- The font information imported is assigned a font ID which is one greater than the largest font ID contained in the font cross-reference file.
- You may need to modify the font ID, typeface, family, and printer specific information for the bitmap fonts after they have been inserted into your current FXR.
- If you install fonts supplied by an outside vendor, you may need to copy those font files to the proper directory in your resource library, such as:

C:\FAP\MSTRRES\FMRES\DEFLIB\

Remember, the master resource setup tells Image Editor the directory location of your fonts and your FXR file.

Inserting PostScript and TrueType Fonts

To insert PostScript and TrueType fonts into an FXR file, you must have these files and INI options:

File name	Description
IFW32.DLL	Intellifont Runtime DLL Postscript, TrueType font reader You must store this file in the system's DLL directory.
IF.FNT	Intellifont Typeface Index file Store this file in the directory specified in the DEFLib option in the FMRes control group. See below for more information.
UIF.SS	Intellifont Symbol Set file Store this file in the directory specified in the DEFLib option in the FMRes control group. See below for more information.
UMT.SS	MicroType Symbol Set file Store this file in the directory specified in the DEFLib option in the FMRes control group. See below for more information.

File name	Description
UTT.SS	TrueType Symbol Set file Store this file in the directory specified in the DEFLib option in the FMRes control group. See below for more information.
PLUGIN.TYQ	Intellifont Plugin and Typeface Library file Store this file in the directory specified in the DEFLib option in the FMRes control group. See below for more information.
PLUGIN.TTF	TrueType Plugin and Typeface Library file Store this file in the directory specified in the DEFLib option in the FMRes control group. See below for more information.
CODEPAGE.INI	List of code pages Store this file in the directory specified in the DEFLib option in the FMRes control group. See below for more information.

For example, if your INI file contained this option:

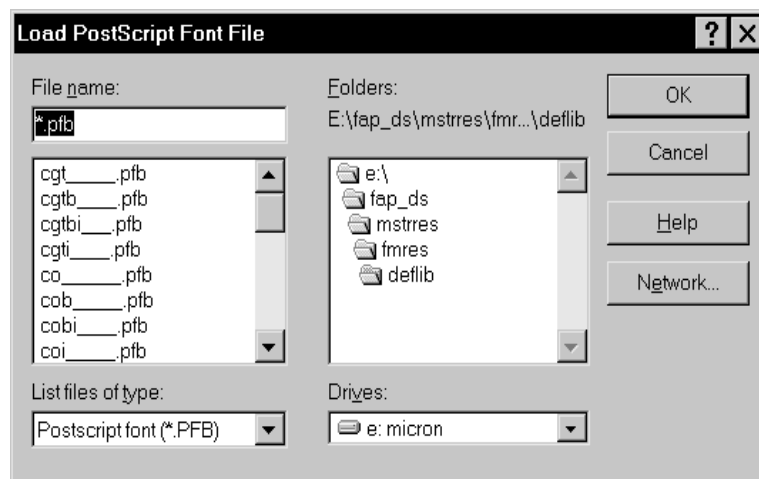
```
< FMRes >
  DEFLib = c:\fap\fonts
```

the system would look for these files (IF.FNT, UIF.SS, UMT.SS, UTT.SS, and so on) in the C:\FAP\FONTS directory. If you omit the DEFLib option, the system looks for these files in the ..\MSTRRES\FMRES\DEFLIB\ directory.

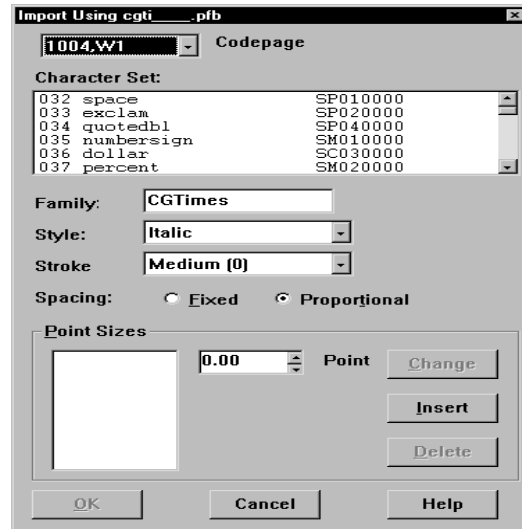
Follow these steps to insert PostScript and TrueType fonts:

- 1 Open your font cross-reference file and click Insert. Select the font type in the Insert Fonts window. The system shows you the appropriate Load Font File window. If you select a PostScript font from the Insert Fonts window, the Load PostScript Font File window appears.

NOTE: The insertion procedure is the same for both font types.



- 2 Select the file you want to insert and click Ok. The Import Using [font name] window appears.



Because PostScript and TrueType fonts are scalable fonts, you can select the point sizes you want to import. See [Font Terminology on page 172](#) for help understanding this window's fields.

- 3 Select 1004 in the Codepage field. The system displays the characters associated with the code page in the Character Set field. See [Using Code Pages on page 178](#) for information about various code pages.
- 4 Click the font you want to insert.
- 5 Select any changes you want to make in the Style, Stroke, and Spacing fields.
- 6 In the Point field, use the scroll arrows to select the font point size you want to include in the font set and click Insert. The font point size appears to the left of the Point Sizes field. Repeat this step for each point size you want to include in your font set.

NOTE: To select the point size, enter -1 in the point field of the Import Using (font name) window. Then click Insert. The system tells you that you have entered an illegal character in a numeric only field and asks you to correct the entry before continuing. Click Ok in the message. The system will then fill in the point values in the Point Sizes field for you.

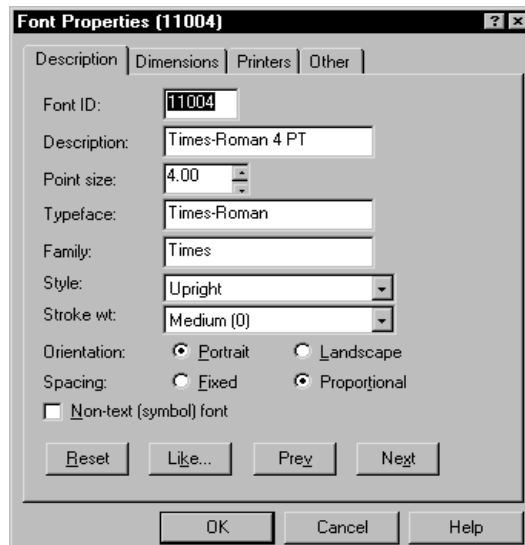
- 7 Click Ok to include the various font point sizes in your font set. The font information imported from the PostScript or TrueType font is assigned a font ID starting at one greater than the largest font ID contained in the font cross-reference file (FXR).
- 8 Image Editor redisplay the Font List. You can insert additional fonts in your font set if necessary.

CHOOSING SCREEN FONTS

After inserting fonts into the FXR, or if your company uses fonts other than those distributed with the system, you can use the Like button in the Font Properties window to select screen fonts which more closely resemble the printer fonts.

Follow these steps to choose a screen font:

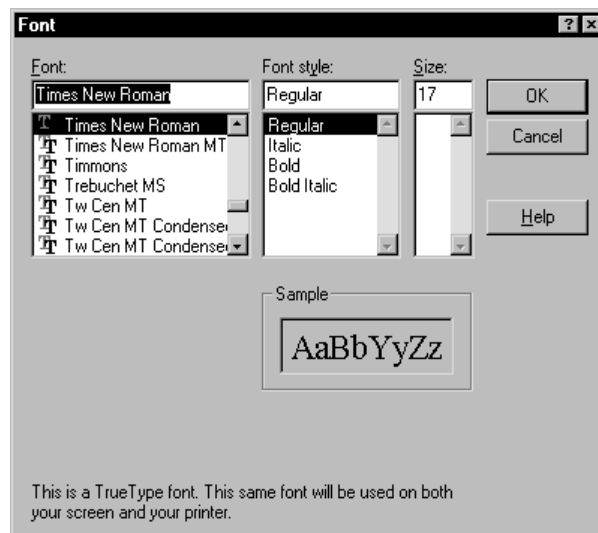
- 1 Open the FAP font set in Font Manager to display the Font List window.
- 2 Select a font in the font list and click Edit. The system displays the Font Properties window.



The FXR file contains the names of the screen and printer fonts to use. When you insert fonts into the FXR, the Font File fields on the Printers tab display the names of the printer fonts to use. The Family field on the Description tab contains the name of the Windows screen font to use based on the printer font file you inserted into the FXR. This means the Family field may not contain the name of a valid Windows screen font.

- 3 Click Like to display the Font window.

NOTE: Using the Like option can affect GDI printing which creates output based on what appears on the screen.



The Font window lists the Windows screen fonts installed on your computer.

- 4 Select the Windows display font that most closely resembles the printer font and make any necessary adjustments to the style and size. Click Ok to return to the Font Properties window. The screen font you selected will be displayed in the Family field and the style and size will be set for the font. This affects any object to which you can assign a font ID.
- 5 Click Ok in the Font Properties window to apply these changes and close the window. You can elect to save the changes to your FXR file when you select Close in the Font List window.

Understanding the System

If you plan to use your modified FXR file to create FAP files, be sure to make the FXR file available to anyone working with the FAP files. Keep in mind that those users will also need the same fonts installed on their systems.

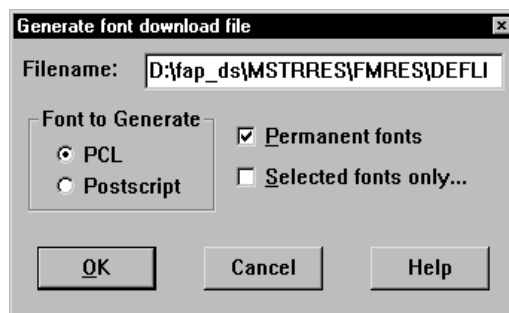
GENERATING FILES USING FONT MANAGER

Use Font Manager's File, Generate menu option to quickly generate FNT, XRF, and PFM files. FNT files are used to download fonts to a PCL or PostScript printer. XRF files are FormMaker II font cross-reference files. PFM files are Docuflex font format files.

GENERATING AN FNT FILE

The Generate FNT File option lets you automatically generate an FNT file for downloading fonts to a PCL or PostScript printer. Follow these steps to generate an FNT file:

- 1 Select the Resources, Fonts option from the system menu or click on the Fonts icon. The Open FAP Font Set window appears.
- 2 Select the font set you want to generate an FNT file for and click Ok. The Font List window appears.
- 3 Drag the screen to the right to display the Font Manager window. Then select File, Generate, FNT File. The Generate Font Download File window appears.



- 4 Check the name in the File name field. The system defaults to the name of the font cross-reference file with the *FNT* extension. Make any necessary changes.
- 5 Select either PCL or Postscript for the type of font to generate.
- 6 Select Permanent fonts or Selected fonts as necessary. This does not apply to Postscript fonts.

Your permanent, selected, and temporary font choices are defined in this table:

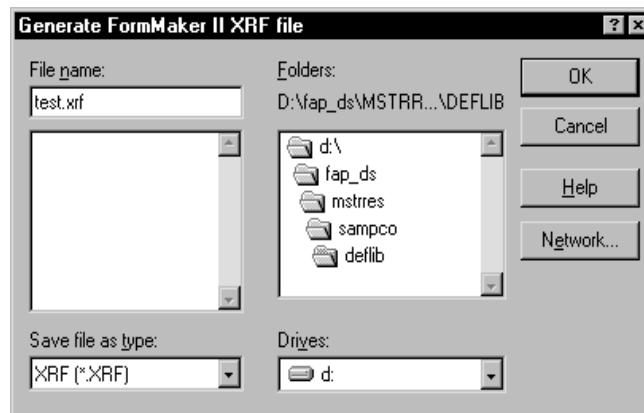
Fonts	Description
Temporary fonts	Downloaded PCL fonts that are deleted during a printer reset. A printer can be reset via a command in a PCL print stream or from the console. Temporary fonts are deleted from memory when the printer is reset or turned off. This field has no effect on PostScript fonts.
Permanent fonts	Downloaded PCL fonts that remain in the system during a printer reset. Permanent fonts are deleted from memory when the printer is turned off. Permanent fonts are not deleted during a printer reset. This field has no effect on PostScript fonts.
Selected fonts	Selected fonts lets you select specific font IDs to build the FNT file. If you leave this field unchecked, the system includes all font IDs in the FXR file.

- 7 Click Ok.

GENERATING AN XRF FILE

With Font Manager you can use a FAP font set to generate a FormMaker II XRF file. To begin, open Font Manager by selecting Resources, Font. The Open FAP Font Set window appears.

- 1 Select the font set you want to work with and click Ok. The system displays the Font List window.
- 2 Drag the screen to the right to display the Font Manager window; then select File, Generate, XRF File. The Generate FormMaker II XRF File window appears.



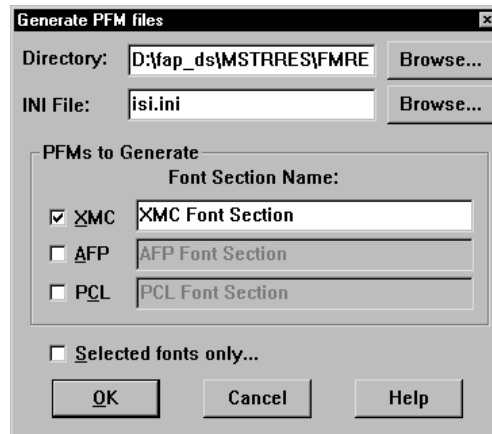
- 3 Enter a name for the XRF file in the File Name field; be sure you have the file path set to the library you want. You can use the Windows Network button if you need to map a drive path. Click Ok to generate the XRF file.
- 4 The system displays a confirmation message stating that the XRF file generated successfully.

GENERATING PFM FILES FROM AN FXR FILE

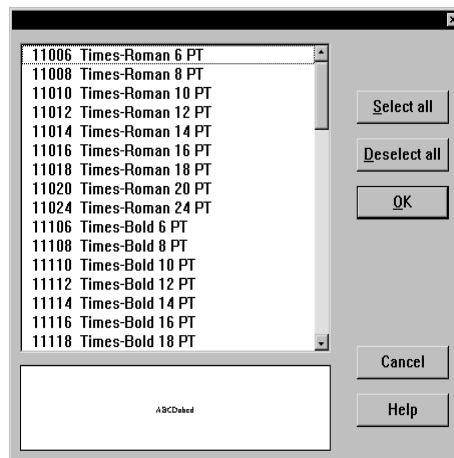
Font Manager can generate Docuflex font format files, known as PFM files. These files are listed in the font section of Docuflex INI files and are used when converting a document from one format to another.

To create a PFM file, select Resources, Fonts to start Font Manager. The system displays the Open FAP Font Set window.

- 1 Select a font cross reference file (FXR) and click Ok. The Font List window appears.
- 2 Drag the screen to the right to display the Font Manager window; then select File, Generate, PFM Files. The following window appears.



- 3 Enter the destination directory for the PFM files in the Directory field. Use the Browse button to change the destination directory.
- 4 Enter the name of the INI file in the INI File field, or use the Browse button to select an INI file.
- 5 In the PFMs to Generate field, select which type of PFM files for which printer type to generate.
- 6 If you want to use all the fonts in the FXR file, leave the Selected Fonts Only field unchecked and click Ok. Font Manager generates the PFM files.
- 7 If you only want to generate PFM files for specific fonts, click the Selected Fonts Only field then click Ok. The system displays a list of the fonts in the FXR file:



- 8 Select the fonts and click Ok. The system displays a message confirming that the PFM files were successfully generated.

A PFM file is generated for a particular printer type and information for that font goes in the font record. For example, in the REL95SM.FXR file there are 68 fonts, but there is only setup data for all 68 fonts for the AFP printer. There is XMC and PCL information for only 64 of the fonts, so there could be a maximum of 64 PFM files of the XMC and PCL printer types for that FXR file.

The name of each PFM file is determined from the setup data in the font record for each printer type. For example, in REL95SM.FXR for font number 11012, the XMC font is FXTINB.PFM, the AFP font is XoDATINB.PFM, and the PCL font is FPTINB.PFM.

When Font Manager generates the PFM files, it also creates an extract file which is, by default, called *ISI.INI*. You can insert the contents of this file into your Docuflex INI files as one of the FontSection control groups. The extract files are placed in the same directory as the generated PFM files.

NOTE: If the ISI.INI file exists, the system appends the font section information in the existing file. If you do not include a path for the ISI.INI file, the system creates or updates the file in the directory from which you ran Font Manager, such as *..\FAP\DLL*.

Here's a sample ISI.INI file:

```
[XMC Font Section] Fonts=64 FontThreshold=70000
Font1=E:\LIBRS\FMRES\DEFLIB\FXTIN6.pfm; 11006 Times-Roman 6 point
Font2=E:\LIBRS\FMRES\DEFLIB\FXTIN8.pfm; 11008 Times-Roman 8 point
Font3=E:\LIBRS\FMRES\DEFLIB\FXTIN0.pfm; 11010 Times-Roman 10 point
Font4=E:\LIBRS\FMRES\DEFLIB\FXTINB.pfm; 11012 Times-Roman 12 point
Font5=E:\LIBRS\FMRES\DEFLIB\FXTIND.pfm; 11014 Times-Roman 14 point
```

When you use these PFMs to read or write FAP files in the Docuflex system, be sure to use the accompanying extract files in the FontSection control groups in the Docuflex INI files. The numbers after the semicolons are used by the system to identify font numbers in FAP records.

MAPPING FONTS FOR FILE CONVERSIONS

When converting a file from one format to another, you may need to convert the fonts used in the document. You can use INI control groups and options to map fonts in a source document to the fonts you want to use in the destination document. For instance, if you are converting an RTF file into a FAP file, you can use the following control group:

```
< RTFFontMAP >  
    Arial = Swiss
```

This tells the system to convert all Arial fonts into Swiss fonts. Use this control group when converting DCD files into FAP files:

```
< FontFamilyMatching >  
    Arial = Swiss
```

Place these control groups and options in the FAPCOMP.INI file.

RTF and DCD files contain font information about the generic font families used. For example, Arial and Univers, both sans serif proportional fonts, belong to a generic font family called *Swiss*.

The RTF and DCD converters in the system use the RTFFontMap and FontFamilyMatching control groups to assign a font when other means of mapping fonts from the RTF or DCD file fails.

In Windows environments, there are several generic font families, as shown in this table:

Family	Description
Decorative	Specifies a novelty font, such as Old English.
Dontcare	Specifies a generic family name. This name is used when information about a font does not exist or does not matter. The default font is used.
Modern	Specifies a monospace font with or without serifs. Monospace fonts are usually modern fonts, such as Pica, Elite, and Courier New.
Roman	Specifies a proportional font with serifs, such as Times New Roman.
Script	Specifies a font that is designed to look like handwriting, such as Script and Cursive.
Swiss	Specifies a proportional font without serifs, such as Arial.

CHAPTER 6

Setting Up Printers

The system supports printing on a variety of printers ranging from network laser printers to high volume production printers. This chapter describes how to set up the system to print on this wide array of printers.

In this chapter you will find information on the following topics:

- [AFP Printers on page 246](#)
- [Metacode Printers on page 263](#)
- [PCL Printers on page 289](#)
- [PostScript Printers on page 301](#)
- [Using the GDI Print Driver on page 311](#)
- [Using Pass-through Printing on page 318](#)
- [Creating PDF Files on page 320](#)
- [Creating RTF Files on page 321](#)
- [Using the VIPP Print Driver on page 324](#)
- [Emailing a Print File on page 342](#)
- [Choosing the Paper Size on page 346](#)
- [Creating Print Streams for DocuSave on page 358](#)
- [Handling Multiple Paper Trays on page 363](#)

For each type of printer, this chapter discusses set up issues, printer resources, special features, performance considerations, troubleshooting, and more.

AFP PRINTERS

IBM created the Advanced Function Printing (AFP) language. The data streams produced by Documaker applications for AFP printers are called *Mixed Object Document Content Architecture* (MO:DCA) data streams. MO:DCA data streams are sometimes referred to as AFP data streams (AFPDS).

You must have a program such as IBM's Print Services Facility (PSF) to convert AFP data stream into the printer's native language. PSF is the umbrella software that brings the AFP resources (created by AFP or system utilities) together in one print job and sends it to the printer.

NOTE: All system print drivers support 24-bit color graphics. If your printer does not support color, the print driver will automatically convert the color graphics into monochrome graphics. Keep in mind that for the best performance you should avoid color graphics.

AFP INI OPTIONS

You define the necessary printer options for the system to produce AFP data streams. These options specify how the system creates AFP output. Most of the AFP-related options are found in a `PrtType:XXX` control group, where `XXX` indicates the different printer types. `PrtType:AFP` is a common control group name used to contain AFP settings. The most common AFP printer options are shown below (default values are bold):

Option	Values	Description
Device	Any file or device name	The name of the file or device (LPT1) where the AFP data stream should be written. This setting is ignored by the GenPrint program but is used by Documaker Studio and other system programs.
Module	AFPPRT	The name of the program module which contains the system's AFP print driver. See also the discussion of the Class option. See also Using defaults for the Module and PrintFunc options on page 250 .
PrintFunc	AFPPrint	The name of the program function that is the main entry point into the system's AFP print driver. See also Using defaults for the Module and PrintFunc options on page 250 .
Resolution	240/300	The dots per inch (dpi) resolution of the printer which receives the AFP data stream
SendOverlays	Yes/No	Set to Yes if you created AFP overlays for each FAP file

Option	Values	Description
ChartResolution	120/150/ 240/300	Used when printing charts as inline bitmap graphics on an AFP printer that does not have graphics (GOCA) support. Defaults to one-half of the Resolution option setting.
LandscapeSupport	Yes/ No	Although not required for printing, you can set this option to <i>Yes</i> if your printer supports landscape medium maps. Generally, AFP printers using cut-sheet paper <i>do not</i> support landscape medium maps.
SplitText	Yes/ No	Used to minimize the print differences between 240 and 300 dpi printing.
SplitPercent	0 to 100 (50)	Percentage of the width of the space character used to determine when the rounding error between 240 and 300 dpi printing has caused a significant difference and the text string should be split into smaller strings.
FudgeWidth	any number (0)	Can be used when building page overlays for sections smaller than a page.
GraphicSupport	0, 1, 2, 3	0 = no graphics (GOCA) support 1 = inline bitmap graphics support 2 = GOCA charts support 3 = inline bitmap graphics and GOCA charts support
PageNumbers	Yes/ No	Set to <i>Yes</i> to turn on form or form set page numbering
PrintViewOnly	Yes/ No	If set to <i>Yes</i> , the view only sections will print. This does not apply to entry only sections, which are never printed. Entry only sections are usually worksheets. If the section is marked as hidden and view only, it will not print.
PrePrintedPaper	Yes,Disabled	Determines if the check box which lets you print or not print pre-printed objects appears on the Print window. Also determines the default for this check box—checked or unchecked. You must add this option to the INI file if you want the check box to appear on the Print window. The default for this option includes the checkbox on the Print window and leaves it unchecked. All objects except fields can be designated as pre-printed on the object's Properties window.

Option	Values	Description
Class	<i>(first three characters of the Module option)</i>	<p>Specifies the printer classification, such as AFP, PCL, XER, PST, or GDI. If you omit this option, the system defaults to the first three letters from the Module option.</p> <p>Some internal functions expect a certain type of printer. For instance, all 2-up functions require an AFP printer. The internal functions check the Class option to make sure the correct printer is available before continuing.</p>
OnDemandScript		<p>Use this option to add comments to the print stream. This lets you handle archiving using OnDemand.</p> <p>Enter the name of the DAL script you want the system to run. This DAL script creates the On Demand records and adds them as comments.</p> <p>The AddComment function is also used in DAL scripts to add OnDemand command records. For more information about this and other functions, see the DAL Reference.</p>
TLEScript		<p>Enter the name of the DAL script to execute to add Tagged Language Element (TLE) records to the print stream.</p> <p>See Adding TLE Records on page 362 for more information.</p>
TLESeparator		<p>Enter the character you want to use to separate the key and value portions of the TLE comment string.</p>
TLEEveryPage	Yes/ No	<p>Optional. If you enter Yes, the TLE DAL script will be executed at the start of every page. If you enter No, the TLE DAL script is executed at the start of every form set. The default is No.</p>
PaperSize	0 , 1, 2, 3, 98	<p>Use this option to set a default paper size when converting AFP print streams using the Internet Document Server or the MRG2FAP utility.</p> <p>Enter 0 (o) for letter size (default)</p> <p>Enter 1 for legal size</p> <p>Enter 2 for A4 size</p> <p>Enter 3 for executive size</p> <p>Enter 98 for a custom size</p>

Option	Values	Description
DocusaveScript		<p>Use this option to add comments to the print stream. This lets you handle archiving using Docusave.</p> <p>Enter the name of the DAL script you want the system to run. This DAL script creates the Docusave records and adds them as comments.</p>
SendColor	Yes/No	<p>Enter Yes to send color information to the printer. AFP highlight color printing on printers from Xerox and Oce is supported.</p> <p>Make sure the objects you want to print in color (text, lines, shades, and so on) are set to print in color. The Print in Color option is on the Color Selection window. You can display this window by clicking the Color button on the object's Properties window.</p>
NamedColors		<p>Use this option to tell the system to use only specific AFP named colors. For example, if you wanted all highlight (non-black) colors mapped to blue, you would set the NamedColors option to <i>blue</i>.</p> <p>To allow the mapping of the colors you assigned to the objects in the FAP file to multiple colors, separate each color with a semicolon (;). For example, to use red, blue, and magenta, set the NamedColors option as shown here:</p> <pre>NamedColors = red;blue;magenta</pre> <p>The order you list the colors does not matter.</p>
SkipChartColorChange	Yes/No	<p>Enter Yes to suppress color changes normally done to enhance 3D bar charts.</p>
SuppressLogoUnload	Yes/No	<p>Enter Yes to suppress the unloading of graphics (LOG) files during a conversion of AFP files to FAP (or PDF) format. The default is No.</p>
ReplaceBitmap	LIGHT, LIGHTER, LIGHTEST, MEDIUM, DARK, DARKER, DARKEST, NOSHADE, SOLID, HORIZONTAL , VERTICAL, DIAGRRIGHT, DIAGLEFT, HATCH, or DIAGHATCH	<p>Enter the name of the bitmap you want to replace followed by one of the replacement patterns.</p> <p>The default is LIGHT.</p> <p>Keep in mind your entry must be in all caps.</p> <p>See Using Documaker shading patterns instead of shaded bitmaps on page 251 for more information.</p>

Option	Values	Description
DisplayCodedFont	Yes/No	Enter No to include the character set/code page combinations in the AFP font list, instead of the coded fonts. The defaults is Yes, which tells the system to include the coded fonts. See Outputting character set and code page information on page 252 for more information.

There are some additional options you can use to print inline graphics (LOG files). Be aware that not all AFP printers support these settings. You'll find these options in the AFP control group.

AFP Options	Values	Description
OutputHalfRes	Yes/No	Scales the bitmap loaded from the graphic to half resolution in memory before writing the output.
DoubleOutputRes	Yes/No	Does not change the bitmap loaded from the graphic, but would tell the printer to double its resolution when printed. This lets the system load graphics that are half resolution already.
SuppressZeroData	Yes/No	Suppresses data containing a series of zeros (white space in the bitmap).
TrimWhiteSpace	Yes/No	Suppresses data containing zeros (white space) at the right edge of the bitmap.
MultiLinesPerCommand	Yes/No	Tries to combine AFP commands into fewer records when printing the bitmap. You cannot use this option with the SuppressZeroData option.

Using defaults for the Module and PrintFunc options

Default values for the Module and PrintFunc options in the PrtType:xxx control group are provided when you use a standard print type name or print class, such as AFP, PCL, PDF, PST, VPP, XER, XMP, or GDI.

These defaults keep you from having to enter the Module and PrintFunc names in your INI file. For example, if you want to generate AFP print files, you can specify these INI options:

```
< Printer >
  PrtType      = MYAFP
< PrtType:MYAFP >
  Class       = AFP
```

And the system will default these options for you:

```
< PrtType:MYAFP >
  Module      = AFPPRT
  PrintFunc   = AFPPrint
```

Using Documaker shading patterns instead of shaded bitmaps

You can replace the shading bitmaps in AFP files with Documaker's internal FAP shading patterns. Using Documaker's internal FAP shading patterns results in smaller and more efficient FAP files and you will have more flexibility in choosing patterns.

To use Documaker FAP shading patterns, include the ReplaceBitmap INI option, as shown here:

```
< PrtType:AFP >
  ReplaceBitmap =
```

NOTE: The system ignores this option if the AFP output file being loaded is one generated by Documaker because it automatically replaces shading bitmaps from internally-generated AFP files with FAP shading patterns when appropriate.

The system replaces all occurrences of the bitmap you specify with the shading pattern you choose. The system places the replacement shading pattern in the same location as the AFP bitmap. To replace multiple bitmaps, repeat the ReplaceBitmap option as necessary.

The bitmap patterns that are replaced must be named in bytes 10-17 of the Begin Image (D3 A8 7B) AFP structured field and the bitmap name listed in the ReplaceBitmap option must match the bitmap name in the Begin Image structured field. All Begin Image structured fields encountered that have names that match the name in the ReplaceBitmap option are replaced.

NOTE: While the system does support color text, color bitmaps are not supported by the AFP loader of the MRG2FAP utility.

Printing highlight colors

The system supports AFP highlight color printing on printers from Xerox and Océ. Like other color printer support, the SendColor option must be set to Yes and the objects, such as text, lines, and shades must be set to *Print In Color*.

The RGB (red,green,blue) color setting for each FAP object is mapped to the closest AFP named color. The names of the available colors are as follows: blue, red, magenta, green, cyan, yellow, dark_blue, orange, purple, dark_green, dark_cyan, mustard, gray, and brown.

You use the NamedColors option in the AFP printer group to specify certain AFP named colors. For example, if you wanted all FAP (non-black) colors to be mapped to brown, you would use this INI option:

```
NamedColors = brown
```

To let the system map FAP colors to multiple colors, separate each color with a semicolon (;). For example, to use all of the default AFP named colors except brown, you would use this INI option:

```
NamedColors = Red;Blue;Magenta;Green;Cyan;Yellow
```

NOTE: The order in which you name the colors does not matter. In addition, the LOG2PSEG and FAP2OVL utilities include a /C=color parameter, where color is the one of the named AFP colors.

Character set and code page font information

When loading AFP, the system uses the information in the Character Set and Code Page Font fields in the FXR file instead of using the font information contained in the IBMXREF.TBL.

The AFP loader expects the AFP file's Map Coded Font (MCF) structured fields to contain references to AFP coded fonts. However, MCF structured fields can contain character set and code page information instead of the coded font information the FXR file requires.

Before version 11.2, for MCF structured fields that contained character set and code page information instead of coded fonts, you had to manually set up the IBMXREF.TBL file to resolve the character set/code page information to coded fonts in the FXR file.

Since the system includes character set and code page information in the FXR file, the AFP loader first checks the FXR file for this information and, if it exists, uses it. If the information does not exist, the AFP loader loads the information from the IBMXREF.TBL file.

Outputting character set and code page information

You can output the AFP character set and code page combination instead of the coded font in the font list when you generate normalized AFP files. If you want the character set/code page combinations to be output in the AFP font list, instead of the coded fonts, you must add the DisplayCodedFont option, as shown here:

```
< PrtType:AFP >  
    DisplayCodedFont = No
```

Keep in mind the FXR file must contain the character set and code page entries in the AFP font record for this option to work. If you set the INI option to No and the character set and code page entries are not in the FXR file, the font list in the AFP file will contain only the coded fonts.

NOTE: The AFP output record can only contain *either* coded fonts *or* character set/code page entries — it cannot contain a combination. It will default to coded fonts for all if the font for one or more objects does not contain character set/code page entries.

Using multiple code pages

You can use multiple code pages for creating AFP output. While the standard 37 code page is the default code page, alternate code pages are frequently used for fonts set up for them. Here is a summary of the new font definition files which were created to let you specify code pages:

File	Description
CODED.FNT	The coded font definitions. This file specifies which AFP code page and AFP font character set make up the coded font.
CPDEF.FNT	The code page definitions. This file maps each AFP code page to a Windows character set.
CPGID.CP	The code page map file. This file contains the character identifiers (and associated EBCDIC hexadecimal code points) for an IBM code page and maps them to character identifiers (and associated ASCII code points) for a Windows ANSI or SYMBOL character set.

Here are the general syntax rules for all new font definition files:

- A semicolon (;) in the first column of any of these files will cause the line to be treated as a comment statement and ignored.
- Section headers within files are enclosed either in brackets (< > or []) with *no* spaces and must *not* be removed or changed.
- All values are case insensitive.
- If a parameter value is invalid and a default value exists, it will be substituted.
- All parameters are positional.
- Blanks are allowed between parameter values.
- The question mark (?) is used in some areas as a single wildcard character.
- If the resource file exists in DEFLIB directory and contains valid data conforming to these specifications, it will be loaded and used.
- If bad data is encountered in the file, either the offending record is ignored or a warning is issued. If the file is considered corrupt or invalid enough, it may not be used at all.

CODED.FNT FILE. This file specifies which AFP code page and AFP font character set make up the coded font. The CODED.FNT file is necessary for basic multiple code page support.

When creating this file, keep these rules in mind:

- The coded font name and both parameters are required.
- A question mark (?) can be used as the wild-card character only for the second character in the coded font name and for any character of the character set name. This allows all the character rotations of the coded fonts to be handled with one entry for searching.
- After the coded font name, the character set name must be listed first, followed by the code page name.
- The character set and code page must be separated by a comma.

Here is an example of this file:

```
X?COL8=C?420080,T1000850
X?COL7=C?420070,T1000850
;Core
X?H210AC=C?H200A0,T1V10500
X?H210FC=C?H200F0,T1V10500
;FormMaker Fonts
X?FA????=C?FA????,T100ASC4
X?DA????=C?FA????,T1DOC037
X0P09X12=C0P09X12,T1DOC037
X0P12X16=C0P12X16,T1DOC037
```

CPDEF.FNT FILE . This file maps each AFP code page name to its code page global identifier (CPGID) and to a Windows character set. If you do not have at least one valid entry in this file for each code page you want to use, the system uses the default code page.

When creating this file, keep these rules in mind:

- Parameters must be separated by a comma.
- AFP code page name and code page identifier are required.
- If you create your own code page, you must assign it a unique code page identifier. Leading zeros are invalid.
- Code Page Global Identifier (CPGID) attribute's possible values: IBM-defined CPGID or your own defined CPGID between 65280 and 65534, inclusively. This value matches the name of a code page map file.
- For each CPDEF.FNT entry, you must have a corresponding code page map file with the same name as the CPGID.
- Windows character set attribute's possible values: ANSI or SYMBOL.

Here is an example of this file:

```
<CODEPG>
;codepage = cpgid,wincp
;*****Put User-defined/Custom code pages Here *****
T100ASC4=361,ANSI
T1DOC037=37,ANSI
T1OMR=5280,ANSI
T1POSTBC=5280,ANSI
;***** End User-defined/Custom code pages *****
T1000259=259,SYMBOL
T1000290=290,ANSI
T1000293=293,ANSI
T1000310=310,ANSI
DEFAULT=361,ANSI
```

CPGID.CP (CODE PAGE MAP FILE) . You must have a separate *CPGID.CP* file for each AFP code page entry in the CPDEF.FNT file. Each code page map file contains the character identifiers (and associated EBCDIC hexadecimal code points) for an IBM code page and maps them to character identifiers (and associated ASCII code points) for a Windows ANSI or SYMBOL character set. Code page map files are necessary for basic multiple code page support.

NOTE: The actual file name is not CPGID.CP, but rather the *CPGID* value from the CPDEF.FNT file with an extension of *CP*. For instance, in the CPDEF.FNT example, the first two lines are:

```
T100ASC4=361,ANSI
T1DOC037=37,ANSI
```

So, since those two entries are in the CPDEF.FNT file, that means that there must be code page map files with named *361.CP* and *37.CP*.

Also, if these two entries are in the CPDEF.FNT file, but the corresponding *361.CP* and *37.CP* code page map files are not in DEFLIB, the translations for those fonts will not be correct.

When creating this file, keep these rules in mind:

- Parameters must be separated by blanks.
- All four parameters are required.
- “NOMATCH” means there is not a matching character in the Windows character set.

Here is an example of this file:

(395.cp for the T1000395 code page mapped to the Windows ANSI character set):

```
;T1000395 to ANSI
SP010000 40 SP010000 20
LA150000 42 LA150000 E2
LA170000 43 LA170000 E4
LA130000 44 LA130000 E0
SP180000 8B SP180000 BB
SM560000 8C SM560000 89
SA000000 8D SP100000 2D
LI510000 8E NOMATCH 00
LI570000 8F NOMATCH 00
SM190000 90 SM190000 B0
LJ010000 91 LJ010000 6A
LF510000 A0 NOMATCH 00
;;;;;;;; ; SD150000 5E
;;;;;;;; ; SD130000 60
```

Using LLE records to link to external documents

For AFP files, LLE (Link Logical Element) records let you link internal or external documents into the AFP presentation space. For example when you are creating a PDF file, you might want to include in the text hotspots that link to a URL. These hotspots, when clicked, open that document.

NOTE: The LLE records are for use with text fields.

Place the LLE record immediately before the *BPT* – Begin Presentation Text record. Then, following the BPT record, you can have any number of PTX records containing a *TRN* (Transparent Data) control sequence, followed by a terminating *EPT* – End Presentation Text.

Here is an example of the LLE format:

Element	Description
5A	
00 32	record length
D3B490	LLE
00	Flags
00 00	reserved
01	Navigation Link Type
00	reserved
00 05	triplet length including this value
02	Link Source specification
/N	source text limited by triplet size) See below explanation of /N
00 11	triplet length including this value) 0x11 (17 decimal (2+1+14)
03	Link Target specification
http://xyz.com	target text limited by triplet size

In the above example, the text fields */N* and *http://xyz.com* would be encoded as hex EBCDIC. For example a source link such as:

```
00 05 02 /N
```

would be encoded as...

```
00 05 02 61 D5
```

The FAP library does not use the name (link source) member of the FAPLINK, therefore it is used for feature steering.

By specifying a /N (NEXT) as the source name, the system applies the current instance of the LLE to the first occurrence of a PTX record containing a TRN (Transparent Data) control sequence record. Once the LLE link information has been applied to that particular PTX FAPOBJECT, the system clears the LLE status so subsequent PTX records are rendered as non-hyperlinked text.

By default the LLE is applied to all subsequent PTX / TRN records until either an LLE is encountered with a /C as its source link to enable the clearing of the active instance of the LLE, or to use a normal valid LLE to supersede the prior usage.

If you are not using a /N or /C, you may use the source name area of the LLE for a brief descriptive label.

NOTE: The system does not support the use of the attribute link type or internal target links within FAP and therefore PDF documents.

The system only supports the conversion of LLE records in FAPSTEXT objects and linking to external documents.

AFP PRINTER RESOURCES

FormDef The system uses copy groups from its own FormDef named *F1FMMST.DAT*. Each copy group in a FormDef contains information about paper size, duplex, tray selection, jog, orientation, and so on. The FormDef must be available to PSF to print AFP data streams. You can use the AFPFMDEF utility to create or modify the FormDef.

Fonts AFP fonts are designed solely for AFP printers. For more information about fonts, see [Working with Fonts on page 171](#). In IBM AFP terminology, a font is described by three components:

CODED FONT. A coded font file contains references to specific character set and specific code page. Coded font files always begin with the letter *X*, such as *XoDATIN8*.

CODE PAGE. In IBM AFP terminology, a code page file maps code points to an AFP character name in a character set file. Code page files always begin with the letter *T*, such as *T1DOCo37*.

CHARACTER SET. A character set file contains the bitmap graphic of each character in the character set. Character set files always begins with the letter *C* (such as *CoFATIN8.240* or *CoFATIN8.300*). The character set file name extension (*240* or *300*) indicates whether the bitmap graphics are drawn at 240 or 300 dots per inch.

Monotype fonts Oracle Insurance has licensed for use and distribution with its systems, fonts from Monotype Imaging, Inc. The system includes both 240 and 300 dpi AFP fonts.

Overlays Use the FAP2OVL utility to create AFP overlays from FAP files. The OVLCOMP utility also lets you create AFP overlays from FAP files. These overlays must be available to PSF to print AFP data streams when the SendOverlays option is set to Yes.

Page segments Use the LOG2PSEG utility to create AFP page segments from graphics (LOG files). These page segments must be available to PSF to print AFP data streams.

NOTE: For information on system utilities, see the [Docutoolbox Reference](#).

AFP 2-up support The system include rules you can use to generate and merge print streams for AFP printing for printers that support 2-up printing. See [Handling 2-up Printing on page 68](#) in the [Documaker Server System Reference](#) for more information.

AFP TROUBLESHOOTING

Floating section limitations

The system lets you compose a page from several sections. The system also lets you create overlays for these sections. There is one limitation when you print these sections on a landscape page. Overlays on a landscape page can only be placed vertically on the page. Overlays on a landscape page *cannot* be placed horizontally on the page.

This means, in your SetOrigin rule, you *cannot* specify any non-zero, positive number for the X-relative displacement. Create your FAP files accordingly, but keep in mind that they can be moved down but not across. This limitation exists only for AFP overlays, and only in landscape mode.

Objects extending beyond the edges

Another type of error can occur if the overlay for a custom-sized section is too small for the objects (text, lines, graphics, and so on) contained within it. If the AFP overlay's page size is too small, objects may be clipped to the page size, printed as solid black rectangles, or trigger error messages.

Documaker Studio and Image Editor offer an Auto-size option which you can use to make sure the custom-sized section is large enough to contain all objects placed within it. Use this feature to prevent most custom page size problems.

Be careful placing text at the extreme left edge of the section because it may cause errors that the Auto-size option cannot detect. For instance, suppose you have this text label positioned on the left edge of the FAP file (left offset = 0):

Beneficiary

When printed, black rectangles or an error message may appear instead of the text.

This can occur because some of the characters in the italic font (Times New Roman) have a *negative left offset*. This means that the characters print to the left of where they would normally start. A negative left offset may be easier to understand by looking at these characters:

ef

Notice how the bottom of the *f* goes under the *e*. This is an example of a negative left offset. Because it is positioned to the left of where it would normally start, the character is now positioned off the left edge of the overlay.

This kind of detailed character information is not stored in the FXR file so Documaker Studio and Image Editor have no way to know there may be a problem. You can, however, move the text labels in the FAP file to correct the problem.

Conflicts between page and form orientation

If you create a custom-sized page, be aware of any conflict between page orientation and the form orientation. If the form orientation is not the same as the page orientation, the page will not print according to the page orientation, but will follow the form's orientation.

NOTE: This happens only in case of custom size pages. Standard size pages obey the page orientation.

Multi-page FAP limitation

There is a problem when a landscape, multi-page FAP has different page sizes on each page. All pages of a multi-page FAP file should be the same size. As a workaround, use Documaker Studio or Image Editor to correct the page sizes. After saving the FAP file, you can then generate proper AFP overlays.

Printing rotated variable fields

Here is a list of field options you can specify in the NAFILE.DAT file:

Option	Description
E	Error
M	Manual
P	Protected
G	Global scope (entire form set)
F	Form scope
H	Hidden field – a dummy field, not displayed or printed
N	Nonprintable field (displayed, not printed)
C	Send-copy-to field (receives current recipient name at print time)
9	Rotated 90 degrees
8	Rotated 180 degrees
7	Rotated 270 degrees

Some of these options require the FAP field attributes to be available at runtime, since the DDT file does not include the necessary information. Use the CheckImageLoaded rule to make sure this information is available.

AFP 240 dpi print problems

Due to differences in resolution on 240 and 300 dpi printers, a text string may print with slightly different lengths. One example where this may be noticeable is when the text is printed inside of a boxed region. Another example where this may be noticeable is when a text area contains an embedded variable field.

To minimize the print differences between 240 and 300 dpi printing, use the SplitText option. Make sure these options are in your printer PrtType:xxx control group:

```
< PrtType:AFP >
  SplitText      = Yes/No (default is No)
  SplitPercent   = ###    (% of space-width as max rounding error)
  Resolution     = ###    (default is 300)
```

If you set the SplitText option to Yes, each text string is checked to see if it needs to be split into sections for printing. The SplitPercent value helps determine when a text string must be split into sections for printing.

The SplitPercent option sets the percentage of the width of the space character to use as the maximum amount of rounding error that can accumulate in a string before it is broken into sections.

The SplitPercent value is from zero (0) to 100. Do not enter a value greater than 100. For example, if you set the SplitPercent option to 75, the string is broken into sections if the accumulated rounding error is greater than 75% of the width of the space character. This value is set to 50 by default.

NOTE: Using 50 as the SplitPercent value is a good trade-off between the appearance and the performance impact on the GenPrint program and print spool size. Setting the SplitPercent option to a smaller value gives you a more accurate printout but slows the GenPrint program, increases the size of the print spool, and increases the amount of time it takes to print.

The Resolution option determines the rounding error. Most FXRs are built using 300 dpi fonts. This causes rounding errors when the FXR is used for printing to a 240 dpi printer. If you omit the Resolution option, the system uses the default setting of 300.

You need to know whether the FXR you are using was built by importing 300 dpi fonts or 240 dpi fonts. The standard FXRs are built using 300 dpi fonts. When an FXR is built using 300 dpi fonts, there are rounding errors when printing to a 240 dpi printer.

Here are some examples of options to use in different situations:

- If your font cross-reference (FXR) file was built from 300 dpi fonts and your printer resolution is 240 dpi, set the options as shown here:

```
< PrtType:AFP >
  SplitText      = Yes
  SplitPercent   = 50
  Resolution     = 240
```

- If your font cross-reference file was built from 240 dpi fonts and your printer resolution is 300 dpi, set the options as shown here:

```
< PrtType:AFP >
  SplitText      = Yes
  SplitPercent   = 50
  Resolution     = 300
```

- If your font cross-reference file was built from 300 dpi fonts and your printer resolution is 300 dpi, you do not need to set the SplitText option.
- If your font cross-reference file was built from 240 dpi fonts and your printer resolution is 240 dpi, you do not need to set the SplitText option.

INCLUDING DOCUMERGE FORM-LEVEL COMMENT RECORDS

You can include Documerge form-level comments in AFP print streams produced by Documaker. You may want to include form-level comments if you have a reprint utility program that needs information about a form before it can reprint it.

To include form-level comment records, add the `FormNameCR` option in your AFP printer control group and set it to `Yes`, as shown here:

```
< PrtType:AFP >
  FormNameCR      = Yes
  Module          = AFPPRT
  PrintFunc       = AFPPrint
  SendOverlays    = Yes,Enabled
  ...
```

Here is an example of the AFP records in an AFP print stream which includes the Documerge form level comment (NOP) records:

```
000, Begin, Document, 29,
001, Data, NOP, 84, %%DMGFORMBEG%% DEC PAGE          00001
AFP Docucorp 000001
002, Map, Medium Map, 16, PLUD
...
033, End, Page, 16,
034, Data, NOP, 84, %%DMGFORMEND%% DEC PAGE          00001
AFP Docucorp 000001
035, Data, NOP, 84, %%DMGFORMBEG%% LETTER            00001
AFP Docucorp 000002
036, Begin, Page, 16,
...
053, End, Page, 16,
054, Data, NOP, 84, %%DMGFORMEND%% LETTER            00001
AFP Docucorp 000002
173, End, Document, 16,
000, Begin, Document, 29,
001, Data, NOP, 84, %%DMGFORMBEG%% OP714             00001
AFP Docucorp 000001
002, Map, Medium Map, 16, PLUO
...
```


METACODE PRINTERS

The Metacode language is the native mode language for Xerox 4000 and 9000 series printers. This language is superior to printing using line data with Xerox Laser Printing Systems (LPS). The advantages of using Metacode over line data printing include support for portrait and landscape text on the same page, support for different fonts on the same line, precise text positioning, and text justification. In addition, Metacode lets you merge multiple forms onto a single page.

NOTE: All system print drivers support 24-bit color graphics. If your printer does not support color, the print driver will automatically convert the color graphics into monochrome graphics. Keep in mind that for the best performance you should avoid color graphics.

Required JSL INI Options

The system does not require you to use a special JSL on your printer to print its Metacode output. The Xerox Metacode printer driver is configurable based on options to produce Metacode which match your existing JSL settings. Here is an example of the PRTType:XER control group which contains these options:

```
< PRTType:XER >
  DJDEIden      = A'@@@DJDE'
  DJDEOffset    = 0
  DJDESkip      = 8
  OutMode       = BARR
  ImageOpt      = No
  CompressMode  = LIN
  JDEName       = META
  JDLCode       = NONE
  JDLData       = 0,255
  JDLHost       = IBMONL
  JDLName       = CBA
  PaperSize     = 0
  Device        = dummy.txt
  RelativeScan  = Yes
```

Several of these options are based on the comparable parameter values in the settings of the printer's JSL. A JSL may contain many JDLs from which to choose, or there may be multiple JSLs compiled into multiple JDLs.

A portion of a JDL may look like the following:

```
CBA:      JDL;
T1:       TABLE      CONSTANT=X'1212121212121212';
T2:       TABLE      CONSTANT=X'1313131313131313';
T3:       TABLE      CONSTANT=X'FFFF26FFFF';
C1:       CRITERIA     CONSTANT=(0,9,EQ,T1);
C2:       CRITERIA     CONSTANT=(0,10,EQ,T2);
C3:       CRITERIA     CONSTANT=(1,5,EQ,T3);
VOLUME    HOST=IBMONL;
LINE      DATA=(0,255);
IDEN      PRE=A'@@@DJDE',
          OFF=0,
          SKIP=8;
ROFFSET   TEST=C1;
RSTACK    TEST=C2,DELIMITER=YES,PRINT=NONE;
RPAGE     TEST=C3,SIDE=NUFRONT,WHEN=NOW;

/* 8.5 x 11 job */
USA1: JDE;          /* JOB can be used in place of JDE */
OUTPUT      PAPERSIZE=USLETTER;

/* 8.5 x 14 job */
META: JOB;
VOLUME      CODE=NONE

/* Default job */
DFLT: JDE;
VOLUME      CODE=EBCDIC
END;
```

Here are the required options which are based on settings in the printer's JSL file.

JDLName Represents the name of the JDL to use. The following table shows the relevant JSL statement for the earlier example and the proper option to use based on the JSL example.

JSL statement	CBA: JDL;
INI option	JDLName = CBA

JDEName Represents the name of the job to use. A JDL may contain many jobs (JDEs) from which to choose. This JDE must contain a *VOLUME CODE=NONE* statement. The following table shows the relevant JSL statements for the earlier example and the proper option to use based on the JSL example.

JSL statements	META: JOB; VOLUME CODE=NONE
INI option	JDEName = META

**DJDEIden, DJDEOffset,
and DJDESkip**

Represent the IDEN statement of the JDL. The value of the DJDEIden setting is a string constant. The types of string constants supported are ASCII (A'string'), EBCDIC (E'string'), Character ('string'), and Hex (X'string'). Octal, H2, and H6 strings *are not* supported.

Strings containing repeat counts, embedded hex values, and upper/lower case toggles *are not* supported. The following table shows the relevant JSL statements for the earlier example and the options to use based on the JSL example.

JSL statements	IDEN PRE=A'@@@DJDE', OFF=o, SKIP=8;
INI options	DJDEIden = A'@@@DJDE' DJDEOffset = o DJDESkip = 8

JDLCode

Represents the type of input format expected by the Xerox printer. Character translation occurs as necessary. Currently, the supported code types are *EBCDIC*, *ASCII*, *NONE* (same as ASCII), *BCD*, *H2BCD*, *H6BCD*, *IBMBCD*, and *PEBCDIC*. User-defined code translations are not supported.

Referring to the sample JSL, if the printer is normally started with STA DLFT,CBA then the JDLCode option must be set to *CODE = EBCDIC*. The system's option must contain the value of the CODE statement for the printer's normal operation. This table shows the relevant JSL statements for the earlier example and the proper option to use based on the JSL example.

JSL statements	DFLT: JDE; VOLUME CODE=EBCDIC
INI option	JDLCode = EBCDIC

JDLData

Represents the starting position and length of the print line data within an input data record. The LINE statement contains a DATA entry that holds these values. This table shows the relevant JSL statement for the earlier example and the proper option to use based on the JSL example.

JSL statement	LINE DATA=(o,255);
INI option	JDLData = o,255

JDLHost

Represents whether the printer is normally in an on-line or off-line state. Currently, the only values we accept for this option are *IBMONL* (on-line) and *IBMOS* (off-line). The following table shows the relevant JSL statement for the earlier example and the proper option to use based on the JSL example.

JSL statement	VOLUME HOST=IBMONL;
INI option	JDLHost = IBMONL

Additional Required INI Options

Below are the additional required options not based on the printer's JSL file.

OutMode

The OutMode option indicates the output format for the Metacode data stream generated by Documaker applications.

Use *BARR*, if the Metacode output is to be transmitted to the Xerox printer via BARR SPOOL hardware and software. When using the BARR setting, a length byte is placed at the start and end of each Metacode record.

Use *BARRWORD*, if the Metacode output is to be transmitted to the Xerox printer via BARR SPOOL hardware and software. BARRWORD should be used *only* if the Xerox printer can handle records longer than the 255 characters.

Use *PCO*, if the output is transmitted to the Xerox printer via PCO hardware and software (from Prism). When using the PCO setting, a 4-byte length field is placed at the start of each Metacode record.

NOTE: The PCO interface has not been tested, but should work.

Use *JES2*, if the Environment option is set to MVS.

Use *MRG4*, if you will transmit the Metacode output to the mainframe using Commcommander or if you will archive it in Docusave (see [Creating Print Streams for Docusave on page 358](#) for more information).

Use *LAN4235*, if the output is generated for a Xerox 4235 printer attached to a network.

Here is an example:

```
OutMode = BARR
```

ImageOpt

The ImageOpt option specifies if the graphics are being saved on Xerox printer as *IMG* files or as *FNT* files.

To use IMG files, the printer needs a special GVG or GHO hardware installed. Also, in the JSL you have to specify GRAPHICS = YES.

If you are using IMG files, vectors, in-line bitmaps or want to print charts, set the ImageOpt option to *Yes*; otherwise set it to *No*. Here is an example:

```
ImageOpt = No
```

If the system detects a problem when you are printing in-line bitmaps and vectors, it will display a message that tells you the type of graphic and image name. If the graphic is an in-line bitmap, it includes the name.

NOTE: Metacode printers have a limit of 16 IMG files on a page.

CompressMode

The CompressMode option compresses bitmaps output as inline graphics, such as charts and graphics with the inline graphics flag set. There are four compression modes available, which you can specify using the CompressMode option in the PrtType:XER control group:

- CompressMode = UNC

- CompressMode = ENC
- CompressMode = HTN
- CompressMode = LIN

UNC is the uncompressed or raw bitmap mode. If none is specified, the system defaults to *HTN* mode.

To demonstrate the effects of Metacode graphics compression, the following chart shows the GenPrint program run times and file sizes with the different compression options for a test environment containing in-line images.

Test	GenPrint time	File size
No charts (ImageOpt=No)	182 seconds (3:02)	697,599
UNC – uncompressed	309 seconds (5:09)	9,011,058
LIN compression	290 seconds (4:50)	1,589,226
ENC compression	301 seconds (5:01)	2,248,302
HTN compression	296 seconds (4:56)	1,831,050

Which compression method yields the smallest file size or the quickest compression time depends on the graphic bitmaps you are printing. In general, HTN or LIN compression provides the best results. HTN generally does best with graphics which contain more filled-in or shaded areas, while LIN performs better with graphics which contain more line art. Experiment with your sections to determine the best compression method.

The results of compression can be dramatic, as the table shows. The uncompressed print-ready file is over nine megabytes in size, while the compressed file size ranges from 18% to 25% of the uncompressed file. However, keep in mind that while the reduced file sizes save disk space and reduce transmission times, these files must be decompressed by the printer at print time, which is done automatically by the print controller.

CompileInStream

The CompileInStream option determines whether the FAP files have been loaded. If set to *Yes*, the print driver compiles the print stream using FAP files. Make sure the DownloadFAP option in the RunMode control group is set to *Yes*. If set to *No*, pre-compiled MET files are used.

The print driver creates the print stream using pre-compiled Metacode files. Use the FAP2MET utility to create pre-compiled Metacode files. The GenPrint program loads pre-compiled Metacode members from the PMETLIB PDS under z/OS. On other platforms, the PMetLib option specifies the directory which contains the pre-compiled MET files. If you do not set this option, the system uses the setting for the FormLib option in the MasterResource control group.

NOTE: To use FRM files in your Metacode print stream, set the CompileInStream INI option to No in the Xerox printer control group. Using FRM files enhances performance in high volume situations that use a repeated background form on every page.

Device This is the name of the file or device, such as LPT1, where the Metacode print stream should be written. This option is ignored by the GenPrint program but should not be left blank or omitted. For instance, you could enter *dummy.txt*.

RelativeScan When set to Yes, the RelativeScan option tells the system to consolidate all records in the print stream. When set to No, this option tells the system to omit Relative Scan records when consolidating records. If you are using GenPrint version 9.0 or higher you will probably want to leave this option at its default setting (Yes) for maximum optimization.

Specifying Installable Functions

For the Xerox print driver, you must specify the following set of installable functions in the PrtType:XER control group:

```
OutputFunc    = XEROutput
OutMetFunc    = XEROutMet
InitFunc      = XERInit
TermFunc      = XERTerm
Module        = XERW32
PrintFunc     = XERPrint
```

Using defaults for the Module and PrintFunc options

Default values for the Module and PrintFunc options in the PrtType:xxx control group are provided when you use a standard print type name or print class, such as AFP, PCL, PDF, PST, VPP, XER, XMP, or GDI.

These defaults keep you from having to enter the Module and PrintFunc names in your INI file. For example, if you want to generate XER print files, you can specify these INI options:

```
< Printer >
  PrtType      = MYXER
< PrtType:MYAFP >
  Class       = XER
```

And the system will default these options for you:

```
< PrtType:MYAFP >
  Module       = XERPRT
  PrintFunc    = XERPrint
```

Optional INI Options

Setting the end of the report

Use the JDLRStack option to set the criteria which signals an *end of report* condition to the printer. In the JDL sample listed earlier, the RSTACK statement performed a criteria test named C2. The C2 test checks a specific part of each input line against the string named T2. If the string T2 matches an input data record at position zero (o) for a length of 10 bytes, an *end of report* condition is signaled. Only CONSTANT criteria using an EQ operator are supported.

Setting the JDLRStack option is optional. If your printer is used for both Metacode and text file print jobs, you *must* set this option. Using the JDL sample listed earlier, the option should be:

JSL statements	<pre>T2: TABLE CONSTANT=X'13131313131313131313'; C2: CRITERIA CONSTANT=(o,10,EQ,T2); RSTACK TEST=C2,DELIMITER=YES,PRINT=NONE;</pre>
INI option	JDLRStack = o,10,EQ,X'13131313131313131313'

Starting new pages

Use the JDLRPage option to set the criteria which signals a *jump to front side of a new sheet* to the printer. In the JDL sample listed earlier, the RPAGE statement performed a criteria test named C3. The C3 test checks a specific part of each input line against the string named T3. If the string T3 matches an input data record at position zero (o) for length of 5 bytes, a *jump to new sheet* condition is signaled because of the *SIDE=NUFRONT* statement. Only CONSTANT criteria using an EQ operator are supported. For the JDLRPage option to work properly, the *SIDE=NUFRONT* and *WHEN=NOW* statements must be used as a part of the RPAGE settings in the JSL file.

Setting the JDLRPage option is optional. If the print job contains duplex pages alternating with simplex (one-sided) pages, this option provides a way to leave blank the backsides of certain pages. Using the JDL sample listed earlier, the option should be:

JSL statements	<pre>T3: TABLE CONSTANT=X'FFFF26FFFF'; C3: CRITERIA CONSTANT=(1,5,EQ,T3); RPAGE TEST=C3,SIDE=NUFRONT,WHEN=NOW;</pre>
INI option	JDLRPage = 1,5,EQ,X'FFFF26FFFF'

The Metacode print driver automatically places the *SIDE=NUFRONT* statement on all front pages when operating in duplex mode. This lets the system support print stream sorting facilities such as Mobius InfoPak. Also, the *SIDE=NUBACK* statement is now added to blank back pages when in duplex mode.

These statements eliminate the need for the ADDPAGES utility which some systems used with Mobius InfoPak support. Without this functionality the first page of an output may print on the back of a previous output.

You will need to add the *SIDE=NUFRONT* statement on all front pages printed, not only those pages that specify a tray change. This is necessary to handle the end of job condition where the last page prints on the front and is moved by InfoPak.

Also, the system will now add a `SIDE=NUBACK` statement for pages that start on the back side of the page, leaving the front side blank.

NOTE: You cannot configure these statements. The system automatically enters them into the print stream. You do not need to add `SIDE=NUFRONT` and `SIDE=NUBACK` statements to your Xerox printer control group (PrtType:XER).

Adding an OFFSET command

Prior to version 11.3, the first Metacode print stream the system produced would include this statement:

```
DJDE SIDE=NUFRONT,END
```

while the remaining print streams the system produced would include this statement:

```
DJDE SIDE=(NUFRONT,OFFSET),END
```

This means the first Metacode print stream will not have a statement which includes the `OFFSET` command.

If your printer requires the `OFFSET` command to be in all statements, including the first `DJDE` statement, add the `DJDEForceOffsetEnd` option to your INI file, as shown here:

```
< PrtType:XER >
CodeDef          = dcascii9
Device           = X.MET
DJDEIden         = E'$XEROX'
DJDEOffset       = 0
DJDESkip         = 8
DJDEForceOffsetEnd = Yes
```

Option	Description
DJDEForceOffsetEnd	<p>Enter Yes to make sure there is an <code>OFFSET</code> command in every <code>DJDE</code> statement, including the <code>DJDE</code> statement for the first print stream.</p> <p>The default is No, which omits the <code>OFFSET</code> command from the <code>DJDE</code> statement in the first Metacode print stream.</p> <p>Only set this option to Yes if you must include the <code>OFFSET</code> command for your printer. Most printers do not require <code>OFFSET</code> in the first <code>DJDE</code> statement.</p>

Logging pages

Use the `JDLROffset` option to set the criterion that tells the printer to initiate a page offset in the output bin. This option has not been fully implemented.

In the JDL sample, the `ROFFSET` statement performed a criteria test named `C1`. The `C1` test checks a specific part of each input line against the string named `T1`. If the string `T1` matches an input data record at position zero (0) for length of 9 bytes, a page offset is initiated. Only `CONSTANT` criteria using an `EQ` operator are supported.

Setting the `JDLROffset` option is optional. Using the JDL sample listed earlier, the option should be:

JSL statements

```
T1: TABLE    CONSTANT=X'1212121212121212';
C1: CRITERIA  CONSTANT=(0,9,EQ,T1);
ROFFSET TEST=C1;
```


INI option

```
JDLROffset = 0,9,EQ,X'1212121212121212'
```

You can also jog form sets by transaction instead of by batch. In some situations, this can make manual assembly easier. To do this, set the `OffsetLevel` option to `Formset`, as shown here:

```
< PrtType:XER >
  OffsetLevel = Formset
```

This adds an additional 'OFFSET' parameter to the `SIDE=NUFRONT` command, which tells the printer to jog after each transaction.

Specifying spot color

Use the `PrinterInk` option to specify the color of ink loaded on a Xerox highlight color printer. You can set this option to one of the following colors:

Blue	Red	Green	Ruby	Violet	Brown
Gray	Cardinal	Royal	Cyan	Magenta	

Blue is the default if you omit this option. This option is used with the `SendColor` option. If you set the `SendColor` option to `Yes`, be sure to also set the `PrinterInk` option. Here is how you would specify cyan as the color of the ink stored on the printer:

```
PrinterInk = cyan
```

Chart performance and print quality

By default, charts are rendered at 150 dpi (dots per inch) in a Metacode print stream. This setting typically provides for a smaller print stream and optimal performance from the GenPrint program.

Charts are scaled by the printer to their proper size and are printed as 300 dpi bitmaps. Because fewer dots are used at these lower dpi settings, you may notice some loss of detail in the printed output and effects such as:

- The circle which makes up the pie chart is less precise
- The lines used in a chart are thicker

Test charts printed to see if the loss of detail is acceptable. In general, horizontal and vertical lines scale with little or no loss of precision. Arcs and diagonal lines may lose some detail.

To disable rendering charts at 150 dpi, add the following option to the Xerox printer control group, usually named `PrtType:XER`:

```
ChartResolution = 300
```

The only other acceptable value for this option is 150. This option does not affect graphics printed as inline graphics.

Optimizing Metacode print streams

The GenPrint program lets you produce optimized Metacode print streams. You may want to consider using optimization if your Metacode output causes the printer to cycle down (wait) while printing.

This condition can occur when Metacode records cannot be transferred fast enough to the printer. Optimization helps remedy this situation by combining Metacode print records into larger and fewer records. Reducing the number of records that must be transmitted reduces the amount of time needed to spool the Metacode print stream to the printer. The cost is decreased GenPrint performance. You can also use the METOPT utility to optimize normal (non-optimized) Metacode output. For more information on this utility, see the [Docutoolbox Reference](#).

To have the GenPrint program produce optimized Metacode output streams, add this FSISYS.INI option to have the GenPrint program sort and consolidate records to create more efficient print streams:

```
< PrtType:XER >
    Optimize = Yes
```

The Optimize option defaults to No, which tells the GenPrint program to run without sorting and consolidating records.

You can enable some extra error checking during optimization. If optimization encounters critical errors, such as the inability to find or open a file, it will notify you and stop immediately. It can report actual or potential non-critical problems it encounters while it runs. For instance, if optimization finds Metacode records that may prevent the file from printing, it can warn you.

To have optimization notify you if it spots potential problems, add the following option to your PrtType:XER control group:

```
< PrtType:XER >
    ValidLevel = 0 (default)
```

Enter zero (0) to tell the utility not to report non-critical problems. Enter one (1) to tell the utility to report warnings for non-critical problems, but continue optimizing. Enter two (2) to tell the utility to report warnings for non-critical problems and attempt to fix the problems. Enter three (3) to tell the utility to report warnings for non-critical problems and exit immediately.

Regardless of the option you choose, if you receive any warnings, be sure to closely check both the original and, if applicable, the optimized file.

Using a common font list

The METOPT utility and the Metacode print driver let you use common font lists at the beginning of a Metacode print stream. A common font list names all of the Xerox fonts that will be used by the print job.

By knowing all of the fonts up front, the Metacode driver can issue a single DJDE FONTS command once at the beginning of the job and avoid issuing DJDE FONTS commands on subsequent pages. This helps some Metacode printers print jobs at their highest rated speed.

In the CommonFonts control group, you will see a list of options similar to these:

```
< CommonFonts >
    Names = 28
    Name1 = FORMSX
    Name2 = FXUNBD
    Name3 = FXUNN6
    Name4 = FXCON6
    Name5 = FXUNN8
    Name6 = FXUNN0
    Name7 = FXUNBH
    ...
    Name28 = FXUNI0
```

The first option, Names, defines the number of font name entries that follow. The following options specify the Xerox fonts which will be used in the print job.

NOTE: The format used for the CommonFonts control group is the same as that used by Documerge. Therefore, if you used this in Documerge, you can copy that INI control group into your Documaker INI file.

To use common font lists, you must use the METOPT utility or use the Metacode print driver and have the following INI options in the Xerox print group:

```
< PrtType:XER >
  Optimize = Yes
  MaxFonts =
```

Option	Description
Optimize	To use common font lists, set this option to Yes.
MaxFonts	Set this option to the maximum number of fonts your printer can handle in a single DJDE command. This number will vary based upon the printer's memory and configuration. The maximum value is 99 and the default is 20.

If the number of fonts in your common font list exceeds the MaxFonts value, the system outputs the MaxFonts number of fonts in the DJDE FONTS command. The DJDE FONTS command will contain the names of the fonts used on that page plus additional fonts from the common fonts list until the MaxFonts number of fonts is reached.

If the system encounters a page that uses a font not specified in the common fonts list (or the prior DJDE FONTS command to be more precise), it issues a new DJDE FONTS command which appends to the common font list the new fonts for that page.

Setting a default paper size

Use the PaperSize option to set a default paper size when converting Metacode print streams using the Internet Document Server or the MRG2FAP utility.

```
< PrtType:XER >
  PaperSize = 0
```

Enter	Description
zero (0)	for letter size (default)
1	for legal size
2	for A4 size
3	for executive size
98	for a custom size

Automatically sizing sections

You can have the system automatically size FAP files converted from Metacode files, (usually Documerge EDL members). This lets you create the FAP files as custom sized sections that are the minimum size required to contain all of the converted objects from the Metacode file.

To have the system automatically size the FAP files, include this INI option in the Xerox printer group you are using to convert the Metacode file:

```
< PrtType:XER >
```

```
AutoSize = Yes
```

If you omit this option, the system creates full page size sections.

Keep in mind...

- The system will not automatically size the section if the converted Metacode file results in a multi-page section.
- If the section is automatically sized and the result is a custom sized section, the Metacode loader does not try to determine if the section is landscape and does not rotate landscape objects.

Inline graphic performance and print quality

Graphics at 75, 100, or 150 dpi, printed using inline graphics, are scaled by the printer to their proper size and printed as 300 dpi bitmaps. Because fewer dots are used at these lower dpi settings, you may notice some loss of detail in the printed output and effects such as:

- Arcs and circles are less precise
- The lines used in a graphic are thicker

Test LOG files printed as inline graphics to see if the loss of detail is acceptable. In general, horizontal and vertical lines scale with little or no loss of precision. Arcs and diagonal lines may lose some detail.

To avoid scaling inline graphic LOG files, use Documaker Studio or Logo Manager to scale your graphics to 300 dpi. Most graphics are normally 300 dpi and most graphics are not printed as inline graphics.

Adding color to charts

Use the ColorCharts option to print the graphic portion of the chart in color.

```
ColorCharts = Yes
```

This option is used with the SendColor and PrinterInk options.

Using named paper trays

By default, Metacode output specifies the main tray for pages that use Tray 1. The AUX tray is specified for all other trays. If you have named trays in your JSL, specify these named trays in your options. An example of this option is shown here:

```
Tray1 = ONE1  
Tray2 = TWO2  
Tray3 = THREE3  
Tray4 = FOUR4
```

Specifying the printer model

Use the PrinterModel option to specify the particular printer model you are using. There may be subtle differences between printer models that can affect the output sent to the printer. Currently, only the 3700 printer requires this setting. An example of this option is shown here:

```
PrinterModel = 3700
```

Specifying the resolution

Use the Resolution option to specify the printer's dots per inch resolution. Currently, only 300 dpi is supported, which is also the default.

```
Resolution = 300
```

Displaying console messages

Use the OTextString option to display a message on the printer console. The text you specify is sent before the print job starts. For example, this lets you display the message, *Put BLUE paper in tray 1* before a print job starts. Here is an example:

```
OTextString = "Put BLUE paper in tray 1"
```

The system also supports multiple OTEXT messages in the Metacode print driver at a print batch level. Additionally, the system lets OTEXT messaging generate multiple messages per print batch. To turn on multiple OTEXT messaging, add this option to the FSISYS.INI file

```
< PrtType:XER >
MultipleOText = Yes
```

The default is No.

This tells the system to ignore the OTextString value in the PrtType control group and instead use the ones found in the appropriate print batch group.

For example, if you have three print batches, called BATCH1, BATCH2, and BATCH3, under each separate batch group, put required number of sequential messages for that batch:

```
< BATCH1 >
...
OTextString1 = "Batch 1 OText String1"
OTextString2 = "Batch 1 OText String2"
OTextString3 = "Batch 1 OText String3"
< BATCH2 >
...
OTextString1 = "Batch 2 OText String1"
OTextString2 = "Batch 2 OText String2"
OTextString3 = "Batch 2 OText String3"
< BATCH3 >
...
OTextString1 = "Batch 3 OText String1"
OTextString2 = "Batch 3 OText String2"
OTextString3 = "Batch 3 OText String3"
***
```

Keep in mind that the index tags OTextStringX (where X is a number) must start with one (1) and be sequential. The system stops writing OTEXT records to the batch when it finds a tag that is out of sequence. Here is an example:

```
OTextString1 = "Batch 3 Otext String 1"
OTextString3 = "Batch 3 Otext String 3"
```

In this example, only the first one would display on the screen, because OTextString2 is not encountered next.

Stapling forms

Some Metacode printers include a stapling feature. The system supports this feature, but it has not been tested and is not warranted.

Using this feature, forms printed on certain Metacode printers can be stapled if you specify a StapleJDEName option in the PrtType control group. This causes a new JDE to be specified on forms that need to be stapled.

It is assumed that the Staple JDE option has the same settings as the normal JDE specified except for the additional STAPLE command. You specify which forms should be stapled using the Form Set Manager, which is part of Docucreate or via Documaker Studio.

This option only affects implementations which print to Metacode printers with the optional stapling feature. For more information on this feature, see the [Docucreate User Guide](#). An example of this option is shown here:

```
StapleJDEName = JDESTP
```

Duplex switching

In earlier versions of the system, a Metacode print stream began and continued as a simplex job until the system encountered a page that needed to be duplex. At that point, the duplexing option was turned on. From that point forward, the print stream remained in duplex mode. For performance reasons, the system did not switch out of duplex mode. Research showed that for most cases, this was the most efficient way to drive the printer.

If, however, you are directing the printer output stream to a *value-added* process, you may want to include the actual duplex selection information with each form set. Without the commands to specify the duplex state, some value-added processes may not work properly. By setting the DJDELevel option to **Formset**, each form set will include a duplex command which specifies either simplex or duplex mode (DJDE DUPLEX=YES or NO always appears at the beginning of every new form set). A value other than *Formset* causes the duplex commands to be output as before. Here is an example:

```
DJDELevel = Formset
```

Using VSAM to store resources

The system lets you store DDT files, precompiled Metacode resources, NA and POL files, and transaction trigger files in VSAM KSDS (Virtual Storage Access Method/Key Sequence Data Set) data sets. If you use this feature, you must set the following options in the VSAM control group in the FSISYS.INI file:

```
< VSAM >
DDTVSAM = DD:DDTVSAM      DDT files
METVSAM = DD:PMETVSAM     PreCompiled Metacode files
VSAMRCPTB = DD:SETRCPVS   Transaction Trigger file
VSAMNA = DD:NAFILE        NA and POL files
```

For more information on implementing VSAM support under z/OS, see [Optimizing Performance](#) in the [Documaker Server Installation Guide](#).

PrintViewOnly

If set to Yes, this option tells the system to print the view only sections. The default is No. This does not apply to entry only sections, which are never printed. Entry only sections are usually worksheets. If the section is marked as hidden and view only, it will not print.

Caching files to improve performance

The following options let you minimize the opening and closing of frequently used PDS members by retaining, or caching, file handles and file data. In many cases the default values are sufficient, but for specific cases in which you use many different sections, you may need to increase these caching values to improve performance.

Here are the options you can customize:

```
< Control >
CacheFAPFiles =
RuleFilePool =
LogCaching =
CacheMethod =
```

Option	Description
CacheFAPFiles	Specifies the number of FAP files to keep available for re-use without re-loading them from disk. The default is 100.
RuleFilePool	Specifies the number of DDT files to keep available for re-use without re-loading them from disk. The default is 100.
LogCaching	Enter No if you do not want the system to log caching statistics. The default is Yes.
CacheMethod	Use to set the type of caching method. You can choose from <i>LFU</i> (least frequently used), <i>LRU</i> (least recently used), or <i>LFUO</i> (least frequently used optimized). LFUO is the default.

MET files contain pre-compiled Metacode information produced by the FAP2MET utility. The GenPrint program loads MET members from the PMETLIB PDS under z/OS. On other platforms, the PMetLib option specifies the directory containing the pre-compiled MET files.

If not set, the system uses the setting for the FormLib option in the MasterResource control group. The CacheFiles option keeps frequently used MET members available for re-use. This option is placed in the PrtType:XER control group in the FSISSYS.INI file, as shown here:

```
< PrtType:XER >
  CacheFiles = 100    (default is 100)
  InitFunc   = XERInit
  TermFunc   = XERTerm
```

Caching statistics for FAP files, DDT files and Xerox resources such as pre-compiled Metacode files (PMETs) and forms (FRMs) are collected and can be placed in the LOGFILE.DAT file. These statistics show the following information:

Item	Description
Method	The caching method you are using (LFUO, LFU, or LRU).
Size	The size of the caches. The default is 100.
Hits	The number of times the system tried to load a resource from the cache and found it there.
Misses	The number of times the system tried to load a resource from the cache and did not find it there.
Total	The combined hits and misses. This represents the number of times the system tried to load a resource from the cache.
Purges	The number of times the system had to remove a resource from the cache to put another resource into the cache. The system decides which resource to remove based on the method. If you are using LFUO or LFU, the least frequently used resource is removed. If you are using LRU, the least recently used resource is removed.

Using the loader

The system lets you load print-ready Metacode files. For this feature to work, the print-ready Metacode file must have the same extension as the Ext option in the Loader:MET control group in the FAPCOMP.INI file. Here is an example:

```
< Loader:MET >
  Desc      = Xerox Metacode (*.MET)
  Ext       = .MET
  LoadFunc  = XERLoadMet
  Module    = XERW32
< Loaders >
  Loader    = MET
```

Along with the Metacode loader feature, another INI option is required in the PrtType:XER control group. The DefaultFont option defines the default font to use to indicate the names of any graphics in the print-ready Metacode file.

The graphics do not display in Image Editor when the print-ready Metacode file is opened. Instead the name of the graphic appears, in the default font, and the space taken by the graphic is indicated. In addition, the default font is also used for displaying any text that references a font not present in the font cross-reference file.

To set the default font, enter the name of a Xerox font file contained in the font cross-reference file as shown here:

```
< PrtType:XER >
  DefaultFont = FXTIN8
```

If there are any graphics in the MET file, the system requires a LOGO.DAT file in the FormLib directory so it can display graphics properly for all rotations. The LOGO.DAT file, which is a semicolon-delimited file, should look similar to this:

```
[file name for 0° rotation];[file name for 90° rotation];[file name for 180° rotation];[file name for 270° rotation];
```

Here are a few points to keep in mind when using this feature:

- The PrtType settings must match the setting used to produce the print-ready Metacode file.
- Rotated text will not display properly.
- Blank pages are created for simplex forms printed in duplex mode.
- This feature slows the printing of large print-ready files (more than 100 pages).
- If there is a reference to a FRM file in the MET file, the system cannot display the MET file.
- The system cannot display charts and graphics.

Using the Class option

You can use the following INI option to specify the printer classification, such as AFP, PCL, XER, PST, or GDI. If you omit this option, the system defaults to the first three letters from the Module option.

```
< PrtType:XER >
  Class = XER
```

Some internal functions expect a certain type of printer. For instance, all 2-up functions require an AFP printer. The internal functions check the Class option to make sure the correct printer is available before continuing.

Adding user-defined DJDE statements

You can place the AdditionalDJDE option anywhere in the PrtType:XER control group. Each AdditionalDJDE value represents a distinct and separate DJDE statement, given verbatim. You can include as many AdditionalDJDE statements as needed. All of the located AdditionalDJDE statements are inserted into the print stream. You can also specify the batch in which to output the DJDE statement. Here is an example:

```
< PrtType:XER >
  AdditionalDJDE = "BATCH1";FEED=COVER,;
  InitFunc      = XERInit
  ...
  AdditionalDJDE = "BATCH1";STOCKS=BLUE,;
  ...
  AdditionalDJDE = JDL=DPLJDL,JDE=STRTON,;
```

The first two occurrences only apply to the BATCH1 batch. The third occurrence has no batch specified, so this DJDE statement is written to all print batches.

Keep in mind that these user-defined DJDE statements are placed after the BEGIN DJDE record and before the other DJDEs that are always inserted, such as FONTS. Make sure the DJDE syntax is correct and that the new DJDE records do not interfere with the ones automatically inserted into the print stream by the system.

Also, it is very important that you follow the correct syntax when coding the INI line. If you enter an invalid batch name, no corresponding batch will be found and the DJDE line will be ignored or not output in any batch. And, if the DJDE syntax is incorrect, the printer will issue error messages or unpredictable print results may occur.

Using third-party software to read Metacode files

If you use third-party software to read Documaker-produced Metacode files and that software needs the DJDE, RSTACK, and RPAGE commands to begin with a carriage control value other than the default value of *oxo1*, you can use the DJDECarrControl option to handle this. You simply enter a value in the form of a string constant. These string constants are supported:

- ASCII (A'string')
- EBCDIC (E'string')
- character ('string')
- hex (X'string')

NOTE: The character string produces an EBCDIC string, same as E'string'.

The default value is 1 (X'01'). Here is an example:

```
< PrtType:XER >
  DJDECarrControl = X'09'
```

Keep in mind that any carriage control value will be accepted and no attempt is made to make sure a valid carriage control is used.

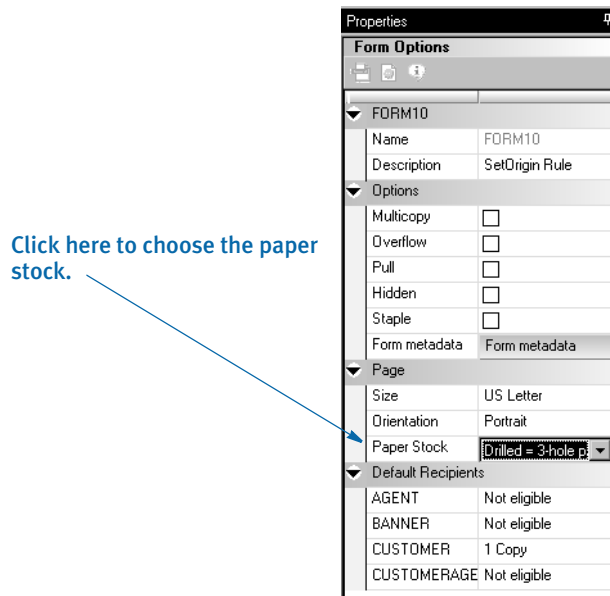
Specifying the paper stock

Using Documaker Studio you can specify what paper stock the form should print on. This will help users who have more than nine types of paper stocks. Here is an example of the INI options you could set up:

```
< PaperStockIDs >
  PaperStockID = Drilled
```

```
PaperStockID    = 201b
...(and so on)
< PaperStockID:Drilled >
  Description    = 3-hole paper
< PaperStockID:201bW >
  Description    = 201b White Paper
  DJDE           = DJDE name
```

Once you have set up the appropriate PaperStockID options, you will see those options available via Studio's Form manager. Just open a form and select the appropriate paper stock in the Paper Stock field on the Properties tab, as shown here:



Your selection is reflected in the POL file produced by the GenData program. In this example, the form called DEC PAGE has a paper stock ID of *Drilled*.

```
;SAMPCO;LB1;DEC
PAGE;||FORMPAPERSTOCK=Drilled||;R;;QPRUNA|DL(3360,18600)<AGENT,COMP
ANY,INSURED>
```

In the Metacode printer control group, you must set the TrayUsePaperStockID option to Yes, as shown here:

```
< PrtType:XER >
  TrayUsePaperStockID = Yes
```

If the TrayUsePaperStockID option is set to Yes, the Metacode print driver takes the form's PaperStockID and tries to find the DJDE INI option for it in the INI file when it emits the tray command.

Keep in mind...

- The paper stock selection applies to the entire form
- Only the Metacode print driver uses the paper stock selection
- Only Documaker Studio lets you select the paper stock

USING MOBIUS METACODE PRINT STREAMS

You can use Mobius to archive Metacode print streams and also use Docupresentment to retrieve archived Metacode print streams and produce or present PDF files.

You can retrieve the archived Metacode print streams using Mobius' ViewDirect APIs. The ViewDirect APIs are built to communicate with the Mainframe Mobius Archive via TCP/IP. If you license the Mobius' ViewDirect APIs, you can write a custom rule to retrieve your archived Metacode print streams.

To do this, include these options in your FAPCOMP.INI file (for Image Editor) or your FISISYS.INI file (for Studio and the MRG2FAP utility):

```
< PrtType:XER >
  OutMode = MOBIUS
< Loader:MOBIUS >
  Desc    = Mobius Metacode files (*.MET)
  Func    = XERLoadMobius
  Module  = XEROS2
< Loaders >
  Loader  = MOBIUS
< Control >
  Mobius  = XER
```

Where *XER* is the printer control group that contains the Mobius Metacode information.

To use the Mobius Metacode loader in Docupresentment, use the same MTCLoadFormset rule you would use to load a Documerge Metacode print stream.

To specify a Mobius Metacode print stream, instead of a Documerge print stream, the Xerox printer control group must include this INI option:

```
< PrtType:XER >
  OutMode = MOBIUS
```

Metacode print streams retrieved from a Mobius archive have a special record blocking scheme and use special comment records to indicate the fonts used. This version adds support for reading Metacode print streams retrieved from a Mobius archives.

Use XERLoadDocuMerge as the loader function. It checks for an OutMode setting of MRG2, MRG4, or ELIXIR. You must add MOBIUS to the list of allowed OutMode settings and you must add your Mobius comment checking to XERLoadMet, when the OutMode option is set to *MOBIUS*.

NOTE: The loader functions convert a particular type of file, such as a PCL print stream, a Metacode print stream, an RTF file, and so on, into an internally formatted file. Once converted, the system can then do a variety of things with that file, like display it in Studio, print it on a supported printer, or save it as another type of file, such as a FAP file, RTF file, or a print stream file.

The loader included in this version can also be used in other Documaker products. For instance, Studio can use it to load Mobius Metacode, then display, modify, and save the result as a FAP file or print to a supported printer. It can also be used by the METDUMP utility to dump information about the Mobius Metacode print stream.

METACODE PRINTER RESOURCES

A number of resources are used in the printing process. These resources generally reside on the printer's disk drive.

Fonts Xerox fonts are ASCII fonts. Xerox fonts are not scalable and do not rotate. There is one font file for each rotation and different files are required for different sizes. The file extension is *FNT* and file names are up to six characters long. Oracle Insurance has licensed for use and distribution with its systems, fonts from Monotype Imaging, Inc. Xerox fonts for all four rotations are included.

Forms Xerox forms are precompiled electronic files containing static text, boxes, graphics, and so on, ready to be merged with variable data. Forms always have the extension *FRM*. Like fonts, the maximum file name is six characters. You use the FAP2FRM utility to create Xerox forms from FAP files.

Images Xerox images are large bitmaps or raster patterns that are stored in a special file format. These images are merged onto the forms which are then merged with the variable data. The file extension is *IMG* and the maximum file name is six characters.

NOTE: You must install a GVG hardware card on the printer to print IMG files. You can use the LOG2IMG utility to create Xerox images from LOG files. For more information on this utility, see the [Docutoolbox Reference](#).

Logos Logos are small bitmaps stored in a different format than IMG files. The extension is *LGO* and the file name is six characters long. You can only use Xerox logos inside a FRM file. You cannot invoke them directly in the data stream.

NOTE: These *LGO* files are quite different than the graphics (*LOG*) files used in Documaker Studio and Logo Manager. Documaker software does not use Xerox LGO files.

METACODE LIMITATIONS

Xerox images The maximum number of images and inline graphics per page is 16.

HMI support HMI (horizontal motion index) is supported for zero (o) and 270 degree rotated text on portrait forms only. HMI combines separate text labels which are positioned on the same line and which use the same font into a single Metacode record. FAP files with justified paragraphs can benefit from this feature. Use the FAP2MET utility to implement HMI into pre-compiled MET files.

Changing the paper size on the 4235 printer You can not easily change paper sizes in one job. Each job is controlled by a JDE. If you need to pull paper from bins of different sizes, you have to call a different JDE each time you change from one paper size to another. This is similar to staple support. There is no code to invoke different JDEs for change of paper size.

Xerox forms

If a Xerox form (FRM file) contains more than 48 blocks (each block is 512 bytes), your printer may not have enough memory to print it.

The CD (Character Dispatcher) memory is divided into three regions. The first region loads all fonts used on a page. The second region is used for TL/DLs which contain inline Metacode (may only be variable data if you use an FRM). The third region loads the TL/DLs from an FRM file, if one is being used for the page.

If you have version 2 of the printer software, your printer supports eight TL/DL buffers of 3K each (same as 48 blocks of 512 bytes each) for inline Metacode. With version 3.5 of the printer software, the limit was increased to 16 buffers of 3K each.

NOTE: Our testing shows that with version 3.5, TL/DLs from FRMs (the third region of CD memory) are still limited to 8 TL/DL buffers of 3K each (same as 48 blocks of 512 bytes each).

Typically, Xerox 9700 and 9790 printers still have the older release installed. If so, you may want to upgrade to version 3.5. The Xerox 4000 series printers (4050, 4850, and so on) always come with version 3.5 or higher.

When you are not using FRM files in a print stream, the system does not use the CD memory reserved for FRM files.

METACODE TROUBLESHOOTING

Unexpected color output

Even though you set the SendColor option set to *No*, you still get color output when printing. This occurs when:

- You specified *Print in color* for some elements of the FAP file
- You precompiled the FAP files with the */C* option on FAP2MET
- A *SUB INK* command was issued on the printer

If ink substitution occurred because of an operator command, such as *SUB INK BLUE* (or *RED* or *GREEN*), the colored components of the precompiled MET file will be brought in with color attribute turned on and printed with color. This happens regardless of how you set the SendColor option. To print in black and white, either re-run the FAP2MET utility with no */C* flag, or use the *END* command to cancel ink substitution on the printer.

Unexpected black and white output

Even though you set the SendColor option to *Yes*, you still get black and white output when printing. Use this checklist to make sure you have done everything to print in color:

- Make sure you specified *Print in color* for the color elements in your FAP file, such as text, shaded areas, lines, and so on.
- Make sure you precompiled the FAP files with the */C* option on FAP2MET;
- If you are using precompiled FAP files, make sure you compiled those FAPs using the FAP2CFA utility.
- Make sure you run the GenPrint program with the SendColor option set to *Yes*.

Highlight color should match the PrinterInk option

The PrinterInk option causes a DJDE ILIST command which specifies the highlight color to use. If a different highlight color is installed on the printer, the printer follows the procedure specified in the ABNORMAL statement in the JDL and JDE loaded. The ABNORMAL procedure specifies whether the job should continue, abort, or stop. If no ABNORMAL procedure is declared, the default is for the printer to stop so a new ink cartridge can be loaded. Besides the ABNORMAL statement, the printer operator can override the ink setting using the SUB command (for example, SUB INK BLUE or SUB INK CURRENT).

LOG file orientation

To print a portrait section which contains a graphic on a landscape form using pre-compiled MET files, set the LoadFAPBitmap option to Yes. This is necessary because the graphic name must change from the portrait (zero degrees) name to the landscape (270 degrees) name.

Output catching up with the input

If your printer cycles down and displays a message stating that the output caught up with the input, it indicates the average number of records per physical page is greater than the maximum number of records that can be transferred across the channel in the time allowed for a page.

This situation causes the printer to cycle down so it can buffer more pages before it continues. This table shows the maximum average number of records that can be transferred across the channel in time to support the printer running at rated speed:

Printer	Maximum Records Per Page
4050	285
4090	155
DP96	149
41/4635	105
DP180	78

To resolve this problem, you need to optimize the Metacode print stream. For more information, see [Optimizing Metacode print streams on page 271](#).

Printing rotated variables

Here is a list of field options you can specify in the NAFILE.DAT file:

Option	Description
E	Error
M	Manual
P	Protected
G	Global scope (entire form set)
F	Form scope
H	Hidden field (such as a dummy field, neither displayed nor printed)

Option	Description
N	Nonprintable field (displayed, not printed)
C	Send-copy-to field (receives current recipient name at print time)
9	Rotated 90 degrees
8	Rotated 180 degrees
7	Rotated 270 degrees

NOTE: For legacy MRLs, some of these options require the FAP field attributes to be available at runtime because the DDT file does not include the necessary information. You can use the CheckImageLoaded rule to make sure this information is available.

There are no DDT files in MRLs created using Documaker Studio

Multi-page sections

When you use multi-page FAP files and pre-compiled MET files, you must use the EjectPage rule. This rule enables the printing of multi-page sections. Here are the steps to apply the rule in Image Editor:

- 1 Open the FAP file in Image Editor.
- 2 Select Format, Image Properties and then click the Load DDT button.

Image Editor detects that the image contains multiple pages and inserts into the DDT file as many EjectPage rules as there are pages.

You must have a variable field on each page. The variable field can be a dummy field that is hidden.

NOTE: Documaker Studio automatically handles EjectPages for you.

When you implement multi-page FAP files and pre-compiled MET files, keep these requirements in mind:

- Only multi-page FAP files are applicable.
- Multi-page FAP files cannot be mixed with single page FAP files on the same form. The system cannot easily determine the page number in this case.
- The multi-page FAP file came from Documaker Studio or Image Editor and therefore there is only one section per page, hence, each page on the form has an section list that contains one and only one section.
- The index of the page on which that section resides within that form is the number of the page.
- Multi-page sections can be duplexed by setting the form to either *Front* (long edge binding) or *Short* bind (short edge binding). Internally created sections will be set to *Rolling* for the remaining pages.

NOTE: If a form begins with a rolling duplex option, the print drivers begin printing on the blank back page of the previous form. Any form that starts with rolling and begins a form set is treated as the front page of a rolling set.

Operator command,
FEED, causes duplex
problems

If you enter an operator command to specify an input tray—because for instance, one paper tray is empty and while you refill it you want the printing to continue using another tray—you can no longer select trays from DJDEs in the job stream. Instead, you will get messages which tell you tray selection was suspended by an operator override.

All paper feed from that point forward, will be from the tray specified in the operator command. This can cause duplex jobs to print incorrectly if you have completed printing on a front page and the next page should print from a different tray.

To correct this situation, enter a *FEED=MAIN* command. This command tells the printer to switch to tray 1 and enables tray selection through DJDE commands so the next paper selection command is obeyed.

Line density errors

As the speed of the printer increases, there is less and less time available to the character dispatcher to form the scan line and send it to the image generator. Here is some information on how this affects the various Xerox printers:

- Since the 4135 printer is the fastest of the Xerox printers using the older CD/IG, the chances of running out of time and causing a *line density error* are greatest with this model.
- The Xerox 4050 and 4850 printers are too slow for this to be a problem. These printers allow more fetches from the font memory per scan line.
- The Xerox 4635 printer's image generation module has been revamped to such an extent that Xerox almost guarantees there will never be a line density exceeded error on a 4635 printer.
- The 4235 printer is slow and works quite differently than the centralized printers.
- If a job works fine on a 9790 printer but fails on a 4135, the number of character fetches is likely on the borderline of failure.

If you experience line density problems, check your FAP files for the following:

- Text superimposed on shaded areas.
- Large number of text lines with small fonts.
- Large number of horizontal lines whose thickness is measured in an odd number of dots. If you change the thickness of a horizontal line from three dots to two or four dots (0.01" to .006666" or 0.013333" — 24 FAP units to 16 or 32 FAP units), it reduces the character count from two to one.

The Xerox line drawing font has three horizontal line drawing characters which specify lines with thicknesses of two, four, and eight dots (.006666", .013333" and .026666" or 16, 32, and 64 FAP units). Odd thicknesses require the printer to overlay or overlap multiple lines.

- Large number of small boxes, many of which have common boundaries. On paper it looks like one line. Actually, there may be two or more character fetches for the same black dots. Create these kinds of boxes by drawing lines rather than boxes.

Output data length validation

Metacode printer JSL specifies the length of data that can be received. This data length must match the value output into the Metacode print stream. You specify the data length in the JSL as shown here:

```
LINE DATA = (0,213)
```

You specify the data length in the PrtType:XER control group in the FISISYS.INI file, as shown here:

```
< PrtType:XER >
JDLData = 0,213
```

In this example, the JSL specifies a maximum data length of 213, so the INI option has a matching value. The maximum length value is also used in the Metacode print driver to make sure no more than the specified amount of data is output in any Metacode record. If the amount of data to be emitted in the record exceeds this amount, an error message such as the following appears:

```
Record Length 214 is too long - maximum length is 213.
```

NOTE: Under z/OS, Metacode output files are VB datasets. The JCL specifies a maximum length of a record (LRECL). If an attempt is made to write a record longer than the LRECL value, the write will fail and an error message appears.

Be advised that under z/OS, with VB datasets, the LRECL size includes a 4-byte record length, known as the *RDW*. The RDW is implicitly added to the front of each variable length record. Therefore, you should set the LRECL value for the Metacode output dataset to a number equal to the JSL maximum length plus four to account for the RDW bytes at the front of the record. For the above example, set the LRECL of the Metacode output file to 217.

Using Xerox Forms (FRMs)

The system lets you use Xerox form (FRM) resources when you print to Xerox Metacode printers. FRMs are printer resident resources that contain static full-page images. The system can use FRMs during the print process.

You can convert frequently used static full-page images into FRMs using the FAP2FRM utility. To indicate an image is resident on the printer as a FRM file, use the Form Set Manager. The Printer Resident field indicates the image is a pre-compiled resource resident on the printer—as opposed to a pre-compiled resource that needs to be downloaded to the printer. For more information on the Form Set Manager and the Printer Resident field, see the [Docucreate User Guide](#).

Here are some guidelines for using Xerox forms (FRMs):

- Create one FAP file per page. If there is a text area, do not put variable data within the text area.
- The image size must be one of the standard paper sizes, such as US Letter, Legal, A4, or Executive.
- Because Xerox printers can only accept file names up to six characters in length, the image name can be up to six character long. If, however, it is a multi-page FAP file, the name can consist of no more than four characters to accommodate the two-character number added by the FAP2FRM utility. Here are some examples: *TEST01.FRM* for the first page, *TEST02.FRM* for the second, and so on.

- Use the FAP2FRM utility to convert FAP files into FRM files. For multi-page FAP files, create multiple FRM files. The names are appended with two-digit numeric suffixes.
- On workstations, store the FRM files in the same directory as the FAP files. On z/OS, keep them in a PDS attached to the PFRMLIB DD name. On AS400 systems, use the FRMFile option in the Data control group to specify to store the FRM files.
- Use the Printer Resident field in the Form Set Manager to mark individual forms as *printer resident*. After you do this, the FORM.DAT file contains the V image option which indicates the image is resident on the printer. When you run the GenPrint program, a DJDE FORMS=*fname* command is inserted for the corresponding images. The remainder of the images are printed by inline Metacode, possibly using precompiled MET files.
- Install the FRM files on the Xerox printer using the XERDNLD utility. Copy the resulting *.DAT files to the printer. To make sure the forms are installed on the printer, use the SAMPLE console command to print the form files.

BARRWRAP

The BARRWRAP utility converts Metacode output from JES2 format into BARR format.

The BARR interface attachment for Metacode printers requires that the Metacode print stream files contain BARR specific information. The BARRWRAP utility adds this information to an existing Metacode print stream file, which lets you print the output file via the BARR interface.

After you run the utility on a Metacode file, *76 1A FF 00* is added at the beginning of the file. This tells BARR the file is a Metacode file. A byte denoting the record length is also added at the beginning and end of each record in the file.

Use this utility when you test the GenPrint program on z/OS. If the z/OS system is not directly channel-attached to the Xerox printer, you must download the print streams to an OS/2 system—use no ASCII translation, but do use CRLF. Then, using BARRWRAP, the print stream is packaged to successfully pass through BARR/SPOOL.

NOTE: Occasionally, the binary data contained in a Metacode file has a sequence of hex bytes (*x'oDoA'*) which could be misinterpreted as a carriage return/line feed. This is true particularly for charts and other inline graphics. Convert such data streams using the BARRWRAP utility on the z/OS platform before you download them with the no ASCII and no CRLF (binary) options.

Transferring Files from Xerox Format Floppies

Resources saved on a 5 1/4-inch floppy, using *FLOPPY SAVE file.ext*, are saved in a special Xerox format. For use in the system, or for transferring to a 4235 printer, you must convert these resources into DOS format. You can use the following software packages to perform this required conversion:

- FloppyCopy by Lytrod Software – Inexpensive, easy to use
- Elixir – More expensive, but includes additional features.
- LaserLinx – No longer marketed.

PCL PRINTERS

Hewlett-Packard created the Printer Control Language (PCL) to provide a way for application programs to control a range of printer features across a wide array of printing devices.

The PCL language has evolved over time. For the most part, system-produced PCL output will run on any printer that supports PCL 5 or PCL 6. There are separate drivers for these two versions of the PCL language.

To support color printing, the printer must support PCL 5c, which contains color extensions. To support more than two paper trays, the printer must support PCL 5e.

NOTE: All system print drivers support 24-bit color graphics. The PXL (PCL 6) driver supports monochrome, 8-bit color (256 color), and 24-bit color graphics.

If your printer does not support color, the print driver will automatically convert the color graphics into monochrome graphics. Keep in mind that for the best performance you should avoid color graphics.

PCL INI OPTIONS

You must define the necessary printer options for the GenPrint program to produce PCL output. These options specify PCL output and are located in a *PrtType:xxx* control group, such as *PrtType:PCL* for PCL 5 or *PrtType:PXL* for PCL 6. Common PCL printer options are shown below, with default values in bold:

Option	Values	Description
Device	any file or device name	The name of the file or device (LPT1) where the PCL print stream should be written. This setting is ignored by the GenPrint program but is used by Studio, Image Editor, and other Documaker system programs.
Module	PCLW32	The name of the program module which contains the PCL print driver. See also the Class option. For PCL6, enter PXLW32 . See also Using defaults for the Module and PrintFunc options on page 268 .
PrintFunc	PCLPrint	The name of the program function that is the main entry point into the PCL print driver. For PCL6, enter PXLPrint . See also Using defaults for the Module and PrintFunc options on page 268 .
Resolution	300	The dots per inch resolution of the printer which will receive the PCL data stream.
SendOverlays	Yes/No	Set to Yes if you created PCL overlays for each FAP file. This option is not supported for PCL 6.

Option	Values	Description
OverlayPath	any directory	<p>Set to the directory containing the PCL overlays for each FAP file. The default is the FormLib option of the MasterResource control group. Here is an example:</p> <pre>< MasterResource > FormLib = <CONFIG:Batch Processing> FormLib = <CONFIG:Batch Processing> FormLib = ./forms/</pre> <p>This option is not supported for PCL 6.</p>
OverlayExt	any file extension (OVL)	<p>The file extension of the PCL overlays. This option is not supported for PCL 6.</p>
PageNumbers	Yes/No	<p>Set to Yes to enable form or form set page numbering.</p>
SendColor	Yes/No Enabled/ Disabled/ Hidden	<p>Set to Yes to enable color printing.</p> <p>Enabled = Option appears in the Print window and is active (available to be checked).</p> <p>Disabled = Option appears in the Print window but is grayed out (not available to be checked).</p> <p>Hidden = Option does not appear in the Print window.</p>
HighlightColor	Yes/No	<p>Set this option and the SendColor option to Yes to use simple color mode. See Using Simple Color Mode on page 297 for more information. This option is not supported for PCL 6.</p>
DownloadFonts	Yes/No	<p>Set to Yes to enable downloading of PCL fonts. For PCL6, you must enter Yes because internal font selection is not supported.</p>
TemplateFields	Yes/No	<p>Set to Yes to test print Xs in variable fields.</p>
FitToWidth	Yes/No	<p>Not supported by either PCL print driver.</p>
AdjLeftMargin	Yes/No	<p>Automatically adjusts the left margin to compensate for the 1/4-inch left margin added by PCL printers.</p> <p>Yes = Automatically adjust the left margin. Forms print exactly as they appear on screen (default).</p> <p>No = Do not adjust the left margin. Forms may not print correctly on PCL printers after performing a retrieve function.</p> <p>This option is not supported for PCL 6.</p>

Option	Values	Description
SelectRecipients	Yes/No Enabled/Disabled/Hidden	Set to <i>No</i> to disable the ability to select recipients. Enabled = Appears in the Print window and is active (available to be checked). Disabled = Appears in the Print window but is grayed out (not available to be checked). Hidden = Does not appear in the Print window.
PrintViewOnly	Yes/ <i>No</i>	If set to Yes, the view only sections will print. This does not apply to entry only sections, which are never printed. Entry only sections are usually worksheets. If the section is marked as hidden and view only, it will not print.
PrePrintedPaper	Yes,Disabled	Determines if the check box which lets you print or not print pre-printed objects appears on the Print window. Also determines the default for this field — checked or unchecked. You must add this option to the INI file if you want the field to appear on the Print window. The default includes the field on the Print window and leaves it unchecked. All objects except fields can be marked pre-printed on the object's Properties window.
Class	<i>(first three characters of the Module option)</i>	Specifies the printer classification, such as AFP, PCL, XER, PST, or GDI. If you omit this option, the system defaults to the first three letters from the Module option. Some internal functions expect a certain type of printer. For instance, all 2-up functions require an AFP printer. The internal functions check the Class option to make sure the correct printer is available before continuing.
StapleBin		Set this option to the PCL printer escape sequence that selects the bin that contains the staple attachment. Use a tilde character (~) in place of the binary escape character. This option is not supported for PCL 6.
PJLCommentScript		To add PJI comments to a PCL print stream, enter the name of the DAL script you want the system to run. This DAL script creates the control strings and adds them as ASCII comments. This option is not supported for PCL 6.
PJLCommentOn	batch /formset	Use this option to add PJI comment records to the beginning of every form set or batch. This option is not supported for PCL 6.

Option	Values	Description
OutputBin		Enter the printer escape sequence to select the normal output bin (for non-stapled forms) if non-stapled forms are being sent to the wrong bin. This option is not supported for PCL 6.

NOTE: The default FAPCOMP.INI file should include the PrtType:GDI control group and options in addition to the PrtType:PCL or PrtType:PXL control group.

Using defaults for the Module and PrintFunc options

Default values for the Module and PrintFunc options in the PrtType:xxx control group are provided when you use a standard print type name or print class, such as AFP, PCL, PDF, PST, VPP, XER, XMP, or GDI.

These defaults keep you from having to enter the Module and PrintFunc names in your INI file. For example, if you want to generate PCL print files, you can specify these INI options:

```
< Printer >
  PrtType      = MYPCL
< PrtType:MYAFP >
  Class        = PCL
```

And the system will default these options for you:

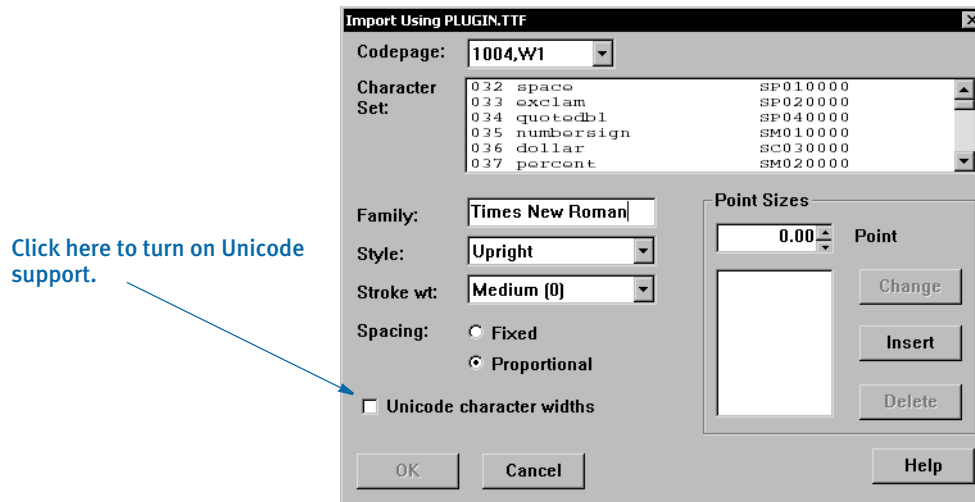
```
< PrtType:MYAFP >
  Module       = PCLPRT
  PrintFunc    = PCLPrint
```

Using PCL 6

PCL 6 is a stack-based protocol (similar to PostScript) composed of attributes and operators that let you define paths, clip paths, pens, brushes, fonts, raster patterns, and so on. PCL6 also supports 16-bit character codes which makes it a better choice for supporting Unicode than PCL 5.

The PCL 6 driver lets you download both PCL bitmap fonts and TrueType fonts. You must specify the TrueType font file name in the Font File entry of the PCL printer section in the font cross-reference (FXR) file.

To turn on Unicode support, check the Unicode Character Widths field when you insert a TrueType font into the FXR file. Unicode support lets you use additional characters and languages supported by the TrueType font.



Keep in mind...

- The PCL 6 driver supports PCL bitmap fonts so you can use master resource libraries (MRLs) designed for PCL 5. Just remember to make the appropriate changes to your INI options.
- When printing using a TrueType font, only the characters used on the form are downloaded into the print stream. This reduces the size of print stream files, particularly if the TrueType font includes support for Asian languages.

In comparison to the PCL 5 printer driver, the PCL 6 driver has these limitations:

- No overlay support
- No support for a separate downloadable font file which contains multiple PCL fonts
- No internal printer font support
- Less paper tray support, no INI options to specify which PCL commands to use
- No INI options to specify PCL commands to output bin or staple bin
- No highlight color support
- No comment script support

Printing Under Windows

Windows XP/2000 does not recognize printer ports such as LPT1. If you are using Windows XP/2000, you must change the PrtType control group in the FSIUSER.INI file to reflect the print server name and print device. Here is an example:

```
< PrtType:PCL >
Device = \\FSISRV03\\OPTRA1
```

Using High-Capacity Trays 3 and 4 on HP 5SI Printers

The system defines document attributes in a device-independent fashion. In prior versions, PCL support was based on options available to PCL 5 and similarly configured printers. The newer HP 5SI printer offers additional capabilities which depend upon (at least somewhat) commands that exist in PCL 5e. To add to the confusion, HP is not always consistent with its own terminology. Here is how the system treated PCL in prior versions:

NOTE: The ability to define trays or use the Tray# option is not supported for PCL 6.

System term	PCL command	PCL term	HP 4 term	HP 4si term	HP 5si term
Tray 1 (Main)	~&l1H	Tray 2 (upper)	PC Tray	Upper tray	Tray 2 (upper drawer)
Tray 2 (Aux)	~&l4H	Tray 3 (lower)	MP	Lower tray	Tray 3 (lower drawer)
Tray 3 (Man)	~&l2H	Manual feed	Tray 1	Manual feed	Tray 1 (manual side feed)
Tray 4 (Env)	~&l3H	Envelope feed	Tray 1	Manual feed	Tray 1 (manual side feed)
n/a	~&l5H	HCI, first tray	LC Tray	n/a	First tray of HCI
n/a	~l20H	HCI, second tray	n/a	n/a	Second tray of HCI

The terms for the current version are shown below, with changes highlighted:

System term	PCL command	PCL term	HP 4 term	HP 4si term	HP 5si term
Tray 1 (Main)	~&l1H	Tray 2 (upper)	PC Tray	Upper tray	Tray 2 (upper drawer)
Tray 2 (Aux)	~&l4H	Tray 3 (lower)	MP	Lower tray	Tray 3 (lower drawer)
Tray 3	~&l5H	HCI, first tray	LC Tray	n/a	First tray of HCI
Tray 4	~l20H	HCI, second tray	n/a	n/a	Second tray of HCI

The command ~&l5H (first high-capacity tray) is supported by PCL 5, but the hardware is not typically found on HP printers. The command ~&l20H requires PCL 5e.

You can use these INI options:

```
< PrtType:PCL >
  Tray1 = pcl command sequence (default is ~&l1H)
  Tray2 = pcl command sequence (default is ~&l4H)
  Tray3 = pcl command sequence (default is ~&l5H)
  Tray4 = pcl command sequence (default is ~&l20H)
```

Keep in mind the paper size overrides the tray selection.

NOTE: See also for [Handling Multiple Paper Trays on page 363](#) more information.

If you depend on the prior sequence, you can return to the original operation by specifying:

```
< PrtType:PCL >
  Tray3 = ~&l2H
  Tray4 = ~&l3H
```

NOTE: The tilde (~) represents the escape character and is translated internally. The third character in each sequence shown is a lowercase *L*.

Using a staple attachment

In your PCL printer group, usually PrtType:PCL, add the StapleBin option to use a staple attachment on your PCL printer.

Set the StapleBin option to the PCL printer escape sequence that selects the output bin which contains the staple attachment. Use a tilde (~) in place of the binary escape character.

Here is an example:

```
~&l2G (tilde, ampersand, lower case l, 2, upper case G)
```

This example shows the escape sequence used to select an optional lower (rear) output bin that may have a staple attachment. Check with your printer manual for the escape sequence you should use.

The OutputBin option should contain the printer escape sequence needed to select the normal output bin (for non-stapled forms). Using the OutputBin option is not necessary unless you notice the non-stapled forms are being sent to the wrong output bin. This INI option is only necessary when you have both stapled and non-stapled forms in the same print batch.

Overriding Paper Size Commands and Tray Selections

You can include additional PCL 5 printer commands which you can use to override both the paper size and the tray selection. For instance, you can use this technique to get an envelope feeder to work.

NOTE: Before the release of version 11.1, you could only specify the PCL 5 command for the system to emit when a form is specified to use a certain paper tray (for more information, see [Using High-Capacity Trays 3 and 4 on HP 5SI Printers on page 294](#)).

When you include a PCL paper (page) size command, the system does not emit its own paper (page) size PCL command based on the form's page size. This lets you use a page size the system does not support.

For example, suppose you want to print on #10 business envelopes ($4\frac{1}{8}$ inch by $9\frac{1}{2}$ inch) using an optional envelope feeder on your PCL printer. The PCL command to select a paper (page) size for printing COM-10 (Business $4\frac{1}{8}$ x $9\frac{1}{2}$ inches) is shown here:

```
~&l81A
```

The PCL command to feed an envelope from an optional envelope feeder is shown here:

```
~&l6H
```

If your system contained a form for printing on an envelope and the form was specified to print from tray 4, you would use this INI setting:

```
< PrtType:PCL >
...
Tray4 = ~&l81A~&l6H
```

Because some characters are hard to distinguish, refer to this table for an explanation of the characters shown for the Tray4 field, in order:

Character	Description
~	A tilde
&	An ampersand
l	A lowercase L
8	The numeral eight (8)
1	The numeral one (1)
A	An uppercase A
~	A tilde
&	An ampersand
l	A lowercase L
6	The numeral six (6)

Character	Description
H	An uppercase <i>H</i>

When writing PCL commands as an INI setting, the tilde (~) is used as a substitute for the PCL escape character (x1B).

The PCL 5 Technical Reference manual contains information on PCL commands used to select paper trays and paper sizes. You can get a copy of the PCL 5 Technical Reference manual by going to www.hp.com and entering the phrase *PCL technical reference* in the search window.

NOTE: When printing envelopes, you may want to design your form (section) in landscape mode. When printing on PCL printers, there are unprintable margins on the left/right edge of 1/4 inch and top/bottom edge of 1/6 inch. These unprintable margins apply when printing envelopes. Remember to account for these unprintable margins when designing your form (section).

Using Simple Color Mode

The PCL print driver supports PCL simple color mode in addition to full RGB color support. PCL simple color mode uses a 3-plane CMY palette. The 3-plane CMY palette contains these indexed colors:

- 0 - White
- 1 - Cyan
- 2 - Magenta
- 3 - Blue
- 4 - Yellow
- 5 - Green
- 6 - Red
- 7 - Black

To specify highlight color printing for PCL, include these INI options:

```
< PrtType:PCL >
  SendColor = Yes
  HighlightColor = Yes
```

Marking objects to print in color

For any object, such as lines, boxes, or text, select the Print in Color option on the Color Selection window if you want the object to print in a color other than black. Keep in mind...

- If the object is black and is not marked as Print in color, the system prints the object using a black color index.
- If the object has a color other than black and is marked as Print in color, the system prints it using a highlight color index.
- Charts print in black, although you can print chart labels in the highlight color.

Specifying the highlight color to use

You can use these INI options to specify the PCL color commands to use for printing the black and highlight colors. The default values are shown here:

```
< PrtType:PCL >
  HighlightColorCmd = ~*v3S
  HighlightBlackCmd = ~*v7S
```

Note that the tilde (~) is used in place of the PCL escape character (hex 1B) and that the number used in the command corresponds to the color indexes specified earlier, such as 3=Blue and 7=Black.

To use a different highlight color, include the HighlightColorCmd option. To use a different black color, specify the HighlightBlackCmd option.

Printing on different types of printers

Printing black and white, highlight color, and full color print streams on black and white, highlight color, and full color PCL printers will produce varying results.

You can usually send a color PCL print stream to a black and white PCL printer without any problem—everything comes out black and white. PCL printers usually ignore any commands they do not understand.

If, however, you send a highlight color PCL print stream to a full color PCL printer, the results may be slightly different than if you had sent the print stream to a highlight color printer.

Bitmap graphics in a highlight color print stream may print as cyan on a full color printer. Bitmaps are a sequence of binary data—zeros (0) and ones (1)—so the zeros may print as white, while the ones may print as cyan. On a highlight color printer, the bitmap is printed as expected using the black or highlight color.

If you send a full color PCL print stream to a highlight color printer, your results may vary based on the printer model and printer settings.

Creating Compressed PCL Files

You can create compressed PCL files using Documaker. This capability is typically used with IDS because Windows does not let you print files that are a mixture of simplex and duplex pages from Acrobat. The whole document has to be printed the same way. IDS, however, lets you print a file to a local PCL printer which preserves the file's duplex information.

Use these options, which call the PRTZCompressOutputFunc function, to compress an output file, such as a PCL print batch file:

```
< PrtType:PCL >
  OutputMod = PRTW32
  OutputFunc = PRTZCompressOutputFunc
```

NOTE: The output is compressed, regardless of the file's extension. You must decompress the file before you can print it.

Bitmap compression

The PCL print driver also supports bitmap compression. To disable bitmap compression, add the following INI option to the PCL printer control group:

```
< PrtType:XXX >
  Compression = No
```

Adding Printer Job Level Comments

Printer Job Language (PCL) comments are supported by some PCL printers (not PCL 6). One type of command lets you add a comment to your PCL print stream. The PJL comment does not affect printing but can pass information to other products that look for specific information in PJL comment records, such as an imaging system.

NOTE: Imaging products can be used to archive PCL print streams. These products often require a control record at the beginning of the PCL print stream. These options and DAL functions let you create that control record.

To add PJL comments, add the following INI option to the PCL print group:

```
< PrtType:PCL >
    PJLCommentScript = imaging.DAL
```

The PJLCommentScript option specifies the DAL script you want to run. This DAL script creates a control string and adds it as an ASCII comment. Here is an example of the DAL script:

```
* Add imaging comment - use default APPIDX record.
Comment = AppIdxRec( )
AddComment(Comment,1)
Return('Finished!')
```

Notice the use of the second parameter to the AddComment DAL function. The **1** indicates the string should be an ASCII string. If you omit this parameter, the system converts the string into an EBCDIC string.

You can also use the PJLComment option to tell the system to add PJL comments to the beginning of every form set or print batch. Here is an example:

```
< PrtType:PCL >
    PJLCommentScript    = imaging.DAL
    PJLCommentOn        = formset
```

Adding Data for Imaging Systems

The PCL print driver can add free form text or data at the beginning of a batch or each form set within the batch. This can help you interface with imaging systems such as RightFax.

Use the TEXTScript INI option to specify the DAL script you want to run. This DAL script creates a free form data or text buffer and adds it to the print stream.

Here is an example of the DAL script:

```
* Populate the PCL stream comment with these values from RCBDFFD
faxnum = trim(GVM('FaxNumber'))
faxname = trim(GVM('FaxName'))

AddComment('<TOFAXNUM:' & faxnum & '>',1)
AddComment('<TONAME:' & faxname & '>',1)

Return
```

Notice the use of the second parameter to the AddComment DAL function. The 1 indicates the string should be an ASCII string. If you omit this parameter, the system converts the string into an EBCDIC string. You can also use the TEXTCommentOn option to tell the system to add free form text or data to the beginning of every form set or print batch. Here is an example:

```
< PrtType:PCL >
  TEXTScript      = imaging.DAL
  TEXTCommentOn   = formset
```

Limiting the Embedded PCL Fonts

When using the PCL Print Driver with the GenPrint program, you can force the PCL Print Driver to create print streams that contain only those fonts used in the document. To have the PCL Print Driver only embed the fonts used, include these INI options in your PCL printer control group:

```
< PrtType:PCL >
...
InitFunc      = PCLInit
TermFunc      = PCLTerm
DownloadFonts= Yes
...
```

PCL PRINTER RESOURCES

A number of resources are used in the printing process. These resources reside in directories specified in the MasterResource control group.

Fonts The system supports PCL bitmap fonts. These fonts reside in the directory specified in the FontLib option in the MasterResource control group when you set the DownloadFonts option to Yes. The system includes utilities for creating PCL fonts from PostScript, TrueType, Xerox, or AFP fonts.

Overlays Use the OVLCOMP utility to create PCL overlays from FAP files. These overlays must reside in the directory specified in the OverlayPath option in the MasterResource control group when you set the SendOverlays option to Yes.

NOTE: Because the PCL 6 driver supports PCL bitmap fonts, you can use master resource libraries (MRLs) designed for PCL 5. Just remember to make the appropriate changes to your INI options.

POSTSCRIPT PRINTERS

Adobe Systems created the PostScript language. It is an interpretive programming language with powerful graphics capabilities. For the most part, system-produced PostScript output will run on any printer that supports PostScript Level 2.

NOTE: The PostScript print driver supports monochrome, 4-bit, 8-bit, and 24-bit color bitmaps. If your printer does not support color, the print driver will automatically convert the color graphics into monochrome graphics. Keep in mind that for the best performance you should avoid color graphics.

POSTSCRIPT INI OPTIONS

You must define the necessary printer-related options for the GenPrint program to produce PostScript output. These options specify PostScript output and are located in a `PrtType:xxx` control group, such as `PrtType:PST`. Common PostScript printer options are shown below, with default values in bold:

Option	Values	Description
Device	any file or device name	The name of the file or device (LPT1) where the PCL print stream should be written. This setting is ignored by the GenPrint program but is used by Documaker Studio, Image Editor, and other system programs.
Module	PSTW32	The name of the program module which contains the PostScript print driver. See also the Class option. See also Using defaults for the Module and PrintFunc options on page 304 .
PrintFunc	PSTPrint	The name of the program function that is the main entry point into the PostScript print driver. See also Using defaults for the Module and PrintFunc options on page 304 .
Resolution	300	The dots per inch resolution of the printer which will receive the PostScript data stream.
SendOverlays	Yes/ No	Set to Yes if you have created PostScript overlays for each FAP file. See also Creating Smaller PostScript Output on page 305 .
DSCHeaderComment		Use to specify PostScript Document Structure Convention (DSC) comments you want added to the header portion of the generated PostScript print stream. You can include as many DSCHeaderComment options as are necessary. See Adding DSC Comments on page 306 for more information.

Option	Values	Description
OverlayPath	any directory	<p>Set to the directory which contains the PostScript overlays for each FAP file. The default is the FormLib option of the MasterResource control group.</p> <p>Instead of using the above control groups and options, you could use the following options:</p> <pre>< MasterResource > OverlayPath = <CONFIG:Batch Processing> OverlayPath = < CONFIG:Batch Processing > OverlayPath = .\PstOvl\</pre> <p>The default is the FormLib directory pointed to by the FormLib option in the MasterResource control group., as shown here:</p> <pre>< MasterResource > FormLib = <CONFIG:Batch Processing> FormLib = < CONFIG:Batch Processing > FormLib = ./forms/</pre>
OverlayExt	any file extension (OVL)	The file extension of the PostScript overlays.
PageNumbers	Yes/ No	Set to <i>Yes</i> to enable form or form set page numbering.
SendColor	Yes/ No Enabled/ Disabled/ Hidden	<p>Set to <i>Yes</i> to enable color printing.</p> <p>Enabled = Option appears in the Print window and is active (available to be checked).</p> <p>Disabled = Option appears in the Print window but is grayed out (not available to be checked).</p> <p>Hidden = Option does not appear in the Print window</p>
DownloadFonts	Yes / No	<p>Set to <i>Yes</i> to enable downloading of PostScript fonts.</p> <p>See also Creating Smaller PostScript Output on page 305.</p>
PrinterModel	file name (omit extension)	Contains the name of the PostScript Printer Definition (PPD) file. This file contains information about printer-specific features. This file must be in the directory specified by the DefLib option of the FMRES control group.
TemplateFields	Yes / No	Set to <i>Yes</i> to test print Xs in variable fields
FitToWidth	Yes / No	Not supported by the PostScript print driver

Option	Values	Description
PrintViewOnly	Yes/No	If set to Yes, the view only sections will print. This does not apply to entry only sections, which are never printed. Entry only sections are usually worksheets. If the section is marked as hidden and view only, it will not print.
PrePrintedPaper	Yes,Disabled	Determines if the check box which lets you print or not print pre-printed objects appears on the Print window. Also determines the default for this check box—checked or unchecked. You must add this option to the INI file if you want the check box to appear on the Print window. The default for this option includes the checkbox on the Print window and leaves it unchecked. All objects except fields can be designated as pre-printed on the object's Properties window.
Class	<i>(first three characters of the Module option)</i>	Specifies the printer classification, such as AFP, PCL, XER, PST, or GDI. If you omit this option, the system defaults to the first three letters from the Module option. Some internal functions expect a certain type of printer. For instance, all 2-up functions require an AFP printer. The internal functions check the Class option to make sure the correct printer is available before continuing.
LanguageLevel	Level1 Level2	Level2 is the default setting and is required for complex printing tasks, such as duplexing, tray selection, and so on. Only use Level1 if your printer only supports PostScript Level 1 language features.
StapleOn StapleOff	see description	These options work in a similar fashion to the Tray# options which let you specify PostScript commands directly as a quoted string or to look up the PostScript commands to use in your printer's PPD file. For detailed information, see Stapling Forms on page 308 .
SelectRecipients	Yes/No Enabled/Disabled/Hidden	Enabled = Option appears in the Print window and is active (available to be checked). Disabled = Option appears in the Print window but is grayed out (not available to be checked). Hidden = Option does not appear in the Print window.

Option	Values	Description
SetOverprint		<p>Enter Yes if you are using a highlight color printer, such as the Xerox DocuTech/DocuPrint 180 Highlight Color printer, and you want to remove the white outline that appears around black letters printed on a highlight color background.</p> <p>If you are using pre-compiled overlays, be sure to re-create the overlays after you set this option to Yes.</p> <p>If you still see a small white outline around the characters in your printed output, your printer may need to be re-calibrated. Contact your printer vendor to fine tune your printer calibration.</p>

Using defaults for the Module and PrintFunc options

Default values for the Module and PrintFunc options in the PrtType:xxx control group are provided when you use a standard print type name or print class, such as AFP, PCL, PDF, PST, VPP, XER, XMP, or GDI.

These defaults keep you from having to enter the Module and PrintFunc names in your INI file. For example, if you want to generate PST print files, you can specify these INI options:

```
< Printer >
  PrtType      = MYPST
< PrtType:MYAFP >
  Class        = PST
```

And the system will default these options for you:

```
< PrtType:MYAFP >
  Module       = PSTPRT
  PrintFunc    = PSTPrint
```

Avoiding a white outline around letters

On some highlight color printers, such as the Xerox DocuTech/DocuPrint 180 Highlight Color printer, if you print black text on a colored shaded area, the black text is printed with a white outline around the letters. To eliminate the white outline, add the SetOverprint option to your PostScript printer INI control group and set it to Yes.

Printing under Windows

Windows XP/2000 does not recognize printer ports such as LPT1. Change the PrtType control group in the FSIUSER.INI file to reflect the print server name and print device. Here is an example:

```
< PrtType:PST >
  Device = \\FSISRV03\\OPTRA1
```

Generating PostScript Files on z/OS

You can generate PostScript output files on z/OS systems with an updated (version 11.0 or later) PSTLIB. Be sure to include these settings in your FSISYS.INI file to print PostScript on z/OS:

```
< Printer >
  PrtType = PST
< PrtType:PST >
  Module = PSTW32
  Printfunc = PSTPrint
  SendOverlays = (Yes or No)
  SendColor = (Yes or No)
  DownloadFonts = (Yes or No)
```

Creating Smaller PostScript Output

The PostScript print driver automatically downloads (embeds) only the fonts that are needed. This results in smaller output files.

NOTE: To produce a PostScript print stream that only downloads (embeds) the minimum set of fonts required by the PostScript print stream, you *cannot* use overlays.

All PostScript fonts referenced in the FXR file are downloaded if the SendOverlays option is set to Yes because the system does not know which fonts are used by the overlays.

You must set these PostScript INI options as shown to tell the PostScript print driver to download the minimum set of fonts required by a print stream:

```
< PrtType:PST >
  DownloadFonts = Yes
  SendOverlays = No
```

If you are running the GenPrint program, you will need to tell GenPrint to load the FAP files (instead of overlays) by using the DownloadFAP option:

```
< RunMode >
  DownloadFAP = Yes
```

Bitmap compression

The PostScript print driver supports bitmap compression. Compression is enabled by default. To disable compression, add this option to the PostScript printer control group:

```
< PrtType:XXX >
  Compression = No
```

Color bitmaps are compressed in JPEG format.

Monocolor bitmaps are compressed using Run Length Encoding (RLE) compression. If compression or color is disabled, 4-bit and 8-bit color bitmaps are printed as monochrome bitmaps. For compatibility with previous releases, 24-bit color bitmaps are printed in color when compression is disabled and color is enabled.

PostScript print streams with bitmap compression are often smaller and may be produced faster than PostScript print streams without bitmap compression. PostScript print streams with compressed multi color bitmaps will see the greatest reduction in terms of file size and time to produce.

The 4-bit and 8-bit color bitmaps printed in color with compression will likely produce larger print streams than 4-bit and 8-bit color bitmaps which have been converted to monocolored (black and white) bitmaps.

Keep in mind:

- For any bitmap to print in color, you must make sure the bitmap (LOG) is marked as *Print in Color* in the FAP file. Also make sure you set the SendColor option to Yes in the PCL or PostScript printer control group before printing.
- When using Forms Integrity Manager (FIM) to compare a version 11.2 or later PostScript print stream with bitmap compression against an older PostScript print stream without bitmap compression, FIM will report that some bitmaps are not identical. Older PostScript print streams without bitmap compression generated the bitmap data in multiple streams while the newer compressed bitmaps are always generated within a single stream. In this case, FIM will report the older print streams contains multiple *Overlay Images* entries while the new print streams contain a single *Overlay Images* entry. Also, FIM may report differences in some attributes (height, width, raster size, and so on) of *Overlay Images* and *Variable Images* due to differences in how bitmaps are emitted.

Adding DSC Comments

Use the DSCHeaderComment option to specify the PostScript Document Structure Convention (DSC) comments you want added to the header portion of the generated print stream. You can include as many DSCHeaderComment options as are necessary.

This example shows how, in addition to specifying PostScript commands in the Tray# options, you can also include DSC comments you want added to the header portion of the generated PostScript print stream:

```
< PrtType:PST >
  Device = test.ps
  DownloadFonts = Yes,Enabled
  DSCHeaderComment = %%DocumentMedia:Media1 612 792 75 (White)
(Tray1)
  DSCHeaderComment = %%+ Media2 612 792 75 (White) (Tray2)
  DSCHeaderComment = %%+ Media3 612 792 75 (White) (Tray3)
  DSCHeaderComment = %%+ Media4 612 792 75 (White) (Tray4)
  LanguageLevel = Level2
  Module = PSTW32
  PageNumbers = Yes
  PrinterModel = XDP92C2
  PrintFunc = PSTPrint
  Resolution = 300
  SendColor = No,Enabled
  Tray1 = "<< /MediaType (Tray1)/MediaColor(White) /MediaWeight
75>>
          setpagedevice"
  Tray2 = "<< /MediaType (Tray2)/MediaColor(White) /MediaWeight
75>>
          setpagedevice"
```

```

Tray3 = "<< /MediaType (Tray3)/MediaColor(White) /MediaWeight
75>>
        setpagedevice"
Tray4 = "<< /MediaType (Tray4)/MediaColor(White) /MediaWeight
75>>
        setpagedevice"
SendOverlays = Yes,Enabled

```

The DSC header comments are added at the beginning of the generated PostScript print stream as shown here:

```

%!PS-Adobe-3.0
%%Title: INSUREDS COPY
%%Creator: FormMaker PostScript Driver
%%CreationDate: Thu Apr 04 17:50:57 2002
%For: INSURED
%%Pages: (atend)
%%DocumentData: Clean7Bit
%%DocumentSuppliedResources: font (atend)
%%DocumentMedia:Media1 612 792 75 (White) (Tray1)
%%+ Media2 612 792 75 (White) (Tray2)
%%+ Media3 612 792 75 (White) (Tray3)
%%+ Media4 612 792 75 (White) (Tray4)
%%EndComments

```

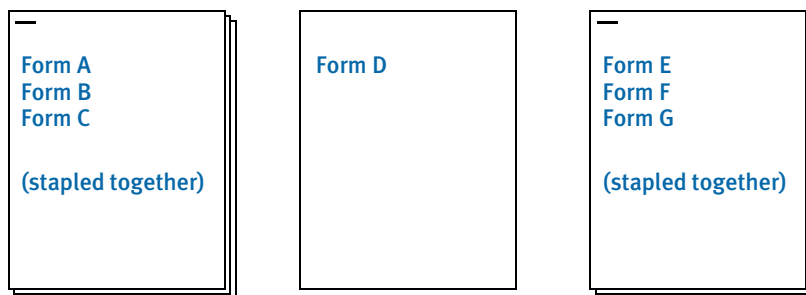
Stapling Forms

Use the StapleOn and StapleOff INI options in the PostScript printer control group to control staple support. These options work in a similar fashion to the Tray# INI options which let you specify PostScript commands directly as a quoted string or to look up the PostScript commands to use in your printer's PPD file.

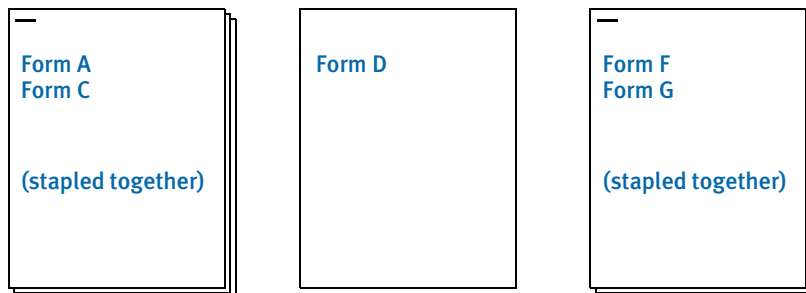
Here is an example. Suppose you have seven forms in the form set and all but one (Form D) are to be stapled. There are two recipients who are to receive these forms as shown in this table:

Form	Staple?	Recipients
A	Yes	INSURED, AGENT
B	Yes	INSURED
C	Yes	INSURED, AGENT
D	No	INSURED, AGENT
E	Yes	INSURED
F	Yes	INSURED, AGENT
G	Yes	INSURED, AGENT

The INSURED recipient's forms print as:



The AGENT recipient's forms print as:



By default, the PostScript print driver will use these commands:

```
< PrtType:PST >
...
StapleOn = "<</Staple 3 >> setpagedevice"
StapleOff = "<</Staple 0 >> setpagedevice"
```

You can override PostScript staple commands by providing an alternate PostScript command to use via the StapleOn and StapleOff options in your PostScript printer control group.

You can issue PostScript staple commands in these forms:

- A quoted string containing the PostScript commands. The quoted string should contain the appropriate PostScript commands for turning stapling on or off. Here is an example:

```
StapleOn = "1 dict dup /Staple 0 put setpagedevice"
```

- A UI keyword from a PPD file. UI keywords represent features that commonly appear in a user interface (UI). They provide the code to invoke a user-selectable feature within the context of a print job, such as the selection of an input tray or manual feed. The entries of UI keywords are surrounded by these structure keywords:

```
*OpenUI/*CloseUI or *JCLOpenUI/*JCLCloseUI
```

Here is an example of an OpenUI structure for XRXFinishing:

```
*OpenUI *XRXFinishing/Finishing: PickOne
*OrderDependency: 60.0 AnySetup *XRXFinishing
*DefaultXRXFinishing: None

*XRXFinishing None/None: "
1 dict dup /Staple 0 put setpagedevice"
*End

*XRXFinishing Single_Portrait_Staple/Single Portrait Staple: "
2 dict dup /Staple 3 put
  dup /StapleDetails 2 dict dup /Type 1 put dup /StapleLocation
  (SinglePortrait) put
  put setpagedevice"
*End

*XRXFinishing Single_Landscape_Staple/Single Landscape Staple: "
2 dict dup /Staple 3 put
  dup /StapleDetails 2 dict dup /Type 1 put dup /StapleLocation
  (SingleLandscape) put
  put setpagedevice"
*End

*XRXFinishing Dual_Portrait_Staple/Dual Portrait Staple: "
2 dict dup /Staple 3 put
  dup /StapleDetails 2 dict dup /Type 1 put dup /StapleLocation
  (DualPortrait) put
  put setpagedevice"
*End

*XRXFinishing Dual_Staple/Dual Landscape Staple: "
2 dict dup /Staple 3 put
```

```
dup /StapleDetails 2 dict dup /Type 1 put dup /StapleLocation
(DualLandscape) put
put setpagedevice"
*End
```

```
*?XRXFinishing: " (Unknown) = flush"
```

```
*CloseUI: *XRXFinishing
```

A PostScript Printer Definition (PPD) file is supplied with a PostScript printer. This file contains information about printer-specific features. You specify the PPD file you want to use in the PrinterModel option in your PostScript printer control group (just the file name, no drive, path, or file extension). If the PrinterModel option contains the name of a PPD file, this file must be in the directory specified in the DefLib option in the FMRes control group.

This example shows a PostScript printer group that uses a PPD file for a DocuPrint 65 printer (XRD60651.PPD) and specifies StapleOn and StapleOff options using keyword settings from the PPD file:

```
< PrtType:PST >
...
PrinterModel = XRD60651
StapleOn = *XRXFinishing Single_Portrait_Staple/Single Portrait
Staple:
StapleOff = *XRXFinishing None/None:
```

POSTSCRIPT PRINTER RESOURCES

A number of resources participate in the total printing process. They reside in directories specified in the MasterResource control group.

Fonts

The system supports PostScript Type 1 fonts. These fonts must reside in the directory specified in the FontLib option in the MasterResource control group when the DownloadFonts option is set to Yes.

Overlays

Use the OVLCOMP utility to create PostScript overlays from FAP files. These overlays must reside in the directory specified in the OverlayPath option in the MasterResource control group when the SendOverlays option is set to Yes.

PostScript Printer Definition (PPD) Files

A PostScript Printer Definition (PPD) file is supplied with a PostScript printer. This file contains information about printer-specific features. If the PrinterModel option contains the name of a PPD file, this file must be in the directory specified in the DefLib option in the FMRES control group.

USING THE GDI PRINT DRIVER

Oracle Insurance developed a Graphics Device Interface (GDI) print driver because it provides many opportunities for Windows platform users. For example, by using a GDI driver, you can now fax, since fax drivers can be installed into Windows as a GDI Windows printer driver.

Also, printing using GDI lets you print to printers that do not support any of the printer languages the system supports, such as inkjet printers. To make this driver even more useful, it includes the ability to scale output, which lets you shrink the printed output to the size of the paper.

The advantages of using the Graphics Device Interface (GDI) include:

- Ability to print to any printer attached via a Windows print driver
- Ability to print to any fax machine attached via a Windows print driver
- Ability to scale edge to edge forms to print within the printable area defined by the Windows print driver.

The disadvantages of using the Graphics Device Interface (GDI) include:

- Print quality is often poorer
- Inability to print a mixture of portrait and landscape forms
- Inability to print a mixture of simplex and duplex forms
- Inability to address the same printable area available when using our native print drivers.

NOTE: If you do not specify the option for sending color to a GDI printer, the system converts color (4-, 8-, or 24-bit) graphics into monochrome before sending them to the printer driver. Depending on the bitmap, this conversion from color to monochrome may not yield acceptable results. Be sure to consider your printers capabilities when you are creating graphics.

If you elect to send color, including color graphics, to a GDI printer that does not support color, the printer driver determines what to do. Some ignore the color commands (printing in black), and some apply a gray-scale adjustment to the output to simulate the color changes. Some GDI printer drivers cannot accept color commands at all. If printing to your Windows-attached printer causes a program fault, or print failure, try turning off the Send Color option via the system's Print window and sending the output again.

How it works

Most Windows applications print using the Windows GDI application programming interface. Essentially, the application uses commands similar to display commands to send print commands to the operating system. Windows, in turn, sends the commands to the currently installed Windows printer driver.

NOTE: Printer manufacturers provide Windows printer drivers for their printers. These come on install disks from the manufacturer, or sometimes ship with Windows itself. Other types of drivers (such as fax drivers) can be installed as Windows printer drivers.

When a Windows program talks to the operating system using GDI, printer commands are not emitted in the native language of the printer by the program. The program *prints* to Windows, and Windows then *prints* to the installed printer driver.

The printer driver then produces the native printer language commands, including the bitmap font definitions. If the printer driver belongs to a PCL printer, the print driver issues PCL commands, including fonts. In contrast, our PCL printer modules produce the PCL commands and fonts.

When you use our GDI driver, a Windows print driver will use the Windows screen fonts to print the document with its goal being to make the document look like it does on your screen.

Understanding the System

In Documaker implementations, users typically decide what fonts they want to use and then install those fonts on the production printer. Documaker applications try to make the screen look like the printed output, not the other way around. Information from the production printer fonts is loaded into the font cross-reference file. The system uses this information to try to represent the printer fonts on screen. The system can also convert production printer fonts into PCL bitmap fonts. The PCL fonts the system produces look like the fonts used on your production printer.

GDI print quality, by definition, is based on the fonts used for display. The attributes which describe fonts in the font cross-reference file determine which screen fonts are used. The screen fonts used determine what you see on the screen and how GDI printed output will look.

So, the key to improving GDI print is to improve the fonts used in the display system. Some of this can be improved by making sure the font's character widths and family name is correct. There are INI options for improving the screen font substitutions, if names cannot be matched up.

For the best results, you should use exact matching screen fonts. The system comes with a set of TrueType fonts that match the printer fonts included with the system. Install and use these fonts for best results.

NOTE: If you are instead working backward from existing production fonts, as is often the case, either an approximation must take place, or you have to find screen fonts built from the printer fonts.

GDI PRINTER DRIVER INI OPTIONS

You define the necessary printer options to print using the GDI printer driver. These options specify GDI output and are located in a `PrtType:xxx` control group, such as `PrtType:GDI`. Common GDI options are shown below, with default values in bold:

Option	Values	Description
Device	any file or device name	Not used by the GDI print driver.
Module	GDIW32	The name of the program module which contains the system's GDI print driver. See also the Class option. See also Using defaults for the Module and PrintFunc options on page 315 .
PrintFunc	GDIPrint	The name of the program function that is the main entry point into the system's GDI print driver. See also Using defaults for the Module and PrintFunc options on page 315 .
Resolution	300	Not used by the GDI print driver.
SendOverlays	Yes/ No	Not used by the GDI print driver.
OverlayPath	any directory	Not used by the GDI print driver.
OverlayExt	any file extension (OVL)	Not used by the GDI print driver.
PageNumbers	Yes/ No	Set to Yes to enable form or form set page numbering.
SendColor	Yes/ No Enabled/ Disabled/Hidden	Set to Yes to enable color printing. Enabled = Option appears in the Print window and is active (available to be checked). Disabled = Option appears in the Print window but is grayed out (not available to be checked). Hidden = Option does not appear in the Print window
DownloadFonts	Yes /No	Not used by the GDI print driver.
FitToWidth	Yes/ No	Scale pages to fit on the paper. This option will, if necessary, reduce the size of the page. It will not increase it.
TemplateFields	Yes/ No	Set to Yes to test print Xs in variable fields.

Option	Values	Description
SelectRecipients	Yes/No Enabled/ Disabled/Hidden	Enabled = Option appears in the Print window and is active (available to be checked). Disabled = Option appears in the Print window but is grayed out (not available to be checked). Hidden = Option does not appear in the Print window.
PrintViewOnly	Yes/No	If set to Yes, the view only sections will print. This does not apply to entry only sections, which are never printed. Entry only sections are usually worksheets. If the section is marked as hidden and view only, it will not print.
PrePrintedPaper	Yes,Disabled	Determines if the check box which lets you print or not print pre-printed objects appears on the Print window. Also determines the default for this check box—checked or unchecked. You must add this option to the INI file if you want the check box to appear on the Print window. The default for this option includes the checkbox on the Print window and leaves it unchecked. All objects except fields can be designated as pre-printed on the object's Properties window.
Class	<i>(first three characters of the Module option)</i>	Specifies the printer classification, such as AFP, PCL, XER, PST, or GDI. If you omit this option, the system defaults to the first three letters from the Module option. Some internal functions expect a certain type of printer. For instance, all 2-up functions require an AFP printer. The internal functions check the Class option to make sure the correct printer is available before continuing.
SuppressDlg	Yes/ No	Set to Yes to suppress the Windows Print window.
GDIDevice		Specifies the Windows printer name. Click Start, Settings, Control Panel, Printers to see a list of the printers you can choose from. If you set the SuppressDlg option to Yes and leave this option blank, the system suppresses the Print window and automatically prints to the default printer.

Include these options in your FSISYS.INI file (for Documaker Workstation) and FAPCOMP.INI files (for Docucreate).

In addition, you can add the following INI setting to automatically select landscape mode when printing any of the specified sections:

```
< VBPrOptions >
    Landscape = (list of landscape sections)
```

Beside the Landscape option, list the sections you want printed landscape. Separate each section with a comma.

Users can override this option at print time.

Understanding the System

If you do not set the *SuppressDlg* option to *Yes*, the Windows Print window appears when you use the print device to spool the job. If you omit the *SuppressDlg* option or set it to *No*, the user can select which Windows print device to spool the output through. By setting this option to *Yes*, the Windows Print window (not the system's Printer window which normally appears first), will be automatically completed for the user.

If you set the *SuppressDlg* option to *Yes*, the default Windows printer is used unless the *GDIDevice* option specifies a printer. You can use the *GDIDevice* option to name a specific Windows print device for spooling the raw output. The name you specify must match one of the installed printers. You can see these printer names by going to the Control Panel and clicking the Printers icon.

If you misspell the printer name or specify one not installed for the *GDIDevice* option, the system will send the output to the default printer device or you will get an error and printing will stop. On Windows, an incorrect setting sends the raw output to spool to the default printer device.

Don't confuse the *SuppressDlg* option with the *SuppressDialog* option in the Printer control group in the FSISYS.INI file. The *SuppressDialog* option suppresses the system's internal Printer Selection window—the one that names which *PrtType:XXX* group from the INI file you wish to use. The *SuppressDlg* option suppresses the operating system's (Windows 32-bit) Printer Selection window.

Using defaults for the Module and PrintFunc options

Default values for the *Module* and *PrintFunc* options in the *PrtType:xxx* control group are provided when you use a standard print type name or print class, such as AFP, PCL, PDF, PST, VPP, XER, XMP, or GDI.

These defaults keep you from having to enter the *Module* and *PrintFunc* names in your INI file. For example, if you want to generate GDI print files, you can specify these INI options:

```
< Printer >
  PrtType      = MYGDI
< PrtType:MYAFP >
  Class       = GDI
```

And the system will default these options for you:

```
< PrtType:MYAFP >
  Module       = GDIPRT
  PrintFunc    = GDIPrint
```

AVOIDING PROBLEMS WITH FAX DRIVERS

Use the FullSupport option to prevent problems with FAX drivers which can occur when you are printing from Documaker Workstation or PPS.

The GDI driver first looks for this INI option in the control group whose name reflects the Windows print driver, such as *HP LaserJet 4050 Series PS*.

If the FullSupport option is set to Yes, the GDI driver assumes the Windows print driver contains full print support and can handle form sets with mixed simplex and duplex forms (some FAX drivers crash when presented these kinds of forms).

Here is an example:

```
< HP LaserJet 4050 Series PS >
FullSupport = Yes
```

If not found there, the GDI driver looks for the FullSupport option in the control group for the printer type, such as *PrtType:GDI*. If you place the FullSupport option in the *PrtType:GDI* control group, it serves as a default for all GDI printers. Putting the option in for specific devices overrides this default.

BATCH PRINTING TO FILES

You can use the GDI print driver to print to a file by adding the PrintToFile option in your GDI printer control group. This lets you direct output to the path and file you specify — equivalent to checking the Print to File field on the Print window.

```
< PtrType:GDI >
PrintToFile = Yes
```

Option	Description
--------	-------------

PrintToFile	Enter Yes to have the GDI print driver use the Port options as the output print file names for each batch when running GenPrint. The default is No.
-------------	---

In the GenPrint program, output print file names for each batch are specified using the Port INI option. When you use the GenPrint program with most Documaker print drivers, the Port option determines the name of the print stream created for each batch.

Normally, the GDI print driver prints directly to a Windows print driver and does not create files written to disk. By setting the PrintToFile option to Yes in your GDI printer control group, the GDI print driver creates a print stream for each batch based on the names specified in the Port options — just like the other Documaker print drivers.

Because the Documaker GDI print driver is not designed for batch print, these additional GDI print options are recommended when you set the PrintToFile to Yes:

```
< PtrType:GDI >
...
SuppressDialog = Yes
GDIDevice = (Windows printer name)
FullSupport = Yes
```

Option	Description
SuppressDialog	Enter Yes to suppress the Windows Print window from appearing.
GDIDevice	Enter the name of the Windows print driver you want to use.
FullSupport	Enter Yes to tell the Windows driver to fully support duplexing, tray selection, and so on.

This feature is limited to using the GDI driver with GenPrint (multi-step batch print) to produce output print files and is limited to simple GenPrint (batch print) environments.

Keep in mind that all normal GDI print limitations (fidelity, tray selection, duplexing, and so on) apply, plus the following:

- Banner page processing may not work.
- Cannot use the SetDeviceName and BreakBatch DAL functions.
- Callback functions may not work.
- Single step processing does not work correctly (all transactions are printed to a single file).
- Multiple driver routers may not work.
- Printing from Studio or Image Editor may work but the Device setting will be used to create the file. Printing from Documaker Workstation may not work.
- Printing to fax drivers, email drivers, and so on may not work and other types of print or print features not previously discussed may not work.

In other words, trying to use PrintToFile option with anything except GenPrint running in a simple batch mode using a normal Windows print driver is not supported.

USING PASS-THROUGH PRINTING

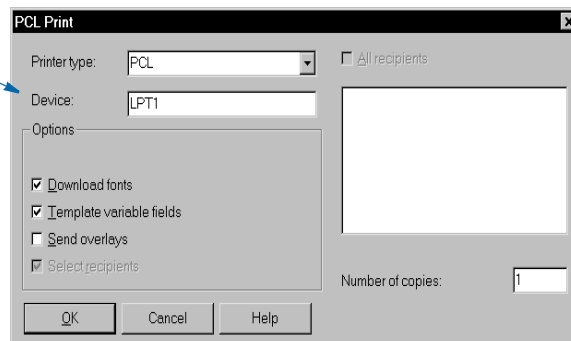
There are some problems which occur when you print to LPT1 on Windows platforms. One problem occurs if you run Netware Client 32 for Windows 95. Although you can open LPT1 from the system, you may receive errors when printing large amounts of data. Downloading PCL fonts usually causes this.

Another problem occurs when a print queue adds additional printer commands to system-created print jobs. This causes invalid output to be sent to the printer. The HP 5si print driver can cause this problem. Another problem affects other software which redirects printers and expects all print output to use the Windows GDI mechanism.

Documaker includes a GDI print driver that uses Windows-native calls for printing, which is how most applications print under Windows. However, the Windows system print drivers have problems handling some system printing requirements, such as enhanced font selection, the ability to combine duplexing with landscape forms, and so on.

To solve these problems, our print drivers can produce the commands for controlling the printer while still using an installed Windows printer device. To use this feature, leave the Device field blank on the Print window, where you select the printer driver you want to use.

Leave this field blank



Normally, the Device field contains the name of the device (LPT1) or the name of the file (D:\OUTPUT.PCL) the system should print to. When you leave this field blank, you tell the print driver you want to print through an installed printer device. After you click Ok, the Windows Print window appears so you can select which printer device to send the print job.

This printer device must be associated with a printer supported by the system's print driver. If you have a printer device available that is associated with a printer not supported by the system's print driver, the results are unpredictable. For example, if you select PCL as the system printer type (print driver), but choose a printer device associated with an AFP printer, the AFP printer will not understand the PCL output and will print garbage.

Unlike the GDI driver, our print drivers control the printed output. The Windows Print window is the standard print window provided by Windows. Documaker applications cannot control or change this window. In addition, since our print driver is controlling the printer, most of the options on the Windows Print window will be ignored. The only options you can use are:

- Select a printer device.
- Select the Cancel button and the print process is canceled.
- Check the Print to File field and the system will print the document to the file you specify.

NOTE: Not all Windows print drivers support pass-through printing. If you receive an error while printing in this manner, you are probably using a Windows print driver that does not support pass-through printing.

CREATING PDF FILES

Adobe Systems created the Portable Document Format (PDF). It is the native file format of the Adobe Acrobat family of products. The original PDF file format was version 1.0.

The system produces PDF files which adhere to PDF file format version 1.3 (or version 1.4 if 128-bit encryption is used). This version supports compression and page-at-a-time downloading. With page-at-a-time downloading (byte-serving), a web server sends only the requested page of information to the user, not the entire PDF document.

NOTE: When you use Acrobat Reader to view a PDF document, you do not have to do anything to make it download a page at a time. Acrobat Reader and the web server handle this for you.

If you want the entire PDF document to continue downloading in the background while you view the first page in Acrobat Reader, choose File, General Preferences and select the Allow Background Download of Entire File option.

For additional information about creating PDF files with Documaker applications, please refer to the following documents:

For	See
Docupresentment	Please see the Internet Document Server User Guide for more information on PDF support.
Documaker Server (z/OS)	Please see the additional configuration steps in the Documaker Server Installation Guide .
For all other products and for general PDF information	Please see Using the PDF Print Driver .

CREATING RTF FILES

The RTF print driver lets you create a medium-fidelity export of the contents of a form set in a format you view or edit with most popular word processors. The email print driver uses this capability to email form sets. See [Emailing a Print File on page 342](#) for more information.

To use the RTF print driver, you need these INI settings:

```
< Printers >
  PrtType      = RTF
< PrtType:RTF >
  Module       = RTFW32
  PrintFunc    = RTFPrint
```

You will also need to specify an output device name on the Print window.

NOTE: The RTF print driver does not support graphics (bitmaps), charts, or bar codes.

Generating separate files

You can generate separate files for each transaction when you choose RTF (or PDF) from WIP or batch print.

The name of the files will have a rolling number appended to the end of the name that starts the process and is filled in on the Print window. This is automatically handled and you do not have to set INI options to get the WIP or batch print to work as long as your PrtType name is PrtType:RTF.

There are several INI options you can use to override the naming process and also name other print drivers that require this unique handling.

```
< BatchPrint >
  NoBatchSupport = RTF
  PreLoadRequired= RTF
```

These are the default settings and cannot be overridden. However, you can specify other PrtType print driver definitions you want to fall into these same categories.

Option	Description
NoBatchSupport	Indicates that the named PrtType items, separated by semicolon, do not really support batch transactions and require special handling.
PreLoadRequired	Lets you specify all the PrtType items, separated by semicolon, that should be forced to load the form set prior to the starting print. Most print drivers don't require this special requirement, but some, such as PDF do.

Also, you can name PrtType specific items under the BatchPrint control group to override the normal Device naming option. Here is an example:

```
< BatchPrint >
  PDF = ~HEXTIME .PDF
  RTF = ~HEXTIME --KeyID .RTF
```

Any batch print sent to PrtType:PDF (picking PDF on the Print window) will override the name and store the current hexadecimal date and time, such as BCF09CA4.PDF, which is an eight-character name, as the name of each transaction's output.

Also, you can combine INI built-in calls as shown in the RTF example. Here any WIP or batch print sent to RTF will name the files using the HEXTIME and the KeyID from the WIP transaction. This will result in names similar to this: BCF099A4-123456.RTF

Note that you must leave a space after the built-in INI function name for it to work properly. That space will not appear in the resulting output name.

Adding or removing frames

By default, the RTF print driver uses frames to replicate the look of a document. If you do not want the frames, which print as boxes around the various document objects, to appear, set the WriteFrames option to No.

```
< PrtType:RTF >  
WriteFrames =
```

For instance, you can use the RTF driver to print form sets to an RTF file. Once the RTF file is created, you can then open it in a word processor. To avoid having frames in the file, you would set this option to No.

Creating form fields

You can use the RTF print driver to convert variable fields into RTF form fields. For example, a variable address field is converted into an RTF form field. The format of the field is retained. If, for example, the address field contained all uppercase characters, this would be reflected in the corresponding RTF form field.

To print form fields, include this INI option:

```
< PrtType:RTF >  
AllowInput = Yes
```

NOTE: This works with print types RTF and RTF_NoFrame.

You may also need to include the WordTimeFormats and WordDateFormats control groups. You can use these control groups in case you are using a time or date format that has no equivalent in Word. The following groups and options let you map a Documaker format to a Word format.

```
< WordTimeFormats >  
hh:mm XM =  
< WordDateFormats >  
bD/bM/YY =
```

To the left of the equals sign, you list the Documaker format used on the section. To the right, you list the Word format you want to use.

Setting margins

The RTF print driver produces margins by calculating what is required and putting the result in the RTF output. You can, however, set minimum required margins using the RTF print type control group.

You must set the minimum required margins in FAP units (2400 dots per inch). Here are the default settings:

```
< PrtType:RTF >
    MinTopMargin    = 400
    MinLeftMargin   = 600
    MinRightMargin  = 600
    MinBottomMargin = 400
```

Margin values specified in the INI file override those set in the FAP file if the page margins in the FAP file are smaller.

NOTE: The changes in the margins are noticeable when you open the document in an application such as Microsoft Word. You will see the left and right margins shifting based on what you specified in the INI file. The top and bottom margins (seen on the left side of the page) will also vary based on what you specified in the INI file.

Removing the contents of headers and footers

Use these options to remove the contents, including graphics and text, from headers and footers when creating RTF files:

```
< PrtType:RTF >
    EmptyFooters = Yes
    EmptyHeaders = Yes
```

Option	Description
EmptyHeaders	Enter Yes to remove the contents from any headers in the file. This includes both text and graphics. The default is No.
EmptyFooters	Enter Yes to remove the contents from any footers in the file. This includes both text and graphics. The default is No.

USING THE VIPP PRINT DRIVER

Variable Data Intelligent PostScript PrintWare (VIPP) was created by Xerox in the early 1990s to enable high-performance variable data printing on PostScript devices. VIPP is based on PostScript and works by extending the PostScript programming language. VIPP can be used on any PostScript compatible printer, including Xerox and third-party network, workgroup, and production devices that have been licensed for VIPP.

VIPP is supported on these devices:

- DocuPrint NPS (monochrome and color)
- DocuPrint N-series
- DocuSP (Document Services Platform) controllers, including iGen3
- DocuColor, EFI, and Creo controllers, (including iGen3)

The Documaker VIPP print driver requires that you have VIPP version 5.3 or later installed on your printer's controller.

NOTE: Contact your Xerox representative to see if your specific printer supports VIPP and to obtain VIPP licensing and installation of the latest VIPP version. To use the Documaker VIPP print driver, any supported device must have a local file system you can access to transfer resource files. Check with your Xerox representative for any limitations or considerations when using VIPP on your specific printer. For example, DocuColor systems may have limited or no support for stapling, duplexing, and paper tray (media) selection. In addition, older models of DocuTech and DocuPrint printers may have limited or no support for caching resource files.

The Documaker VIPP print driver produces native mode VIPP output. Native mode refers to files composed solely of VIPP commands. VIPP commands are used to place text, lines, boxes, shades, and graphics directly on the page. Native mode is the default VIPP mode.

A VIPP print job can refer to external resource files such as fonts, TIFF and JPEG graphics files, and page overlays (segments).

VIPP provides a mechanism called VIPP Projects that lets you manage all of the resources needed for a VIPP print job.

VIPP Projects allow you to organize the resources of a job under a single name (the project) and group the jobs by family (the folder).

A folder is a collection of projects that share some common features. For example, you may decide to create one folder for each customer, each division, or each line of business. Within each folder, you could define multiple projects. A folder can contain common resources (company logo, standard boilerplate page segments, and so on) that are shared by the projects within the folder. The projects will contain resources that are unique to the project. You can also have resources that are global across all projects and folders.

Having multiple folders and projects provide a great deal of flexibility in how you organize and share your resources. Folders and projects can even provide the logical grouping of the physical resources used by the job at one or more steps during in the job life cycle (development, testing, production, and so on).

This is a sample structure:

```

Folder A - Dallas Division
    Project 1
    Project 2
    Project 3
Folder B - Atlanta Division
    Project 1
    Project 2
    Project 3
Folder C - Silver Springs Division
    Project 1
    Project 2
    Project 3

```

VIPP Resource Files

The resource files referenced by a Documaker VIPP job are:

- Pictures (images) in TIFF or JPEG format
- Overlays (segments) in VIPP format
- PostScript fonts
- Font encoding tables

NOTE: All VIPP resource files stored on the VIPP console that are referenced by a Documaker VIPP job must have lower case file names.

Converting bitmaps into VIPP image files

VIPP supports bitmap files in TIFF and JPEG format. The Documaker VIPP print driver assumes that mono-color (1 bit per pixel) graphics have been converted into TIFF format and multi-color (more than 1 bit per pixel) graphics have been converted into JPEG format.

Scanned images are usually converted into multi-color graphics even though the images can appear to be black and white. There are a number of ways to convert your graphics into TIFF and JPEG files as expected by the VIPP print driver.

- Use Logo Manager. Choose the File, Save As option. On the Save As window, select *VIPP image files (*.*)* in the Save as Type field. Selecting VIPP image files tells the system to create a TIFF file or a JPEG file, based on the number of colors used in the graphic.
- Use the Conversion Wizard in Documaker Studio. Choose the Manage, Conversion option from the main menu. Select *VIPP image files* as the Final Conversion File Type. Selecting VIPP image files tells the system to create a TIFF file or a JPEG file, based on the number of colors used in the graphic.
- Use Docutoolbox RP. Choose the File, Convert, Logos option from the main menu. Select VIPP image files as the output file type. Selecting VIPP image files tells the system to create either a TIFF file or a JPEG file, based on the number of colors used in the graphic.
- Use the LOG2VIPP utility. The utility creates a TIFF file or a JPEG file based on the number of colors used in the graphic. See the [Docutoolbox Reference](#) for details.

NOTE: All VIPP resource files stored on the VIPP console that are referenced by a Documaker VIPP job must have lower case file names. It is usually easier to make sure the resource file names are lower case before they are transferred to the UNIX workstation console attached to the VIPP printer.

Converting FAP files into VIPP segment files

VIPP supports pre-compiled printer overlays (called segments). A segment is a VIPP native mode or a PostScript fragment intended to be reproduced once or several times at specific locations on one or more pages. You can use the OVLCOMP utility to convert Documaker FAP files into VIPP segment files.

Here is an example of the syntax for this utility. For more information, see the [Docutoolbox Reference](#):

```
OVLCOMP /I=fapfile /X=fxrfile /L=VPPW32 /F=VPPPrint /U=VPP /C
```

Parameter	Description
-----------	-------------

/I	Enter the name of the FAP file. Omit the extension.
/X	Enter the name of the FXR file. Omit the extension.
/L	For the VIPP print driver, enter VPPW32 .
/F	For the VIPP print driver, enter VPPPrint . Case is important when using this parameter, therefore, you must enter it exactly as shown here: /F=VPPPrint
/U	(Optional) Enter the name of your VIPP printer group. Here is an example: /U=VPP
/C	(Optional) Include this parameter if you want to use color.

You will need a FSISYS.INI file in the directory that you run the OVLCOMP utility from. Within the FSISYS.INI file, you should have a VIPP printer group defined. For example, below is a subset of the INI settings you might find in a VIPP printer group.

```
< PrtType:VPP >
Module           = VPPW32
OverlayExt       = .seg
PrintFunc        = VPPPrint
SendOverlays     = Yes, Enabled
```

You can specify the overlay (segment) extension you want to use by including the OverlayExt option in your VIPP printer control group and telling OVLCOMP the name of your VIPP printer group (/U=VPP). Use the same OverlayExt setting in your VIPP printer control group when producing a VIPP print stream that uses overlays (segments). If you omit the OverlayExt option, the default file extension for an overlay is .ovl.

Another way to create VIPP overlays (segments) is to use the Conversion wizard in Documaker Studio. Select the Compile Sections (FAPs) to Print Files option and choose Section to VIPP as the conversion type.

You can also use Docutoolbox RP to create VIPP overlays (segments) by choosing the File, Convert, FAP to VIPP option.

NOTE: All VIPP resource files stored on the VIPP console that are referenced by a Documaker VIPP job must have lower case file names. It is usually easier to make sure the resource file names are lower case before they are transferred to the UNIX workstation console attached to the VIPP printer.

VIPP fonts

VIPP supports PostScript fonts as VIPP resources. While VIPP supports any font type (Type 1, Type 3, and composite) supported by the PostScript interpreter, Documaker only supports Type 1 PostScript fonts. The PostScript fonts you use must be defined in your font cross-reference (FXR) file.

If you are using a base FXR file, like REL103.FXR or REL110.FXR, the base PostScript fonts are already set up for you in the FXR file. The same PostScript fonts used for printing with the Documaker PostScript print driver are also used with the Documaker VIPP print driver.

If you are using a custom FXR file and you have not set up your FXR file for printing PostScript, then you will need to add the PostScript fonts to your FXR file. You can use the Import option for the Font manager to import PostScript fonts into your FXR file. The primary fields used by the PostScript and VIPP print drivers are the Codepage field on the Dimensions tab, and the Font File, Font Name, and Char Set ID fields in the PostScript section of the Properties tab.

Here are examples of the Dimensions and Properties tabs in Documaker Studio for a font record in your FXR file:

On the Dimensions tab, you must modify this field to use the VIPP print driver

Font Dimensions - 10006 Arial Narrow 6 PT	
▼ Dimensions	
FAP Height	240
FAP Width	168
FAP Baseline	192
Original Font File	an____.pfb
Codepage	1004

Here are the fields on the Properties tab you must modify to use the VIPP print driver.

Font Properties - 10006 Arial Narrow 6 PT	
▼ PostScript Properties	
Font File	AN____.PFB
Font Name	ArialNarrowMT
Char Set ID	W1
▼ Advanced...	
Typeface ID	
Char Set Name	
Font Index	0
Other Info	
Options	0
Flag	0
▼ PDF Properties	

Field	Enter...
Codepage	Under Windows, the system uses the ANSI code page. Normally, this field is set to 1004 or is left blank.

Field	Enter...
Font File	The PostScript Type 1 font file name, including the .PFB extension. Font Manager fills this field when you insert a PostScript font.
Font Name	The full font name, such as Times-Roman. Font Manager fills this field when you insert a PostScript font.
Char Set ID	<p>A character set (also known as a symbol set) identifies the set of symbols provided by the font. It is used by PostScript printing to build an internal code page. Use W1 for the fonts that use the standard Windows ANSI code page. The character set ID and code page values should match those specified in the CODEPAGE.INI file.</p> <p>Code page 1004 and Char Set ID W1 are used for fonts that use the standard Windows ANSI code page.</p> <p>Code page 9998 and Char Set ID WD are used for DocuDings (Wingdings clone) font.</p> <p>Code page 9999 and Char Set ID MI are used for the base MICR font.</p>

The *Working with Fonts* chapter in the [Docucreate User Guide](#) (and other manuals) contains more detailed information on how to add PostScript fonts to your FXR file.

NOTE: All VIPP resource files stored on the VIPP console that are referenced by a Documaker VIPP job must have lower case file names. It is usually easier to make sure the resource file names are lower case before they are transferred to the UNIX workstation console attached to the VIPP printer.

VIPP font encoding files

A PostScript font is a collection of characters. Each character in a PostScript font has a PostScript-assigned name. For example, the dollar sign (\$) character has a PostScript name of `/$dollar`. While PostScript fonts use PostScript-assigned names for each character, PostScript (and VIPP) print streams use a byte value to represent each character. For example, the dollar sign (\$) is usually represented by a value of 24 hex. An encoding table is used to match a byte value (24 hex) with the character name (`/$dollar`) contained within a PostScript font.

This table shows the relationship between the hex byte value, the equivalent decimal value, the PostScript character name, and the actual printed character using the standard ASCII encoding table.

Hex value	Decimal value	PostScript name	Character
20	32	/space	
24	36	/dollar	\$
2A	42	/asterisk	*
30	48	/zero	o
41	65	/A	A

Hex value	Decimal value	PostScript name	Character
61	97	/a	A
7A	122	/z	z

VIPP font encoding files serve a similar purpose as the Documaker CODEPAGE.INI file and the Codepage and Char Set ID settings in the font cross-reference file. The Documaker VIPP print driver uses the Codepage setting for each font in the font cross-reference to determine the name of the encoding file to use. The Documaker VIPP print driver appends the letters *cp* to the value of the code page setting for each font in the font cross-reference to determine the name of the VIPP font encoding file. Therefore, if a font has a Codepage setting of *1004*, then the Documaker VIPP print driver will use a VIPP font encoding file called *cp1004*.

These VIPP encoding files are provided to correspond to the code pages used by the base Documaker font cross-reference files:

File	Description
cp1004	The VIPP encoding file used for fonts that use the standard Windows ANSI code page. Most text fonts will use this.
cp9998	The VIPP encoding file used for the DocuDings font (clone of Wingdings).
cp9999	The VIPP encoding file used for the base MICR font.

NOTE: All VIPP resource files stored on the VIPP console that are referenced by a Documaker VIPP job must have lower case file names. It is usually easier to make sure the resource file names are lower case before they are transferred to the UNIX workstation console attached to the VIPP printer.

Managing VIPP Resources

Documaker VIPP print jobs use external resources for VIPP images, segments, fonts, and encoding files. By using external resources, the amount of time needed to produce a VIPP print stream is greatly reduced (as well as the size of the print job). Because the resources are not part of the job, the VIPP resources must be deployed to the controller (often a Sun workstation) that houses the VIPP software and ultimately drives the printer.

You will need some means of transferring VIPP resource files to the controller for the VIPP printer such as:

- Windows FTP command line utility
- Third- party FTP file transfer utility
- VIPP Manage (contact Xerox for more information)

You will need to log on with root access onto the controller. For some controllers, you can use the following user ID and password.

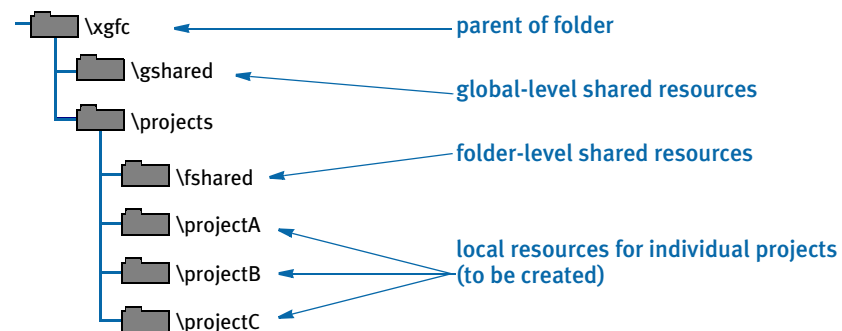
User ID: **root**
Password: **service!**

Contact your Xerox representative if you need help logging onto the controller for your VIPP printer.

As mentioned earlier, VIPP lets you organize the resources required by a VIPP job under a hierarchy of folders and projects. A folder is a collection of projects that share some common features. For example, you can decide to create one folder for each customer, each division, or each line of business. Within each folder, you could define multiple projects. A folder can contain common resources (company logo, standard boilerplate page segments, and so on) that are shared by the projects within the folder. The projects will contain resources that are unique to the project. You can also have resources that global across all projects and folders.

When VIPP is installed on the controller for your printer, VIPP is configured with a file called *xgfunix.run* (stored in the */usr/xgf/src* directory). The *xgfunix.run* file contains VIPP commands that determine the VIPP resource directories.

By default, VIPP is configured with the following VIPP projects repository (collection of VIPP resources and projects):



The root path for *xgfc* will be */usr/xgfc* on controllers that use UNIX systems.

In the *xgfunix.run* file, you might see a SETPPATH (VIPP command) that looks like this:

```
[ (/usr/xgfc/$$FOLDER./$$PROJECT./)    % project local paths
  (/usr/xgfc/$$FOLDER./fshared/)        % project folder shared paths
  (/usr/xgfc/gshared/)                  % project global shared paths
  (/usr/xgfc/fontlib/)                  % project access to font lib
  (/usr/xgf/encoding/)                  % project access to standard encoding
  (/usr/xgf/gshared/)                   % project global shared path
  (/opt/XRXnps/resources/ps/mislib/)    % project access to DocuSP
resource list
] SETPPATH
```

The *\$\$FOLDER.* and *\$\$PROJECT.* strings are placeholders for project folders and project names. In the example listed earlier, *\$\$FOLDER* would be represented by the *projects* folder and *\$\$PROJECT* could be represented by *projectA*, *projectB*, or *projectC*.

Project paths are divided into these three levels of hierarchy or scope:

- Local scope — paths that contain both *\$\$FOLDER* and *\$\$PROJECT*. These libraries will hold resources that pertain only to the project. In the example listed earlier, *usr/xgfc/projects/projectA* would have a local scope.

- Folder scope — paths that contain only `$$FOLDER`. These libraries will hold project libraries and resources shared by projects belonging to the same folder. In the example listed earlier, *usr/xgfc/projects/fshared* would have a folder scope.
- Global scope — paths that contain neither `$$FOLDER` nor `$$PROJECT`. These libraries will hold resources shared by all projects. In the example listed earlier, *usr/xgfc/gshared* would have a global scope.

When a resource is present with the same name in more than one folder (scope), VIPP uses the following order of precedence to determine which resource file to use:

- Local scope folder
- Folder scope
- Global scope

Even the simple default VIPP repository gives you a lot of flexibility in how you manage your VIPP resources.

As you recall, Documaker LOG files are converted to VIPP image files (TIFF or JPEG files). Let's say that some of your LOG files are unique to projectA while others are shared by projectA, projectB, and projectC.

The TIFF or JPEG files that are unique to projectA could be stored in a local scope folder such as *usr/xgfc/projects/projectA*.

The TIFF or JPEG files that are shared between projectA, projectB, and projectC could be stored in a folder scope folder such as *usr/xgfc/projects/fshared*.

Similarly, Documaker FAP files are converted to VIPP segment files. Again, some of your FAP files are unique to projectA while others are shared by projectA, projectB, and projectC.

Like the VIPP image files, the VIPP segment files that are unique to projectA could be stored in a local scope folder such as *usr/xgfc/projects/projectA* while the VIPP segment files that are shared between projectA, projectB, and projectC could be stored in a folder scope folder such as *usr/xgfc/projects/fshared*.

Finally, you have the PostScript fonts and the font encoding resources to consider. Perhaps your company has established standards on the use of the PostScript fonts and font encoding. As a result, you only need one set of PostScript fonts and font encoding files for all projects to use. In that case, you could place your PostScript fonts and font encoding files in a global scope folder such as *usr/xgfc/gshared*.

In the section entitled VIPP INI Settings, you will see how you can define the folder name ("`$$FOLDER`.") and project name ("`$$PROJECT`.") used to represent the directories containing the VIPP resources required by the VIPP print streams produced from the Documaker VIPP print driver. You also see how to set up your own list of libraries containing VIPP resources.

NOTE: All VIPP resource files stored on the VIPP console that are referenced by a Documaker VIPP job must have lower case file names. It is usually easier to make sure the resource file names are lower case before they are transferred to the UNIX workstation console attached to the VIPP printer.

VIPP INI Options

Here are the INI options and settings commonly-used with the VIPP print driver:

Option	Values	Description
Device	any file or device name	The name of the file or device (LPT1) where the VIPP print stream should be written. This setting is ignored by the GenPrint program but is used by Studio, the Image Editor, and other system programs. The default is the first three letters of the entry for the Module option, such as VPP.
Module	VPPW32	The name of the program module that contains the VIPP print driver. See also the Class option. The default is PCLW32, but you must enter VPPW32 to use the VIPP print driver.
PrintFunc	VPPPrint	The name of the program function that is the main entry point into the VIPP print driver. The default is PCLPrint, but you must enter VPPPrint . Case is important when using this option, therefore, you must enter it exactly as shown here: VPPPrint
Resolution	300	The dots per inch resolution of the printer that will receive the PostScript data stream. The default is zero (0) which tells the system to let the print driver to determine the resolution. The VIPP print driver defaults to 300 dpi.
SendOverlays	Yes/No	Set to Yes if you have created VIPP overlays (segments) for each FAP file.
CacheFiles	any number, zero or higher	Set to enable the caching of VIPP segments and images. The first x number of VIPP segments and images in the print job are cached. The default is zero (0).
CacheLogos	Yes/No	Set to enable the caching of VIPP images if CacheFiles is also enabled. The default is No.

Some default settings are determined by the program performing the print operation. The defaults in this table refer to printing from the GenPrint program. The defaults when printing from other applications, such as Documaker Workstation, may differ.

Option	Values	Description
DSCHeaderComment		Use to specify PostScript Document Structure Convention (DSC) comments you want added to the header portion of the generated VIPP print stream. You can include as many DSCHeaderComment options as are necessary. See Adding DSC comments on page 338 for more information.
OverlayExt	any file extension	The file extension of the VIPP overlays (segments). The default is .ovl.
PageNumbers	Yes/No	Set to Yes to enable form or form set page numbering. The default is No.
SendColor	Yes/No Enabled/ Disabled/ Hidden	Set to Yes to enable color printing. Enabled = Option appears in the Print window and is active (available to be checked). Disabled = Option appears in the Print window but is grayed out (not available to be checked). Hidden = Option does not appear in the Print window.
HighlightColor	Yes/No	Set to Yes to enable highlight color support. The default is No. If you set this option to Yes, you must also set the SendColor option to Yes.
DownloadFonts	Yes/No	Set to Yes to embed (download) PostScript fonts within the VIPP print stream. Set to No if you have loaded the PostScript fonts onto the VIPP controller. The default is Yes but you will get better performance if you set this option to No.
TemplateFields	Yes/No	Set to Yes to test print Xs in variable fields
Class	(first three characters of the Module option)	Specifies the printer classification, such as AFP, PCL, XER, PST, GDI, or VPP. If you omit this option, the system defaults to the first three letters from the Module option. Some internal functions expect a certain type of printer. For instance, all 2-up functions require an AFP printer. The internal functions check the Class option to make sure the correct printer is available before continuing.

Some default settings are determined by the program performing the print operation. The defaults in this table refer to printing from the GenPrint program. The defaults when printing from other applications, such as Documaker Workstation, may differ.

Option	Values	Description
SelectRecipients	Yes/No Enabled/ Disabled/ Hidden	This only applies to the Documaker Workstation/PPS systems. Enabled = Option appears in the Print window and is active (available to be checked). Disabled = Option appears in the Print window but is grayed out (not available to be checked). Hidden = Option does not appear in the Print window.
Tray# (where # is a number from 1 to 9)	Media string	Specifies a media string in the form of: <i>MediaType:MediaColor:MediaWeight</i> See Setting up paper trays on page 337 for more details.
Folder	Directory name	Name of the high level directory (folder) under which a project may appear. See Setting up folders and projects on page 335 for more details.
Project	Directory Name	Name of the directory where local resources for a project will reside. See Setting up folders and projects on page 335 for more details.
ProjectPath	Fully qualified directory path	Each ProjectPath setting defines a path that will be used to define a SETPPATH command that overrides the one found in the <i>xgfunix.run</i> file found on the VIPP controller. The path defined by the first ProjectPath setting will be the first directory searched for VIPP resources. If the resource is not found, the path defined by the second ProjectPath will be searched next (and so on). See Overriding the list of libraries for projects on page 336 for more information.

Some default settings are determined by the program performing the print operation. The defaults in this table refer to printing from the GenPrint program. The defaults when printing from other applications, such as Documaker Workstation, may differ.

Setting up folders and projects

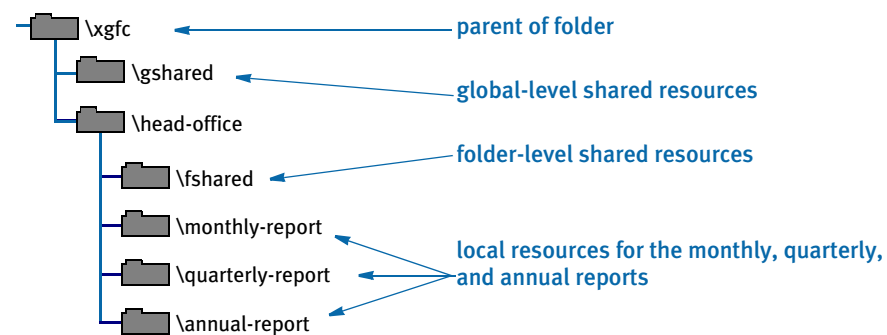
VIPP uses a configuration file named *xgfunix.run* (stored in the */usr/xgf/src* directory) to define a list of libraries (directories) for projects. In the *xgfunix.run* file, you might see a SETPPATH (VIPP command) that looks like this:

```
[ (/usr/xgfc/$$FOLDER./$$PROJECT./)    % project local paths
  (/usr/xgfc/$$FOLDER./fshared/)        % project folder shared paths
  (/usr/xgfc/gshared/)                  % project global shared paths
  (/usr/xgfc/fontlib/)                  % project access to font lib
  (/usr/xgf/encoding/)                  % project access to standard encoding
  (/usr/xgf/gshared/)                    % project global shared path
  (/opt/XXnps/resources/ps/mislib/)      % project access to DocuSP
resource list
] SETPPATH
```

SETPPATH is a VIPP command that defines a list of libraries (directories) for projects. The *\$\$FOLDER.* and *\$\$PROJECT.* strings are placeholders for project folders and project names.

You can use the projects directory for your main folder or create your folder directory. The name of the directory for your local project resources can be anything you wish.

Let's say you wanted to create a series of projects for the head office. Your VIPP projects repository might look like this:



Of course, you would need to create the *head-office* directory along with the subdirectories for the *fshared*, *monthly-report*, *quarterly-report*, and *annual-report* on the VIPP controller. And you would need to transfer the VIPP resource files (images, segments, fonts, and so on) into the appropriate directories.

However, before you can produce one of the reports for the head office, you will need to tell the Documaker VIPP print driver which VIPP folder and project names this report will use. You do this by specifying the Folder and Project options in your VIPP printer control group.

Option	Description
Folder	The Folder option contains the name of the high level directory (folder) under which a project may appear. The value set in the Folder option is substituted automatically as the <code>\$\$FOLDER</code> string in the SETPPATH statement found in the <i>xgfunix.run</i> file on the VIPP controller.
Project	The Project option contains the name of the directory where local resources for a project will reside. The value set in the Project option is substituted automatically as the <code>\$\$PROJECT</code> string in the SETPPATH statement found in the <i>xgfunix.run</i> file on the VIPP controller.

Using the example described earlier, let's say you want to produce a monthly report for the head office. In that case, you would use the following Folder and Project settings:

```
< PrtType:VPP >
  Folder = head-office
  Project = monthly-report
```

Overriding the list of libraries for projects

As mentioned before, VIPP uses a configuration file called *xgfunix.run* (stored in the `/usr/xgf/src` directory) to define a list of libraries (directories) for projects.

In the *xgfunix.run* file, you might see a SETPPATH (VIPP command) that looks like this:

```
[ (/usr/xgfc/$$FOLDER./$$PROJECT./)    % project local paths
  (/usr/xgfc/$$FOLDER./fshared/)        % project folder shared paths
  (/usr/xgfc/gshared/)                  % project global shared paths
  (/usr/xgfc/fontlib/)                  % project access to font lib
  (/usr/xgf/encoding/)                  % project access to standard encoding
  (/usr/xgf/gshared/)                  % project global shared path
  (/opt/XXnps/resources/ps/mislib/)     % project access to DocuSP
resource list
] SETPPATH
```

If you wanted to override the list of project paths with a different set, you can do so by using a series of ProjectPath INI options. Each ProjectPath option defines a path that will be used to define a SETPPATH command that overrides the one found in the *xgfunix.run* file found on the VIPP controller. The path defined by the first ProjectPath option will be the first directory searched for VIPP resources. If the resource is not found, the path defined by the second ProjectPath will be searched next (and so on).

The following ProjectPath settings would produce the same list of paths as described earlier:

```
< PrtType:VPP >
  ProjectPath = /usr/xgfc/$$FOLDER./$$PROJECT./
  ProjectPath = /usr/xgfc/$$FOLDER./fshared/
  ProjectPath = /usr/xgfc/gshared/
  ProjectPath = /usr/xgfc/fontlib/
  ProjectPath = /usr/xgf/encoding/
  ProjectPath = /usr/xgf/gshared/
  ProjectPath = /opt/XXnps/resources/ps/mislib/
```

When defining your own list of project paths, keep in mind:

- In the Local scope category, `$$PROJECT` must immediately follow `$$FOLDER`.

- A path containing \$\$PROJECT without \$\$FOLDER is not allowed.
- If present, \$\$FOLDER and \$\$PROJECT must appear only once in each path.
- No additional path components are allowed after \$\$PROJECT.
- A path ending by \$\$FOLDER is invalid.
- There must be at least one path for each category.
- There may be several paths in each category but they must be defined and grouped by category (local, folder, global) in the SETPPATH list.
- A folder or project name must appear only once in the trees of directories covered by SETPPATH.
- When a resource is present with the same name in more than one scope, the order of precedence is: local, folder, global.
- To improve cross-platform portability, Xerox recommends that FOLDER and PROJECT names do not contain more than 32 characters, and only use the characters “a” to “z”, “o” to “9”, “.” (dot), “-” (dash) and “_” (underscore).

Setting up paper trays

The type of media (paper) stored in each paper tray needs to be defined in terms of its MediaType, MediaColor, and MediaWeight.

The MediaType can be named Plain, Transparency, Drilled, and so on

The MediaColor can be any color such as White, Green, Blue, GoldenRod, and so on

The MediaWeight is measured in grams per square meter. Usually, the media weight is set to 75 g/m² (equivalent to 20 lb. paper).

When designing your form set, you may have specified that certain forms use a specific paper tray to make sure the proper paper (pre-printed forms, colored paper, perforated paper, and so on) was used.

To make sure these forms print on the desired type of paper, you must define a unique MediaType, MediaColor, and MediaWeight combination for the paper tray. This information must be set up on both the printer and in the TRAY# INI settings in your VIPP printer control group.

For example, let's say that on your printer, you defined a type of paper will be stored in TRAY1 as having a MediaType of Plain, a MediaColor of Green, and a MediaWeight of 75 g/m².

For your form set to print from that paper tray, you would add the following INI option to your VIPP printer control group:

```
< PrtType:VPP >
  Tray1 = Plain:Green:75
```

The Tray# INI settings expect a string in the form of:

```
MediaType:MediaColor:MediaWeight
```

You can specify any of the media attributes as null or omit them. When any of the media attributes are omitted or specified as null, those attributes are ignored in the following media selections. This example ignores *MediaType*.

```
Tray1 = null:Green:75
```

If the trailing media attributes are omitted, you can omit the trailing colon (:), as shown in this example:

```
Tray2 = Plain::  
or  
Tray2 = Plain:  
or  
Tray2 = Plain
```

When any of the media attributes such as type, color, or weight are omitted, the last specification or the default value for that attribute remains in effect. Because it may be difficult for you to know the value of the attribute that remains in effect, omitting or media attributes as null should be used with caution.

Finally, the TRAY# INI settings can also be specified with just a tray number from 1 to 9. For example, Tray5=1 maps output for tray 5 to tray 1. The system checks the INI option for overriding Tray1 before it checks the setting for Tray2 and so on.

Because of this, do not specify a tray number *less than* the tray you are overriding. For example, you should not use a setting of Tray5=6.

```
< PrtType:VPP >  
Tray1 = Plain:White:75  
Tray2 = Plain:Yellow:75  
Tray3 = Plain:Pink:75  
Tray4 = Drilled:White:75  
Tray5 = 1  
Tray6 = 1
```

Adding DSC comments

For paper tray selection to work properly on DocuPrint NPS printers, it may be necessary to also include some DSC comments at the beginning of your VIPP print stream.

Use the DSCHeaderComment INI option to specify PostScript Document Structure

Convention (DSC) comments you want added to the header portion of the generated VIPP print stream. You can include as many DSCHeaderComment options as are necessary.

This example shows how, in addition to specifying media commands in the Tray# options, you can also include DSC comments you want added to the header portion of the generated VIPP print stream:

```
< PrtType:VPP >  
DSCHeaderComment = %%DocumentMedia:Media1 612 792 75 (White) (Plain)  
DSCHeaderComment = %%+ Media2 612 792 75 (Yellow) (Plain)  
DSCHeaderComment = %%+ Media3 612 792 75 (Pink) (Plain)  
DSCHeaderComment = %%+ Media4 612 792 75 (White) (Drilled)  
Tray1 = Plain:White:75  
Tray2 = Plain:Yellow:75  
Tray3 = Plain:Pink:75  
Tray4 = Drilled:White:75
```

The form of the DocumentMedia DSC comment is:

```
% Key: <Tag Name> <Width> <Height> <Weight> <Color> <Type>
```

Item	Description
Tag Name	Any unique name, ignored by VIPP
Width	The width of paper stock, measured in 1/72" units
Height	The height of paper stock, measured in 1/72" units
Color	The color of paper stock. You can enter any alphanumeric string.
Type	The type of paper stock. You can enter any alphanumeric string.

The DSC header comments are added at the beginning of the generated VIPP print stream, as shown here:

```
%!
%%Title: INSURED
%%Creator: Documaker VIPP Driver
%%CreationDate: Wed Jul 13 11:55:34 2005

%%DocumentMedia:Media1 612 792 75 (White) (Plain)
%%+ Media2 612 792 75 (Yellow) (Plain)
%%+ Media3 612 792 75 (Pink) (Plain)
%%+ Media4 612 792 75 (White) (Drilled)
%%EndComments
```

VIPP Limitations

The VIPP language does not support Unicode. As a result, the VIPP print driver can not be used as a Unicode print driver.

Troubleshooting

Here are some troubleshooting scenarios:

Scenario 1 A VIPP job stops printing before the last page with the following error message:

```
ERROR: VIPP_unable_to_locate; OFFENDING COMMAND: filename.ext  
Flushing: rest of job (to end-of-file) will be ignored
```

Where *filename.ext* is the name of a VIPP resource file.

This error occurs if the VIPP print job references a VIPP resource file (PostScript font, font encoding table, VIPP segment overlay, VIPP bitmap image) that cannot be found.

Make sure you have loaded the missing file onto the VIPP controller and placed it in a folder defined for your VIPP project. See [Managing VIPP Resources on page 329](#) for more information.

Scenario 2 A VIPP job stops printing before the last page, usually with the following error message:

```
ERROR: undefined  
OFFENDING COMMAND: Selected pages 0 n
```

Where *n* is the page volume limit for that device.

If VIPP is installed without a production license file, then the VIPP program will run in demonstration mode. Demonstration mode is a full-featured version of the VIPP software, however page volume limitations are imposed. The page volume limits are device-dependant and varies between 10 and 200 pages.

On some DocuColor printers, the error does not appear. Instead, jobs simply stop when the demonstration limit is reached. The limit is 57 or 200 pages and depends upon the DocuColor printer model.

Contact your Xerox representative about getting a VIPP license to run VIPP in full production mode.

Scenario 3 If you are not getting the correct characters printing, check the code page setting in the FXR file for the font. For most fonts that use the Windows code page, the code page setting in the font record should be set to 1004.

VIPP known problems

At the time this documentation was written, version 5.3 was the latest version of VIPP. Here are some known problems with VIPP version 5.3:

- When caching is used in a VIPP print job, some VIPP segments and images may not print in the correct location or at all, or may cause a fatal system error on the printer. This is a known issue on some printers, such as older model DocuTech and DocuPrint printers. You can remove the CacheFiles INI option and reproduce your print job without using caching.

Or, you can open a console window on the printer's workstation, login with root access and type (or ask your Xerox analyst or engineer to do so):

```
/opt/XXNps/bin/setimagepath -f 0
```

This will disable VIPP caching for all print jobs.

- There is a VIPP bug when using a vector object to draw a circle and the line width exceeds a certain size (noticeable at 1/6 inch or higher). The outside edge of the circle does not draw completely around the border of the circle. The Xerox says it will be fixed in the next VIPP release (after version 5.3).
- There is a problem when using Univers Condensed Bold and Italic fonts on DocuPrint or DocuTech 65 printers. When printing a line of text using the Univers Condensed Bold font followed by a second line of text using the Univers Condensed Italic font, some of characters in the second line may print using the Univers Condensed Bold font (instead of the Univers Condensed Italic font). This bug reported to Xerox but will not be fixed.

NOTE: The SPAR problem was analyzed by Xerox's VIPP and DocuSP development staffs who determined the problem lies in the Adobe PS decomposer. The problem was tested against the latest DT/DP75/90 product release and the fonts printed correctly, indicating the problem has been corrected by Adobe. Unfortunately, the DT65 is, according to Xerox, at its end of life and no further software support will be provided for this product.

EMAILING A PRINT FILE

The system lets you set up an RTF (Rich Text Format) print driver which lets you create a print-ready file that you can email to another user. The recipients can immediately print the file.

NOTE: If you have the Internet Document Server, you can also use the included PDF print driver to create print-ready files you can email.

You install the email print driver (EPTLIB) by setting up INI options so the system will know how to use the driver. Since EPTLIB is essentially a *wrapper* for a real print driver, the INI options must also include a reference to the actual print driver the system will use to create the print-ready file, such as the PDFLIB or PCLLIB. There are also INI options for the email processing, in addition to the regular email INI options.

Creating EPTLIB print files for Documaker Workstation

The INI options for EPTLIB are as follows:

```
< Printers >
PrtType = EPT
```

This option lets the system know that EPTLIB is a print driver so it will include it on the Print window when you print from Documaker Workstation.

You can use the PrtType:EPT control group to further customize the email print driver. For instance, you can add subject and message information and use the email address book when printing from Documaker Workstation using the EPT print driver. This lets you select print, choose form set (form or page), then select the EPT print type.

The system would then display the email address book. You select the recipients and a window appears into which you can enter the subject and message text. You then choose to send or cancel the message.

Here is an example of the INI options you would set up:

```
< PrtType:EPT >
Device      =
Filename    = EPTFILE.RTF
InitFunc    = EPTInit
KeepFile    = No
Message     = Please respond ASAP
Module      = EPTW32
PrintFunc   = EPTPrint
PrtType     = RTF
RecipFunc   = CSTSetMailRecipgvm
RecipMod    = CSTW32
Recipient   =
Subject     = New Application
TermFunc    = EPTTerm
KeepFile    = No
```


Creating EPTLIB print files for Documaker Server

Set up your INI options as shown here:

```
< Printer >
  PrtType = EPT
< PrtType:EPT >
  Module = EPTW32
  PrintFunc = EPTPrint
  InitFunc = EPTInit
  TermFunc = EPTTerm
```

These options tell the system which functions to call to execute the printing process.

```
PrtType = RTF
```

This tells the EPTLIB print driver which real print driver to use to create the print-ready file. If omitted, it defaults to the RTF print driver (Rich Text Format).

```
FileName = EPTFILE.RTF
```

This option gives the name of the output file to create. This is only used if the Device Name field is empty in the GUI print window (the batch file name is used for GenPrint). If the device name is empty and the FileName option is omitted, a temporary file name is used. Use a file name with an extension that matches the print driver type, such as *RTF*. For GenPrint, the file name is the name of the print batch.

```
KeepFile = No
```

The KeepFile option tells EPTLIB whether or not to keep the output file after it has been emailed. The default is No.

```
< Print >
  CallbackFunc = MultiFilePrint
  MultiFileLog = data\rtflog.dat
```

These options tell the system to divide large RTF files into smaller RTF files. If you omit these options, you will be able to view the first transaction, but not the following ones.

The RTFLOG.DAT file stores the information that defines which RTF file contains which transaction for which batch.

```
Recipient = Email Recipient
Subject = File from Documaker User
Message = PDF file attached
```

Use these INI options to set mail settings for EPTLIB. The Subject and Message options specify the Subject line and Message text for the email message. For the Recipient option, you can either include the actual email recipient or you can specify a field name where the system can go to look up the recipient. Here are some examples:

```
Recipient = Stephen Petersen; send to internal email recipient
Recipient = spetersen@oracle.com; send to Internet email address
Recipient = Fieldname:ADDRESS2; use text in ADDRESS2 field
```

If the email system cannot resolve recipients, or if you leave the Recipient option blank, an email address window appears so you can select an email address from the address book. The field lookup is a feature of the default recipient function in EPTLIB, which you can replace using these INI options:

```
RecipMod = CSTW32
RecipFunc = CSTSetMailRecip
```

These options tell the system which module and function to use to determine the recipient. Omit these options and the system uses EPTLIB's default recipient function.

The `CSTSetMailRecip` function displays a window which shows the subject and message text and lets you edit this text. This window also lets you provide the email recipient for Documaker Workstation. Documaker Server lets you use these functions to set up recipients:

```
RecipMod = CUSW32
RecipFunc = CUSSetMailRecip
```

or

```
RecipFunc = CUSSetMailRecipGVM
```

Function	Description
<code>CUSSetMailRecip</code>	<p>This function finds the print recipient and looks up the recipient in the <code>RECIP_MAIL</code> control group to get the email address of the recipient. Here is an example:</p> <pre>< RECIP_MAIL > AGENT = myagent@sampco.com COMPANY = support@sampco.com</pre>
<code>CUSSetMailRecipGVM</code>	<p>This function finds the recipient in a global variable, the name of which is defined in this INI option:</p> <pre>< PrtType:EPT > Recipient = EAddress</pre> <p>Instead of using <code>EAddress</code> as the recipient name, the system uses it as the variable name to look up to find the recipient name. This global variable can have any name.</p>

The recipient functions have the following syntax:

```
DWORD _VMMAPI EPTDefSetRecipient(VMMHANDLE objectH,
                                char FAR * recip,
                                size_t len);
```

Parameter	Description
<code>objectH</code>	The object being printed (form set, form, or page)
<code>recip</code>	The recipient buffer
<code>len</code>	Length of the buffer, currently 80 characters

The return value should be `SUCCESS` or `FAILURE`. If `FAILURE`, then the message is not sent and `FAILURE` is returned from `EPTPrint`. To set the recipient function without INI options, use the `EPTSetRecipFunc` function:

```
EPTRECIPFUNC _VMMAPI EPTSetRecipFunc(EPTRECIPFUNC newfunc);
```

Call it with the address of the recipient function:

```
EPTSetRecipFunc(func);
```

The `EPTSetRecipFunc` function returns the previous installed function, which can be used to set it back.

Creating PDF print files

If you are creating PDF files, use these INI options:

```
< Printers >
  PrtType      = PDF
< PrtType:PDF >
  Module       = PDFW32
  PrintFunc    = PDFPrint
```

Keep in mind that when the PDF driver is called from the EPT driver, the current printer control group remains PrtType:EPT, not PrtType:PDF. Therefore, unless you add PDF-specific options, the system uses the INI settings it finds for PrtType:EPT.

Many print options, such as the DownloadFonts option, are set before the system calls EPT, which then redirects the print to another driver. So, to have the system use the correct PDF options, set your PrtType:EPT control group to look like this:

```
< PrtType:EPT >
  PrtType      = PDF
  DownloadFonts = [PrtType:PDF] DownloadFonts =
  SendColor    = [PrtType:PDF] SendColor =
```

This way, if you change the options in the PrtType:PDF control group, those changes are automatically picked up in the PrtType:EPT control group.

Overriding attached files

Keep in mind that the EPT (email print) driver can use the FSRSetFileAttachment API. This lets you create custom hooks to override the attached file and handle situations where you need to remove the attached file but still send the message.

Using email aliases

Multiple recipient addresses are not supported with the EPT PrtType. If you need to send an email to, for instance, all agents, use an Email Application Server, such as Microsoft Exchange (MailType = MSM) or ccMail (MailType = CCM). With these products you can define an alias to represent a group of email addresses. You cannot set the MailType option to SMTP unless your SMTP server understands aliases.

Email Application Servers usually run on top of an SMTP service and let you manage email messaging more efficiently. When using an application such as Exchange, you can create a group (such as *TestGroup*) and you can specify the group name when you specify the Recipient option.

For example, if you set the MailType option to MSM in the Mail control group and you have this defined for the Recipient option:

```
< PrtType:EPT >
  Recipient = TestGroup
```

This option is sent to the Exchange server which converts the alias (*TestGroup*) into its SMTP equivalent value, such as a list of email address similar to this:

```
hbean@oracle.com;jgaramond@oracle.com;tbottle@oracle.com...
```

The result is a message sent to the entire group represented by *TestGroup*.

NOTE: To use this feature, you must set up email-related INI options. These options are discussed in the [Documaker Workstation Supervisor Guide](#).

CHOOSING THE PAPER SIZE

The system supports a variety of paper sizes including US and international sizes. The following tables show the paper sizes you can choose from:

- [US Standard Sizes on page 347](#)
- [ISO Sizes on page 348](#)
- [Japanese Standard Sizes on page 351](#)

You can also find the following related information in this topic:

- [Printer Support for Paper Sizes on page 352](#)
- [Paper Sizes for AFP Printers on page 356](#)

NOTE: Please note that the NA file stores the actual section height and width for custom sized sections. This information is stored in the SIZE entry in the NAFILE.DAT file. Here is an example:

```
\NA=q1snam, LN=1, DUP=LB, SIZE=3360x18600, TRAY=U, X=600, Y=600 . . .
```

The height and width are in FAP units (2400 per inch).

In Studio you use the Size property to specify the page size for a section. There is also a Size property at the form level.

For a section, you can choose from the available standard page sizes or choose Custom here.

The screenshot shows the 'Properties' dialog box with the 'Section Options' tab selected. Under the 'Paper' section, the 'Size' property is set to 'Custom'. Other properties like 'Height' (3600 FAPs), 'Width' (20415 FAPs), 'Orientation' (Portrait), and 'Auto size' are also visible. A blue arrow points from the text 'For a section, you can choose from the available standard page sizes or choose Custom here.' to the 'Size' property.

If, for a section, you choose Custom, the system defaults to the size of paper that will best contain the custom section, but you must tell it what paper is installed on your printer. For sections small enough to fit on letter size paper, the system defaults to letter.

NOTE: This affects section printing from Documaker Studio and Image Editor but has no effect on Form Set Manager or Form (FOR) definitions.

US STANDARD SIZES

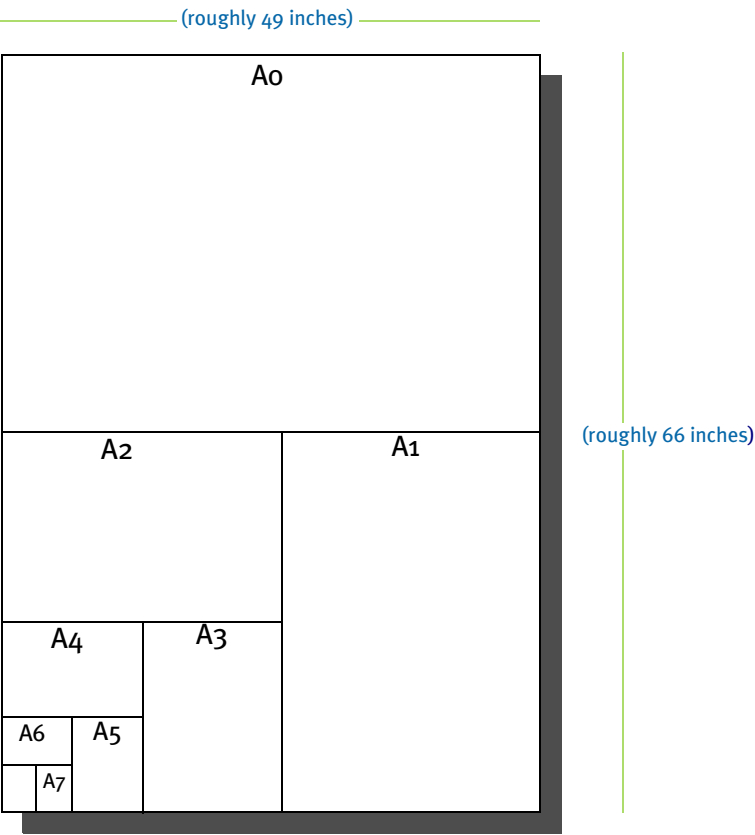
These paper sizes are commonly used in the United States and Canada. The height and width are in FAP units (2400 per inch), millimeters, and inches. The inch dimensions are approximate.

Name	Code	Width x Height		
		FAP units	Millimeters	Inches (approximate)
US letter	0	20400 x 26400	216 × 279	8½ x 11
US legal	1	20400 x 33600	216 × 356	8½ x 14
US executive	3	17400 x 25200	190 × 254	7¼ 10½
US ledger	4	40800 x 26400	432 x 279	17 x 11
US tabloid	5	26400 x 40800	279 × 432	11 x 17
US statement	6	13200 x 20400	140 x 216	5½ x 8½
US folio	7	20400 x 31200	216 x 330	8½ x 13
US fanfold	8	35700 x 26400	378 x 279	14⅞ x 11
Custom	98	any x any	any x any	any x any

ISO SIZES

The International Organization for Standardization (ISO) paper sizes, which are based on the earlier Deutsche Industrie Norm (DIN) sizes, are used throughout the world except in Canada, the United States, and Japan. There are three main series of paper sizes: A, B, and C.

ISO A sizes The A series of sizes are typically used for correspondence, books, brochures, and other printed materials. This diagram shows most of the various A sizes. The height and width are in FAP units (2400 per inch), millimeters, and inches. The inch dimensions are approximate.



Width x Height				
Name	Code	FAP units	Millimeters	Inches (approximate)
ISO Ao	20	79464 x 112345	841 x 1189	33 ³ / ₈ x 46 ¹ / ₄
ISO A1	21	56125 x 79464	594 x 841	23 ³ / ₈ x 33 ³ / ₈
ISO A2	22	39685 x 56125	420 x 594	16 ¹ / ₂ x 23 ³ / ₈
ISO A3	23	28063 x 39685	297 x 420	11 ³ / ₄ x 16 ¹ / ₂

Name	Code	Width x Height		
		FAP units	Millimeters	Inches (approximate)
ISO A4	2	19842 x 28063	210 x 297	8 $\frac{1}{4}$ x 11 $\frac{3}{4}$
ISO A5	25	13984 x 19842	148 x 210	5 $\frac{7}{8}$ x 8 $\frac{1}{4}$
ISO A6	26	9921 x 13984	105 x 148	4 $\frac{1}{8}$ x 5 $\frac{7}{8}$
ISO A7	27	6992 x 9921	74 x 105	2 $\frac{7}{8}$ x 4 $\frac{1}{8}$
ISO A8	28	4913 x 6992	52 x 74	2 x 2 $\frac{7}{8}$
ISO A9	29	3496 x 4913	37 x 52	1 $\frac{1}{2}$ x 2
ISO A10	30	2457 x 3496	26 x 37	1 x 1 $\frac{1}{2}$
ISO 2A	32	112345 x 158927	1189 x 1682	46 $\frac{3}{4}$ x 66 $\frac{1}{4}$
ISO 4A	34	158927 x 224690	1682 x 2378	66 $\frac{1}{4}$ x 93 $\frac{5}{8}$

ISO B sizes

The B series of sizes are designed primarily for posters, wall charts, and similar items where the difference between each A size represents too large a jump. The height and width are in FAP units (2400 per inch), millimeters, and inches. The inch dimensions are approximate.

Name	Code	Width x Height		
		FAP units	Millimeters	Inches (approximate)
ISO B0	40	94487 x 133605	1000 x 1414	39 $\frac{1}{8}$ x 55 $\frac{1}{8}$
ISO B1	41	66802 x 94487	707 x 1000	27 $\frac{7}{8}$ x 39 $\frac{1}{8}$
ISO B2	42	47244 x 66802	500 x 707	19 $\frac{5}{8}$ x 27 $\frac{7}{8}$
ISO B3	43	33354 x 47244	353 x 500	13 $\frac{7}{8}$ x 19 $\frac{5}{8}$
ISO B4	44	23622 x 33354	250 x 353	9 $\frac{7}{8}$ x 13 $\frac{7}{8}$
ISO B5	45	16630 x 23622	176 x 250	7 x 9 $\frac{7}{8}$
ISO B6	46	11811 x 16630	125 x 176	5 x 7
ISO B7	47	8315 x 11811	88 x 125	3 $\frac{1}{2}$ x 5
ISO B8	48	5858 x 8315	62 x 88	2 $\frac{1}{2}$ x 3 $\frac{1}{2}$
ISO B9	49	4157 x 5858	44 x 62	1 $\frac{3}{4}$ x 2 $\frac{1}{2}$
ISO B10	50	2929 x 4157	31 x 44	1 $\frac{1}{4}$ x 1 $\frac{3}{4}$

Name	Code	Width x Height		
		FAP units	Millimeters	Inches (approximate)
ISO 2B	52	133605 x 188974	1414 x 2000	55 ³ / ₄ x 78 ³ / ₄
ISO 4B	54	188974 x 267209	2000 x 2828	78 ³ / ₄ x 111 ¹ / ₄

ISO C sizes

The C series of sizes are designed for making envelopes and folders to take the A series of sizes. The height and width are in FAP units (2400 per inch), millimeters, and inches. The inch dimensions are approximate.

Name	Code	Width x Height		
		FAP units	Millimeters	Inches (approximate)
ISO Co	60	86645 x 122550	917 x 1297	36 ¹ / ₈ x 51
ISO C1	61	61228 x 86645	648 x 917	25 ¹ / ₂ x 36
ISO C2	62	43275 x 61228	458 x 648	18 x 25 ¹ / ₂
ISO C3	63	30614 x 43275	324 x 458	12 ³ / ₄ x 18
ISO C4	64	21638 x 30614	229 x 324	9 x 12 ³ / ₄
ISO C5	65	15307 x 21638	162 x 229	6 ³ / ₈ x 9
ISO C6	66	10772 x 15307	114 x 162	4 ¹ / ₂ x 6 ³ / ₈
ISO C7	67	7653 x 10772	81 x 114	3 ¹ / ₄ x 4 ¹ / ₂
ISO C8	68	5386 x 7653	57 x 81	2 ¹ / ₄ x 3 ¹ / ₄
ISO C9	69	3779 x 5386	40 x 57	1 ⁵ / ₈ x 2 ¹ / ₄
ISO C10	70	2646 x 3779	28 x 40	1 ¹ / ₈ x 1 ⁵ / ₈
ISO DL	71	10394 x 20787	110 x 220	4 ¹ / ₃ x 8 ² / ₃

The DL size is for a sheet 1/3 of the A4 size. This is the most common size of envelope.

JAPANESE STANDARD SIZES

Japan has its own standard paper sizes, called the Japan Industrial Standard (JIS). The JIS A series is identical in size to the ISO A series. The JIS B series, however, does not match the ISO B series. There is no equivalent to the ISO C series. This table shows the JIS paper sizes. The height and width are in FAP units (2400 per inch), millimeters, and inches. The inch dimensions are approximate.

Name	Code	Width x Height		
		FAP units	Millimeters	Inches (approximate)
JIS B0	80	97322 x 137573	1030 x 1456	40½ x 57¼
JIS B1	81	68787 x 97322	728 x 1030	28¾ x 40½
JIS B2	82	48661 x 68787	515 x 728	20¼ x 28¾
JIS B3	83	34393 x 48661	364 x 515	14¼ x 20¼
JIS B4	84	24283 x 34393	257 x 364	10⅞ x 14¼
JIS B5	85	17197 x 24283	182 x 257	7¼ x 10⅞
JIS B6	86	12094 x 17197	128 x 182	5 x 7¼
JIS B7	87	8598 x 12094	91 x 128	3½ x 5
JIS B8	88	6047 x 8598	64 x 91	2½ x 3½
JIS B	89	4252 x 6047	45 x 64	1¾ x 2½
JIS B10	90	3024 x 4252	32 x 45	1¼ x 1¾

PRINTER SUPPORT FOR PAPER SIZES

This table outlines the various paper sizes supported by the different print drivers. The table includes information for the PDF, RTF, HTML, Metacode, PCL 5, PCL 6, GDI, PostScript, and AFP print drivers. The PDF, RTF, HTML, and Metacode print drivers support all paper sizes.

Paper size	PDF, RTF, HTML, and Metacode	PXL ¹	PCL ²	GDI ²	PST ³	AFP ⁴
US letter	X	X	X	X	X	X
US Legal	X	X	X	X	X	X
US executive	X	X	X	X	X	X
US ledger	X	X	X	X	X	X
US tabloid	X	Y	US letter	X	X	X
US statement	X	JIS B5	US executive	X	X	X
US folio	X	US legal	US legal	X	X	X
US fanfold	X	US ledger	US ledger	X	X	X
ISO 4A	X	Y	US letter	US letter	US letter	C
ISO 2A	X	Y	US letter	US letter	US letter	C
ISO A0	X	Y	US letter	US letter	X	C
ISO A1	X	Y	US letter	US letter	X	C
ISO A2	X	Y	US letter	US letter	X	C
ISO A3	X	X	X	X	X	X

Sizes marked with an *X* are fully supported by the corresponding driver.

Sizes marked with a *Y* are supported by sending the paper dimensions in millimeters to the printer.

Sizes that refer to another size substitute the referred size when *paper size matching* is turned on.

If paper size matching is not turned on, the behavior depends upon the specific driver. To turn on paper size matching, use this INI option:

```
< PrtType:XXX >
    PaperSizeMatching = Yes
```

¹ When paper size matching is not turned on, the PCL 6 (PXL) driver sends the paper dimensions in millimeters to the printer.

² When paper size matching is not turned on, these drivers substitute US letter.

³ This driver does not use paper size matching. US letter is substituted for the unsupported paper sizes

⁴ Sizes marked with a *C* are supported, but are commented out of the AFP formdef source file called F1FMMST.DAT, See [Paper Sizes for AFP Printers on page 356](#) for more information.

Paper size	PDF, RTF, HTML, and Metacode	PXL ¹	PCL ²	GDI ²	PST ³	AFP ⁴
ISO A4	X	X	X	X	X	X
ISO A5	X	X	X	X	X	X
ISO A6	X	X	X	X	X	X
ISO A7	X	ISO A6	ISO C5	ISO A6	X	C
ISO A8	X	ISO A6	ISO C5	ISO A6	X	C
ISO A9	X	ISO A6	ISO C5	ISO A6	X	C
ISO A10	X	ISO A6	ISO C5	ISO A6	X	C
ISO 4B	X	Y	US letter	US letter	US letter	C
ISO 2B	X	Y	US letter	US letter	US letter	C
ISO B0	X	Y	US letter	US letter	X	C
ISO B1	X	Y	US letter	US letter	X	C
ISO B2	X	Y	US letter	US letter	X	C
ISO B3	X	Y	US letter	US letter	X	C
ISO B4	X	JIS B4	US ledger	X	X	X
ISO B5	X	JIS B5	X	X	X	X
ISO B6	X	JIS B6	ISO C5	X	X	X
ISO B7	X	ISO A6	ISO C5	ISO A6	X	C
ISO B8	X	ISO A6	ISO C5	ISO A6	X	C

Sizes marked with an X are fully supported by the corresponding driver.

Sizes marked with a Y are supported by sending the paper dimensions in millimeters to the printer.

Sizes that refer to another size substitute the referred size when *paper size matching* is turned on. If paper size matching is not turned on, the behavior depends upon the specific driver. To turn on paper size matching, use this INI option:

```
< PrtType:XXX >
    PaperSizeMatching = Yes
```

¹ When paper size matching is not turned on, the PCL 6 (PXL) driver sends the paper dimensions in millimeters to the printer.

² When paper size matching is not turned on, these drivers substitute US letter.

³ This driver does not use paper size matching. US letter is substituted for the unsupported paper sizes

⁴ Sizes marked with a C are supported, but are commented out of the AFP formdef source file called F1FMMST.DAT, See [Paper Sizes for AFP Printers on page 356](#) for more information.

Paper size	PDF, RTF, HTML, and Metacode	PXL ¹	PCL ²	GDI ²	PST ³	AFP ⁴
ISO B9	X	ISO A6	ISO C5	ISO A6	X	C
ISO B10	X	ISO A6	ISO C5	ISO A6	X	C
ISO Co	X	Y	US letter	US letter	X	C
ISO C1	X	Y	US letter	US letter	X	C
ISO C2	X	Y	US letter	US letter	X	C
ISO C3	X	Y	US letter	X	X	C
ISO C4	X	JIS B4	US ledger	X	X	C
ISO C5	X	X	X	X	X	C
ISO C6	X	JIS B6	ISO C5	X	X	C
ISO C7	X	ISO A6	ISO C5	ISO A6	X	C
ISO C8	X	ISO A6	ISO C5	ISO A6	US letter	C
ISO C9	X	ISO A6	ISO C5	ISO A6	US letter	C
ISO C10	X	ISO A6	ISO C5	ISO A6	US letter	C
ISO DL	X	X	X	X	X	X
JIS B0	X	Y	US letter	US letter	X	C
JIS B1	X	Y	US letter	US letter	X	C
JIS B2	X	Y	US letter	US letter	X	C
JIS B3	X	Y	US letter	US letter	X	C

Sizes marked with an X are fully supported by the corresponding driver.

Sizes marked with a Y are supported by sending the paper dimensions in millimeters to the printer.

Sizes that refer to another size substitute the referred size when *paper size matching* is turned on. If paper size matching is not turned on, the behavior depends upon the specific driver. To turn on paper size matching, use this INI option:

```
< PrtType:XXX >
    PaperSizeMatching = Yes
```

¹ When paper size matching is not turned on, the PCL 6 (PXL) driver sends the paper dimensions in millimeters to the printer.

² When paper size matching is not turned on, these drivers substitute US letter.

³ This driver does not use paper size matching. US letter is substituted for the unsupported paper sizes

⁴ Sizes marked with a C are supported, but are commented out of the AFP formdef source file called F1FMMST.DAT, See [Paper Sizes for AFP Printers on page 356](#) for more information.

Paper size	PDF, RTF, HTML, and Metacode	PXL ¹	PCL ²	GDI ²	PST ³	AFP ⁴
JIS B4	X	X	X	US fanfold	X	X
JIS B5	X	X	X	X	X	X
JIS B6	X	X	X	X	X	X
JIS B7	X	ISO A6	ISO C5	ISO A6	X	C
JIS B8	X	ISO A6	ISO C5	ISO A6	X	C
JIS B9	X	ISO A6	ISO C5	ISO A6	X	C
JIS B10	X	ISO A6	ISO C5	ISO A6	X	C

Sizes marked with an X are fully supported by the corresponding driver.

Sizes marked with a Y are supported by sending the paper dimensions in millimeters to the printer.

Sizes that refer to another size substitute the referred size when *paper size matching* is turned on. If paper size matching is not turned on, the behavior depends upon the specific driver. To turn on paper size matching, use this INI option:

```
< PrtType:XXX >
    PaperSizeMatching = Yes
```

¹ When paper size matching is not turned on, the PCL 6 (PXL) driver sends the paper dimensions in millimeters to the printer.

² When paper size matching is not turned on, these drivers substitute US letter.

³ This driver does not use paper size matching. US letter is substituted for the unsupported paper sizes

⁴ Sizes marked with a C are supported, but are commented out of the AFP formdef source file called F1FMMST.DAT, See [Paper Sizes for AFP Printers on page 356](#) for more information.

PAPER SIZES FOR AFP PRINTERS

The AFP formdef source file (F1FMMST.DAT) contains support for the following paper sizes, but since this file contains support for so many paper sizes, its size could affect printer performance. To limit the effect, some of the paper sizes are commented out, as shown in this table:

Size	Commented out?
Letter	No
Legal	No
Executive	No
Ledger	Yes
Tabloid	Yes
Statement	Yes
Folio	Yes
Fanfold	Yes
ISO A3	Yes
ISO A4	No
ISO A5	Yes
ISO A6	Yes
ISO B4	Yes
ISO B5	Yes
ISO B6	Yes
ISO DL	Yes
JIS B4	Yes
JIS B5	Yes
JIS B6	Yes

NOTE: The F1FMMST.DAT and F1FMMST.FDF files can be found in the FMRES master resource library (MRL).

The commented source line begins with an asterisk (*). To add support for another paper size, you open the F1FMMST.DAT file and delete the asterisk at the beginning of each line that references the paper size you want to add.

Because the AFP formdef is composed on medium map names that specify page orientation, paper size, tray selection, and duplex settings, there are 31 groups of medium map settings. Each of these groups contains the 57 possible paper sizes. So, for each paper size you add, there are 31 sources lines you must *uncomment* to fully support a paper size for all orientations, trays, and duplex settings.

After you uncomment the lines that reference the paper size you want to add, run the AFPFMDEF utility to rebuild your AFP formdef file with the new information. For more information on this utility, see the [Docutoolbox Reference](#).

CREATING PRINT STREAMS FOR DOCUSAVE

Docusave can archive AFP, Metacode, and PCL print streams that are in a Docusave-compatible format and contain special records used to index the archive.

For AFP and Metacode, you use the OutMode option in the PrtType:AFP or XER control group to tell the GenPrint program to create a Metacode or AFP print stream in a Docusave-compatible record format. You can choose between these Docusave-compatible formats: JES2 and MRG4.

For PCL, the process is similar but there is not OutMode option to set. You include comment records in the print streams to index the archive. You can use a DAL script to add those comment records.

For details, see...

- [Archiving AFP Print Streams on page 358](#)
- [Archiving Metacode Print Streams on page 359](#)
- [Archiving PCL Print Streams on page 360](#)

ARCHIVING AFP PRINT STREAMS

Set the OutMode option to MRG4 to produce a print stream for Docusave from non-z/OS platforms.

Here is an example:

```
< PrtType:AFP >  
    OutMode = MRG4
```

When you set the OutMode option to MRG4, the GenPrint program creates print stream records with a 4-byte sequence that precedes them. This sequence defines the record lengths. Records are grouped into blocks with one or more records in each block. Both records and blocks have a 4-byte sequence that precedes them, defining their length.

These length indicators are formed by taking the high-order byte of length followed by the low-order byte of length followed by two bytes of zeros.

The maximum number that can be displayed is a 16-bit quantity. The value in each includes the length of the structure itself. A one-byte data record in its own block would have five for the record length and nine for the block length. This table shows what a 3-byte record would look like:

Byte offset	Value (Hex)	Meaning
0	00	Block length high-order
1	0B	Block length low-order
2	00	Always 0
3	00	Always 0
4	00	Record length high-order
5	07	Record length low-order
6	00	Always 0

Byte offset	Value (Hex)	Meaning
7	00	Always 0
8	31	'1'
9	32	'2'
10	33	'3'

In addition to using the OutMode option, you must include comment records in the print streams to index the archive. You can use a DAL script to add comment records into the print stream. Use the DocusaveScript option in the PrtType:AFP control group to have the system execute a DAL script at the times when Docusave comments can be added to the print streams.

To add Docusave comments to an AFP print stream, you must add the DocusaveScript option and the name of a DAL script to execute. The DAL script should call the AddDocusaveComment function to add a string as a Docusave comment record. Here is an example:

```
< PrtType:AFP >
  DocusaveScript = Docusave.DAL
  OutMode = MRG4
```

ARCHIVING METACODE PRINT STREAMS

Set the OutMode option to JES2 to produce print streams under z/OS. Here is an example:

```
< PrtType:XER >
  OutMode = JES2
```

When you set the OutMode option to JES2, the GenPrint program creates print stream records that are native to a mainframe environment.

Also include comment records in the print streams to index the archive. You can use a DAL script to add comment records into the print stream. Use the DocusaveScript option in the PrtType:XER control group to have the system execute a DAL script at the times when Docusave comments can be added to the print streams.

To add Docusave comments to a Metacode AFP print stream, add the DocusaveScript option and the name of a DAL script to execute. The DAL script should call the AddDocusaveComment function to add a string as a Docusave comment record. Here is an example:

```
< PrtType:XER >
  DocusaveScript = Docusave.DAL
  OutMode = JES2
```

ARCHIVING PCL PRINT STREAMS

NOTE: Docusave is adding support for archiving PCL 5 print streams. In anticipation of Docusave's PCL archive capability, Documaker version 10.2 and later can produce PCL 5 print streams with the necessary Docusave comment information.

You must include comment records in the print streams to index the archive. You can use a DAL script to add comment records into the print stream. Use the DocusaveScript option in the PrtType:PCL control group to have the system execute a DAL script when Docusave comments can be added to the print stream.

To add Docusave comments to an PCL print stream, add the DocusaveScript option and the name of a DAL script to execute. The DAL script should call the AddDocusaveComment function to add a string as a Docusave comment record.

Here is an example:

```
< PrtType:PCL >  
DocusaveScript = DOCUSAVE.DAL
```

Here is an example of what the DOCUSAVE.DAL file might look like:

```
* Add DocuSave Comment - use default: APPIDX record!  
COMMENT = AppIdxRec()  
PRINT_IT(COMMENT)  
ADDDOCUSAVECOMMENT(COMMENT)  
RETURN('FINISHED!')
```

NOTE: PCL 6 print streams cannot be archived into Docusave.

USING DAL FUNCTIONS

For all types of print streams, you can use these DAL functions to create archive keys to use with Docusave.

Function	Description
AddDocusaveComment	Adds a Docusave comment string to the print stream
AddComent	Adds a comment string to the print stream
AppIdxRec	Gets an archive record based on APPIDX.DFD and Trigger2Archive INI settings
HaveGVM	Verifies if a GVM variable exists
SetGVM	Updates the contents of a GVM variable
GVM	Gets the contents of a GVM variable
MajorVersion	Gets the system's major version number

Function	Description
MinorVersion	Gets the system's minor version number
PrinterClass	Gets the type of print being produced
PrinterGroup	Gets the name of the print group being used
Print_It	Debug tool to print a string to the console

For more information on these functions, see the [DAL Reference](#).

ADDING TLE RECORDS

You can add TLE (Tag Logical Element) records into AFP print streams which can be used by some 3rd-party archive systems to archive AFP print streams in a manner similar to archiving AFP or Metacode print streams in Docusave.

You must include comment records in the print streams to index the archive. You can use a DAL script to add comment records into the print stream. Use the TLEScript option in the PrtType:AFP control group to name the DAL script to execute when TLE records can be added into the print stream. The DAL script should call the AddComment function to add a string as a TLE comment record.

The TLE comment string must include a key and a value. Separate these components with a special character. This character can be any printable character as long as it is a unique character not found in the key or value portion of the comment string.

For example, you might build a comment string using a colon (:) as a separator as in the following example:

```
PolicyNum:7SAMPCO
```

The key portion of the string is *PolicyNum*, the value portion of the string is *7SAMPCO*, and the separator character is a colon (:).

Here is an example of what TLE DAL script might look like:

```
cidlabel = 'PolicyNum'  
clientid = GVM("PolicyNum")  
colon = ':'  
AddComment (cidlabel & colon & clientid);  
RETURN('FINISHED!')
```

Notice that the key portion remains constant (PolicyNum) while the value portion changes based on the contents of the GVM variable, PolicyNum.

Add these options to the PrtType:AFP control group to enable TLE record support:

```
< PrtType:AFP >  
TLEScript    = TLE.DAL  
TLEEveryPage= No  
TLESeparator= :
```

Option	Description
TLEScript	Enter the name of the DAL script to execute.
TLESeparator	Enter the character you want to use to separate the key and value portions of the TLE comment string.
TLEEveryPage	Optional. If you enter Yes, the TLE DAL script will be executed at the start of every page. If you enter No, the TLE DAL script will be executed at the start of every form set. The default is No.

HANDLING MULTIPLE PAPER TRAYS

You can set up PCL, PostScript, GDI, AFP, and Metacode print drivers to support up to nine paper trays. Setting up nine tray printer support for the various types of printers is outlined below.

NOTE: You can also use the Form Set Manager to specify tray settings. See the [Docucreate User Guide](#) for more information.

For PCL printers

You can override PCL tray commands by providing an alternate PCL command to use. Here are the default PCL INI settings:

```
< PrtType:PCL >
  Tray1 = ~&l11H
  Tray2 = ~&l14H
  Tray3 = ~&l15H
  Tray4 = ~&l120H
  Tray5 = ~&l121H
  Tray6 = ~&l122H
  Tray7 = ~&l123H
  Tray8 = ~&l124H
  Tray9 = ~&l125H
```

When writing PCL commands as an INI setting, the tilde (~) is used as a substitute for the PCL escape character (x1B).

For PostScript printers

You can override PostScript tray commands by providing an alternate PostScript command to use. You issue PostScript tray commands in these forms:

- A quoted string containing the PostScript commands. The quoted string should contain the appropriate PostScript commands for selecting a paper tray. Here is an example:

```
Tray1 = "statusdict /lettertray get exec"
```

- A tray number from 1 to 9. You can use tray numbers to map non-existent trays. For example, Tray5=1 maps output for tray 5 to tray 1. The system checks the INI setting for overriding Tray1 before it checks the setting for Tray2 and so on. Because of this, do not specify a tray number *less than* the tray you are overriding. For example, you should not use a setting of Tray5=6.
- A UI keyword from a PPD file. UI keywords represent features that commonly appear in a user interface (UI). They provide the code to invoke a user-selectable feature within the context of a print job, such as the selection of an input tray or manual feed. The entries of UI keywords are surrounded by these structure keywords:

```
*OpenUI/*CloseUI or *JCLOpenUI/*JCLCloseUI
```

Here is an example of an OpenUI structure for MediaColor:

```
*OpenUI *MediaColor: PickOne
*OrderDependency: 30 AnySetup *MediaColor
*DefaultMediaColor: white
*MediaColor white: "1 dict dup /MediaColor (white) put setpagedevice"
*MediaColor clear: "1 dict dup /MediaColor (clear) put setpagedevice"
*MediaColor blue: "1 dict dup /MediaColor (blue) put setpagedevice"
*MediaColor buff: "1 dict dup /MediaColor (buff) put setpagedevice"
*MediaColor green: "1 dict dup /MediaColor (green) put setpagedevice"
```

```
*MediaColor goldenrod: "1 dict dup /MediaColor (goldenrod) put
setpagedevice"
*MediaColor pink: "1 dict dup /MediaColor (pink) put setpagedevice"
*MediaColor yellow: "1 dict dup /MediaColor (yellow) put
setpagedevice"
*?MediaColor: "
save
  currentpagedevice /MediaColor
  {get} stopped
    {pop pop (white)} {dup null eq {pop (white)} if} ifelse
  = flush
  restore
"
*End
*CloseUI: *MediaColor
```

Input media (paper trays) are often selected on PostScript printers by specifying **PageSize**, **MediaColor**, **MediaWeight**, and **MediaType**. In the above example, media (paper) colors were defined for white, clear, blue, and so on. If you wanted to specify that the paper assigned to tray 5 uses blue paper, you could use one of these INI settings:

```
Tray5 = *MediaColor blue:
```

or

```
Tray5 = "1 dict dup /MediaColor (blue) put setpagedevice"
```

The first uses the UI keyword in the PPD file while the second uses the actual PostScript commands in a quoted string. When you use the UI keyword in an INI setting, always include the beginning asterisk (*) and the terminating colon (:).

Here are the default PostScript INI settings:

```
< PrtType:PST >
; UI keyword is used if PPD is specified and keyword is found.
; Otherwise, quoted string is used.
Tray1="0 statusdict /setpapertray get exec" or Tray1=*InputSlot
Upper:
Tray2="1 statusdict /setpapertray get exec" or Tray2=*InputSlot
Lower:
Tray3="2 statusdict /setpapertray get exec" or Tray3=*InputSlot
Manual:
Tray4="3 statusdict /setpapertray get exec" or Tray4=*InputSlot
Envelope:
; Make trays 5 through 9 use the PostScript commands for tray 1
Tray5=1
Tray6=1
Tray7=1
Tray8=1
Tray9=1
```

For GDI printers

You can override the GDI tray commands by specifying an alternate paper tray to use. Here are the default GDI INI settings:

```
< PrtType:GDI >
Tray1 = 1
Tray2 = 2
Tray3 = 3
Tray4 = 4
```

```

Tray5 = 1
Tray6 = 1
Tray7 = 1
Tray8 = 1
Tray9 = 1

```

For AFP printers

You can override the AFP tray commands by specifying an alternate paper tray to use. Here are the default AFP INI settings:

```

< PrtType:AFP >
  Tray1 = 1
  Tray2 = 2
  Tray3 = 3
  Tray4 = 4
  Tray5 = 1
  Tray6 = 1
  Tray7 = 1
  Tray8 = 1
  Tray9 = 1

```

For Metacode printers

You can override the Metacode tray commands by specifying an alternate tray name to use. Here are the default Metacode INI settings:

```

< PrtType:XER >
  Tray1 = MAIN
  Tray2 = AUX
  Tray3 = AUX
  Tray4 = AUX
  Tray5 = AUX
  Tray6 = AUX
  Tray7 = AUX
  Tray8 = AUX
  Tray9 = AUX

```

INCLUDING TRAY SELECTIONS IN A PRINT STREAM BATCH

To include the header with the tray selection in a print stream batch, the first section written or triggered to the batch must have a tray, such as Tray 1 or Tray 2, listed in its FORM.DAT file. Otherwise, the information is not written to that batch print stream. Here is an example of header information from a PostScript print stream that had these INI options:

```

< PrtType:PST >
  Tray1 =*InputSlot Upper:
  Tray2 =*InputSlot Lower:

```

Here is the example header:

```

GenericDict begin
%%BeginSetup
%%BeginFeature: *Duplex
false statusdict /setduplexmode get exec false statusdict /settumble
get exec
%%EndFeature
%%BeginFeature: *InputSlot Upper
0 statusdict /setpapertray get exec
%%EndFeature

```


CHAPTER 7

Setting Up Error Messages and Log Files

This chapter discusses the how the system creates error and log messages and describes how you can customize these messages to meet your company's needs.

In this chapter, you will find information about...

- [Overview on page 368](#)
- [Configuring the Message System on page 369](#)
- [Creating Messages on page 374](#)
- [Using the Message Token File on page 380](#)

OVERVIEW

The message system is enabled by default. Without making any modifications, it is fully functional. Each executed system program (GenTrn, GenData, GenPrint, and so on) appends output messages to the appropriate log or error file.

When an error or log message occurs, the system writes the information to a token file named *MSGFILE.DAT*. A second step converts or translates the output into log and error files, which are typically named *LOGFILE.DAT* and *ERRFILE.DAT*.

By default, this translation step occurs before each program's termination so the system is compatible with earlier versions. You can, however, delay this step and execute it manually using the TRANSLAT utility (see the [Docutoolbox Reference](#) for more information). This lets you translate the message and error information after all system programs have completed their processing cycle for a given batch run.

NOTE: Typically, you will want to use system defaults as you implement your system. This lets you spot errors after each processing step. Once your system is implemented and is running without error, you may want to delay the translation process to improve performance. See [Controlling the Translation Process on page 372](#) for more information.

Delaying the translation process can sometimes improve throughput performance—especially in batch implementations that typically run without errors.

This translation process, delayed or not, gives you flexibility in the type of options you can use; increases the amount of information that can be generated; and lets you control message formatting and language.

CONFIGURING THE MESSAGE SYSTEM

As with most system features, you can configure the messaging system. Typically you use INI options in the FSISYS.INI file (or whatever your INI file is named) to configure the message system.

For example, you can turn off or on the log and error files, assign different output file names or directories, and so on. As mentioned earlier, you can also configure the message translation process to occur during normal system processing or as a final, separate step.

The system automatically prefixes an error code before each error message. Each code begins with the two-character identifier. Here is an example:

```
DM10825: Warning in TextMergeParagraph(): Rule used in image that
does not have any text areas. Image name is <q1snam>. Processing will
continue
```

ENABLING AND DISABLING MESSAGES

Messages output from system programs fall into two categories—log and error messages. Unless specifically turned off via INI options, the message system produces both error and log files.

Error messages contain information about the problems encountered during the execution of the program. The generation of error information cannot be disabled. It is possible to not translate the results into an actual error file, instead the informational *tokens* output by the programs are written to a *message token file* named MSGFILE.DAT.

Log messages are a different matter. This type of message is informational, but not generally tied to the success or failure of the job. In general, these messages are transactional in nature—meaning that they provide information about each transaction as it proceeds through the processing cycle.

You can suppress the log information output by the programs. The LogTransactions option enables or disables the generation of log messages:

```
< Control >
LogTransactions = Yes
```

The LogTransactions option defaults to *Yes*. To disable the logging of messages, set it to *No*. By disabling this option, you suppress the informational tokens written to the intermediate file and prevent the translation of the log file.

When you set the LogTransactions option to *No*, system programs do not output the informational tokens, so you cannot generate the log file even if you use the TRANSLAT utility.

NOTE: For more information on the TRANSLAT utility, see the [Docutoolbox Reference](#).

Logging INI Files and Options Used

By default, the GenTrn, GenData, GenPrint, and GenArc programs log the INI *files* being used. This tells you which files were used and if they were opened successfully. To turn off the logging of INI file names in the LOGFILE, make the following change in your INI files:

```
< Control >
LogINIFileNames = No
```

You can also log which INI *options* are being used. This feature is turned *off* by default. To turn on the logging of INI options, use this INI option:

```
< Control >
LogINIOptions = Yes
```

You can also use the /L command line parameter to log the INI options being used. For example, if you are running on a Windows 32-bit system, you would use this command to tell the GenData program to log the INI options used to the LOGFILE:

```
gendaw32 /L
```

CLEARING MESSAGES

If you are using single-step processing, you can use the following INI option to delete all MSGFILE.DAT, ERRFILE.DAT, and LOGFILE.DAT files before the system begins the single step process.

```
< GenData >
ClearMsgFile = Yes
```

The default is No.

DEFINING THE OUTPUT MESSAGE FILES

Several files are used by the message system. You identify the output files and their locations with these INI options:

```
< Data >
ERRFile = errfile.dat
LOGFile = logfile.dat
MSGFile = msgfile.dat
TranslationFile = translat.ini
```

NOTE: The values for the LOGFile and ERRFile options are probably already set correctly if you are upgrading your system from an earlier version.

The values you specify for each option identify the file name for that option. You can also specify a directory path for each file. If you omit the path and include only the file name, the setting for the DataPath option is used as the default location for these files.

Option	Description
ERRFile	Identifies the file which contains the error messages.

Option	Description
LOGFile	Identifies the file which contains the log messages.
MSGFile	Identifies the message token file the system programs produce.
TranslationFile	<p>Contains the message text. Normally defaults to TRANSLAT.INI. Use this option to specify the file name and location.</p> <p>Unlike the other files, the TRANSLAT.INI file is static—it does not change during the batch process and is not considered a data file. This file's location does not default to DataPath option as do the other files.</p> <p>In the MVS environment, the DefLib option identifies the TRANSLAT.INI file's default location if you do not specify a path in the TranslationFile option.</p>

Initializing the Output Message Files

In a standard implementation, the GenTrn program is the first program run in the batch process. As the first program, it re-initializes the data files by first deleting the existing data files.

If your implementation does not use the GenTrn program, you either have to set up the implementation to manually delete these files or you must include an additional INI option.

The ErrorFileOpenMode option lets you tell system programs to delete old message files before beginning its processing cycle. Here is an example of this option:

```
< Control >
ErrorFileOpenMode = Create
```

If you set this option to *Create*, the system deletes existing files and creates new ones for the processing run. If you leave this option blank or enter any other value, the system appends information onto existing files.

Turning Off Date Stamps

You can turn off date stamps in batch processing error and log files using these INI options:

```
< Control >
    ErrorFileDateStamp = No
    LogFileDateStamp   = No
```

Option	Description
ErrorFileDateStamp	Enter No to disable date stamps in error files. The default is Yes.
LogFileDateStamp	Enter No to disable the date stamp in log files. The default is Yes.

Entering No to turn off these options can be of use when regression testing.

Use this option to disable date stamps in the batch trace file:

```
< Debug_Switches >
    PrintTimeStamp = No
```

Option	Description
PrintTimeStamp	Enter No to disable date stamps in the batch trace file. The default is Yes.

Controlling the Translation Process

By default, the GenTrn program deletes the old message file at the beginning of its execution and starts a new file with output information. All other programs, such as GenData, GenWIP, and so on, append information to the end of the message file created by the GenTrn program.

The default translation options are set so the log and error files are created after each system program executes. You can, however, set the ImmediateTranslate option to *No* to delay the translation process until all system programs finish processing—at the end of the batch process.

Here is an example:

```
< Control >
    ImmediateTranslate = No
```

Once processing stops, you can then use the TRANSLAT utility to translate the messages. By delaying the translation process and only executing it once per batch cycle, you can reduce job throughput times.

NOTE: If you set the ImmediateTranslate option to No, the system will not create the ERRFILE.DAT file.

DBLib Trace Messages

DBLib-related trace (or log) messages are written to the trace file. The name of this file defaults to *trace* but you can set it to another file name using the TraceFile option:

```
< Data >
TraceFile = xxxxx
```

We recommend you use the default name of *trace*.

NOTE: Before version 11.0, DBLib-related logging messages were written to the file indicated by this option:

```
< Data >
DBLogFile = (file name)
```

The default was DBLOGFLE.DAT.

Keep in mind, all types of tracing, including DBLib tracing, slow performance. You should only activate DBLib tracing during development and testing or if requested by Documaker support personnel.

In the Rules Processor, the trace file for DBLIB log messages is the default logging file. You can activate DBLib tracing by specifying these INI options in the FSISYS.INI file:

```
< Debug_Switches >
Enable_Debug_Options = Yes
DBLib                = Yes
```

In IDS, the default logging file is the DPRTRC.LOG file DBLIB log messages. You can enter the INI options in the DAP.INI file or the MRL-specific INI file.

CREATING MESSAGES

System messages fall into these categories:

- Log messages
- Error messages

Log messages record information about the processing run. These messages are informative rather than diagnostic. Types of information that fall into this category include transaction IDs that are processed; the start, ending and elapsed time of the run; transaction counts and statistics; and the program description that is producing the information.

Error messages are also informative, but usually help diagnose problems encountered during the processing run. These messages include such things as invalid data recognition; improper options; input/output errors; and resource validation.

The way these messages are produced is exactly the same. In general, the only real distinction between these two message classes is the destination file to which each is written.

USING THE RPERRORPROC AND RPLPROC FUNCTIONS

Use these two functions when you specify information to be output to the log or error files. You can use these functions to install the custom error and log procedures called from within these functions. The system lets the calling function provide the details of a message without having to specify the exact formatted text.

Here is an example:

```
RPErrorProc(pRPS, (WORD)EMIT_WARNING, (DWORD)10012,  
            "OutBuff", pRPS->OutBuff,  
            "Image", IMAGENAME(pRPS->CurrentFapImageH),  
            LASTERRORTOKEN);  
RPLPROC(pRPS, (WORD)EMIT_MESSAGE, (DWORD)10775,  
        LASTERRORTOKEN);
```

Each parameter is discussed below:

RP Struct

The first parameter represents the pointer to the RP Struct active during the run.

Message Types

The second parameter identifies the type of message being reported. There are these classes of messages:

Class	Description
EMIT_MESSAGE	Indicates the resulting information is simply a message.
EMIT_WARNING	Indicates the information is a warning to the user.
EMIT_ERROR	Indicates an error has been encountered by the program.

The message system recognizes the type of message if you use one of the above defines. Use the *EMIT_???* keywords for this parameter and do not rely upon the underlying numeric value. This lets you later change these values or add new values and recompile without invalidating the meaning of a particular message.

Message Number

Use this parameter to specify the message number to associate with the output data. Message numbers fall within the range of 100 to 1,000,000.

Message numbers are associated with the TRANSLAT.INI file. This file contains all the static text for each message. Later, the static text is merged with the variable information to produce the messages written into the log or error files. This table shows the range categories for messages:

Range	Description
100 to 9999	Used for general messages, universally shared
10000 to 499999	Reserved range for Documaker base system messages
500000 to 999999	Can be used for custom messages

Within the reserved range, there are sub-ranges for each library (DLL) and program:

Range	Library or Program
10000 to 10999	RULLIB
11000 to 11999	GENLIB
12000 to 12999	RPLIB
13000 to 13999	RCBLIB
14000 to 14999	AzWBLIB
15000 to 15999	GenTrn
16000 to 16999	GenData
17000 to 17999	GenPrint
18000 to 18999	GenArc
19000 to 19999	GenWIP
20000 to 20999	CUSLIB (Base)

This gives each program or DLL one thousand possible messages. We reserve the first five hundred thousand numbers for base system use (0 - 499,999). If a library eventually exceeds the 1000 messages currently assigned, we will assign an additional range. Likewise, this reserves enough numbers to allow for new libraries and programs which may be added in the future.

Assigning numbers to custom messages

The range 500,000 to 999,999 is for customization messages, which are generally added when you customize your system. Although you can use previously defined messages, it is better to assign an unused number within the custom range for each message you add.

This makes sure the intended meaning of an existing message is not changed in case someone modifies the text of the assumed custom message in the external file. In addition, if you develop a numbering system for the custom range, you can provide additional debugging information through the message number.

USING MESSAGE TOKENS

The remaining parameters passed to the RPErrProc or RPLogProc functions are variables which represent *token-data* pairs used to define the content of the message.

In this example, there are two pairs of *token-data*.

```
RPErrProc(pRPS, (WORD)EMIT_WARNING, (DWORD)10012,  
         "OutBuff", pRPS->OutBuff,  
         "Image", IMAGENAME(pRPS->CurrentFapImageH),  
         LASTERRORTOKEN);
```

Token	Description
OutBuff	Represents a token name. The data for that token is defined in pRPS->OutBuff.
Image	A second token name, with appropriate data text following. Token and data must be character text. Therefore, if the data to be represented is anything other than text, it must be converted before you call the message function.
LASTERRORTOKEN	Not really a single token, but rather is a macro that contains several <i>token-data</i> pairs. These pairs identify the source module name and the line number of the statement being compiled. The last component of <i>LASTERRORTOKEN</i> is a NULL pointer used by the internal message formatter to recognize the end of the <i>Token-Data</i> pairs. <i>LASTERRORTOKEN must be the last variable passed to both the RPErrProc and RPLogProc functions.</i>

There are several points to remember about *tokens* which will become apparent as you examine the TRANSLAT.INI file—the file that contains the rest of the message text.

- The message text from the TRANSLAT.INI file does not have to use all, or for that matter any, of the tokens output from a particular function. This means you can output more information (in *token-data* format) than would normally be required in the message. This information, however, might prove useful to a programmer during closer examination of the message file.
- Token names live forever. This means that a token logged earlier in the session can be referenced by messages that occur later. For instance, if an early message outputs a token (with a value) named *ID*, any message text translated after that point may refer to *ID* and receive that same value.
- Token names are reusable. You should reuse token names whenever it makes sense. For instance, each time a function is required to emit the section (image) name, use the same token name. This conserves space in the token list (because a new entry does not have to be created) and if subsequent messages rely upon the last known value of a given token, it is more likely to be correct.

- Tokens are not case sensitive. A token named *Image* can be referred to as *IMAGE*, *Image*, *image*, *ImageE*, and so on.

Also note, that the example refers to one-word tokens. Although, this is the most efficient use of space, tokens can be longer and include spaces. The only character you cannot use in a token is the ampersand (&)—ampersands are used in defining the static message text. For instance, you can define a token such as *One A Day*, but you cannot define a token such as *Will Not&Work*.

Understanding the System

Legacy systems expected the fourth parameter to be a string representing a format. This format string might be the complete message or contain *flags* indicating where subsequent variables will be substituted—such as *%d*, *%s*, *%X*, and so on.

The RPErrProc or RPLogProc functions distinguish how these remaining parameters are handled (legacy or new) by first determining if the Message Type and Message Number parameters are values expected by the new functionality.

The new use of the functions does not require a format string. Instead, the variables represent *token-data* pairs until the *LASTERRORTOKEN* is encountered.

SETTING UP MESSAGE TEXT

Message output from system programs is typically destination bound to the error or log files. All static message text is isolated into an external file for easy maintenance. The static portion of all messages is contained in the TRANSLAT.INI file.

NOTE: The *INI* designation is one of convenience, since the TRANSLAT.INI file is not intended to be used like a conventional INI file. INI references intended for other program functionality do not work when placed in this file. Likewise, you cannot add static message text intended for the log or error files into the FSISYS.INI or FSIUSER.INI files.

The TRANSLAT.INI file associates a message number with the static text for each message. Each entry takes the form:

```
AA99999 = message text
```

The numeric value is the *message number* which defines the text associated with the message. You can prefix the message number with a two-character alpha code, such as *AA*.

All messages must have a unique message number. You must make sure the proper message number is referenced in the code.

Message examples

Here are some examples:

```
10529 = Error in rundate(): Unable to GENFmtDate(<&RunDate>,).
10536 = Error in lookup(): Missing Key offset in lookup.
20261 = \nProcessing Batch:<&Name> File:<&File> Port:<&Print>
```

There are several points to note in these messages.

- Each line specifies a unique message number and associates the static text portion of the message with that number.
- The words bounded on each end with an ampersand (&) are token *placeholders* for value replacement (see message numbers 10529 and 20261). This is where the token-data pairs passed to the RPErrProc and RPLogProc functions are matched and substituted into the static text. For example, assume the following statement is in the code of one of the system programs.

```
RPErrProc(pRPS, (WORD)EMIT_ERROR, (DWORD)10529,
    "RunDate", "April 1, 1999",
    LASTERRORTOKEN);
```

This would cause message number 10529, shown above, to print this text in the log file.

```
Error in rundate(): Unable to GENFmtDate(<April 1, 1999>,).
```

- Since token names are identified between ampersand characters, two ampersand characters together (&&) signals that the output text is to contain a single ampersand character.

Undefined tokens

Messages in the TRANSLAT.INI file can have any number of token replacements. If, however, a token is undefined when the messages are translated, the token name is left in the text. So, if you view the log or error file and find a message which includes a word bounded by ampersands, it means one of these things:

- The token is misspelled in the TRANSLAT.INI file.
- The token is misspelled in the code that called the RPLogProc or RPErrProc function.
- The token and data was not included in the parameters to the message functions.
- This is not a token and was intended to print in this manner. Either it is data associated with a token or two ampersands were included at each end of the word in the static message text.

The first place to begin diagnosing this type of result is by examining the text included for the message in the TRANSLAT.INI file.

Adding a new line

In message number 20261, you can see the use of another format convention. The \n in the text is translated as a new line character. This causes the following text to print on the next line. The layout of the TRANSLAT.INI file requires that all of the text for each message must fit onto a single line. Using \n in text expands your formatting possibilities.

Determining where the message originated

Examine message number 10536. This message does not contain any tokens. Therefore there is no variable text that is required to print within this message.

The fact that the message does not contain any tokens does not mean that no tokens were output from the system program when the RPErrProc function was called. In fact, there are at least two tokens associated with this message.

LASTERRORTOKEN is the last required parameter to calls to the RPErrProc and RPLogProc functions. This macro defines the *FSIFileName* and *FSILineNumber* tokens. If you include the *FSIFileName* token in the message text, the name of the module that contained the code calling the RPErrProc or RPLogProc function is substituted into the message. Likewise, *FSILineNumber* is substituted with the source line number of the statement calling the RPLogProc or RPErrProc function.

This information can be quite useful if you are trying to determine what code is issuing a particular message. All you have to do is edit the message and include *&FSIFileName&* and *&FSILineNumber&* into the message text defined in the TRANSLAT.INI file.

USING THE MESSAGE TOKEN FILE

While a system program is running and emitting information, the token-data pairs are written to the message file (*MSGFILE.DAT*). Typically, you do not have to examine the message file. The translation process that produces the error file and log file will do that for you and will make the final text more readable.

On occasion, however, examining the file reveals more information than is provided by the translation process. For instance, if you see a particular message in the error file and want to know where in the code this message originated, you can do one of two things.

You could edit the TRANSLAT.INI file to add the *FSILineNumber* and *FSIFileName* tokens to the message. Then, by rerunning the translation process, you would get the additional message information. (See [Determining where the message originated on page 379](#) for more information)

Or, if you know what you are looking for, you could peek into the message file and locate the information more readily. Here is an excerpt from a message token file.

```
T DestField/PREM PAY INCEPTION
T Image/qmdc2
T FSIFilename/..\C\rulbsfl.c
T FSILineNumber/364
E 10010
T FSIFilename/..\C\rccbatpr.c
T FSILineNumber/418
E 13027
T FSIFilename/..\C\rulbsfs.c
T FSILineNumber/185
L 10775
T ID/3234567
T GrpName1/SAMPCO
T GrpName2/LB1
T GrpName3/
T Buff/T1
T FSIFilename/..\C\gentrans.c
T FSILineNumber/1187
L 11190
```

The first character on the line is a letter code which designates the meaning of the line. Valid codes are shown here:

Code	Description
E	Followed by a message number bound for the error file. (error or warning)
L	Followed by a message number bound for the log file. (informational)
T	Followed by a token-data pair, separated by a forward slash (/).

The token-data pairs for a given message will occur in the file on lines before the *E* or *L* lines. Knowing this, you can see that the excerpt from the message file shown above contains the information for four different messages.

The first message number occurs at the line that contains *E 10010*. This is a message bound for the error file. Four tokens are defined before translation:

- DestField
- Image
- FSIFileName
- FSILineNumber

This means that if the message text for 10010 contains any of these tokens the appropriate data will be substituted. Remember, however, if the message refers to a token that has not been defined prior to this point, the token will be left in the output text to indicate a problem might have occurred.

The next message number occurs at the line that contains *E 13027*. This too is a message bound for the error file. Notice that two tokens occur between the location of the first and second message—*FSIFileName* and *FSILineNumber*. These use the same token names used before, however, now their data values are different.

Also note that although only two additional token (changes) occurred before message 13027, four tokens are defined. If you could look into the program memory at this moment, you would see that the token list has these values:

Token	Value
DestField	PREM PAY INCEPTION
Image	qmdc2
FSIFileName	..\C\rcbbatpr.c
FSILineNumber	418

All tokens remain active after they have been translated. Tokens that are reused are updated with new values, but no tokens are removed until the translation process is complete.

Therefore, it is permissible (but at this point not likely) that a message can use tokens output by a prior message. This is why it is important to reuse token names when it makes sense, such as when all references to an section's (image) name should use the same token.

Continuing with the examination of the message file excerpt, the next message is identified via the line that reads, *L 10775*. This is a message bound for the log file, not the error file. It too redefines the *FSIFileName* and *FSILineNumber* tokens, as do all messages that use *LASTERRORTOKEN*.

The last message in this example is defined by the line, *L 11190*. Five new tokens were introduced before this message. Peeking into program memory again, the token list now looks something like this:

Token	Value
Buff	T1
DestField	PREM PAY INCEPTION
GrpName1	SAMPCO
GrpName2	LB1
GrpName3	
ID	3234567
Image	qmdc2
FSIFilename	..\C\gentrans.c
FSILineNumber	1187

Note that the most recent values for *FSIFilename* and *FSILineNumber* are reflected. Also note that the tokens previously defined still exist. Finally, note that one of the tokens appears to have no data (GrpName3) and is therefore blank. This is permissible.

CHAPTER 8

Archiving and Retrieving Information

The GenArc program lets you store completed form sets for later retrieval. The GenArc program can be run as an independent program or from within the Documaker system using the archive and retrieval options.

When you run the archive module, the information the system uses to compose the form sets is compressed and stored in an archive file along with certain indexing information.

Once the form set information has been archived, those form sets can be regenerated by retrieving the form set information from the archive file. The archive index file is used to aid in the retrieval of particular form set information through the use of keys. These keys can be set to meaningful search criteria such as policy or account numbers, claim or invoice numbers, company names, customer names, and so on.

This chapter includes information on the following topics:

- [Terminology on page 384](#)
- [System Scenarios on page 386](#)
- [Archive and Retrieval Features on page 388](#)
- [Processing Overview on page 389](#)
- [Running GenArc on page 392](#)
- [Using WIP and the Archive Index File on page 413](#)
- [Retrieving Archived Forms on page 416](#)
- [Working with Documanage on page 419](#)

TERMINOLOGY

The GenData program creates the NEWTRN file (which contains one record for every transaction to be processed), the NAFILE (which contains section and variable field information and possibly some in-line data), the POLFILE (which contains form and section inclusion information) and the recipient batch files, such as BATCH1, BATCH2, and so on (which look similar to the NEWTRN file).

The GenArc process accepts as input the NEWTRN, NAFILE and POLFILE files and archives this data. Here are some terms you need to be familiar with:

- Files and tables** The term *file* refers to a non-database data structure, such as a flat file, while the term *table* refers to data structures within some database management system, such as DB2, SQL Server, and so on. However, the terms file and table might be used interchangeably in this chapter.
- Commit** The term *commit* is a database term which means to make table changes *permanent*. As data is written to tables, the data is not really made permanent until a *commit* is performed. Before performing a commit, if you determine that you really don't want to make the changes to the table, you can perform a *rollback* which will undo any table changes you have made since the last commit point. The GenArc program performs periodic commits based on an INI value you set.
- Rollback** The term *rollback* is a database term which means to undo any table changes that have been made since the last commit point. As table rows are inserted, deleted and updated, these changes do not become *permanent* until a *commit* is performed.
- GenArc** The program name for the process which performs batch archive. The program names vary slightly, depending on the operating system you are running. For example, the GenArc program on Windows is called GENACW32.EXE.
- AFEMAIN** The program name for the Processing System. The AFEMAIN program contains a graphical user interface. It lets you enter key information and retrieve a list of archived form sets you can display. The program name may vary slightly, depending on the operating system platform you are using. For example, on Windows it is called AFEMNW32.EXE.
- CARFILE** *Compressed Archive File*. The CARFILE may also be referred to as the ARCHIVE file. The GenArc program compresses the NAFILE/POLFILE data for each transaction and writes archives this data to the CARFILE. The GenArc program writes one or more records to the CARFILE for each transaction it archives.
- APPIDX** *Application Index*. The GenArc program archives indexing information to the APPIDX file. The GenArc program writes one record to the APPIDX file for each transaction it archives.
- TEMPIDX** *Temporary Application Index*. The TEMPIDX file is used as a temporary storage for records to be added to the Application Index file. The TEMPIDX file is used only when the GenArc program is archiving to a DBASE IV database. TEMPIDX is not used by the GenArc program when archiving to DB2, SQL Server, Oracle, or other databases.

- CATALOG** Refers to the CATALOG file. As the GenArc program archives data to the CARFILE and the APPIDX, it connects the CARFILE and APPIDX files with a *key* (by default called ARCKEY). Part of this key is a field called the CATALOGID. The GenArc program generates a unique CATALOGID (timestamp) each time it runs and writes this CATALOGID to the CATALOG file. The GenArc program writes one record to the CATALOG file for each GenArc run.
- RESTART** The Restart table. The Restart table describes whether a GenArc run was successful or if the run failed. The GenArc program writes one record to the Restart table for each distinct GenArc run. GenArc runs are made distinct by passing the GenArc program a parameter called *JOBID*.
- DFD** *Data Format Definition*. A DFD file is used to describe the fields a file's records are composed of. DFD files have a particular format and are frequently used to map the layout of system-related data files. The archive-related files defined above all have default DFD files that describe their layout.

SYSTEM SCENARIOS

You can run the batch archive GenArc program, on a variety of platforms. This program creates and indexes the archived copy of the form set and its corresponding data.

You use Documaker's Archive module to retrieve, display, and print archived form sets from their workstations. The Archive module runs under various Windows 32-bit operating systems such as Windows 2000 and Windows XP. The following tables describe the various platforms and types of archives you can create and access.

NOTE: If your company has needs not covered below, contact your sales representative.

Scenarios for OS/390 (MVS)

Server

Operating system	OS/390	OS/390
Database	DB2 8.1	Oracle 8.1.7 or higher
Communications	SNA 6.2	SNA 6.2

Client

Operating system	Windows 32-bit	Windows 32-bit
Database	DB2 for Windows 7.2	na
Product	SNA Server 6.2	SNA Server 6.2
Communications	DDCS 2.3.2	DDCS 2.3.2
Archive (Documaker Workstation)	Yes	Yes

Scenarios for Windows 32-bit

Server

Operating system	Windows	Windows	Windows	Windows	Windows
Database	DB2 8.1	xBase	SQL Server 7.0	Sybase	Oracle 8.1.7
Communications	na	na	ODBC	ODBC	ODBC

Client

Operating system	Windows	Windows	Windows	Windows	Windows
Database	DB2 8.1	xBase	SQL Server 7.0.	Sybase	Oracle 8.1.7
Communications	ODBC	na	ODBC	ODBC	ODBC
Archive (Documaker Workstation)	Yes	Yes	Yes	Yes	Yes

Scenarios for UNIX

Server

Operating system	AIX version 5 or higher	Linux (x86) Kernel version 2.4.21	Solaris 9 or higher
Database	DB2 8.1 or higher Oracle 8.1.7 or higher xBASE	DB2 8.1 or higher xBASE	DB2 8.1 or higher Oracle 8.1.7 or higher xBASE
Communications	na	na	no

Client

Operating system	Windows	Windows	Windows
Database	DB2 8.1		DB2 8.1 Oracle
Communications	ODBC	see below	ODBC
Archive (Documaker Workstation)	Yes	see below	Yes

The DB2 database uses DB2LIB on if you are running the UNIX version of the GenArc program. If you are archiving to UNIX from the Windows version of the GenArc program, the system uses ODBC as the database communications layer.

You can also retrieve to Windows using DB2LIB or ODBC from tables created from the UNIX version of the GenArc program.

For Oracle databases, the UNIX processes use ORALIB as the communications client to the Oracle database server so the UNIX version of the GenArc program uses ORALIB. The Oracle database server can reside on UNIX/Linux or on Windows and you can set up ORALIB to communicate with the Oracle database server.

After the tables are populated by the UNIX version of the GenArc program, Windows applications such as AFEMAIN can retrieve archived form sets using ODBC as the Oracle database client communication layer.

ARCHIVE AND RETRIEVAL FEATURES

Regardless of the platform being used, the system has many features, including:

- Multiple media support

The archive and index files can be automatically or manually divided into separate files which may be stored on multiple storage devices. This allows for the segregation of archive data chronologically to improve search and retrieval performance. Also, as archive files grow in size, they are not limited by the physical space available on a single drive. This feature also lets you easily copy older archive files to long-term media for storage without inhibiting the retrieval capabilities.

- Stability and redundancy

The archive files are designed to be reliable. Indexing information is stored redundantly in separate files so that the index can be regenerated independently in the event of index corruption. There are a variety of archive utilities you can use to repair archive files damaged by user error or hardware failure.

- Flexible indexing

The archive index can be configured to use certain field keys within the data, allowing for retrieval based on the specified keys. This lets you design your archive system to store information for later retrieval using the most relevant data fields.

- Network-ready

The system lets you use both local and network drives for storing of archive files. The archive files are independent, so archive files can be split up over combinations of local and network drives. The system keeps track of where specific files are stored, so users do not need to know the physical or logical file storage locations.

- Unattended operation

If configured to do so, the archive module can be executed as part of the batch process. This allows data to be archived automatically.

- Restarting the archival process

Should the archive process get interrupted, you can easily restart the GenArc program and have it automatically begin where it was interrupted. You can also use command line options to process a specified range of transactions or a specific job if you are running the GenArc program on multiple computers simultaneously.

PROCESSING OVERVIEW

The GenArc program can archive form set data to files and/or Database Management Systems (DBMS). By default (if the INI file is not configured otherwise), the GenArc program archives form set data to a DBASE IV DBMS (actually a combination – APPIDX is DBASE IV file and CARFILE is a flat file). Below is a list containing some of the DBMS systems the GenArc program can archive to.

NOTE: For information on the various INI option settings, see the appropriate installation manual for your operating system and the technical documentation.

- | | |
|-------------------|--|
| DBASE IV | The APPIDX, TEMPIDX and CATALOG files are created as DBASE IV files. This results in the GenArc program creating DBF and MDX database files for the APPIDX, TEMPIDX and CATALOG and a CAR file (non-DBASE IV) for the CARFILE. The restart option is not available for DBASE IV archive. |
| DB2 | The APPIDX, ARCHIVE, CATALOG and RESTART files are all created as DB2 tables. GenArc communication to DB2 can be done through either the DB2's native API or DB2's ODBC interface. The restart option is available for DB2 archive. |
| SQL server | The APPIDX, ARCHIVE, CATALOG and RESTART files are all created as SQL Server tables. SQL Server is an ODBC-compliant DBMS. The restart option is available for SQL Server archive. |
| Oracle | The APPIDX, ARCHIVE, CATALOG, and RESTART files are all created as Oracle tables. Oracle is an ODBC-compliant DBMS. The restart option is available for Oracle archive. |

FILES GENARC USES

- | | |
|---------------------|---|
| Input files | <ul style="list-style-type: none"> • NEWTRN file • NAFILE file • POLFILE file |
| Output files | <ul style="list-style-type: none"> • Compressed Archive (CAR) file • Application Index file • Catalog file • Restart file |

HOW THE GENARC PROGRAM WORKS

Below is a brief description of how GenArc processing is performed. Most of the restart information has been omitted but is covered in [Using the Restart Option on page 395](#).

- 1 Store the command line parameters, load INI files, and check and update the Restart table.

the GenArc program parses and stores any command line parameters passed to it. INI files are read and loaded. The Status column of the Restart table is checked (if archiving to a DBMS, not DBASE IV) to determine if the previous GenArc run by this JOBID (DEFAULT_JOB_ID by default) was successful or whether it failed. If the last GenArc run was successful the Status column of the Restart row is initialized to *Failed*.

- 2 Get a CATALOGID and then check and update the CATALOG table.

the GenArc program gets a timestamp from the system and constructs a 10-character CATALOGID. The CATALOG table is checked to make sure this CATALOGID is not already in the table. If the CATALOGID is already in the table, the GenArc program gets additional timestamps, until it finds one that is not already in the table. Once it has a unique CATALOGID, the GenArc program constructs a row containing this CATALOGID (CATALOGID column) and writes this row to the CATALOG table so future runs of the GenArc program will not be able to use this CATALOGID.

- 3 Read the NEWTRN file, get form set data from the NAFILE and POLFILE, then combine and compress the information.

The NEWTRN file is opened and the first record (transaction) is read. The NEWTRN record contains offset values into the NAFILE and POLFILE for the transaction. The GenArc program uses these offset values to retrieve the NAFILE data and POLFILE data for the transaction and it then combines and compresses this data.

- 4 Construct the ARCKEY, construct and archive the rows to the ARCHIVE table.

An eight-character sequential number (which will be incremented for each transaction) is appended with the 10-character CATALOGID to form an 18-character ARCKEY. This ARCKEY will be unique for each transaction. A record (or row) is constructed to be written to the ARCHIVE table. This row (whose columns are described by the CARFILE DFD file) contains the ARCKEY and the combined and compressed NAFILE/POLFILE data (CARDATA column). If the CARDATA is too large to fit on a single row, additional rows are constructed—each row will have the same ARCKEY but will have an incremented Sequence Number (SEQ_NUM column). The constructed rows are archived to the ARCHIVE table.

- 5 Construct and archive the rows to the APPIDX table.

The index information for the transaction is gathered and a row is constructed to be written to the APPIDX table. This row (whose columns are described by the APPIDX DFD file) contains the ARCKEY used to construct the row for the ARCHIVE table above, as well as other information, such as Company, Line of Business, PolicyNumber, and so on (columns identified in the INI group Trigger2Archive). Once this APPIDX row is constructed it is archived to the APPIDX table. Only one record is written to the APPIDX table for each transaction.

- 6 Repeat the process, update the Restart table, issue messages, and terminate processing.

Steps 3 through 5 are repeated until all the NEWTRN records have been read. Once all the NEWTRN records have been read and the archiving is complete for all transactions, the Status column of the Restart table row, which was set to failed in step 1, is updated to reflect that the GenArc run was successful. The GenArc program issues console messages indicating how many transactions were *read*, *archived*, *in error*, and *rolled back*. The GenArc program then terminates processing.

RUNNING GENARC

The name of the GenArc program and how you run it varies somewhat depending on the operating system you are using. The concepts are the same, though, for all operating systems. For our example let's assume you are running the GenArc program on Windows 2000. To run the GenArc program on Windows 2000, you enter a command like this:

```
C:\FAP\MSTRRES\RPEX1\genacw32
```

Notice the command includes the program name (GENACW32) and its full path—from the RPEX1 master resources directory. This command starts the GENACW32 program (GENACW32.EXE) and attempts to locate a FSIUSER.INI file in the c:\fap\mstres\rpex1 directory.

The GenArc program messages will look something like the sample below if you have the LogToConsole option set as shown here:

```
< Control >
LogToConsole = Yes
```

Here are the sample messages:

```
--- GenArc ---
==> Processing: TransactionId-GroupName1-GroupName2-GroupName3-
TransactionType
==> Processing: 1234567-SAMPCO-LB1--T1
==> Processing: 2234567-SAMPCO-LB1--T1
==> Processing: 5SAMPCO-SAMPCO-LB2--T1
==> Processing: 6SAMPCO-SAMPCO-LB2--T1
==> Processing: 7SAMPCO-SAMPCO-LB2--T1
==> Processing: 8SAMPCO-SAMPCO-LB2--T1
==> Processing: 9SAMPCO-SAMPCO-LB2--T1
==> Processing: 4234567-FSI-CPP--T1
==> Processing: 5234567-FSI-GL--T1

==> Transactions Read      :      9
==> Transactions Archived :      9
==> Transactions In Error :      0
==> Transactions Rolled Back:      0

==> Warning count:      0
==> Error count:      0
Elapsed Time: 2 seconds
--- GenArc Completed ---
```

Logging archived transactions

If you want the GenArc program to produce a log of the archived transactions, include the following INI option in the ArcRet control group:

```
< ArcRet >
ExportIndex = <file name>.
```

Be sure to include the full path and file name of the log file. If you omit the ExportIndex option, the system does not create the log file.

Archiving to a database

The system lets you archive information to a database, such as DB2, as an alternative to archiving to flat files (CAR files). You use the ArchiveMem option in the FSI SYS.INI file to enable database archiving, as shown here:

```
< Archival >
ArchiveMem = Yes
```

NOTE: When running on z/OS, the GenArc program sets the ArchiveMem option to Yes if it was not in the FSI SYS file and produces a warning. This prevents an error (running with non-VSAM NA and POL files) or an abend (running with VSAM NA and POL files) which will occur if the ArchiveMem option is set to No.

Sorting records in a database

Use the DefaultTag option to specify the default tag in ODBC and DB2. This tag is then used by the ORDER BY clause in the SQL database to sort records.

```
< DBTable:MYTABLE >
DefaultTag =
```

For the DefaultTag option, enter the name of the key from the DFD file.

Keep in mind this only works with ODBC and DB2. It does not work with xBase files.

Preparing SQL

Add the AlwaysSQLPrepare option to make sure the ODBC driver always performs the _SQLPrepare() function. Here is an example:

```
< DBHandler:ODBC >
AlwaysSQLPrepare = Yes
```

Omitting this option can the S1010 o [Oracle][ODBC]Function sequence error.

COMMAND LINE OPTIONS

The GenArc program accepts several command line options. Command line options are prefixed with either a backslash (/) or a dash (-). Here is an example of starting the GenArc program with command line options:

```
C:\FAP\MSTRRES\RPEX1\genarcw32 /ini=my.ini /jobid=tuesday1
```

The command line options are explained below:

INI Use the INI command line option to tell the GenArc program to open and read a FSIUSER.INI file other than the one in the current directory.

JOBID (Abbreviation: J)

Use the JOBID command line option to associate a Job Identifier with this particular run of the GenArc program. By default the GenArc program associates a run with the identifier, *DEF_JOB_ID*. This identifier (either the default identifier or the identifier specified with the JOBID option) is used when the Restart row in the Restart table is searched for and/or updated. Using JOBID allows for concurrent runs of the GenArc program.

DPASSWD (Abbreviation: DP)

Use the DPASSWD command line option to indicate the password to be used when connecting to a DB2 database management system (DBMS). Use this option along with the DUSERID option. You can also specify the DPASSWD option in the INI file as shown below:

```
< DBHandler:DB2 >
Passwd = xxxxxxxx
```

DUSERID (Abbreviation: DU)

Use the DUSERID command line option to indicate the User ID to use when connecting to a DB2 database management system. Use this option along with the DPASSWD option. You can also specify the DUSERID option in the INI file as shown below:

```
< DBHandler:DB2 >  
UserID = xxxxxxxx
```

OPASSWD (Abbreviation: OP)

Use the OPASSWD command line option to indicate the password to be used when connecting to an ODBC-compliant database management system. Use this option along with the OUSERID option. You can also specify the OPASSWD option in the INI file as shown below:

```
< DBHandler:ODBC >  
Passwd = xxxxxxxx
```

OUSERID (Abbreviation: OU)

Use the OUSERID command line option to indicate the password to be used when connecting to an ODBC-compliant database management system. Use this option along with the OPASSWD option. You can also specify the OPASSWD option in the INI file as shown below:

```
< DBHandler:ODBC >  
UserID = xxxxxxxx
```

RESTART (Abbreviation: R)

Use the RESTART command line option to tell the GenArc program to start processing with the n'th record in the NEWTRN file. The GenArc program will skip n-1 NEWTRN records and begin with the n'th record. When you use the RESTART command line option you are *explicitly* restarting the GenArc program.

SQLID (Abbreviation: SQL)

Use the SQLID command line option to tell the GenArc program to perform a *SET CURRENTSQLID=SQLID* at initialization time. You can also specify the SQLID option in the INI file as shown below:

```
< DBHandler:DB2 >  
CurrentSQLID = xxxxxxxx
```

STOPREC (Abbreviation: S)

Use the STOPREC command line option to tell the GenArc program to stop processing on the n'th NEWTRN record.

Using the Restart Option

The Restart option is *only* available if you are archiving both APPIDX and ARCHIVE data into a database management system. The Restart option is not available if you are using DBASE IV, which is the default archive method.

If the GenArc program detects an error during its processing, it can skip the transaction in error and continue processing with the next transaction in the NEWTRN.DAT file. The INI option listed below tells the GenArc program whether it should terminate processing when it encounters errors:

```
< GenArcStopOn >
  DBErrors = No
```

The default value for the DBErrors option is Yes, which means the GenArc program stops processing when it receives an error. If you set the DBErrors option to No, the GenArc program tries to skip the transaction in error and then continues with the next transaction in the NEWTRN.DAT file.

Below is a brief description of how the GenArc program performs restart processing. The description below does not include all of the information provided in [How the GenArc Program Works on page 389](#) but all of that information applies to restart processing as well.

- 1 Check the command line for parameters, load INI files, and then check and update the Restart table.

The GenArc program parses and stores any command line parameters passed to it. INI files are read and loaded. If the JOBID parameter was passed, the GenArc program will attempt to locate a row in the Restart table whose JOB_ID column equals the JOBID value. If the GenArc program cannot locate a row whose JOB_ID column matches the JOBID value passed in, the GenArc program issues an error message and terminates.

If the RESTART parameter was passed, this is an *explicit* restart, meaning we are supposed to restart on the n'th record of the NEWTRN.DAT file (skipping the first n-1 records).

If the RESTART parameter was not passed, either the prior run of the GenArc program was successful (and there is no need to try to restart) or the prior run was unsuccessful but the operator made some change since encountering the error that should allow the GenArc program to continue where it left off (implicit restart).

- 2 Determine the restart point and check the Restart table.

If this is an *explicit* restart, the GenArc program simply skips the first n-1 records of the NEWTRN file and reads the n'th record. It begins the archiving process with that record.

If this is either a *no restart* or an *implicit restart*, the GenArc program first locates the appropriate row of the Restart table (based on the JOBID described in Step 1). The GenArc program then checks the Status column of the Restart table to determine if the previous GenArc run by this JOBID was successful or whether it failed. If the last GenArc run was successful the Status column of the Restart row is initialized to *Failed*.

If the last GenArc run failed, the COMM_RECS column is checked to see how many transactions were committed during the prior GenArc run. The GenArc program also retrieves the value of the LASTREC column – this column contains the actual NEWTRN record for the last successful transaction. If the value of COMM_RECS is, for example, X, the GenArc program then skips to the x'th record in the NEWTRN.DAT file and compares the NEWTRN record with the value of the LASTREC column – if the values do not match, the GenArc program issues an error message indicating there is a consistency problem and terminates processing. If the values of the x'th NEWTRN record and the LASTREC column do match, the GenArc program positions itself to the x+1'th NEWTRN record and will begin the archiving process with that record.

3 Archive form sets and then perform regular commits.

Before beginning the actual archive processing of the NEWTRN records, the GenArc program checks the INI file to determine how often to perform *commits* to the DBMS tables. The GenArc program checks the INI option listed below:

```
< ArcRet >
CommitEvery = 10
```

The default value for the CommitEvery option is 10. This value tells the GenArc program to perform a commit every 10 transactions.

Once the GenArc program is positioned to the appropriate NEWTRN record where it is to begin processing, it processes each NEWTRN record. *Processing* means the NAFILE and POLFILE data are combined and compressed and archived to the ARCHIVE table, an index record is constructed and is archived to the APPIDX table.

Also, the Restart table is updated: the COMM_RECS column receives the NEWTRN record number—the record number of the most recently archived NEWTRN transaction—and the LASTREC column receives a full copy of the actual NEWTRN record itself. If at any time GenArc processing fails, a *rollback* is performed which will restore all the GenArc tables to the last point of consistency, which is the last commit point.

4 Finish processing the NEWTRN.DAT file and then update the Restart table.

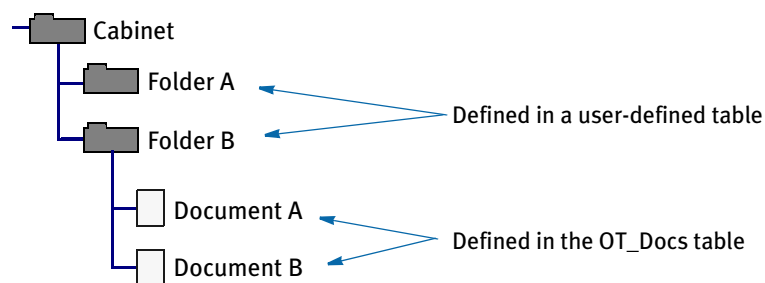
The archiving and committing process described in step 3 is performed until all of the NEWTRN records have been processed. When the final NEWTRN record is processed, the Status column of the Restart table is updated from *F* (failed) to *S* (successful) and a final *commit* is performed to make the last few table changes permanent.

The GenArc program issues messages indicating how many transactions were read from the NEWTRN.DAT file, how many transactions were skipped (if this was a restart), how many transactions were successfully archived, how many transactions were in error and how many transactions were rolled back. The sum of the number of transactions skipped, archived, in error and rolled back should equal the number of transactions read.

USING GENARC WITH DOCUMANAGE

You can use Documanager to archive files created from the GenArc program. This is done using the PO Handler. Set up the Documanager Administrator in this order:

- Map to database
- Business tables
- Cabinets
- Document types
- Authorities



The user-defined table contains a record for each folder in the cabinet. The OT_Docs table includes one record for each document in the folder.

What happens when a transaction is archived:

- 1 The PO Handler searches the cabinet for a folder that matches the transaction data. The FolderBy option in the Cabinet control group defines the fields used to identify the correct folder.
- 2 If the folder exists, the data needed to create the document is checked into the folder. A folder is created if a matching folder was not found. Creating the folder adds a record to the table that defines the cabinet. Adding the document adds a record to the OT_Docs table. The document is named by the fields defined in the NameDocBy INI option. The document appears by this name in Documanager.

When you display a transaction using the Entry system:

- 1 Folders are searched based on the fields defined in the FolderBy option. If a folder exists, the documents in the folder that match the type are searched. If no documents match, the folder is ignored. The document type is defined in the FileType option in the Cabinet control group. The system then creates a row in the Formset Selection window for each document where the folder has matching properties and document types.
- 2 When you select a document, the body of the document (CARDATA) is extracted into a temporary file. The data is then retrieved into the ARCHIVE record and the form set is displayed.

Here are examples of the INI options you use. These options set all archive tables to use the PO Handler:

```
< DBTable:APPIDX >
    DBHandler = PO
< DBTable:ARCHIVE >
    DBHandler = PO
```

These options set up the PO Handler:

```
< DBHandler:PO >
    UserID    = EZPOWER
    Password  = EZPOWER
    Cabinet   = ARCCAB
    Domain    = FSI
```

The Cabinet option contains all of the fields in all tables. You would use the Domain option if you are executing Documaker Workstation or the GenArc program in a different domain than the server machine.

Here are the options for the cabinet:

```
< PO:ARCCAB >
    FileType  = dap
    FolderBy  = KEY1,KEY2,KEYID
    NameDocBy = KEY1,KEYID,TRANCODE
```

Option	Description
FileType	Use this option to define the file types that can be placed in the folder.
FolderBy	Use this option to define the fields you want the system to use to sort the document into the various folders. For instance, if you enter Key1,Key2,KeyID, the system places documents which have the same data in these fields in the same folder.
NameDocBy	Use this option to tell the system which field contains the document name. If you omit this field, the systems uses ARCKEY.

Use this control group to map the DFD fields to the OT_Docs fields. For instance, this example assumes that the AddedOn option is in the OT_Docs table:

```
< POField2Document >
    AddedOn = CreateTime
```

Use this control group to map the OT_Docs fields to the DFD fields:

```
< PODocument2Field >
    CreateTime = AddedOn
```

This control group is required for the GenArc program. The Restart table is not supported by Documanager:

```
< Archival >
    ArchiveMem = Yes
    UseRestartTable = No
```


These field names are reserved in the Documanager/PO Handler environment:

Field	Description
CARData	This field must be present in the CARFILE.DFD file. Never folder on this field. Should never be in the DB table under Documanager only in the DFD. Must be defined in the CARFILE.DFD as a BLOB. Always associated with the document.
ARCKey	This field is the archive key. It must be in both the APPIDX.DFD and CARFILE.DFD files. Required in the table under Documanager.
DESC	(optional) The document description. By default, this field is associated with document.
RunDate	(optional) The document's run date. By default, this field is associated with document.

Other fields are associated with the folder unless you specify otherwise in the PODocument2Field or POField2Document control group.

Here are samples of the FSIUSER.INI, APPIDX.DFD, and CARFILE.DFD files:

NOTE: Make sure you use upper- and lowercase correctly in DFD and INI files.

Forcing folder updates

You can now use the ForceFolderUpdate option to force folder updates when the folder already exists. This lets Documanager Folder Update Authorities, when set to No, prevent duplicate archive entries from being sent to the Documanager archive repository.

Here is an example of the ForceFolderUpdate option:

```
< PO:Prod >
  FileType = PROD
  FolderBy = DOC_TYPE_CODE,DOC_NUM,DOC_REV_NUM
  NameDocBy = DOC_TYPE_CODE,DOC_NUM,DOC_REV_NUM
  ForceFolderUpdate = Yes
```

The default is No.

FSIUSER.INI sample

```
< Archival >
  ArchiveMem      = Yes
  UseRestartTable = No
< ArcRet >
  AppIdx          = ARC\APPIDX
  AppIdxDFD       = DefLib\AppIdx.Dfd
  ArcPath         = [CONFIG:Batch Processing] ARCPATH =
  Arrangement     = Stack
  CARFile         = ARCHIVE
  CARFileDFD      = .\DEFLIB\ODBC\carfile.dfd
  CARPath         = [CONFIG:Batch Processing] CARPath =
  Catalog         = ARC\CATALOG
  ExactMatch      = No
  Key1            = Company
  Key2            = Lob
```

```

        KeyID           = Policynum
        LBLimit         = 500
        TempIdx         = ARC\Temp
    < Config:Batch Processing >
        ARCPATH         = ARC\
        BaseDef         =
        CARPath         = arc\
        CompLib         = COMPLIB\
        DALFile         =
        DefLib          = DEFLIB\
        FntFile         = REL95SM.fnt
        FontLib         = ..\fmres\deflib\
        Form7x          =
        FormDef         = FORM.DAT
        FormFile        =
        FormLib         = FORMS\
        FormsetTrigger  = SETRCPTB.DAT
        HelpLib         = help\
        LogoFile        =
        TableLib        = table\
        WIPPath         = wip\
        XrfFile         = REL95SM
    < Configurations >
        Config          = Batch Processing
    < Control >
        XrfExt          = .FXR
    < DBHandler:PO >
        Cabinet         = RPEX1
        Domain          = FSI
        PassWord        = astros3
        UserID          = erm
    < DBTable:APPIDX >
        DBHandler       = PO
    < DBTable:ARCHIVE >
        DBHandler       = PO
    < DefaultTextArea >
        Chars           = 10
        Font            = 16010
        Lines           = 2
    < DefaultVarField >
        Font            = 12010
        Length          = 1
        Type            = x
    < Environment >
        FSISYSINI       = .\FSISYS.INI
        FSITemp         = TEMP
    < MasterResource >
        BaseDef         = [CONFIG:Batch Processing] BaseDef =
        CompLib         = [CONFIG:Batch Processing] CompLib =
        DalFile         = <CONFIG:Batch Processing> DalFile =
        DDTFile         = [CONFIG:Batch Processing] DDTFile =
        DDTLib          = [CONFIG:Batch Processing] DDTLib =
        DefLib          = [CONFIG:Batch Processing] DefLib =
        DictionaryFile  = [CONFIG:Batch Processing] DictionaryFile =
        FieldBaseFile   = [CONFIG:Batch Processing] FieldBaseFile =

```

```

      FntFile          = [CONFIG:Batch Processing] FntFile =
      FontLib          = [CONFIG:Batch Processing] FontLib =
      Form7x           = [CONFIG:Batch Processing] Form7x =
      FormDef          = [CONFIG:Batch Processing] FormDef =
      FormFile         = [CONFIG:Batch Processing] FormFile =
      FormLib          = [CONFIG:Batch Processing] FormLib =
      FormsetTrigger   = [CONFIG:Batch Processing] FormsetTrigger =
      HelpLib          = [CONFIG:Batch Processing] HelpLib =
      LbyLib           = [CONFIG:Batch Processing] LbyLib =
      LogoLib          = [CONFIG:Batch Processing] LogoLib =
      LogoFile         = [CONFIG:Batch Processing] LogoFile =
      TableLib         = [CONFIG:Batch Processing] TableLib =
      Xrffile          = [CONFIG:Batch Processing] Xrffile =
> PO:RPEX1 >
      FileType         = DAP
      FolderBy        = Company,Lob,Polycynum
      NameDocBy        = ARCKEY
< PODocument2Field >
      CreateTime       = AddedOn
< POField2Document >
      AddedOn          = CreateTime
< SignOn >
      UserID           = FORMAKER
< WIPData >
      File             = Wip\Wip
      Path             = [CONFIG:Batch Processing] WIPPath =

< FIELDS >
      FIELDNAME = UNIQUE_ID
      FIELDNAME = Company
      FIELDNAME = Lob
      FIELDNAME = Polycynum
      FIELDNAME = RunDate
;   FIELDNAME = InvFlag
;   FIELDNAME = ClaimFl
      FIELDNAME = ARCKEY
      FIELDNAME = FormsetId
      FIELDNAME = RECNUM
      FIELDNAME = CONFIG
< FIELD:UNIQUE_ID >
      INT_TYPE = CHAR_ARRAY
      INT_LENGTH = 26
      EXT_TYPE = CHAR_ARRAY
      EXT_LENGTH = 26
      KEY = Y
      REQUIRED = Y
< FIELD:Company >
      INT_TYPE = CHAR_ARRAY
      INT_LENGTH = 6
      EXT_TYPE = CHAR_ARRAY
      EXT_LENGTH = 6
      KEY = Y
      REQUIRED = Y
< FIELD:Lob >
      INT_TYPE = CHAR_ARRAY

```

APPIDX.DFD sample

```
INT_LENGTH = 3
EXT_TYPE = CHAR_ARRAY
EXT_LENGTH = 3
KEY = Y
REQUIRED = Y
< FIELD:Polycynum >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 7
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 7
  KEY = Y
  REQUIRED = Y
< FIELD:RunDate >
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 8
  EXT_PRECISION = 0
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 8
  INT_PRECISION = 0
  KEY = N
  REQUIRED = Y
< FIELD:InvFlag >
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 1
  EXT_PRECISION = 0
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 1
  INT_PRECISION = 0
  KEY = N
  REQUIRED = Y
< FIELD:ClaimFl >
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 1
  EXT_PRECISION = 0
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 1
  INT_PRECISION = 0
  KEY = N
  REQUIRED = Y
< FIELD:ARCKEY >
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 18
  EXT_PRECISION = 0
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 18
  INT_PRECISION = 0
  KEY = Y
  REQUIRED = Y
< FIELD:FormsetId >
  EXT_TYPE = NOT_PRESENT
  EXT_LENGTH = 0
  EXT_PRECISION = 0
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 8
  INT_PRECISION = 0
```

```

        KEY = N
        REQUIRED = Y
    < FIELD:RECNUM >
        EXT_TYPE = NOT_PRESENT
        EXT_LENGTH = 0
        EXT_PRECISION = 0
        INT_TYPE = LONG
        INT_LENGTH = 4
        INT_PRECISION = 0
        KEY = N
        REQUIRED = Y
    < FIELD:CONFIG >
        EXT_TYPE = CHAR_ARRAY
        EXT_LENGTH = 10
        EXT_PRECISION = 0
        INT_TYPE = CHAR_ARRAY
        INT_LENGTH = 10
        INT_PRECISION = 0
        KEY = Y
        REQUIRED = Y
    < KEYS >
        KEYNAME = UNIQUE_ID
        KEYNAME = Company
        KEYNAME = Lob
        KEYNAME = Policynum
    < KEY:Company >
        EXPRESSION = Company
        FIELDLIST = Company
    < KEY:Lob >
        EXPRESSION = Lob
        FIELDLIST = Lob
    < KEY:PolicyNum >
        EXPRESSION = Policynum
        FIELDLIST = Policynum
    < KEY:UNIQUE_ID >
        EXPRESSION = UNIQUE_ID
        FIELDLIST = UNIQUE_ID

```

CARFILE.DFD sample

```

    < FIELDS >
        FIELDNAME = ARCKEY
        FIELDNAME = SEQ_NUM
        FIELDNAME = CONT_FLAG
        FIELDNAME = TOTAL_SIZE
        FIELDNAME = CARDATA
    < FIELD:ARCKEY >
        INT_TYPE = CHAR_ARRAY
        INT_LENGTH = 18
        EXT_TYPE = CHAR_ARRAY
        EXT_LENGTH = 18
        KEY = N
        REQUIRED = N
    < FIELD:SEQ_NUM >
        INT_TYPE = CHAR_ARRAY
        INT_LENGTH = 5
        EXT_TYPE = CHAR_ARRAY

```

```

EXT_LENGTH = 5
KEY = N
REQUIRED = N
< FIELD:CONT_FLAG >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 1
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 1
  KEY = N
  REQUIRED = N
< FIELD:Total_Size >
  INT_Type = LONG
  INT_Length = 4
  EXT_Type = LONG
  EXT_Length = 4
  Key = N
  Required = N
< FIELD:CARData >
  INT_Type = BLOB
  INT_Length = 8
  EXT_Type = BLOB
  EXT_Length = 8
  Key = N
  Required = N
< Keys >
  KeyName = ARCKEY
  KeyName = SEQ_NUM
  KeyName = CAR_KEY
< KEY:ARCKey >
  Expression = ARCKEY+SEQ_NUM
  FieldList = ARCKEY,SEQ_NUM
< KEY:SEQ_NUM >
  Expression = SEQ_NUM
  FieldList = SEQ_NUM
< KEY:CAR_Key >
  Expression = ARCKEY
  FieldList = ARCKEY

```

Using the Oracle ODBC Driver

The Oracle ODBC driver is supported on all Windows platforms. The DFD and INI files shown on previous pages require special consideration when using the Oracle driver. Here are samples of CARFILE.DFD and FSIUSER.INI files.

CARFILE DFD

To use a library using the Oracle ODBC driver, you must use an Oracle Insurance-supplied CARFILE DFD file that differs from the standard (internal) DFD definition. The supplied CARFILE.DFD file is located in the sample RPEX1 resources in the directory:

```
.. \DEFLIB\ODBC_ORA\CARFILE.DFD
```

The contents of the CARFILE.DFD are listed below:

```

; CARFILE.DFD - this DFD is to be used when referencing a library or
; archive with the Oracle ODBC driver.
< FIELDS >
  FIELDNAME = ARCKEY
  FIELDNAME = SEQ_NUM

```

```

        FIELDNAME = CONT_FLAG
        FIELDNAME = TOTAL_SIZE
        FIELDNAME = CARDATA
< FIELD:ARCKEY >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 18
    EXT_TYPE = CHAR_ARRAY
    EXT_LENGTH = 18
    KEY = N
    REQUIRED = N
< FIELD:SEQ_NUM >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 5
    EXT_TYPE = CHAR_ARRAY
    EXT_LENGTH = 5
    KEY = N
    REQUIRED = N
< FIELD:CONT_FLAG >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 1
    EXT_TYPE = CHAR_ARRAY
    EXT_LENGTH = 1
    KEY = N
    REQUIRED = N
< FIELD:TOTAL_SIZE >
    INT_TYPE = LONG
    INT_LENGTH = 4
    EXT_TYPE = DOUBLE
    EXT_LENGTH = 4
    KEY = N
    REQUIRED = N
< FIELD:CARDATA >
    INT_TYPE = BLOB
    INT_LENGTH = 252
    EXT_TYPE = BLOB
    EXT_LENGTH = 252
    KEY = N
    REQUIRED = N
< KEYS >
    KEYNAME = ARCKEY
    KEYNAME = SEQ_NUM
    KEYNAME = CAR_KEY
< KEY:ARCKEY >
    EXPRESSION = ARCKEY+SEQ_NUM
    FIELDLIST = ARCKEY, SEQ_NUM
< KEY:SEQ_NUM >
    EXPRESSION = SEQ_NUM
    FIELDLIST = SEQ_NUM
< KEY:CAR_KEY >
    EXPRESSION = ARCKEY
    FIELDLIST = ARCKEY

```

To use the supplied CARFILE.DFD file, do the following:

- 1 Copy the CARFILE.DFD file into the directory where you store other DFD files, such as the \DefLib directory.

2 Make the system use the CARFILE.DFD file by adding this entry into the INI file:

```
< ArcRet >
CARFileDFD = ..\DEFLIB\CARFILE.DFD
```

Creating the Database and Tables

Use these INI options to tell Library Manager to create a library using the Oracle ODBC driver and to load resources from that library:

```
< MasterResource >
DALFile = LBYI
DDTFile = LBYI
FormFile = LBYI
LOGOFile = LBYI
< LibraryManager >
LBYLOGFile = LBYLOG
< Library:LBYI >
DBTable = LBYD
< DBTable:LBYI >
DBHandler = ODBC
< DBTable:LBYD >
DBHandler = ODBC
UniqueTag = ARCKEY+SEQ_NUM
< DBTable:LBYLOG >
DBHandler = ODBC
< DBTable:CATALOG >
DBHandler = ODBC
UniqueTag = CATALOGID
< DBHandler:ODBC >
Server = LBYSQL
Qualifier = LBYSQL
CreateTable = Yes
CreateIndex = No
UserID = userid
Passwd = password
Debug = No
< ODBC_FileConvert >
LBYI = DAP102_LBYI
LBYD = DAP102_LBYD
LBYLOG = DAP102_LBYLOG
```

A description of the above INI options follows:

Option	Description
MasterResource control group	
DALFile	Enter the name of the library from which you want the system to retrieve DAL scripts and DAL script libraries.

Option	Description
DDTFile	<p>Enter the name of the library from which you want the system to retrieve DDT files.</p> <p>If you define this option, the system expects to find all DDT files there, including the MASTER.DDT file. You can use the following option to exclude the MASTER.DDT file from being located in the library:</p> <pre>< RunMode > MasterDDTNotInLibrary = Yes</pre> <p>The only advantage to having an external MASTER.DDT file is if your setup creates the MASTER.DDT file on the fly, before a transaction is run. If that is the case, it is easier to manipulate if it is outside of the library.</p>
FormFile	Enter the name of the library from which you want the system to retrieve FAP files.
LOGOFile	Enter the name of the library from which you want the system to retrieve LOG (logo) files.

LibraryManager control group

LBYLOGFile	Enter the name of the library log file. The library log contains information about resources that are added to, deleted from, or updated in the library. The LBYLOGFile does not have to use the same type of DB handler as the library index and data portions.
------------	--

Library:LBYI control group

DBTable	Enter the name of the data component of the library. In this example, the names LBYI and LBYD are used to emphasize that one table, LBYI, represents the library index and one table, LBYD represents the library data. You can use up to eight characters to give these tables any name you like. See the ODBC_FileConvert control group if you need to map these eight-character names to longer table names.
---------	---

DBTable:LBYI control group

DBHandler	Tells the system to access the LBYI table using the data base handler named ODBC. Because of this INI value, the system expects to find an INI control group named DBHandler:ODBC. Microsoft's SQL Server is an ODBC-compliant database.
-----------	--

DBTable:LBYD control group

DBHandler	Tells the system to access the LBYD table using the data base handler named ODBC. Because of this INI value, the system expects to find an INI control group named DBHandler:ODBC.
UniqueTag	In this example, ARCKEY+SEQ_NUM specifies that the columns ARCKEY and SEQ_NUM can be combined to represent a unique tag for the table. This unique tag is only used for internal purposes. If you do not specify a unique tag for this table, and a column with the name UNIQUE_ID does not exist within the table, you receive warning messages indicating that there is no unique tag defined.

DBTable:LBLOG control group

Option	Description
DBHandler	Tells the system to access the LBYLOG table using the data base handler named ODBC. Because of this INI value, the system expects to find an INI control group named DBHandler:ODBC.

DBTable:CATALOG control group

DBHandler	Tells the system to access the CATALOG table using the data base handler named ODBC. The CATALOG table is used to temporarily store CATALOGID values used to construct an ARCKEY.
UniqueTag	This specifies that the column CATALOGID represents a unique tag for this table. This unique tag is only used for internal purposes. If you do not specify a unique tag for this table, and a column with the name UNIQUE_ID does not exist within the table, you receive warning messages indicating that there is no unique tag defined.

DBHandler:ODBC control group

Server	Specifies the name of the ODBC data source for this database handler, such as LBYSQL. You must also define an ODBC data source with this name.
Qualifier	Specifies the name of the database for this database handler, such as LBYDBASE. If you omit this option, the database set up as the default database for the LBYSQL ODBC data source is used.
CreateTable	Specifies the system should create any tables Library Manager needs, that do not already exist, at run time.
CreateIndex	Specifies the system should create any database indexes it needs, that do not already exist. Always set this option to No.
UserID	Enter the user ID to use when connecting to the data base management system.
Passwd	Enter the password to use when connecting to the data base management system.
Debug	Enter Yes to turn on tracing for the Documaker ODBC DB handler. Enter No or omit this option except in troubleshooting situations.

ODBC_FileConvert control group

This INI control group lets you map table names of eight characters or less to table names longer than eight characters. The table names you specify must follow the table naming conventions for the data base management system.

LBYI	Specifies the name of the table referenced in several INI locations as LBYI on the data base management system.
LBYD	Specifies the name of the table referenced in several INI locations as LBYD on the data base management system.
LBYLOG	Specifies the name of the table referenced in several INI locations as LBYLOG on the data base management system.

Resolving Errors

If the GenArc program produces an error similar to the following example, it indicates the INT_Length or EXT_Length (or both) options in the CARData control group have not been set in the CARFILE.DFD file:

```
Error:
====
GenArc
Transaction Error Report - System timestamp: Fri Sep 07 02:07:33 2001
-->Transaction: 1234567
Error in RPFAPErrorNotify(): FAP library error:
area:..\C\dxmerror.c
Jun 16 2001 12:44:04
400.101.002
DXMSetLastError>, code:<2>, code:<2>, msg<Invalid object handle was
passed>.
```

An example of the correct INI settings is shown in the [FSIUSER.INI sample on page 399](#).

VIEWING ARCHIVES IN DOCUMANAGE

You can use the ARCVIEW utility to view Documaker archive files checked into the Documanager archive system. This utility only runs under 32-bit Windows.

To use this utility, follow these steps:

- 1 Register the Documanager file extension (DPA) in Windows so the operating system will automatically use the ARCVIEW utility to view these files.
- 2 Set the FSIPATH environment variable to point to the directory where the INI file for the AFEMAIN program is stored. Here is an example:

```
FSIPath = d:\rpexl
```

NOTE: The AFEMAIN program is the executable file for Documaker Workstation.

- 3 Place a menu file, similar to the MEN.RES file used by Documaker Workstation, in the directory specified by the FSIPath option. The name of the menu file should be *ARCVIEW.RES*.

NOTE: You can edit this file to remove functionality you do not want to include.

- 4 Edit the FILETYPES.INI file on the computer where the Documanager server runs. Add the DPA file extension to the list of file types to view with the ARCVIEW.EXE program. This causes the Documanager client to use the viewer registered in Windows instead of the default Documanager viewer.

You can now click on Documaker archive files in Windows Explorer to display them.

USING MULTIPLE SIMULTANEOUS ODBC CONNECTIONS

The system supports multiple simultaneous ODBC connections via different ODBC drivers. This will, for instance, let you connect at the same time to multiple:

- Databases on an SQL server
- Databases on an SQL server and Excel spreadsheet databases
- Access databases and Excel spreadsheet databases
- Access databases
- Excel spreadsheet databases
- Databases for which you have an ODBC-compliant driver

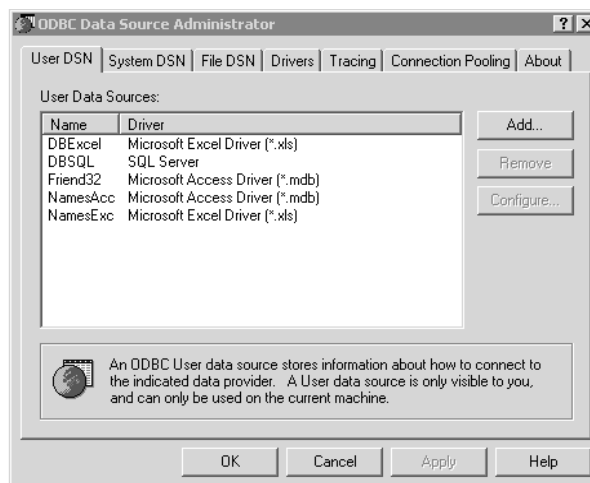
The system does not support multiple different DB2 databases using native DB2 drivers. Support is limited to ODBC-compliant data bases.

NOTE: Keep in mind the ODBC_FileConvert and ODBC_FieldConvert control groups are global and affect all of the handlers.

For example, to access a database on a SQL Server and in a Microsoft Excel spreadsheet simultaneously, you first set up the ODBC Data Sources Administrator panel as illustrated and these INI options:

```
< DBHandler:DBSQL >
  Class = ODBC
  Server = SQL Server
< DBHandler:DBEXCEL >
  Class = ODBC
  Server = MS Excel
```

The database handler name is limited to 22 characters.



For the table you want to open using the appropriate handler add this INI option:

```
< DBTable:MYTABLE >  
    DBHandler = DBSQL
```

Debug INI option can be specified under each of the DBHandler:XXX control group.

If you use the name of the ODBC handler in the appropriate DAL function, you can omit the DBTable:XXX control group. For more information on DAL functions and setting up database handlers for Excel databases, see the [DAL Reference](#).

USING WIP AND THE ARCHIVE INDEX FILE

Since the Archive module supports custom application archive index files, you must create an application archive index record from a WIP record. The following example shows a standard application archive index file.

The Archive option in the AFEProcedures control group defines the DLL and the function name to call when converting a WIP record into an archive record. The standard DLL is AFEW32 and the standard function is called AFEWip2ArchiveRecord. Here is an example of the standard DLL and function:

```
< AFEProcedures >
  Archive = AFEW32-> AFEWip2ArchiveRecord
```

The AFEWip2ArchiveRecord function uses options in the AFEWip2ArchiveRecord control group. Options in the AFEWip2ArchiveRecord control group are:

```
Archive Field Name = WIP Field Name
```

Where *ARCHIVE FIELD NAME* is the actual field name from archive DFD file and *WIP FIELD NAME* is the field name from WIP file. This means that data from WIP record field *WIP FIELD NAME* would be copied into archive record field *ARCHIVE FIELD NAME*.

For a base application archive index file, this control group and options are as follows:

```
< AFEWIP2ArchiveRecord >
KEY1          = KEY1
KEY2          = KEY2
KEYID         = KEYID
RECTYPE       = RECTYPE
CREATETIME    = CREATETIME
ORIGUSER      = ORIGUSER
CURRUSER      = CURRUSER
MODIFYTIME    = MODIFYTIME
FORMSETI      = FORMSETID
TRANCODE      = TRANCODE
STATUSCODE    = STATUSCODE
FROMUSER      = FROMUSER
FROMTIME      = FROMTIME
TOUSER        = TOUSER
TOTIME        = TOTIME
DESC          = DESC
INUSE         = INUSE
ARCKEY        = ARCKEY
APPDATA       = APPDATA
RECNUM        = RECNUM
RUNDATE       = RUNDATE
INVFLAG       = INVFLAG
CLAIMFL       = CLAIMFL
```

FORMATTING ARCHIVE FIELDS

The system lets you format data values that will be mapped to the archive index record from the Trigger2Archive control group. Normally, this group is defined like this:

```
< Trigger2Archive >
  Key1    = Company
  Key2    = LOB
  KeyID   = TransID
  RunDate = RunDate
```

NOTE: These same options in the ArcRet control group are used for searching the key fields in the archive index file.

Where the value on the left of the equals sign designates an archive index field (defined in APPIDX.DFD) and the value on the right represents a GVM variable normally associated with the NEWTRN record (defined by the TRNDFDFL.DFD). These options are used by the GenArc program to add the Key1, Key2, and KeyID information to the archive index file.

You can have the system format these archive fields in several ways:

- Preserving the case of values in the key fields
- Formatting dates
- Storing a constant value

Converting the case of key fields

By default, the system converts the case of information in the Key1, Key2, and KeyID fields to uppercase when it archives a record. It does this to reduce the amount of time it takes to find a record during a search. You can, however, use the CaseSensitiveKeys option to preserve the case of the Key1, Key2, and KeyID values as entered. For example, this option

```
< Archival >
  CaseSensitiveKeys = Yes
```

Tells the system to preserve the case of the Key1, Key2, and KeyID fields as entered. If you enter No or omit the CaseSensitiveKeys option, the system convert the values for these options to uppercase before it archives the record.

Reformatting dates

You can do optional date reformatting and assign a constant data value not associated with a GVM. Here is an example of date reformatting:

```
RUNDATE = TRANDATE;D1-4;D4
```

You still are associating the archive index field with a GVM variable normally loaded from the NEWTRN record. Separated by a semicolon, you can define the date format of the input variable and specify a different format for the final value after the second semicolon.

In this example, the RUNDATE field is to be set from the TRANDATE field from the NEWTRN record. Note the first *D* that follows the semicolon indicates you want a date conversion. This example converts the data from format *1-4* (MM-DD-YYYY) to format *D4* (YYYYMMDD) before storing it in the RUNDATE field of the archive index.

NOTE: Always use YYYYMMDD to store your run date in the archive.

Storing a constant value

Here is an example of how you store a constant value instead of associating the field with a GVM variable from the NEWTRN record.

```
USERID = NULL; ;TOM
```

Keep in mind that *NULL* is a keyword and is not interpreted as the name of a GVM variable associated with any record. When using *NULL*, the system skips to the final destination format section (the second semicolon) and places whatever value is defined there in the resulting archive index field. In this case, that value is *TOM*.

Since this method assumes there will be a constant text value defined after the second semicolon, you can also use INI built-in functions to provide this value. For instance, consider this example.

```
USERID = NULL; ; ~GETENV USERNAME
```

This is similar to the previous example except it uses the GetEnv (Get Environment Variable) INI function to get the value associated with *USERNAME* from the environment to supply the field value.

RETRIEVING ARCHIVED FORMS

Once the form set information has been archived, you can re-create those form sets by retrieving the form set information from the archive file, as long as you have access to the resource library which contains the forms. You do this using the Archive module of Documaker.

NOTE: The Archive module of Documaker can also archive form sets. For more information, see the [Documaker User Guide](#). The following information is provided here so you can have a basic understanding of the retrieval process.

FILES THE ARCHIVE MODULE USES

The Archive module (the AFEMAIN program) uses the archive index file to aid in the retrieval of form set information through the use of keys. You can define these keys to provide meaningful search criteria such as account or policy numbers, company names, or customer names.

Input files

- Compressed Archive (CAR) file
- Application Index file
- Catalog file
- Restart file
- Resource file such as FAPLIB, DEFLIB, and so on

Output files

None.

USING THE ARCHIVE MODULE

To retrieve a document from archive using the Archive module, you select the Retrieve, Formset option. The Retrieve Document window appears.

Company	Line of Business	Policy #	Run Date
			09/30/1997

Policy #	Create Dt	Modify Dt	Tr	St
----------	-----------	-----------	----	----

OK Refresh Options Cancel Help

You can configure the Retrieve Document window using these FSISYS.INI settings:

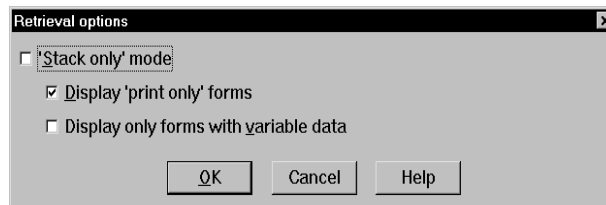
```
< Group1 >  
    Title1 = Company
```

```
Title2 = Line of Business
Title3 = Policy #
Title4 = Run Date
Title5 = Invoice Only
Title6 = Claim Only
Title7 = Policy #           Date           St Tr Description
```

NOTE: *Title5* and *Title6* are not used in the base Documaker Workstation system, but are available if you choose to customize your installation. If you remove these options from the FSISYS.INI file, the system does not display those fields.

Retrieval Options

If you click the Options button on the Retrieve Document window, the Retrieval Options window appears, as shown below.



This window is shown with default text. If you want to change these default values, add values to `DlgTitles` and `ArcRet` control groups as follows:

Beside this DlgTitles option	Enter the title for the...
RetOptionsDlgTitle	window (<i>Retrieve options</i> in this example)
RetrOptionsPrintOnly	Print only field
RetrOptionsOnlyEntry	Display only field
RetrOptionsStackOnly	Stack only mode field

The options in the ArcRet control group define only the default settings for fields users can change actual values by checking or unchecking the fields on the window.

For this ArcRet option	Enter...
Arrangement	StackOnly. If StackOnly mode is on, the system shows one form at a time and the Stack, Tile, and Cascade options are available. In this mode DisplayPrintOnly is set to <i>Yes</i> DisplayOnlyEntry is set to <i>No</i> and cannot be changed.
DisplayPrintOnly	Yes. This setting displays only the forms in the form set defined as Print Only, along with variable data forms included in the form set. These forms do not contain manually-entered data.
DisplayOnlyEntry	No. This setting displays only forms containing variable data. The system will omit reference forms.

WORKING WITH DOCUMANAGE

If you use Documanager as part of your archiving solution, you may want to use Documanager data types when mapping archive index data. You may also want to categorize the documents you archive.

These topics discuss how to do these tasks.

- [Using Documanager Data Type Support on page 420](#)
- [Setting Up Automatic Category Overrides on page 421](#)
- [Mapping Documaker Archive Fields to Documanager Properties on page 422](#)
- [Using Next/Retrieve Cursor on page 424](#)
- [Enhanced Documanager Document Extended Properties Support on page 425](#)

USING DOCUMANAGE DATA TYPE SUPPORT

Pulling Documaker archive documents (DPA files) into Documanage lets you use Documanage-supported data types when mapping the Documaker archive index data into the Documanage folder and document properties tables.

This lets you search, query, and present the data through Documanage clients such as Documanage Workstation and Documanage Bridge-based clients. For example, you can store Documaker date/time data as Documanage date/time data types and enable the use of date ranges and calendar functionality in web page design and for sorting and searching Documaker archive documents. Data mining and reporting can also benefit from better data representation and storage.

The DMIA DBHandler (DMILIB module: [DBHandler:DMIA]) used with the GenArc program and other Documaker Server archive processes lets you use additional Documanage Data Types in Documanage Folder fields instead of only supporting the varchar or char data types.

Keep in mind...

- The date/time data types must be in either a Documaker D4 string format:

YYYYMMDDHHMMSS

The hours, minutes, and seconds (HHMMSS) are optional. For example, the D4 format can be sent in as:

20070131 (Jan. 31, 2007)
2007013113 (Jan. 31, 2007 1PM)
200701311330 (Jan. 31, 2007 1:30PM)
20070131133055 (Jan. 31, 2007 1:30:55 PM)

Or in a Documanage client-supported string format:

YYYY-MM-DD HH:MM:SS.msec

The hours, minutes, seconds and milliseconds (HH:MM:SS.msec) are optional. For example, the Documanage format can be sent in as:

2007-01-31
2007-01-31 13
2007-01-31 13:30
2007-01-31 13:30:55
2007-01-31 13:30:55.800

- Documaker's Archive Application Index Data Format Definition file (APPIDX.DFD) fields must remain as CHAR_ARRAY for the INT_TYPE and EXT_TYPE with the appropriate INT_LENGTH and EXT_LENGTH values for representing the data in string format.

SETTING UP AUTOMATIC CATEGORY OVERRIDES

You can categorize DPA documents from Documaker Server Archive into Documanager. This makes it easier to do searches and queries when retrieving via Documanager Bridge. It also provides more flexibility in using Extended Document Properties (XDPs), which allows for different XDPs in the different document categories so transactions can store different relative data in the XDPs.

You can use input data to set the Documanager document's Category property during archival via the Documaker Server Archive interface (DMIA). The default value for this property comes from the FileType INI option during archival, but you can also dynamically override the default with input data using this INI option:

```
< POField2Document >
  ObjectClass = AppIdx_Field
```

During retrieval, the Category Document property can be loaded into the Documaker AppIdx_Field using this INI option:

```
< PODocument2Field >
  AppIdx_Field = ObjectClass
```

Extended Document Properties (XDPs) are based on the Category value set during ingestion. Mappings to XDPs only occur if the XDP for the Document Category exists by name. Otherwise, they are ignored and no error is generated. This allows different data to be populated into the XDPs based on the category used.

Here is an example of how you would override the default document category of DPA with the APPIDX.DFD field value of the field FormSet:

```
< DMIA:RPEX2ARC >
;   FileType is the default Category/ObjectClass value
  FileType = DAP
< PODocument2Field >
;   Category/ObjectClass is overridden by the value in the AppIdx
;   field FormSet
  FormSet = ObjectClass
< POField2Document >
;   Category/ObjectClass is overridden by the value in the AppIdx
;   field FormSet
  ObjectClass = FormSet
```

Keep in mind the APPIDX.DFD field used to override the document Category in the INI options POField2Document and PODocument2Field can not be used to also set other folder or document properties. For instance, in the example another entry for FormSet can not be used to map FormSet to another folder or document or XDP field.

MAPPING DOCUMAKER ARCHIVE FIELDS TO DOCUMANAGE PROPERTIES

When mapping Documaker archive field names to Documanager Folder and Extended Document Properties, you can use DB Field Name values. This lets you modify the Folder Property Name and Extended Document Property Name values in Documanager Server to effect changes to applications that use these values for input field/control labels without requiring reconfiguring your Documaker to Documanager interface setup.

You can map Documaker archive index data to either the Documanager Folder Property Name field and the Documanager Extended Document Property Name field (default behavior as previously provided) or to the Documanager DB Field Name, which is the database column name, based on the MapByDBName option.

```
< DMIA:cabinetname >  
  MapByDBName =
```

Option	Description
MapByDBName	Enter Yes to map to Documanager DB Field Names values for both Folder Properties and Extended Document Properties. The default is No, which instead maps them to the Folder Property Names and Extended Document Property Names (Display Names).

You can also use these new control groups for even more control over mapping:

- DMIA_FieldConvert_cabinetname
- DMIA_FieldConvert

NOTE: The DMIA_FieldConvert_cabinetname control group overrides any entries in the DMIA_FieldConvert control group.

Also, all filter and order by syntax generated and submitted to the Documanager Server and used in SQL statements now uses qualified column names instead of the Documanager Folder Property and Extended Document Property names to avoid requiring the DB column name to be the same as the Property Name.

Here are some examples:

Example 1

The Documaker archive index (ApplIdx) fields QTY and PreTaxAmt are mapped to Documanager Field or Extended Document Property name Quantity and Pretax Amount. All other Documaker archive index fields map to the same named Field and Extended Document Property names with a test for the name with spaces as they exist and then for spaces replaced with underscores (case-insensitive):

```
< DMIA:RPEX2ARC >  
  MapByDBName = No  
< DMIA_FieldConvert >  
  QTY = Quantity  
  PreTaxAmt = Pretax Amount
```


Example 2 The Documaker archive index fields QTY and PreTaxAmount are mapped to Documanager DB Field Name Quantity and PreTax_Amount. All other Documaker archive index fields map to the same named DB Field Name (case-insensitive):

```
< DMIA:RPEX2ARC >  
  MayByDBName = Yes  
< DMIA_FieldConvert >  
  QTY = Quantity  
  PreTaxAmt = Pretax Amount
```

USING NEXT/RETRIEVE CURSOR

Documanager supports a *next/retrieve cursor* for use by the ARCRET utility when accessing data from Documanager.

The ARCRET utility lets you retrieve records from archive and produce files. You can then send these files to plug-in functions to print or migrate the archive records or to test the archive retrieval results.

NOTE: The ARCRET utility's /REV parameter is only applicable to an archive stored in xBase.

Understanding the System

This eliminates the need to use the /BQ option for a Documanager archive. The previous (before version 11.3) interface to Documanager did not support retrieving documents while sequentially reading the index. The /BQ option told the system to queue batches of records into memory before attempting to retrieve each associated documents. This could be memory intensive and affected performance. With version 11.3 and higher, the system can retrieve the associated document while reading the index rows.

ENHANCED DOCUMANAGE DOCUMENT EXTENDED PROPERTIES SUPPORT

You can populate Documanager Extended Document Properties (XDPs) using Documaker Server archive indexed data. There are no limits to the number, sizes, and data types you can use at the document level. This lets you use XDPs when you are directly archiving to Documanager.

NOTE: Before version 11.1, only Documanager Basic Document Properties could only be used for user data and the number, size and type of data available was limited.

To use this feature, you must...

- Create the extended document properties in Documanager in the proper document categories
- Set up the GenArc program to map to them.
- Add the names you use for the XDP fields into GenArc's application index file (APPIDX.DFD).
- Set up Documaker Server to capture extract data to populate into the XDP fields.

The fields are propagated during GenTrn processing from the XML extract file to the TRNFILE. During GenData processing, the fields are populated from the TRNFILE to the NEWTRN file. Then, during GenArc processing, the fields are populated from the NEWTRN file to the APPIDX structure and into the Documanager XDP fields.

The field names added to the APPIDX.DFD file must have the exact same names as those set up in Documanager's Category Extended Properties. Here are some examples:

- PolicyDate
- PolicyType
- FormSet
- Number
- FinalDate
- Amount
- PreTaxAmt
- QTY
- Percentage
- Ratio
- Overage
- Specifier

For the appropriate fields to end up in the structure mapped by GenArc's APPIDX.DFD file, those fields must be propagated from the NEWTRN.DAT file. This file is created during GenData processing and is mapped using the TRNDFDFL.DFD file.

For the appropriate fields to exist in the NEWTRN file, those fields must be propagated from the TRNFILE. This file is created during GenTrn processing and is mapped by the TRNDFDFL.DFD file.

The TRNFILE is populated with data which is usually retrieved from the extract file. This data is mapped using the INI options in the Trn_Fields control group or by using the Ext2GVM rule in the AFGJOB.JDT file.

NOTE: Documanager Extended Document Properties is not supported by Docusave so the Stacked DPA feature will not propagate the XML header data in the DPA files into Documanager's XDP fields.

To handle the propagation of these fields, you must include additional information in these files:

- FSISYS.INI file or the AFGJOB.JDT file or both
- TRNDFDFL.DFD file
- APPIDX.DFD file
- Extract file

Here are some examples of the additional information required in these files:

FSISYS.INI file

Here is an excerpt from the FSISYS.INI file:

```
< Trn_Fields >
  SYM = 1,3,N
  POL = 4,7,N
  EffectiveDate = 25,6,N;DB;D4
  Module = 38,2,N
  State = 43,2,N
  Trn_Type = 45,2,N
  Company = 35,3,N
  LOB = 40,3,N
  SentToManualBatch = 47,2,N
  Branch = 49,2,N
  RunDate = 51,14,N
  DueDate = 100,8,N
  Cust_Num = 87,10,N
  PKG_Offset = 97,10,N
  TRN_Offset = 107,10,N
  X_Offset = 117,10,N
  NA_Offset = 127,10,N
  POL_Offset = 137,10,N
  TokenLen = 118,316,N
; PolicyDate = 51,14,N
  PolicyType = 45,2,N
  FormSet = 38,2,N
< Trigger2Archive >
  Key1 = COMPANY
  Key2 = LOB
  KeyID = POL
  Customer = customer
  RunDate = RUNDATE
```

```

        DueDate = DueDate
        TokenLen = TOKENLEN
        PolicyDate = PolicyDate
        PolicyType = PolicyType
        FormSet = FormSet
        Number = Number
        FinalDate = FinalDate
        Amount = Amount
        PreTaxAmt = PreTaxAmt
        Qty = QTY
        Percentage = Percentage
        Ratio = Ratio
        Overage = Overage
        Specifier = Specifier
    < Trn_File >
        MaxExtRecLen = 750
        BinaryExt = N

```

TRNDFDFL.DFD file

Here is an excerpt from the TRNDFDFL.DFD file:

```

    < FIELDS >
    FIELDNAME = sym
    FIELDNAME = pol
    FIELDNAME = EffectiveDate
    FIELDNAME = module
    FIELDNAME = state
    FIELDNAME = trn_type
    FIELDNAME = company
    FIELDNAME = lob
    FIELDNAME = SentToManualBatch
    FIELDNAME = branch
    FIELDNAME = RunDate
    FIELDNAME = DueDate
    FIELDNAME = cust_num
    FIELDNAME = customer
    FIELDNAME = PKG_Offset
    FIELDNAME = TRN_Offset
    FIELDNAME = X_Offset
    FIELDNAME = NA_Offset
    FIELDNAME = POL_Offset
    FIELDNAME = TOKENLEN
    FIELDNAME = PolicyDate
    FIELDNAME = PolicyType
    FIELDNAME = FormSet
    FIELDNAME = Number
    FIELDNAME = FinalDate
    FIELDNAME = Amount
    FIELDNAME = PreTaxAmt
    FIELDNAME = QTY
    FIELDNAME = Percentage
    FIELDNAME = Ratio
    FIELDNAME = Overage
    FIELDNAME = Specifier

    < FIELD:PolicyDate >
    INT_TYPE = CHAR_ARRAY

```

```
INT_LENGTH = 24
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
EXT_LENGTH = 23
KEY = N
REQUIRED = Y

< FIELD:PolicyType >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 31
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
EXT_LENGTH = 30
KEY = N
REQUIRED = Y

< FIELD:FormSet >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 41
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
EXT_LENGTH = 40
KEY = N
REQUIRED = Y

< FIELD:Number >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 11
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
EXT_LENGTH = 10
KEY = N
REQUIRED = N

< FIELD:FinalDate >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 24
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
EXT_LENGTH = 23
KEY = N
REQUIRED = N

< FIELD:Amount >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 16
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
EXT_LENGTH = 15
KEY = N
REQUIRED = N

< FIELD:PreTaxAmt >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 16
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
EXT_LENGTH = 15
KEY = N
REQUIRED = N

< FIELD:QTY >
```

```

INT_TYPE = CHAR_ARRAY
INT_LENGTH = 6
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
EXT_LENGTH = 5
KEY = N
REQUIRED = N

< FIELD:Percentage >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 10
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
EXT_LENGTH = 9
KEY = N
REQUIRED = N

< FIELD:Ratio >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 9
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
EXT_LENGTH = 8
KEY = N
REQUIRED = N

< FIELD:Overage >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 11
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
EXT_LENGTH = 10
KEY = N
REQUIRED = N

< FIELD:Specifier >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 2
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
EXT_LENGTH = 1
KEY = N

```

APPIDX.DFD file

Here is an excerpt from the APPIDX.DFD file:

```

[FIELDS]
FIELDNAME=KEY1
FIELDNAME=KEY2
FIELDNAME=KEYID
FIELDNAME=customer
FIELDNAME=RUNDATE
FIELDNAME=DueDate
FIELDNAME=INVFLAG
FIELDNAME=CLAIMFL
FIELDNAME=ARCKEY
FIELDNAME=FORMSETID
FIELDNAME=TOKENLEN
FIELDNAME = PolicyDate
FIELDNAME = PolicyType
FIELDNAME = FormSet
FIELDNAME = Number

```

```
FIELDNAME = FinalDate
FIELDNAME = Amount
FIELDNAME = PreTaxAmt
FIELDNAME = QTY
FIELDNAME = Percentage
FIELDNAME = Ratio
FIELDNAME = Overage
FIELDNAME = Specifier
```

```
< FIELD:PolicyDate >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 24
EXT_TYPE = CHAR_ARRAY
EXT_LENGTH = 24
KEY = N
REQUIRED = Y
```

```
< FIELD:PolicyType >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 30
EXT_TYPE = CHAR_ARRAY
EXT_LENGTH = 30
KEY = N
REQUIRED = Y
```

```
< FIELD:FormSet >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 40
EXT_TYPE = CHAR_ARRAY
EXT_LENGTH = 40
KEY = N
REQUIRED = Y
```

```
< FIELD:Number >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 10
EXT_TYPE = CHAR_ARRAY
EXT_LENGTH = 10
KEY = N
REQUIRED = N
```

```
< FIELD:FinalDate >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 24
EXT_TYPE = CHAR_ARRAY
EXT_LENGTH = 24
KEY = N
REQUIRED = N
```

```
< FIELD:Amount >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 15
EXT_TYPE = CHAR_ARRAY
EXT_LENGTH = 15
KEY = N
```



```
REQUIRED = N

< FIELD:PreTaxAmt >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 15
EXT_TYPE = CHAR_ARRAY
EXT_LENGTH = 15
KEY = N
REQUIRED = N

< FIELD:QTY >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 5
EXT_TYPE = CHAR_ARRAY
EXT_LENGTH = 5
KEY = N
REQUIRED = N

< FIELD:Percentage >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 9
EXT_TYPE = CHAR_ARRAY
EXT_LENGTH = 9
KEY = N
REQUIRED = N

< FIELD:Ratio >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 8
EXT_TYPE = CHAR_ARRAY
EXT_LENGTH = 8
KEY = N
REQUIRED = N

< FIELD:Overage >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 10
EXT_TYPE = CHAR_ARRAY
EXT_LENGTH = 10
KEY = N
REQUIRED = N

< FIELD:Specifier >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 1
EXT_TYPE = CHAR_ARRAY
EXT_LENGTH = 1
KEY = N
REQUIRED = N
```

NOTE: DATE type data must be passed in a format that is accepted by Documanage Server or in a Documaker Server D4 format (YYYYMMDD).

AFGJOB.JDT file**Here is an excerpt from the AFGJOB.JDT file:**

```
;Ext2Gvm;2;11,TOTAL1REC 147,4,Number;  
;Ext2Gvm;2;11,TOTAL1REC 25,24,PolicyDate;  
;Ext2Gvm;2;11,TOTAL1REC 49,23,FinalDate;  
;Ext2Gvm;2;11,TOTAL1REC 143,15,Amount;  
;Ext2Gvm;2;11,TOTAL1REC 158,10,PreTaxAmt;  
;Ext2Gvm;2;11,TOTAL1REC 168,4,QTY;  
;Ext2Gvm;2;11,TOTAL1REC 172,3,Percentage;  
;Ext2Gvm;2;11,TOTAL1REC 175,8,Ratio;  
;Ext2Gvm;2;11,TOTAL1REC 183,6,Overage;  
;Ext2Gvm;2;11,TOTAL1REC 189,1,Specifier;
```

Extract file**Here is an excerpt from a single record in a flat extract file:**

```
SCOREMOVEDHEADERREC00000030194 SCOM1FP GAT1I1B119950123 804-345-8789  
041594 REMOVEDOOO 20000223 MAMTEST TOKEN LENGTH TEST TOKEN LENGTH  
TEST TOKEN LENGTH TEST TOKEN LENGTH TEST TOKEN LENGTH ARCCAB DAP  
SubTypeTest1 TitleTest1 TEST DESCRIPTION 1 19950124 Complete  
UserFlag1Test1 UserFlag2Test1 Keyword1Test1 Keyword2Test1 X  
SCOREMOVEDTOTAL1RECP00002005-01-01 12:00:00.001 2006-01-01  
12:00:00.999 Comprehensive FullLine 1000000.00 1228.98 2 1001.1 98.76  
B X
```

APPENDIX A

Setting Up Archive/ Retrieval Configurations

This section outlines several commonly-used archive/retrieval scenarios. Click on a scenario to quickly go to that discussion:

- [DB2 Server on OS/390 — Windows Client on page 434](#)
- [DB2 Server on Windows — Windows Client on page 446](#)
- [DB2 Server and Client on Windows on page 451](#)
- [SQL Server on Windows — ODBC Client on Windows on page 455](#)
- [IDS on Windows — DB2 Archive on z/OS on page 457](#)
- [Creating a z/OS Database on page 458](#)

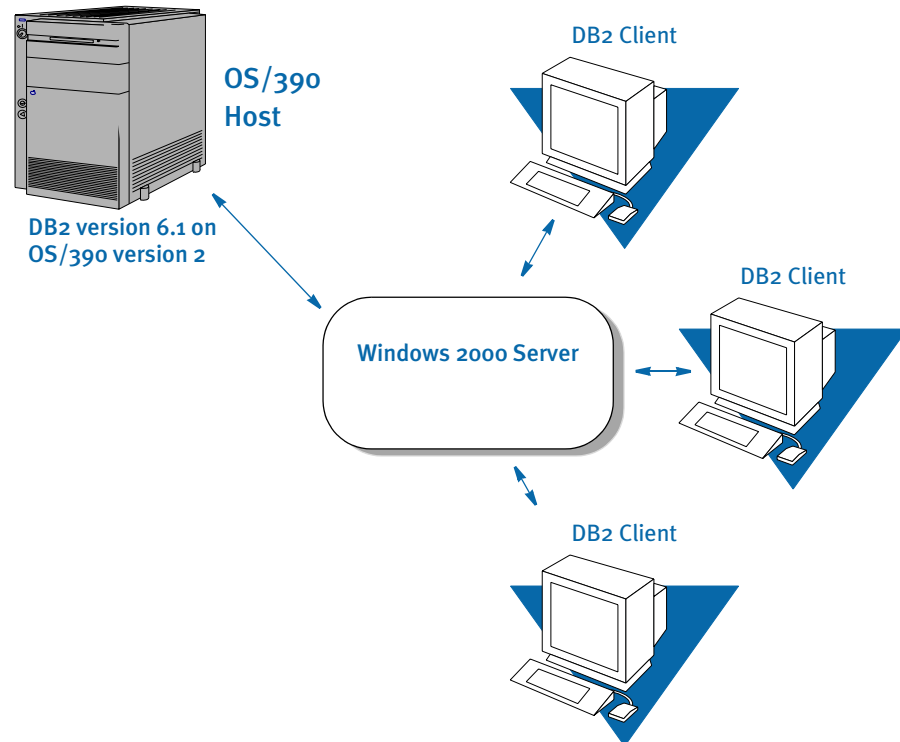
NOTE: Windows refers to 32-bit Windows operating systems, such as Windows 2000 or Windows XP.

We recommend that you only use uppercase for table and column names when storing information in a database. For instance, avoid CustomerName, Customername, or customername and instead use CUSTOMERNAME.

Database management systems (DBMS) vary in how they handle case issues so it is best to standardize on uppercase. With version 11.2, all column names must be in uppercase.

DB2 SERVER ON OS/390 — WINDOWS CLIENT

For this scenario, assume you are running DB2 version 6.1 on OS/390 version 2. For the DB2 client, assume you are running Windows 2000 or Windows XP.



The DB2 Distributed Data Facility is an optional part of the DB2 product on OS/390. The Distributed Data Facility must be configured and running for the DB2 client (on 32-bit Windows) to communicate with the DB2 Server (on OS/390).

CONFIGURING THE SERVER

Getting the DB2 location name and LUNAME

You can use the PRTLOGMP DB2 utility to print a report that lists the communication record of the DB2 Bootstrap Dataset. In the communication record you can find the DB2 location and LU name for that DB2 subsystem. The location and LU name are needed when configuring the SNA Server and DB2 on the 2000 Server.

Here is an example of the JCL used to run PRTLOGMP is shown follows, along with the communication record portion of the output from the PRTLOGMP utility.

```
/* * COPY JOBCARD HERE ...
/* *
//S1 EXEC PGM=DSNJU004
//SYSUT1 DD DSN=TDB1.BSDS01,DISP=SHR
//SYSPRINT DD SYSOUT=*

**** DISTRIBUTED DATA FACILITY ****
COMMUNICATION RECORD
15:35:33 OCTOBER 12, 1999
LOCATION=USFSIMVSTDB1 LUNAME=DB2TDB1 PASSWORD=(NULL)
DSNJ200I DSNJU004 PRINT LOG UTILITY PROCESSING COMPLETED
SUCCESSFULLY
```

Defining the SNA server's APPC LU in VTAM

The following Switched Major Node (SNA) is contained in SYS1.VTAMLST(SWoE40C):

```
* * * * *
*
* VTAM SWITCHED MAJOR NODE
* FOR MICROSOFT SNA SERVER COMMUNICATIONS
*
* * * * *
*
SWOE40C VBUILD TYPE=SWNET,MAXGRP=3,MAXNO=30
*
CP00010 PU ADDR=40,CPNAME=CL00010, X
DISCNT=NO,MAXDATA=16384,USSTAB=USSFSIS, X
MAXPATH=1,MAXOUT=7,PASSLIM=7, X
VPACING=7,PACING=7,SSCPFM=USSSCS
*
CL00010 LU LOCADDR=0.
```

Defining the DB2 Application Major Node in VTAM

The following Application Major Node is contained in SYS1.VTAMLST(DB2TDB1A):

```
* * * * *
*
* VTAM APPLICATION NODE FOR DB2
*
* * * * *
*
DB2TDB1A VBUILD TYPE=APPL
DB2TDB1 APPL APPC=YES, X
ATNLOSS=ALL, X
AUTH=(ACQ), X
AUTOSES=1, X
DMINWNL=25,X
DMINWNR=25, X
DSESLIM=50, X
MODETAB=, X
SECACPT=ALREADYV, X
SRBEXIT=YES, X
SYNCLVL=SYNCPT, X
VERIFY=NONE, X
VPACING=2
```

Setting Up the Windows 2000 Server (Middle Tier)

Installing and configuring Microsoft's SNA Server

To set up the middle tier, first install Microsoft SNA Server version 4, with Service Pack 3 applied, onto a server running Windows 2000 Server. Then Install SNA Server into its own domain called *USRo4SNA*.

Here are the steps for installing the SNA Server:

- 1 Insert the install CD into CD drive. Select Start, Run and enter this command:

```
e:\sna40\i386\setup.exe
```

Go through the normal set up process. Enter this server domain information:

In this field	Enter
Domain	<i>your domain name</i>
Account	<i>your account user name</i>
Password	(leave blank)
Confirm Password	(leave blank)

Click Ok when finished.

- 2 Choose *Primary Configuration Server*. Then choose *Named Pipe, TCP/IP, IPX/SPX*.
- 3 Choose *IPX/SPX Directory Service*. Then choose *Bindery* (Netware 3.x, 4.x, 5.x or 6.x) and *SNA Server Subdomain (USRo4SNA)*.
- 4 Next, use the Microsoft SNA Server Manager to make the following definitions. To start this tool select Start, Programs, Microsoft SNA Server (Common), Manager. Then right click the SNA Server you created in the first three steps. Choose Insert, Link Services. From the Insert Link Services window select your adapter and protocol (DLC 802.2 Link Service).
- 5 Select and right click the SNA Server you configured (USRSRV04). Select Properties.

The control point configured here is for incoming connections only and is not used for this outgoing connection to OS/390. You should, however, configure it. Use the Network Name (P390) and Control Point name (CLO0010). Accept the defaults on the Server Configuration tab.

NOTE: The *network name* matches the value of the NETID parameter in the VTAM startup parameters in SYS1.VTAMLST(ATCSTR00). The control point name (CLO0010) here matches the value of the CPNAME parameter of the VTAM Switched Major Node on OS/390, in SYS1.VTAMLST(SW0E40C).

- 6 Select and right click on Connections under the SNA Server you configured. Move to APPC and select *Local LU*. The Local LU Alias can be whatever you want but in this scenario it's the same as the LU Name (CLO0010).

Enter the network name (P390). Enter an LU Name that matches the control point name used above (CLO0010). Click the Advanced tab. Check *Member Of Outgoing Local APPC LU Pool*. Make sure that the LU 6.2 Type is set to Independent then click Ok.

- 7 Select and right click on Connections under the SNA Server you configured. Move to APPC and select *Remote LU*. Use the Connection List to select your connection (ETH2MVS). The LU Alias can be whatever you want but in this scenario it's the same as the LU Name (DB2TDB1) — remember this is the remote LU Name.

Enter the network name (P390) and LU Name (DB2TDB1) and uninterpreted name (DB2TDB1). Click the Option tab. Accept the defaults. The PLU for DB2 | OS/390 is *independent* to support parallel sessions. Click Ok.

NOTE: Remote LU Name here should match the APPL name of the DB2 application major node in SYS1.VTAMLST(DB2TDB1A). Note that the member name (DB2TDB1A) cannot be the same as the APPL name (DB2TDB1) within it.

- 8 Move to Configured Users, right click, select Insert, and click on User. Highlight *Everyone* and click Add. Go back to the SNA Manager window where you should now see *Everyone* under Configured Users. Right click on *Everyone*, choose Properties and then click the APPC Defaults tab.

Click the list for Local APPC LU and choose (CLOoo10). Click the list for the Remote APPC LU and choose (DB2TDB1).

- 9 Move down to APPC Modes, right click, select Insert, APPC, and click on Mode Definition. Enter the mode name (IBMRDB). Click the Limits tab. Enter the Parallel Session Limit (10), Minimum Contention Winner Limit (3), Partner Minimum Contention Winner Limit (3), and Automatic Activation Limit (2). Accept the defaults on the Characteristics tab and click Ok.

- 10 Move to CPIC Symbolic Names, right click, select Insert, APPC. Click on CPIC Symbolic Name. This name can be anything you want but it must later match something in DB2 on Windows 2000. This name is case sensitive.

For this scenario, use *DB2CPIC* (in all caps). Choose Conversation Security (Same), Mode Name (IBMRDB). Click the Partner Information tab. In the Partner TP Name area click SNA Service TP (in hex) and enter **07F6C4C2**. In the Partner LU Name area click Alias and enter **Partner LU alias (DB2TDB1)**. Click Ok.

The CPIC Symbolic Name (DB2CPIC) must match the destination name when you define the node entry in DB2 on the Windows 2000 Server (see the following section).

Installing and Configuring Microsoft's SNA Server

For this scenario, you should install Microsoft SNA Server version 4, with Service Pack 3 applied, onto a Server running Windows 2000 Server. Install SNA Server into its own domain and call the domain *USRo4SNA*.

Follow these steps to install SNA Server 4.0 SP3:

- 1 Insert the install CD into CD-ROM drive. Go to Start, Run and enter:
`e:\sna40\i386\setup.exe`
 Then click Ok. Go through the normal set up process.
- 2 Choose Primary Configuration Server. Then choose *Named Pipe, TCP/IP, IPX/SPX*.
- 3 Choose IPX/SPX Directory Service. Then choose *Bindery (Netware 3.x, 4.x, 5.x or 6.x)*.
- 4 Choose *SNA Server Subdomain (USRo4SNA)*.
- 5 Next, set up this server domain information:

Field	Enter
Domain	<i>your domain name</i>
Account	<i>your account user name</i>
Password	(leave blank)
Confirm Password	(leave blank)

Click Ok.

Configuring SNA Server 4.0 SP3

The following definitions are made using the Microsoft SNA Server Manager tool. To start this tool select Start, Programs, Microsoft SNA Server, Manager.

- 1 Right click the server you created. Choose Insert, Link Services. From the Insert Link Services window select your adapter and protocol (DLC 802.2 Link Service). Click Add. The properties window for that protocol appears. Click Ok.
- 2 Expand the server. Right click on SNA Service and choose Properties. The control point configured here is for incoming connections only and is not used for this outgoing connection to OS/390.

You should, however, configure it. Enter the network name (P390) and control point name (DL00010). The comment field is optional. Click Ok.

NOTE: The *network name* matches the value of the NETID parameter in the VTAM startup parameters in SYS1.VTAMLST(ATCSTR00). The *control point name* (DL00010) matches the value of the CPNAME parameter of the VTAM Switched Major Node on OS/390, in SYS1.VTAMLST(SWOE40D).

- 3 Highlight SNA Service and on the right hand side of the screen click the Connections tab. Right click the Connections tab. Choose Insert, APPC, Local LU. The Local LU Alias can be set to is whatever you want but for this scenario set it to the LU Name (DL00010). Enter the LU Alias (DL00010). If you tab to the next field the network name and LU name automatically appear in those fields. If this information does not appear, enter **P390** as the network name and **DL00010** as the LU Name. The comment is optional. Click the Advanced tab. Check Member of Default Outgoing Local APPC LU Pool. Make sure *Independent* is selected for the section LU 6.2 type. Click Ok.
- 4 Right click the Connections tab and choose Insert, Connection, 802.2.
- 5 On the General tab, enter a name for your connection, such as ETH2MVS. Choose SNADLC1 (or whatever the option may be) for the link service. The Comment is optional. In the Remote End section, choose *Host System*. In the Allowed Directions section, choose *Outgoing Calls*. In the Activation section choose *On Server Startup*.
- 6 On the Address tab, enter your remote network address, such as 10005A6EA879. Set the Remote SAP Address to **0x04**.

- 7 On the System Identification tab, make sure the following information is filled in. In the Local Node Name section, the network name should be *P390*, the control point name should be *DL00010*, and the local node ID should be *05D FFFF*. In the XID Type section, *Format 3* should be selected.

In the Remote Node Name section, the network name should be *P390* and the control point name should be *USS3270*. Make no changes on the 802.2 DLC tab. Click Ok.

NOTE: The Remote LU Name should match the APPL name of the DB2 application major node in SYS1.VTAMLST(DB2TDB1A). Note that the member name (DB2TDB1A) cannot be the same as the APPL name (DB2TDB1) within it.

- 8 Right click on APPC Modes. Choose Insert, APPC, Mode Definition. On the General tab, enter a mode name, such as *IBMRDB*. The Comment field is optional. On the Limits tab, enter **10** for the parallel session Limit. Enter **3** for the minimum contention winner limit. Enter **3** for the partner minimum contention winner limit. Enter **2** for the automatic activation limit. Leave the Characteristics, Partners, and Compression tabs as is. Click Ok.
- 9 Highlight SNA Service. Right click the Connections tab on the right side of your screen. Choose Insert, APPC, Remote LU. On the General tab, choose *ETH2MVS*. The LU alias can be whatever you want but in this scenario it's *DB2TDB1*. Make sure the following information is in these fields:

Field	Entry
Network Name	P390
LU Name	same as your alias DB2TDB1
Uninterpreted Name	same as your alias DB2TDB1
Comment	optional

On the Options tab, choose *IBMRDB* for the implicit incoming mode. Leave everything else as is. Click Ok.

- 10 Move to Configured Users, right click, select Insert, and click on User. Highlight *Everyone* and click Add. *Everyone* appears in the Add Names box. Click Ok. Go to the SNA Manager Window where you should now see *Everyone* under Configured Users. Right click on *Everyone*, choose Properties, and then click the APPC Defaults tab. Choose *DL0010* as the local APPC LU. Then choose *DB2TDB1* as the remote APPC LU. Click Ok.
- 11 Move to CPIC Symbolic Names, right click, select Insert, select APPC, and click *CPIC Symbolic Name*. This name must match something in DB2 on the Windows 2000 server and is case sensitive. For this scenario, enter **DB2CPIC**.

- 12 Choose *Same* as the Conversation Security and *IBMRDB* as the mode name. The Comment field is optional. Click the Partner Information tab. In the Partner TP Name area, click *SNA Service TP (in hex)* and enter **07F6C4C2**. In the Partner LU Name area, click *Alias* and enter *Partner LU alias (DB2TDB1)*. Click Ok.

NOTE: The CPIC symbolic name (DB2CPIC) must match the destination name when you define the node entry in DB2 on the Windows 2000 Server. This is discussed further in the following topic.

Setting Up DB2 on a Windows 2000 Server

On the Windows 2000 Server, this scenario assumes DB2 version 8.1 for Windows is installed with version 2.3.2 of the Distributed Database Connection Services.

Installing DB2 on a Windows 2000 Server

Follow these steps:

- 1 Insert the installation CD and go to Start, Run. Then enter the following command, substituting the appropriate drive letter for the CD drive:

```
e:\setup /I=LANGUAGE
```

Where *LANGUAGE* represents the two-character country code for your language (for example, EN for English).

Click Ok.

- 2 The installation routine asks if you would like to view the read me file. If not, click Next.
- 3 Check IBM Database 2, select the Server option, and check Distribution Database Connection Services (DDCS). Then select the Multi-User gateway option. Click Next.
- 4 Choose *Try and Buy Only* for both options then click Next.
- 5 Choose Full installation and click Next. Accept the default destination directory and drive letter and click Install. The installation routine asks if you want to reboot:

```
Yes, reboot  
OR  
No, wait to reboot
```

Choose one of these options and click Finish.

All of the following definition descriptions were performed using DB2's Database Director. To start this tool choose Start, Programs, DB2 For Windows, Database Director.

Configure the DB2 instance

Click the plus sign (+) to the left of the Database Managers icon to expand it. Then right click on the DB2 icon and choose Configure. On the Protocols tab, enter db2inst1 in the Service Name field. Click Ok.

Defining an OS/390 node

Click the plus sign (+) to the left of the DB2 icon to expand it, then click the plus sign to the left of the Directories icon to expand it. Double-click the Node Directory icon, then choose Directory Entry, Catalog.

	Enter a Node Name (OS/390), an optional comment, choose the protocol type (APPC) and the destination name (DB2CPIC), and then choose the security type (Program). Click Ok.
Defining a system database entry	<p>Click the plus sign (+) to the left of the DB2 icon to expand it, then click the plus sign to the left of the Directories icon to expand it. Double-click the System Database Directory icon, then choose Directory Entry, Catalog.</p> <p>On the General tab, choose Type for the Remote radio button. Click the Remote tab and enter the database name (ARCDB) and alias (ARCDB). Choose Node from the list (OS/390). <i>Do not</i> click the box labeled DDCS or Back level Database. Click Ok.</p>
Updating TCP/IP values on the Windows 2000 server	The next step is to update TCP/IP-related values on the Windows 2000 server. For information on how to do this, see Updating TCP/IP values on the Windows 2000 server on page 441 .
Defining a database connection services entry	<p>Click the plus sign (+) to the left of the DB2 icon to expand it, then click the plus sign to the left of the Directories icon to expand it. Double-click the Database Connection Services Directory icon, then choose Directory Entry, Catalog.</p> <p>Choose Database (ARCDB). For Target Database, enter the location name for the DB2 subsystem on OS/390.</p>

Installing and Configuring DB2 on a Windows 2000 Server

This scenario assumes DB2 for Windows was installed and DB2 Server was at version 8.1.

All of the following definition descriptions were performed using DB2's Control Center tool. To start this tool choose Start, Programs, DB2 for Windows, Administration Tools, Control Center.

Defining an OS/390 system	Right click on Systems and choose Add. On the Add System window, click the drop down arrow for the operating system. Choose <i>MVS/ESA</i> , and enter P390 for the system name. Click Apply. A confirmation message appears. Click Close.
Defining a DB2 instance	Expand the Host System (P390) you created in the previous topic. Right click on Instances and choose Add. Enter names such as <i>DB2TDB1</i> as a remote instance and <i>DB2</i> as a destination name. Choose <i>APPC</i> as the protocol. In the Security Section of this window, choose <i>Same</i> and click Apply. A confirmation message appears. Click Close.
Defining an OS/390 database	Expand out the newly created Instance from the previous section (DB2TDB1). Right click on Databases and choose Add. Enter a database name and alias, such as ARCDB, and click Apply. A confirmation message appears. Click Close.

Setting Up Universal Database on Windows 2000

Installing Universal Database

This involves installing Universal Database (UDB) version 6.1 EE:

- 1 On the Welcome window, click Next. Then select the DB2 Enterprise Edition option and click Next. Then click Custom.
- 2 Select the components you need. Make sure the Destination folder is correct and click Next. The Configure DB2 Services window appears.

- 3 Make sure there is a DB2 instance (DB2) and an Administration Server (DB2DASoo) then click Next.
- 4 Check the user name and password for the Administration Server.

```
username = dbadmin  
password = (password)
```

Click Next.
- 5 On the Start Configuring Files window, click Next. Then decide if you want to restart your computer and click Finish.

Configuring Universal Database

Follow these steps to configure UDB version 6.1 EE:

- 1 Choose Start, Programs, DB2 for Windows, Client Configuration Assistant.
- 2 Click Add Database if you have just installed. Click Add to add databases if you have already created databases. The Add Database Smart Guide appears.
- 3 On the Source tab (step 1), choose the Manually Configure a Connection to a DB2 Database option and click Next.
- 4 On the Protocol tab (step 2), choose TCP/IP as the protocol. Select OS/390 as the target operating system. Click Next.
- 5 On the TCP/IP tab (step 3), set the following fields:

In this field	Enter
Host Name	os390
Port Number	446
Service Name	(leave blank)

Click Next.

- 6 On the Database tab (step 4), set the following fields:

In this field	Enter
Location Name	USDCIOS39DSN1
Database Alias	ARCDB
Comment	(optional)

Click Next.

- 7 On the ODBC tab (step 5), check the Register this Database for ODBC option. Then select the appropriate data source. Click Done.

Updating TCP/IP-related Values on a Windows 2000 Server

Follow these steps to update TCP/IP values on a Windows 2000 Server.

- 1 Enter these lines into the services file (c:\winnt\system32\drivers\etc\services):

```
db2inst1 3702/tcp # db2 port
db2inst1 3703/tcp # db2 port interrupt
```

- 2 Go to Programs, Start, Settings, Control Panel, System and choose the Environment tab. Enter a system variable called **DB2COMM** and set its value to **APPC, TCP/IP**.

This indicates the communication protocols DB2 will use — APPC talks to the OS/390 Host and TCP/IP talks to the Windows clients.

- 3 Add a system variable called **DB2CODEPAGE** and set its value to **850**.
- 4 Reboot your system to apply these changes.

COMMON DB2 ERRORS

Here is a list of some common DB2 errors:

Error	Description
SQL30073 “119C” Parameter value “” is not Supported	This is a problem with CCSID or code page. Select Start, Control Panel, System, and click the Environments tab. Enter a system variable called DB2CODEPAGE and set the value to 850 . You must reboot for the change to take affect.
SQL30081NA communication error has been detected	This problem is related to the SNA Connectivity parameters.
Protocol specific error 9	First look at the CPIC symbolic destination name and make sure everything is correct. Also check the Partner LU and Local LU definitions. If you change any of these parameters only a stop and restart of SNA Server is required.
Protocol specific error 1	The first thing to look at is the Link. Make sure it has started and you have a valid connection to the host.
Protocol specific error 2	Look at your LU definitions for both the Local LU and Partner LU. Make sure they are correctly defined.

SETTING UP CLIENTS

This scenario assumes DB2 for Windows version 8.1 is installed and the Distributed Database Connection Services is at version 2.3.2.

All of the following definition descriptions were performed using DB2’s Database Director. To start this tool choose Start, Programs, DB2 For Windows, Database Director.

Defining a DB2/2000 node

Click the plus sign (+) to the left of the DB2 icon to expand it, then click the plus sign to the left of the Directories icon to expand it. Double-click the Nodes Directory icon, then choose Directory Entry, Catalog.

Defining a system database entry

Enter a node name (NT04), an optional comment, and choose the protocol type (TCP/IP). For the host name, enter your server name and for the service name enter **DB2INST1**. Click Ok.

Click the plus sign (+) to the left of the DB2 icon to expand it, then click the plus sign to the left of the Directories icon to expand it. Double-click the System Database Directory icon, then choose Directory Entry, Catalog.

On the General tab, choose Remote for Type. Click the Remote tab and enter the database name (ARCDDB) and alias (ARCDDB). Choose Node from the list (NT04).

Do not click the boxes labeled DDCS or Back Level Database. Click Ok.

Updating TCP/IP-related values on a Windows client

Follow these steps so the system can update TCP/IP related values on a Windows client:

- 1 So the system can find the host name (see [Configuring SNA Server 4.0 SP3 on page 438](#)), make this entry in the hosts file (c:\windows\system32\drivers\etc\hosts):

```
10.8.10.211    USRSRV04
```

The left indicates the IP address of the server and right indicates the host name.

- 2 Enter these lines in the services file (c:\windows\system32\drivers\etc\services):

```
db2inst1 3702/tcp # db2 port
db2inst1 3703/tcp # db2 port interrupt
```

- 3 Go to Programs, Settings, Control Panel, System, and click the Environment tab. Enter a system variable called *DB2COMM* and set its value to *TCP/IP*. This indicates the communication protocols DB2 will use (TCP/IP) to talk to the Windows Server. Also add a system variable called *DB2CODEPAGE* and set its value to *850*. Reboot your system to apply these changes.

Setting Up the INI Options for the DB2 Driver

Here are the INI options for the DB2 driver:

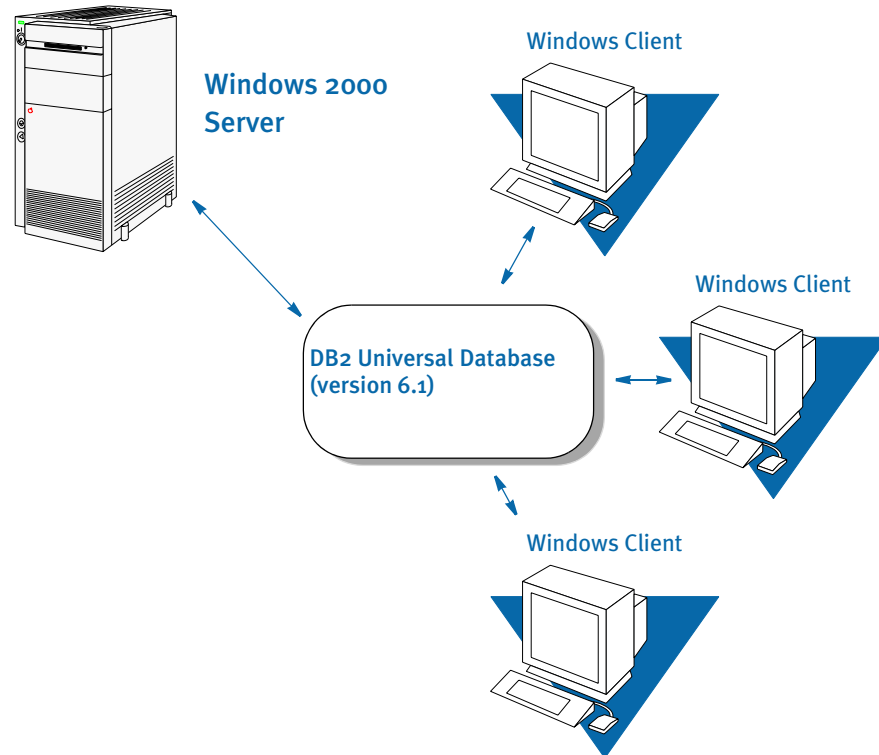
```
< Archival >
  ArchiveMem = Yes
< ArcRet >
  AppIdxDfd = Deflib\AppIdx.dfd
  AppIdx = APPIDX
  CARFile = ARCHIVE
  CARPath =
  Catalog = CATALOG
  RestartTable = RESTART
< DBHandler:DB2 >
  BindFile = c:\rel10\fap400\w32bin\db2lib.bnd
  Database = ARCDDB
  CreateTable = Yes
  CreateIndex = No
  UserID = (OS/390 user ID)
  PassWd = (OS/390 password)
< DBTable:APPIDX >
  DBHandler = DB2
< DBTable:ARCHIVE >
  DBHandler = DB2
```

```
< DBTable:CATALOG >
    DBHandler = DB2
< DBTable:RESTART >
    DBHandler = DB2
< DB2_FileConvert >
    APPIDX = DAP102_APP_R1
    Archive = DAP102_ARC_R1
    Catalog = DAP102_CAT_R1
    Restart = DAP102_RES_R1
< Trigger2Archive >
    Company = Company
    LOB = Lob
    PolicyNum = PolicyNum
    RunDate = RunDate
```

These table names are examples of the names you can use.

DB2 SERVER ON WINDOWS — WINDOWS CLIENT

For this scenario, assume you have a DB2 (version 6.1) Universal Database set up on a Windows 2000 server.



SETTING UP A DB2 DATABASE ON THE SERVER

Follow these steps to set up a DB2 Database on the server.

- 1 Go to Start, Programs, DB2 for Windows, Administration Tools, Control Center. The Control Center window appears. Expand Systems and you should see a server name such as ARCD6.

If so, go to step 3. If the server name is not listed, go to step 2.

- 2 Right click on Systems and choose Add. The Add System window appears. This is where you set up the system information DB2 uses to find the location of the database you are going to archive to.

Go to the Protocol field and select *Named Pipe*. The Protocol Parameters area changes, now displaying a Computer Name field. Click Refresh to retrieve information about the local system. The server name appears under the System Name field. If you click on that name the system places it in the System Name field. Fill in other pertinent information. The Comment field is optional. Click Apply when finished.

A confirmation message appears. Click Close. This should take you back to the Control Center window. The server name should now be listed under Systems. Go to step 3.

- 3 Expand the system name. You will now see Instances listed. Right click on Instances and choose Add. Click Refresh. This retrieves a list of instances on the server.

Choose DB2. Enter **DB2** in the Instance Name field. The Comment field is optional. Click Apply. A confirmation message appears. Click Close. This should take you back to the Control Center window. The DB2 instance should now be listed under Instances.
- 4 Expand DB2. You will see Databases listed, right click on Databases and choose Create, New. The Create Database Smartguide window appears.
- 5 Enter the name of the new database (such as ARCDDB6) in the Database Name field and the Database Alias field. The Comment field is optional. Click Done. This takes you back to the Control Center window. The newly created database will be listed under Databases.

Setting Up a Client for DB2 VERSION 6.1

This topic discusses archiving to a DB2 version 6.1 database (Universal Database) on a Windows 2000 Server using an ODBC driver and the native DB2 driver.

Archiving to a remote DB2 database using an ODBC driver

Follow these steps to set up a DB2 remote database on Windows 2000 Server:

- 1 Go to Start, Programs, DB2 For Windows, Administration Tools, Control Center. The Control Center window appears. Right click on Systems, then choose Add.
- 2 An Add System window appears. This is where you set up the system information DB2 uses to find the location of the database (Windows 2000 Server). Click Refresh and the server name should appear in the box below the System Name field. Click the server name and the server information appears in the fields. Click Apply. A confirmation message appears. Click Close.
- 3 You are now back to the Control Center window again. Make sure the new system name appears when you expand Systems. If the new system name is listed under Systems then expand that out also. You should then find Instances listed under your system name. Right click on Instances and choose Add.
- 4 An Add Instance window will appear. Click Refresh. This will retrieve a list of instances on your local system. Choose DB2 if it is not already in the Remote Instance field. Click Apply. A confirmation message appears. Click Close.
- 5 Expand Instances and expand DB2. There will be Databases listed under the DB2 instance, right click and choose Add.
- 6 An Add Database window appears. Click Refresh to retrieve the names of databases currently set up on the server. Choose the correct database from the list, such as ARCDDB6. Enter the name of the database in the Alias field. The Comment field is optional. Click Apply. A confirmation message appears. Click Close. Expand Databases and make sure the new database appears.

Setting up an ODBC data source

Follow these steps to set up an ODBC data source using Windows 2000:

- 1 Go to Start, Settings, Control Panel, ODBC. You are now viewing User Data Sources.

- 2 Click Add to add an IBM DB2 ODBC driver. The Create New Data Source window appears.
- 3 Choose *IBM DB2 ODBC Driver*. Click Finish. The ODBC IBM DB2 Driver – Add window appears.
- 4 Click the down arrow in the Data Source Name field, choose the correct database name, such as ARCDDB6. The Description field is optional, but it should be there if you specified it when you created the database. Click Ok. The User Data Sources tab of the ODBC Data Source Administrator window appears. Make sure your new data source is there, along with its corresponding driver, then click Ok.

Setting up INI options for the ODBC driver

Follow these steps to set up the INI options specific to the ODBC driver:

- 1 Set up the DBHandler:ODBC control group as shown below.

```
< DBHandler:ODBC >
  CreateTable = Yes
  CreateIndex = No
  Debug = No
  Server = (such as ARCDDB6-the newly-created data source name.)
  BLOBSupportForDB2ODBC =
  UserID = (Windows user ID)
  Passwd = (Windows password)
```

Use the BLOBSupportForDB2ODBC option to tell the Archive/Retrieval programs the version of DB2 being accessed can support BLOB (Binary Large Object) data types. This INI option, along with specifying BLOB as the data type for the CARData field in the CARFILE.DFD file, tells the Archive/Retrieval programs to process the field as a BLOB. If you omit this option or set it to No, the Archive/Retrieval programs translate any CARFILE.DFD data type request of BLOB to LONG VARCHAR.

- 2 The DBTable:XXX control groups determine what tables are used by looking at the ArcRet control group. The ArcRet control group should look like the one shown here:

```
< ArcRet >
  AppIdxDfd = Deflib\AppIdx.dfd
  AppIdx = APPIDX
  CARFile = ARCHIVE
  CARPath =
  Catalog = CATALOG
  RestartTable = RESTART
  ExactMatch = No
  Key1 = Company
  Key2 = Lob
  KeyID = PolicyNum
```

- 3 For all the tables listed above, add these control groups:

```
< DBTable:APPIDX >
  DBHandler = ODBC
< DBTable:ARCHIVE >
  DBHandler = ODBC
< DBTable:CATALOG >
  DBHandler = ODBC
< DBTable:RESTART >
```

```
DBHandler = ODBC
```

- 4 The ODBC_FileConvert control group contains the table names of each table to be created. Here is an example, your table names may differ:

```
< ODBC_FileConvert >
  APPIDX = FSIV100_APPIDX
  Archive = FSIV100_ARCHIVE
  Catalog = FSIV100_CATALOG
  Restart = FSIV100_RESTART
```

- 5 Set the Archival control group as shown here:

```
< Archival >
  ArchiveMem = Yes
```

Archiving to a Remote DB2 Database Using the Native DB2 Driver

Follow these steps to archive to a remote DB2 database using DB2's native driver. These steps assume you are using Windows 2000.

Setting up a DB2 database

First set up a DB2 database:

- 1 Go to Start, Programs, DB2 For Windows, Administration Tools, Control Center. Once the Control Center appears, right click on Systems, then choose Add. An Add System window appears.
- 2 On the Add System window you set up system information DB2 uses to find the location of the database (Windows 2000 Server). Click Refresh and the server name should appear below the System Name field. Click the server name and the server information appears in the fields. Click Apply. A confirmation message appears. Click Close. You return to the Control Center window.
- 3 Make sure the new system name appears when you expand Systems. If the new system name is listed under Systems, expand that out also. You should find Instances listed under your system name. Right click on Instances and choose Add. An Add Instance window will appear.
- 4 Click Refresh to retrieve a list of instances on your local system. Choose DB2 if it is not already in the Remote Instance field. Click Apply. A confirmation message appears. Click Close.
- 5 Expand Instances and expand DB2. There will be Databases listed under the DB2 instance, right click and choose Add. An Add Database window appears.
- 6 Click Refresh to retrieve the names of databases are currently set up on the server. Choose the correct database from the list, such as ARCD6. Enter the name of the database in the Alias field. The Comment field is optional. Click Apply. A confirmation message appears. Click Close. Expand Databases to make sure the new database appears.

Setting up the INI options for the DB2 driver

Follow these steps to add the INI setting the native DB2 driver will use:

- 1 Set up the DBHandler:DB2 control group as shown below.

```
< DBHandler:DB2 >
  BindFile = c:\rel10\fp400\w32bin\db2lib.bnd
  CreateTable = Yes
```

```
CreateIndex = No
Database = (such as ARCD6, a remote database name)
UserID = (Windows 2000 user ID)
Passwd = (Windows 2000 password)
```

- 2** The DBTable:XXX control groups determine what tables are used by looking at the ArcRet control group, which should look like the following.

```
< ArcRet >
  AppIdxDfd = Deflib\AppIdx.dfd
  AppIdx = APPIDX
  CARFile = ARCHIVE
  CARPath =
  Catalog = CATALOG
  RestartTable = RESTART
```

- 3** For all the tables listed above, add the following control groups:

```
< DBTable:RESTART >
  DBHandler = DB2
< DBTable:CATALOG >
  DBHandler = DB2
< DBTable:APPIDX >
  DBHandler = DB2
< DBTable:ARCHIVE >
  DBHandler = DB2
```

- 4** Make sure the DB2_FileConvert control group contains the table names of each table to be created. Here is an example, your table names may differ:

```
< DB2_FileConvert >
  APPIDX = DAP102_APP_R1
  Archive = DAP102_ARC_R1
  Catalog = DAP102_CAT_R1
  Restart = DAP102_RES_R1
```

- 5** Set the Archival control group as shown here:

```
< Archival >
  ArchiveMem = Yes
```

DB2 SERVER AND CLIENT ON WINDOWS

This topic discusses archiving to a local DB2 version 6.1 database using an ODBC driver and the native DB2 driver.

SETTING UP A DB2 DATABASE

This scenario shows how to archive to a DB2 database using an ODBC driver on Windows 2000.

- 1 Go to Start, Programs, DB2 For Windows, Administration Tools, Control Center. Once the Control Center appears, right click on Systems, then choose Add. An Add System window appears.
- 2 On the Add System window you set up system information DB2 uses to find the location of the database (local Windows 2000). Go to the Protocol field and click the down arrow, select *Named Pipe*. The Protocol Parameters area changes, displaying the Computer Name field. Type in the computer's network name here and click Retrieve.

The program retrieves information about the local system. Once that information is retrieved you will see names in the System Name and Remote Instance fields. Click Apply. A confirmation message appears. Click Close. You return to the Control Center window.
- 3 Make sure the new system name appears when you expand Systems. If the new system name is listed under Systems, expand that also. You should then find Instances listed under your system name. Right click Instances and choose Add.
- 4 An Add Instance window will appear. Click Refresh. You will see a list of instances on your local system. Choose DB2 and click Apply. A confirmation message appears. Click Close.
- 5 Expand Instances and expand DB2. You will see Databases listed, right click and choose Add. The Add Database window appears.
- 6 Enter the name of the new database, such as ARCDDBL, in the Database Name field and the Alias field. The Comment Field is optional. Click Apply. A confirmation message appears. Click Close. Expand Databases and make sure that the new database appears.

Setting up an ODBC data source

This scenario uses Windows 2000.

- 1 Choose Start, Settings, Control Panel, ODBC. You are now viewing User Data Sources. Click Add to add an IBM DB2 DBC driver. The Create New Data Source window appears
- 2 Choose *IBM DB2 ODBC Driver*. Click Finish. The ODBC IBM DB2 Driver - Add window appears.
- 3 Click the down arrow in the Data Source Name field and choose the correct database name. The Description field is optional, but should appear if you specified it when you created the database. Click Ok. The User Data Sources tab of the ODBC Data Source Administrator window appears. Make sure that your newly created data source is there and its corresponding driver is correct then click Ok.

Setting up INI options for ODBC

Follow these steps to set up the INI options specific to ODBC:

1 Set up the DBHandler:ODBC control group as shown below.

```
< DBHandler:ODBC >
    CreateTable = Yes
    CreateIndex = No
    Debug = No
    Server = (such as ARCDL - The data source name)
    BLOBSupportForDB2ODBC =
    UserID = (Windows user ID)
    Passwd = (Windows password)
```

Use the BLOBSupportForDB2ODBC option to tell the Archive/Retrieval programs the version of DB2 being accessed can support BLOB (Binary Large Object) data types. This INI option, along with specifying BLOB as the data type for the CARData field in the CARFILE.DFD file, tells the Archive/Retrieval programs to process the field as a BLOB. If you omit this option or set it to No, the Archive/Retrieval programs translate any CARFILE.DFD data type request of BLOB to LONG VARCHAR.

2 Use the DBTable:XXX control groups to determine what tables are used by looking at the ArcRet control group. Here is an example:

```
< ArcRet >
    AppIdxDfd = Deflib\AppIdx.dfd
    AppIdx = APPIDX
    CARFile = ARCHIVE
    CarPath =
    Catalog = CATALOG
    RestartTable = RESTART
```

3 For all the tables listed above, add these control groups:

```
< DBTable:APPIDX >
    DBHandler = ODBC
< DBTable:ARCHIVE >
    DBHandler = ODBC
< DBTable:CATALOG >
    DBHandler = ODBC
< DBTable:RESTART >
    DBHandler = ODBC
```

4 Use the ODBC_FileConvert control group to list the table names of each table to be created. Here is an example, your table names may differ:

```
< ODBC_FileConvert >
    APPIDX = FSIV100_APPIDX
    Archive = FSIV100_ARCHIVE
    Catalog = FSIV100_CATALOG
    Restart = FSIV100_RESTART
```

5 Set the Archival control group as shown here.

```
< Archival >
    ArchiveMem = Yes
```

Archiving to a Local DB2 Database Using the Native DB2 Driver

Setting up the DB2 database

This scenario uses Windows 2000.

- 1 Select Start, Programs, DB2 For Windows, Administration Tools, Control Center. Once the Control Center window appears, right click on Systems, then choose Add. The Add System window appears.
- 2 On the Add System window you set up system information DB2 uses to find the location of the database (local Windows 2000). Go to the Protocol field and click the down arrow, select *Named Pipe*. The Protocol Parameters area then displays a Computer Name field. Enter the computer's network name and click Retrieve.

The program retrieves information about the local system. Once that information appears, you see names in the System Name and Remote Instance fields. Click Apply. A confirmation message appears. Click Close. The Control Center window appears.

- 3 Make sure the new system name appears when you expand Systems. If the new system name is listed under Systems, expand that also. You should then find Instances listed under your system name. Right click Instances and choose Add. The Add Instance window appears.
- 4 Click Refresh to retrieve a list of instances on your local system. Choose DB2 and click Apply. A confirmation message appears. Click Close.
- 5 Expand Instances and expand DB2. You will see Databases listed, right click and choose Create, New. The Create Database Smartguide window appears.
- 6 Enter the name of the new database (ARCDDBL) in the New Database Name field and the Database Alias field. The Comment Field is optional. Click Done.

This should take you back to the Control Center window. Expand Databases if it is not already. Your new database should be listed.

Setting up the INI options for the DB2 driver

Be sure to set up the following INI options for the native DB2 driver.

- 1 Set up the DBHandler:ODBC control group as shown below.

```
< DBHandler:DB2 >
  BindFile = d:\rel10\fp400\w32bin\db2lib.bnd
  CreateTable = Yes
  CreateIndex = No
  Debug = No
  Database = (such as ARCDDBL - Local database name)
  UserID = (Windows user ID)
  Passwd = (Windows password)
```

- 2 Use the DBTable:XXX control groups to determine what tables are used by looking at the ArcRet control group, which should look like the following.

```
< ArcRet >
  AppIdxDFD = Deflib\AppIdx.dfd
  AppIdx = APPIDX
  CARFile = ARCHIVE
  CARPath =
  Catalog = CATALOG
  RestartTable = RESTART
```

3 For all the tables listed above, add the following control groups:

```
< DBTable:CATALOG >  
    DBHandler = DB2  
< DBTable:APPIDX >  
    DBHandler = DB2  
< DBTable:ARCHIVE >  
    DBHandler = DB2  
< DBTable:RESTART >  
    DBHandler = DB2
```

4 The DB2_FileConvert control group contains the table names of each table to be created. Here is an example, your table names may differ:

```
< DB2_FileConvert >  
    APPIDX = DAP102_APP_R1  
    Archive = DAP102_ARC_R1  
    Catalog = DAP102_CAT_R1  
    Restart = DAP102_RES_R1
```

5 Set the Archival control group as shown here:

```
< Archival >  
    ArchiveMem = Yes
```


SQL SERVER ON
WINDOWS —
ODBC CLIENT ON
WINDOWS

This scenario sets up a database in SQL Server using Microsoft SQL Server version 7.0.

- 1 Go to Start, Programs, Microsoft SQL Server 7.0 SQL Enterprise Manager. The Server Manager window appears, SQL 7.0 should already be expanded and there will be server names that appear below, choose the correct server and expand it.
- 2 Highlight the Databases folder, right click and choose New Database. Type in the database name, such as ARCD7, and select a data device. There is a size specified to the right of this field and the device should have a size greater than zero. Click the Create Now button.
- 3 If no login has been defined, highlight the Logins folder under the server and right click. Choose New Login. Type in a login name and a password. Click the Permit field next to the database you would like the login to default to. Then click Add. Confirm your password and click Ok.

SETTING UP A CLIENT

Follow these instructions to set up a Windows client and an ODBC data source using Windows 2000.

- 1 Select Start, Settings, Control Panel, ODBC. The User Data Sources window appears. Click Add to add a new SQL Server data source. The Create New Data Source window appears.
- 2 Choose *SQL Server*. Click Finish. The ODBC SQL Server Setup window appears. Enter the following information:

In this field	Enter
Data Source Name	This is your database name.)
Description	(optional)
Server	(This will drop down and the server should be listed.)

- 3 Click Options and enter the database name, such as ARCD7, you will be archiving to in the Database Name field. Click Ok. The Data Sources window appears.
- 4 Make sure the new data source name appears with the correct driver specified. If all is correct, click Ok.

Setting up the INI options
for ODBC

- 1 Set up the DBHandler:ODBC control group as shown below.

```
< DBHandler:ODBC >
  CreateTable = Yes
  CreateIndex = No
  Debug = No
  Server = (such as ARCD7 - This is the data source name)
  UserID = (SQL Server user ID)
  Passwd = (SQL Server password)
```

The user ID and password must be set up in SQL Server. For more information see [SQL Server on Windows — ODBC Client on Windows on page 455](#).

- 2** In the DBTable:XXX control groups, determine what tables are used by looking at the ArcRet control group, which should look like the one shown here:

```
< ArcRet >
  AppIdxDfd = Deflib\AppIdx.dfd
  AppIdx = APPIDX
  CARFile = ARCHIVE
  CARPath =
  Catalog = CATALOG
  RestartTable = RESTART
```

For all the tables listed above, add the following control groups:

```
< DBTable:APPIDX >
  DBHandler = ODBC
< DBTable:ARCHIVE >
  DBHandler = ODBC
< DBTable:CATALOG >
  DBHandler = ODBC
< DBTable:RESTART >
  DBHandler = ODBC
```

- 3** Add these INI options for DFD files for these tables:

```
< ArcRet >
  CARFileDFD = carfile.dfd
  RestartDFD = restart.dfd
```

DFD files can specify the full file name, otherwise they are located in the directory specified in the DefLib option:

```
< MasterResource >
  DefLib = subdirectory
```

- 4** The ODBC_FileConvert control group contains the table names of each table to be created. Here is an example, your table names may differ:

```
< ODBC_FileConvert >
  APPIDX = FSIV100_APPIDX
  Archive = FSIV100_ARCHIVE
  Catalog = FSIV100_CATALOG
  Restart = FSIV100_RESTART
```

These table names are examples of the names you can use.

- 5** Set the Archival control group as shown here:

```
< Archival >
  ArchiveMem = Yes
```

IDS ON WINDOWS — DB2 ARCHIVE ON z/OS

This scenario features Docupresentment's Internet Document Server (IDS) running on a Windows 32-bit computer and communicating with a DB2 archive residing on a z/OS machine.

To illustrate this scenario, you should download the setup executable to install Docupresentment 10.2 (IDS version 1.8). You can do this from the Support web site:

<http://www.oracle.com/skywiresoftware/index.html>

Follow these steps:

- 1 From the Support site, register, log in, and then click on product installations.
- 2 Select the current version for Windows 32-bit operating systems.

Refer to these documents for installation and configuration information:

- [Internet Document Server Guide](#)
- [SDK Reference](#)

SETTING UP THE DB2 ARCHIVE ON z/OS

Refer to these documents for information on configuring a DB2 archive on z/OS:

- [Documaker Server Installation Guide](#)
- [Documaker Server System Reference](#)

CREATING A z/OS DATABASE

To create a database, you *must* be an administrator on the machine you are creating the database on. Follow these instructions:

- 1 Click Add Database once you enter into the Client Configuration Assistant. On Tab 1 (Source), choose the Manually Configure a Connection to a DB2 Database option. Click Next.
- 2 On Tab 2 (Protocol), choose TCP/IP as the protocol and z/OS as the target operating system. Click Next.
- 3 On Tab 3 (TCP/IP), enter **os390** in the Hostname field. The Port number defaults to 446. Enter **db2ins 1** in the Service Name field. Click Next.
- 4 On Tab 4 (Target Database), enter the database name, such as USDCIOS39DSN1, in the Location Name field. Click Next.
- 5 On Tab 5 (Alias), enter **ARCDB** (or your database name on the mainframe) in the DBAlias field. The DBAlias field gets the first part of the location name from the previous tab. The Description field is optional. Click Next.
- 6 On Tab 6 (ODBC), check the Register this Database for ODBC field. Then select the data source. Click Done.
- 7 The system asks if you want to test your connection, click the Test Connection button. Then enter your user ID and password and click Ok. A window should appear with a message similar to this one:

```
The Connection test was successful.
Database product= DB2 OS/390 7.0
SQL authorization ID= akb
Database alias = ARCDB
To close this window and proceed, click OK.
```

Click Add to add another database or click Close to exit the Client Configuration Assistant.

Updating TCP/IP Values on a Windows 2000 Server

Follow these steps to update TCP/IP-related values on a Windows 2000 server:

- 1 So that the host name you entered can be found, add this entry in the host file

```
(c:\winnt \system32 \drivers \etc \hosts):
10.8.10.210 WIN2000A_1
```

The value on the left is the IP address of the Windows 2000 Server. The value on the right is the host name for that Windows 2000 Server.

- 2 Add these lines in the services file (c:\winnt\system32\drivers\etc\services):

```
db2inst1      446/tcp      #db2 port
db2insti      447/tcp      #db2 port interrupt
```

- 3 Go to Start, Settings, Control Panel, System, and choose the Environment tab. Enter a system variable called **DB2COMM** and set its value to:

```
APPC,TCP/IP
```

This specifies the communication protocols DB2 will use —APPC to talk to the z/OS host and TCP/IP to talk to Windows 2000 clients.

Also add a system variable called *DB2CODEPAGE* and set its value to:

850

- 4 Reboot Windows 2000 for the system variable to take effect.

APPENDIX B

System Files

This appendix includes samples of the various files used by and created by the system. For each file you will find a definition, including information on the tools you can use to modify the files, and a sample of the files.

The sample files are based on the base system. If you or Oracle Insurance's staff have customized your system, your files may differ.

For information on file formats, consult the technical documentation, which is located on your distribution CD and on Oracle Insurance's web site.

This appendix discusses these topics:

- [Overview on page 462](#)
- [Types of Files on page 464](#)
- [Resource Files on page 467](#)
- [Files Created by the GenTrn Program on page 480](#)
- [Files Created by the GenData Program on page 481](#)
- [Files Created by the GenPrint Program on page 483](#)
- [Files Created by the GenWIP Program on page 484](#)
- [Files Used by the GenArc Program on page 485](#)

OVERVIEW

The files discussed in this appendix are arranged in the following order:

Types of files:

- BCH files
- DAT files
- DBF files
- DDT files
- DFD files
- Error files
- Initialization (INI) files
- JDT files
- Log files
- LOG files
- MDX files
- Transaction files

Resource files

- FSISYS.INI
- FSIUSER.INI
- FORM.DAT
- SETRECPTB.DAT
- DFD files
- DDT files
- JDT files
- Extract files

Files created by the GenTrn program as it gathers information:

- TRNFILE.DAT
- LOGFILE.DAT
- ERRFILE.DAT
- MSGFILE.DAT

Files created by the GenData program to make print-ready files:

- NAFILE.DAT
- POLFILE.DAT
- NEWTRN.DAT
- Batch files (*.bch)

- MANUAL.BCH
- Updated log and error files
- Spool files
- MSGFILE.DAT

Files used by the GenWIP program for processing incomplete transactions

- WIP.DBF
- WIP.MDX
- 00000001.DAT
- 00000001.POL

Files used by and created by the GenArc program for archiving information:

- APPIDX.DBF
- ARCHIVE.CAR
- APPIDX.MDX
- APPIDX.DFD

TYPES OF FILES

There are several types of files used in the system. These file types are defined below.

BCH files The GenData program creates files with the extension BCH, called batch files, which list the transactions to be included in each batch, as specified in your FSISYS.INI file settings. Batch files are used as trigger files by the GenPrint and GenWIP programs. Batch files indicate which transactions should be printed in a given batch job. The GenPrint program uses batch files to print completed forms. The GenData program also creates manual batch files which record incomplete transactions. These manual batch files are used by the GenWIP program.

CAR files The GenArc program creates compressed archive (CAR) files in which it stores NAFILES, POLFILES, and archived forms and data. An example of a generated CAR file is ARCHIVE.CAR. You can have multiple CAR files. The GenArc program also creates the APPIDX.DBF file which serves as an index to the archived information stored in the CAR file.

DAT files Data table (DAT) files define various information the system uses as it processes information. All DAT are text files which have the extension DAT. Some DAT files are comma-delimited text files. You can edit DAT files using a text editor.

In many cases, there are tools, such as Form Set Manager, which you can use to edit specific DAT files. For example, the FORM.DAT file tells the system how the various forms are organized in the form set. The SETRCPTB.DAT file contains information about the recipients of a form and the conditions which determine whether or not a form is included in a form set or sent to a recipient. You can edit these files using the Form Set Manager.

The NAFILE.DAT file contains the variable data generated by the GenData program. This file, along with the POLFILE.DAT file, tell the GenPrint program what to print. This file also tells the GenWIP and GenArc programs what to place into WIP and what to archive.

The GenWIP program also creates DAT files for each incomplete transaction it must process. These files are numbered sequentially and for each file there is a corresponding POL file which contains information about the forms to use.

DBF files Database files (DBF) are used in several places in the system. For each DBF file, there is a corresponding MDX file which serves as its index. Examples of DBF files are FDB.DBF, which is created by the Field Database Editor; ARCHIVE.DBF, which is created by the GenArc program; and WIP.DBF, which is created by the GenWIP program.

NOTE: The UNIQUE.DBF file contains the last number for WIP file that was created. Whenever a WIP file is created, a number is generated to uniquely identify it to make sure no WIP file is overwritten.

DDT files The data definition table (DDT) file tells the GenData program, what rules it should use as it processes the data. You can edit DDT files using a text editor or by using the Image Editor.

In the DDT file you store semi-colon-delimited information which defines the source and target fields, field length and offsets, rules to apply to the field, and optional parameters for the rules.

DFD files Data format definition (DFD) files define to the system the database file formats of the files generated by the system. Many common system files are stored in database format. For example, the transaction file, the new transaction, application index, and recipient batch files are all stored in database format. These database files can be in a variety of formats, including Xbase, DB/2, ODBC, and standard sequential files, such as flat text files. The record structure defined in the DFDs remains independent, regardless of the type of database being used—although there are occasionally exceptions for some database specific records.

The GenData program uses TRNDFDFL.DFD to read the TRNFILE which contains the actual transactions GenTrn creates.

Error files The GenTrn program produces an error file to note any transactions it could not process correctly. The other programs, such as GenData, GenPrint, GenWIP, and GenArc, update this file as they perform their processing activities. This file will help you discover and correct any processing errors you may encounter. Errors may be caused by incorrect or missing data. The system records the error information by transaction. You can view this file using a text editor.

The GenData program creates error batch files if it spots an error. In contrast to manual batch files, you cannot correct these errors using the GenWIP program. Instead, you must, for instance, correct the error in the extract file, change the flag to operator required so the transaction will be added to the manual batch file, or change the FAP file and then process the transaction again.

Extract files Extract files are typically text files which contain the data the system processes. Extract files are created by another program, typically a database program, in a format the system can read. The text file format provides a standard interface into the system. For example, your data may be stored in a DB/2 or VSAM database from which you extract the data you want the system to process.

You can customize the system to read almost any type of file layout. The GenTrn program first reads the extract file and, using that extract data and TRNDFDFL.DFD file, creates transaction files (TRN files) the GenData program can use as it applies the processing rules and creates batch files, the NAFILE.DAT, and the POLFILE.DAT file.

NOTE: For use on an z/OS platform, the extract file must be converted to EBCDIC format if the file contains international characters. See [Working with Fonts on page 171](#), for more information on international characters.

Docucreate includes a sample extract file, called EXTRFILE.DAT, which serves as an example of the type of file the base system can read. You can use this file to experiment with the base system and determine how you want to set up your system. Typically, a complete test library is provided with the system. You can use this library to test your installation.

You can use the OpSystem option to specify the origination platform of an extract file:

```
< RunMode >
    OpSystem =
```

If you enter **OS400**, the system loads an EBCDIC conversion table which handles binary number conversions for source extract files originating from an IBM AS/400 system.

FAP files	The information which defines each section (image) is stored in a FAP file. FAP files are text files with the extension FAP. You can edit FAP files using a text editor, but they are most commonly created and edited using Documaker Studio or Image Editor. The FAP file defines the section while the FORM.DAT file defines the sections which comprise a form and form set.
Initialization files	Initialization (INI) files are used by the system to set system parameters and to enable or disable system features. Some examples of system INI files are: FSISYS.INI and FSIUSER.INI. For example, the FSISYS.INI file contains information the GenTrn program uses to determine when a new record starts and other information about the extract files the GenTrn program processes. The FSIUSER.INI file contains information specific to each user, such as the location of files and so on.
JDT files	The job definition table (JDT) file is a text file which tells the system which rules to use as it processes a specific job. Rules defined in the JDT file are run before the system runs rules assigned to specific fields. An example of a JDT file is the AFGJOB.JDT file.
Log files	When you run GenTrn, the program creates log files which record, by transaction, each transaction the program processes. These files have a DAT extension. You can review these log files using any text editor.
LOG files	Graphics, such as scanned signatures or logos, are stored as LOG files in the system. You use Documaker Studio or Logo Manager to view, manage, and manipulate LOG files.
MDX files	<p>The various system programs create MDX files which serve as indexes to the database files (.DBF files). For example, the GenWIP program creates the WIP.DBF file and the corresponding WIP.MDX file to record the incomplete transactions which were not printed.</p> <p>The Field Database editor creates the FDB.MDX file to serve as an index to the FDB.DBF file which contains common variable field definitions.</p>
Transaction files	<p>The GenTrn program creates transaction or TRN files which contains a record for each transaction. The record format for the TRN file can vary to meet your needs. This format is defined in the TRNDFDFL.DFD file. The GenData program uses the TRNDFDFL.DFD file to read the information in the TRN file as it processes the information.</p> <p>Each record in a TRN file contains a series of offsets or pointers. These offsets define the location of the transaction data. For instance, the offsets in a TRN file tell the GenData program where the transaction begins in the extract file, where the data for the transaction is stored in the NAFILE.DAT file, and where the form set for the transaction is stored in the POLFILE.DAT file.</p>

RESOURCE FILES

Resource files are used by the various programs which comprise Documaker Server. These files provide information these programs use to know how to read extract files, how to create print-ready files, which rules to apply, which recipients receive copies of which forms, and so on.

The resource files include:

- FSISYS.INI
- FSIUSER.INI
- FAPCOMP.INI
- FORM.DAT
- SETRCPTB.DAT
- DFD files
- DDT files
- JDT files
- Extract files

FSISYS.INI file

The FSISYS.INI file is one of the initialization (INI) files used by the system to set parameters and to enable or disable features. For example, the FSISYS.INI file contains information the GenTrn program uses to determine when a new record starts and other information about the extract files the GenTrn program processes. You can see examples of this file in the RPEX1 sample resources.

FSIUSER.INI file

The FSIUSER.INI file is one of the initialization (INI) files used by the system to set system parameters. For example, the FSIUSER.INI file contains information specific to each user, such as the location of files and so on. If there are common settings in the FSISYS.INI and FSIUSER.INI files, the system looks at both, but uses the settings in the FSIUSER.INI file. You can see examples of this file in the RPEX1 sample resources.

FAPCOMP.INI

The FAPCOMP.INI is an initialization (INI) file used by the Docucreate tools to set parameters or turn on or off features. For example, this INI file contains the control groups which let you map font families so that if you import an RTF file, the fonts are changed automatically. You can see an example of this file in the FAP\DLL directory.

FORM.DAT file

The FORM.DAT file specifies the forms currently being used in the system. The various elements of the FORM.DAT file specify the print order of the forms, duplex or simplex options, recipient batch information, establish a link between a system form name and the sections associated to it, and descriptive information.

This file, also known as the Form Set Definition Table, contains information about the KEY fields, such as company, line of business, and policy number plus information about each section in the form, its recipients, and the form set itself.

The information is stored in semi-colon-delimited format and you can edit this file using the legacy Form Set Manager or a text editor. The information that comprises individual sections is stored in a FAP file.

The following table describes the syntax of each record of the FORM.DAT file.

NOTE: Some of these options may not apply or may be changed given the specifications for a custom implementation.

```
; <FLD1>; <FLD2>; <SYS NAME>; <DESC>; <FORM OPT>; <not used>; <IMG FILN1> |  
<IMG OPT1> [ <RECP1> ( <CPY1> ) , . . . , <RECPn> ( <CPYn> ) ] / . . . / <IMG  
FILNn> | <IMG OPTn>;
```

Record	Description
<FLD1>	Used to categorize the forms, such as company. (length 20)
<FLD2>	Used to categorize the forms, such as line of business. (length 20)
<SYSNAME>	The name of the form used by the system and in tables. (length 20)
<DESC>	Used to describe the form. (length 30)
<FORM OPT>	Optional. Used for form options. (length 5) B - Indicates forms printed on certain Metacode printers can be stapled. D - Indicates this form is a Dec. page. F - Indicates the form size is fixed and not selectable G - Indicates the form is legal size H - Indicates the form is hidden from view but data can still be embedded on the form for later use. I - Indicates the form is A4 J - Indicates the form is executive size K - Indicates the form is landscape M - Indicates these forms can be repeated. N - Indicates the form is not required and should not display initially. The user has to add this form using the Form Selection window. The system assumes the form is required (see R) by default. O - Indicates overflow. A duplicate form generates to accommodate information which would not fit on the original form. P - Indicates that the form is a pull form. R - Used for default forms. Forms with this option are displayed initially. S - Indicates that this form is a Sub. Dec. page. X - Indicates that this form is a Master Dec. page Z - Line print (- z - z - z - z-)
<not used>	Not currently assigned a value. Can be used in custom development.
	Section file name stored in the master resource library, such as CU54A

	<p>These section options indicate:</p> <p>A - the section has no variable fields and is a print only form.</p> <p>B - the section is duplex and is on the back page.</p> <p>C - the section is for data entry and should not be printed.</p> <p>D - the section is for data entry and should be printed.</p> <p>E - the section is for viewing only and not for data entry.</p> <p>F - the section is duplex and is on the front page.</p> <p>G - the section should be printed on letter size paper (the default)</p> <p>H - the section is not for data entry but should be printed.</p> <p>I - the section will print on standard European paper.</p> <p>J - the section will print on Executive paper (7.25"x10.5").</p> <p>888K - the section will print landscape.</p> <p>L, 2 - the section prints on paper tray 2 (lower)</p> <p>N - the section is an inline FAP file</p> <p>O - the section is copied onto additional pages</p> <p>P - the section is a template</p> <p>Q - the section is hidden and will not print.</p> <p>R - a rolling section which prints on both sides of the paper.</p> <p>S - the section stays on the same page and doesn't flow onto two forms.</p> <p>T - short binding. The section prints on both sides of the paper and duplexes in flip chart fashion.</p> <p>U, 1- the section prints on paper tray 1 (upper).</p> <p>V - the section is a pre-compiled resource that is resident on the printer.</p> <p>W - the section can grow. Size is not fixed.</p> <p>X - the section is a header which appears at the top of the page.</p> <p>Y - the section is a footer which appears at the bottom of the page.</p> <p>Z - this is a flash section and is not used in pagination calculations</p> <p>o (zero) - a variable text merge is created</p> <p>3 - paper tray 3</p> <p>4 - paper tray 4</p> <p>5 - paper tray 5</p> <p>6 - paper tray 6</p> <p>7 - paper tray 7</p> <p>8 - paper tray 8</p> <p>9 - paper tray 9</p>
<RECP>	Contains the name of all possible recipients in which this form can be included. Example: Insured, Home Office, Agent, and so on.
<CPY>	Contains the default number of copies printed for a given recipient.

Here is an excerpt from the FORM.DAT file included with the base application in the RPEX1 sample resources. This excerpt shows the first three forms in the SAMPCO form set, DEC PAGE, LETTER, and LETTER2:

```
;SAMPCO;LB1;DEC PAGE;X;R;;sname|D<INSURED(1),COMPANY(1),AGENT(1)>/  
cmdec1|DS<INSURED(1),COMPANY(1),AGENT(1)>/cmdec2|DS<INSURED(1),  
COMPANY(1),AGENT(1)>/cmdec3|DS<INSURED(1),COMPANY(1),AGENT(1)>;  
  
;SAMPCO;LB1;LETTER;;RD;;sname|D<INSURED(1),COMPANY(1),AGENT(1)>/  
fmlt2a|DS<INSURED(1),COMPANY(1),AGENT(1)>/fmlt2b|DS<INSURED(1),  
COMPANY(1),AGENT(1)>/sal1|DS<INSURED(1),COMPANY(1),AGENT(1)>;  
  
;SAMPCO;LB1;LETTER2;Second Letter;RD;;sname|D<INSURED(1),  
COMPANY(1),AGENT(1)>/fmlt2a|DS<INSURED(1),COMPANY(1),AGENT(1)>/  
b3002|DS<INSURED(1),COMPANY(1),AGENT(1)>/  
ba3006|DS<INSURED(1),COMPANY(1),AGENT(1)>/  
ba3020|DS<INSURED(1),COMPANY(1),AGENT(1)>/  
sal1|DS<INSURED(1),COMPANY(1),AGENT(1)>;
```

You can see other examples of this file in the RPEX1 sample resources.

SETRCPTB.DAT file

The SETRCPTB.DAT file is used with the FORM.DAT file to build form sets and specify recipients given specific transaction types and other dependent conditions. It is also used to describe overflow conditions.

This file, also known as the Form Set Trigger table, contains information which tells the GenData program the recipients of a form set and tells the program which recipients receive which forms or sections.

You can define conditions using the Form Set Manager or by editing the SETRCPTB.DAT file in a text editor.

The following table describes each record in a SETRCPTB.DAT file. You can see examples of this file in the sample resources.

NOTE: Some of these options may not apply or may be changed depending on how your system was implemented.

This table explains the syntax of this file:

```
;COMPANY;LOB;FORM NAME;IMAGE NAME;TRANS CODE;  
RECP LIST; SEARCH MASK;OCCURRENCE (overflow) FLAG; RECS/FIRST IMAGE;  
RECS/OVERFLOW IMAGE;RECIP. COPY COUNT; CONDITION; (CRLF)
```

Field	Purpose
COMPANY	Company name as defined in the form set definition file (FORM.DAT) and the transaction record (TRNDFDFL.DFD)
LOB	Line of business as defined in the FORM.DAT and TRNDFDFL.DFD
FORM NAME	Form name as defined in the FORM.DAT file
IMAGE NAME	Section (image) name as defined in the FORM.DAT file. Section name is included only when you want to set conditions on a particular section in a form.

Field	Purpose
GroupName1	Matches the GroupName1 field in the FORM.DAT file. In an insurance industry application, this would typically contain the <i>company</i> code. <Key1Table> in the FSISYS.INI file.
GroupName2	Matches the GroupName2 field in the FORM.DAT file. In an insurance industry application, this would typically contain the <i>line of business</i> code. <Key2Table> in the FSISYS.INI file.
Form name	The name of the form, as specified in the FORM.DAT file. Note: Form names are descriptive, and do not correlate to any physical file name.
Image name	The name of a section (image) within a form, as specified in the FORM.DAT file. This name also correlates to a physical section file (FAP file) and, in legacy implementations, to a Data Definition Table file (DDT file). Note: A section level trigger record requires an entry in this key field; a form level trigger record must omit any value in this field.
Transaction codes	By including one or more transaction codes in this field, a form is triggered only if the extract file record includes that transaction code. If no transaction code value is mapped from the extract data for a transaction, the system considers all triggers eligible, regardless of whether they specify a transaction code list. Conversely, if a transaction code value is mapped from the data, the system only considers those triggers that have the same value to be eligible for evaluation.
Recipient list	Allows the optional specification of certain recipients.
Search mask 1 (Counter)	Defines the criteria to determine when a form belongs in a form set (or a section within a form). The criteria lets Documaker Server get specific data from the extract file. One form (or section) is added for every occurrence of the search mask per transaction when the overflow flag is set.
Occurrence (overflow) Flag	Indicates the need to calculate overflow conditions. Valid entries are: 0=No overflow and 1=Overflow Also used for Master and Subordinate form and section level flags. Valid entries: M=master (used on form level triggers) and S=subordinate (used on section level triggers) F=tells the system to override any previous copy count settings and use the copy count settings in this trigger file (used on form level triggers) You can choose these options for the occurrence Flag field from a drop-down pick list on the Transaction window.
Records per overflow image	(Used by overflow) Specifies the number of records matching the Counter Search Mask that will fit on the specified overflow form.
Records per first image	(Used by overflow) Specifies the number of records matching the Counter Search Mask that will fit on a specific form before overflowing to a new form.

Field	Purpose
Recipient copy count	Specifies the number of copies a recipient receives.
Search Mask 2 (True/False)	Similar to Search Mask 1, but only one form will be triggered, regardless of how many occurrences of the condition exists.
Custom Rule	Available field for use with custom rules or search masks. Most common custom rule is Reciplf.
Custom Rule Parameters	Specifies parameters for the selected custom rule.

Here is an excerpt from the SETRCPTB.DAT file included with the base application in the /mstrres/rpex1/deflib/ directory. This excerpt shows the recipients for the first three forms in the SAMPCO form set, DEC PAGE, LETTER, and LETTER2:

```
;SAMPCO;LB1;DEC PAGE;;T1;INSURED,COMPANY,AGENT;11,HEADERREC,96,  
~O;0;1;0;1;;  
;SAMPCO;LB1;LETTER;;T1;AGENT,COMPANY,INSURED;11,FRMLSTREC,25,1;0;1;  
0;1;;  
;SAMPCO;LB1;LETTER2;;T1;INSURED,COMPANY,AGENT;11,FRMLSTREC,27,1;0;1;  
0;1;;
```

DFD files

There are several database files, meaning that these files are written and read via calls to Oracle Insurance's DBLIB database software library. These database files can be in several formats, including Xbase (dBase), DB/2, and flat text. Not all database files require a corresponding DFD file because their record structure is coded in the software modules that access them. For instance, here is a list of Oracle Insurance's database files:

- transaction files
- new transaction files
- recipient batch files
- manual batch files
- application index files
- WIP files
- help files
- table files

Only these files require an external DFD file:

- transaction files
- new transaction files
- recipient batch files
- manual batch files
- application index files

The WIP file may optionally have an external DFD. If there is no external WIP DFD file, the internal record structure as coded in the program is used. The help and table files do not support the use of external DFD files.

Of the database files that require external DFD files, only three actual DFD files are needed:

- a transaction file DFD (TRNDFDFL.DFD)
- a recipient batch file DFD (RCBDFDFL.DFD)
- an application index file DFD (APPIDX.DFD)

The transaction file DFD is used by both the transaction file and the new transaction file. The recipient batch file DFD is used by both the recipient batch files and the manual batch files. The application index file DFD is used by the application index file. You can see examples of all these files in the RPEX1 sample resources.

TRNDFDFL.DFD file

The TRNDFDFL.DFD file tells the GenData program how to read the TRN file. If necessary, you can edit this text file in a text editor. The TRNDFDFL, is used by the GenTrn, GenData, GenArc, and GenWIP programs.

The GenTrn program writes out the transaction file using the TRNDFDFL. The GenData program reads the transaction file and writes out the new transaction file using the TRNDFDFL file. And the GenArc and GenWIP programs read the new transaction file using the TRNDFDFL file.

You can define the name of the TRNDFDFL file in the Data control group of the FSISYS.INI file, as shown below:

```
< Data >
    TrnDfdFile = trndfdfl.dfd
```

RCBDFDFL.DFD file

The RCBDFDFL.DFD file, or recipient batch file DFD, is used by the GenData, GenPrint, and GenWIP programs. If necessary, you can edit this text file in a text editor.

The GenData program writes the recipient and manual batch files using the RCBDFDFL.DFD file. The GenPrint program reads the recipient batch files using the RCBDFDFL.DFD file. The GenWIP program reads the manual batch files using the RCBDFDFL.DFD file.

You can set the name of the RCBDFDFL.DFD file in the Data control group of the FSISYS.INI file, as shown below:

```
< Data >
    RcbDfdFile = rcbdfdf1.dfd
```

APPIDX.DFD

The APPIDX.DFD file, or application index file, is used by the GenArc program and the Archive module of Documaker Workstation. The GenArc program writes out the application index file using the APPIDX.DFD. While Documaker Workstation's Entry module reads the application index file using APPIDX.DFD. If necessary, you can edit this text file in a text editor.

You can set the name of the APPIDX.DFD file in the ArcRet control group in the FSUSER.INI file, as shown below:

```
< ArcRet >
    AppIdxDfd = appidx.dfd
```

However, the APPIDX.DFD name does not have to be set as shown above, provided the system is running in a Windows environment. If the APPIDX.DFD name is not specified as shown, the system automatically appends a DFD extension to the APPIDX name specified in the same group, as shown below:

```
< ArcRet >  
AppIdx = AppIdx
```

This will not work in an environment that does not support file name extensions, such as z/OS.

.DDT files

The Data Definition Table (DDT) is used to map data from a source record to fields in a form. The DDT file tells the GenData program what rules it should use as it processes the data. You can edit DDT files using a text editor or by using the Image Editor.

In the DDT file you store semi-colon-delimited information which defines the source and target fields, field length and offset, rules to apply to the field, and optional parameters for the rules.

You can see examples of DDT files in the RPEX1 sample resources. The following table explains the structure of this file:

```
;A;B;sfld;sofst;slen;dflld;dfinx;dlen;fm;frule;data;f1;f2;f3;f4;x;y;  
fontID
```

Element	Size	Description
A		index into File Definition Table (FDT)
B		source record Index
sfld	17	source record field name
sofst		offset of field in source record
slen		length of field in source record
dflld	17	destination field name
dfinx		destination field index
dlen		length of destination field
fm	17	format mask
frule		field level rule
data	1024	data field used by a field level rule
f1	2	not required flag
f2	2	host required flag
f3	2	operator required flag
f4	2	optional required flag

x		x coordinate for the location of the field, in FAP units (2400 per inch)
y		y coordinate for the location of the field, in FAP units (2400 per inch)
fontID		ID of the font

.JDT files

The job definition table (JDT) file is a text file which tells the system which rules to use as it processes a specific job. Rules defined in the JDT file are run before the system runs rules assigned to specific fields.

An example of a JDT file is the AFGJOB.JDT file, which you can see in the RPEX1 sample resources. You can also see examples of JDT files, including the performance JDT file used with single-step processing in the topic, [Single-step Processing Example on page 55](#).

Extract files

Extract files are typically text files which contain the data the system processes. Extract files are created by another program, typically a database program, in a format the system can read. The text file format provides a standard interface into the system. For example, your data may be stored in a DB/2 or VSAM database from which you extract the data you want to process in the system in text format.

You can customize the system to read almost any type of file layout. The GenTrn program first reads the extract file and, using that extract data and TRNDFDL.DFD file, creates transaction files (TRN files) the GenData program can use as it applies the processing rules and creates batch files, the NAFILE.DAT, and the POLFILE.DAT file.

NOTE: For use on a z/OS platform, the extract file must be converted to EBCDIC format if the file contains international characters. See [Working with Fonts on page 171](#), for more information on international characters.

Extract data can be in the form of a flat file, a VSAM file, or it can come directly from a database. The important thing is that the data is organized and presented in a manner that makes it efficient to process. While the system is very flexible, there are things you can do to minimize the need for customizations and to maximize the speed at which the system identifies and processes the data.

Here are some general guidelines to follow when you design an extract file:

- The basic entity of the data is the transaction. Data for transactions is stored in multiple rows.
- To speed the identification of a transaction entity, make the first record for each transaction a general information row.

- Each record should have a standard key structure. Here is an example of a minimum key structure:

Include this key	Which is
Transaction Identifier	unique to each transaction
Record Type Identifier	for each record type
Record Counter	a sequence number

Sequence numbers are not required. In some cases they are nice to have to keep track of which occurrence has been passed. It is, however, not a requirement that you sequence repeating records.

- To make testing easier, use a flat ASCII or EBCDIC extract file. By eliminating packed data fields, you can more easily view the contents of an extract file using standard text editors.
- Speed processing by keeping the extract file as small as possible—minimize the occurrence of repeated information in subsequent records.
- When possible, structure the data in the extract file so the system can read it in the order it should be processed. The less the system has to search for data, the faster it will process the data.
- Keep all related information in one record if possible, to minimize complexity of rules. For example, the layout should look something like this:

Record Name	Layout
GENERALINFO	account number, type transaction
ADDRESSINFO	client name, address, phone

- When information occurs multiple times (occurs clauses) in records, structure the extract file to contain one record for each occurrence. For example, when multiple forms are present on a policy or multiple meters are present on a bill, structure the information into individual records per entity (form, meter, and so on). This increases the likelihood that you can use base system overflow and mapping features to process the data.

NOTE: For overflow, the system first determines the maximum number of lines it can print on a page. When this number is exceeded, the system automatically inserts overflow pages as necessary. If overflow is dependent upon custom conditions to determine line counts, you will need custom code.

- Design records that will recur or overflow to have specific identifiers to sequence the records and to have key identifiers for overflow requirements within one record. This helps to minimize processing time and rule complexity. This is not a requirement, but may ease custom rule complexity with a point of reference.

- It is a good idea to have a header record which contains all global identifiers for a transaction, such as COMPANY, LINE OF BUSINESS, and TRANSACTION. You can then use this header record as the trigger to each transaction and as the basis for building the correct form set.
- When you build a header record, place all of the key fields for WIP, Archive, and the batch sorting fields in this record. This makes it easier for the system to perform searches and simplifies the building of the DFD records used to define the key architecture.
- Where possible, place all conditional data triggers for a form in one record. This may eliminate the need for the RECIPIF rule in the SETRCPTB.DAT file when triggering records. By reducing usage of this rule, you can improve system performance.

NOTE: You can find additional performance considerations for MVS systems in the Installation Guide.

- To maximize performance, provide sub totals and totals for groups of information in the extract data. This eliminates the need for system calculations via DAL scripts or custom rules and speeds performance.
- Provide any data in the extract file that would require the use of the TblLkUp, LookUp, SetState rules. This also improves performance and simplifies your master resource libraries.
- For Year 2000 compliance, make sure all date fields in the extract file are in 4-digit year format, preferably in YYYYMMDD format. (For the Archive application index file, APPIDX.DFD, the rundate field retrieved from the extract file must be in this format).

Docucreate includes a base extract file, called EXTRFILE.DAT, which serves as an example of the type of file the base system can read. You can use this file to experiment with the base system and determine how you want to set up your system.

You can see examples of this file in the sample resources.

DFD File Format

The DFD file contains two control groups. The Fields control group lists all the fields in the record structures and the order those fields appear in the storage media. The fields are automatically stored internally in the same order they appear externally. The second group describes each field. This description includes an external and internal definition of the field where applicable.

Fields Group

The Fields control group appears as follows:

```
< Fields >
  FIELDNAME =
  FIELDNAME =
  FIELDNAME =
```

...where FIELDNAME lists the name of the field. This is the name used by applications to reference data in a DFD record. The order of the FIELDNAME options dictates the order these fields are in, where applicable, on the storage media and how are they are stored in memory.

FIELDNAME has a maximum length of 26 characters, except when using Xbase. Using Xbase, the maximum length is 10 characters.

Field Description Group

The Field Description control group has the following format:

```
< xxxxxx >
EXT_TYPE=
EXT_LENGTH=
EXT_PRECISION=
INT_TYPE=
INT_LENGTH=
INT_PRECISION=
KEY=
REQUIRED=
```

...where xxxxxx is name of field as listed in the Fields control group.

EXT_TYPE	Data format of field on storage media	Possible formats are:
	NOT_PRESENT	not present in this record
	SIGNED_CHAR	a signed char
	CHAR	char
	CHAR_ARRAY	NULL terminated string
	CHAR_ARRAY_NO_NULL_TERM	character array not NULL terminated
	SHORT	16-bit signed integer
	UNSIGNED_SHORT	16-bit unsigned integer
	LONG	32-bit signed integer
	UNSIGNED_LONG	32-bit unsigned integer
	FLOAT	float single precision
	DOUBLE	double precision
	LONG_DOUBLE	long double precision
	DATESTAMP	a FSI date/time field
	TIMESTAMP	a FSI time stamp
	VARCHAR	variable length character array

The external record definition must match the actual records written to or read from the database. The internal record definition is provided for easier programming use.

EXT_LENGTH:	Length of field on storage media. Valid for data types CHAR_ARRAY and CHAR_ARRAY_NO_NULL_TERM only. Ignored for all other data types.
EXT_PRECISION:	Number of digits after decimal point. Valid for data types FLOAT, DOUBLE, and LONG_DOUBLE only. It is ignored for all other data types.
INT_TYPE:	Same as EXT_TYPE.
INT_LENGTH:	Same as EXT_LENGTH except one additional byte is added to length to store null termination byte.
INT_PRECISION:	Same as EXT_PRECISION.
KEY:	Indicates if this field is a key field. Y indicates it is a key field. All other values, or if field is not present, indicates field is not a key field. This field is only used for DB/2 and indicates that the field is required.
REQUIRED:	Indicates if this field is required in order for a record to be stored on or retrieved from a storage media. Y indicates it is required. All other values, or if field is not present, indicates field is not required. If KEY=Y, the field is required regardless of the value of this option.

The options can appear in any order. The system records any errors encountered while loading a field in the log file.

FILES CREATED BY THE GENTRN PROGRAM

The GenTrn program reads data in an extract file and creates transaction records which in turn are processed by the GenData program. The main file created by the GenTrn program is the TRN file which, along with the TRNDFDFL.DFD file, tells the GenData program which transactions to process.

The GenTrn program creates these files as it reads in the extract file and uses the resource files:

- Transaction files
- Error files
- Log files

Transaction files

The GenTrn program creates transaction or TRN files which contains a record for each transaction. The record format for the TRN file can vary to meet your needs. This format is defined in the TRNDFDFL.DFD file. The GenData program uses the TRNDFDFL.DFD file to read the information in the TRN file as it processes the information.

Each record in a TRN file contains a series of offsets or pointers. These offsets define the location of the transaction data. For instance, the offsets in a TRN file tell the GenData program where the transaction begins in the extract file, where the data for the transaction is stored in the NAFILE.DAT file, and where the form set for the transaction is stored in the POLFILE.DAT file.

Error files

The GenTrn program produces this file to note any transactions it could not process correctly. This file will help you discover and correct any processing errors you may encounter. The most common errors are caused by incorrect or missing data. The information is recorded by transaction. You can view this file using a text editor. You can see examples of this file in the RPEX1 sample resources.

Log files

When you run GenTrn, the program creates log files which record by transaction each transaction the program processes. You can review these log files using any text editor. You can see examples of this file in the sample resources.

FILES CREATED BY THE GENDATA PROGRAM

The GenData program takes information created by the GenTrn program and applies processing rules to those transactions and data. The GenData program creates batch files, the NAFILE.DAT, and the POLFILE.DAT file for the GenPrint program. It also creates a manual batch file (MANUAL.BCH) for the GenWIP program. The output from the GenData program is also used by the GenArc program to archive forms and data.

The GenData program creates the following files:

- NAFILE.DAT
- POLFILE.DAT
- NEWTRN.DAT
- Batch files (*.bch)
- MANUAL.BCH
- Updated error and log files

NAFILE.DAT file

The GenData program creates an NAFILE.DAT file, commonly referred to as the NA file, in which it stores section and variable field information. The GenPrint program uses this file, along with the POLFILE.DAT file, which is also produced by the GenData program to print the forms.

If the data is incomplete and GenData cannot complete the form, it creates a manual batch file. The GenWIP program then creates separate DAT and POL files for each incomplete transaction. These files provide the entry system with the information it needs to open the form so a data entry operator can add the missing data. This is a semi-colon-delimited text file. You can see examples of this file in the RPEX1 sample resources.

POLFILE.DAT file

The POLFILE.DAT file, commonly referred to as the POL file, defines the form set used for a specific transaction. The GenData program creates this file which is used by the GenPrint, GenWIP, and GenArc programs. For instance, if the data is complete, GenData creates an NA file and a POL file. These files are used by GenPrint, along with the batch files, to produce the print-ready file.

If the data is incomplete and GenData cannot prepare the form for printing, it creates a manual batch file. The GenWIP program then creates separate files for each transaction to provide the entry system with the information it needs to open the form so a data entry operator can add the missing data. This is a semi-colon-delimited text file. You can see examples of this file in the RPEX1 sample resources.

NOTE: You can use the MaxPolLineLength option to control the output line length when writing out POL file records. The default is 255. You can set it to shorter lengths when testing to more easily view the file in a text editor.

```
< Control >  
MaxPolLineLength = 80
```

Choose a length between 40 to 4000 bytes.

NEWTRN.DAT file	The GenData program creates the NEWTRN.DAT file. This file tells the GenArc program where to find data in the NAFILE.DAT file and which forms to use in the POLFILE.DAT file. You can see examples of this file in the RPEX1 sample resources.
Batch files	The GenData program creates files with the extension BCH, called batch files, list the transactions to be included in each batch, as specified in your FSISYS.INI file settings. Batch files are used as trigger files by the GenPrint and GenWIP programs. Batch files indicate which transactions should be printed in a given batch job. The GenPrint program uses batch files to print completed forms. The GenData program also creates manual batch files which record incomplete transactions. These manual batch files are used by the GenWIP program.
MANUAL.BCH file	The GenData program creates this file if it is unable to complete the processing of a form set. Typically, this occurs because the forms are missing information. This file is then used by the GenWIP program so a data entry operator can manually complete the form and resubmit it for processing.
Error batch	The GenData program creates error batch files if it spots an error. In contrast to manual batch files, you cannot correct these errors using the GenWIP program. Instead, you must, for instance, correct the error in the extract file, change the flag to operator required, or change the FAP file and then process the transaction again.
Updated log, error, and message files	As the GenData program processes information, it updates the log, error, and message files. You can review these files in a text editor to review when transactions were processed or to resolve errors.

FILES CREATED BY THE GENPRINT PROGRAM

The GenPrint program takes information produced by the GenData program and creates a printer spool file for use with PCL, AFP, Metacode, and PostScript printers. Specifically, the GenData program produces batch files, an NAFILE.DAT, and a POLFILE.DAT file which the GenPrint program uses to create printed forms

The GenPrint program creates the following files:

- Spool files
- Updated log and error files

Spool files

The spool files are print-ready files the GenPrint program creates from information received from the GenData program and from resource files.

Updated log and error files

As the GenPrint program processes information, it updates the log and error files. You can review these files in a text editor to review when transactions were processed or to resolve errors.

FILES CREATED BY THE GENWIP PROGRAM

The GenWIP program receives information about incomplete transactions from the GenData program. This information is stored in manual batch (MANUAL.BCH) files. The GenWIP program then creates separate files for each incomplete transaction. The data for these incomplete transactions is stored in a file with the extension DAT, such as 00000001.DAT. The corresponding form set information is stored in a file with the extension POL, such as 00000001.POL.

The GenWIP program also creates the WIP.DBF file, a database file which contains records of all the incomplete transactions extracted from the NAFILE.DAT file produced by the GenData program. The WIP.MDX file, also created by the GenWIP program serves as an index to the WIP.DBF file.

This gives the entry program the information it needs to display the form so you can fill in the missing information and complete the form in Documaker Workstation. Once completed, you can resubmit the form for processing by the GenData program.

The GenWIP program uses these files as it prepares incomplete transactions for further processing with the entry system.

- WIP.DBF
- WIP.MDX
- 00000001.DAT files
- 00000001.POL files
- UNIQUE.DBF

WIP.DBF file The WIP.DBF file contains information about the incomplete transactions which the GenWIP program extracted from the NAFILE.DAT and POLFILE.DAT file created by the GenData program. The WIP.MDX file serves as an index to this file.

WIP.MDX file This file serves as an index to the WIP.DBF file.

00000001.DAT file Using the MANUAL.BCH file produced by the GenData program. The GenWIP program creates from the NAFILE.DAT file, a separate data file for each incomplete transaction. These files are numbered and have the extension DAT. In essence, they are like the NAFILE.DAT except there is only one transaction per file.

00000001.POL file Using the MANUAL.BCH file produced by the GenData program, the GenWIP program creates from the POLFILE.DAT file, a separate POL file for each incomplete transaction. These files are numbered to correspond with their matching data file and contain information about the form set on which the system should place the data. In essence, they are like the POLFILE.DAT except there is only one form set per file.

UNIQUE.DBF file The UNIQUE.DBF file contains the last number for WIP file that was created. Whenever a WIP file is created, a number is generated to uniquely identify it to make sure no WIP file is overwritten. You should not modify, rename, or delete this file. The highest number it will generate for WIP files is FFFFFFFF, which is 4,294,967,295. After this number, the counter resets to 00000001.

The GenWIP program uses this information to create separate data and form information files for the incomplete transaction information it receives from the GenData program.

FILES USED BY THE GENARC PROGRAM

The GenArc program archives forms and data so you can store the information efficiently and retrieve it quickly. This program receives information stored in the APPIDX.DFD. Using this information, the GenArc program creates CAR files to store the information and forms and a DBF files which serves as an index to the data in the CAR files. The GenArc program can create multiple CAR files, as needed.

The GenArc program uses and creates these files as if archives information:

- APPIDX.DBF
- APPIDX.DFD
- ARCHIVE.CAR
- APPIDX.MDX

APPIDX.DBF file The APPIDX.DBF file is created by the GenArc program and contains records about the archive information stored in the ARCHIVE.CAR file.

APPIDX.DFD file The GenData program creates this file to tell the GenArc program how to store data and forms to be archived. The actual information is stored in ARCHIVE.CAR files.

ARCHIVE.CAR file The GenArc program creates CAR files in which it stores archived forms and data. An example of a generated CAR file is ARCHIVE.CAR. You can have multiple CAR files.

APPIDX.MDX file This file serves as an index to the APPIDX.DBF file.

APPIDX.DFD file The APPIDX.DFD file, or application index file, is used by the GenArc program and the Entry module. The GenArc program writes out the application index file using the APPIDX.DFD. While the entry module reads the application index file using the APPIDX.DFD file.

You can set the name of the APPIDX.DFD file in the ArcRet control group in the FSIUSER.INI file, as shown below:

```
< ArcRet >
AppIdxDfd = AppIdx.Dfd
```

However, the APPIDX.DFD name does not have to be set as shown above, provided the system is running in a Windows environment. If the APPIDX.DFD name is not specified as shown, the system automatically appends a DFD extension to the APPIDX name specified in the same group, as shown below:

```
< ArcRet >
AppIdx = AppIdx
```

This will not work in an environment that does not support file name extensions, such as z/OS systems.

Glossary

All components of the system use specific terminology. We suggest you familiarize yourself with these terms before you begin using the system. The following terms include definitions of system tools and files as well as commonly-used terms.

NOTE: The Data control group in the FSISYS.INI file lets you specify many of the file names you want to use. For instance, by modifying the settings in this group, you can change the name of the error file (ERRFILE.DAT) to any file name you want. In this manual, we refer to the default names for these files.

oooooooo1.DAT File

Using the MANUAL.BCH file produced by the GenData program, the GenWIP program creates from the NAFILE.DAT file, a separate data file for each incomplete transaction. These files are numbered and have the extension DAT. In essence, they are like the NAFILE.DAT except there is only one transaction per file.

See also ooooooooo1.POL and the [GenWIP Program on page 494](#).

oooooooo1.POL File

Using the MANUAL.BCH file produced by the GenData program, the GenWIP program creates from the POLFILE.DAT file, a separate POL file for each incomplete transaction. These files are numbered to correspond with their matching data file and contain information about the form set on which the system should place the data. In essence, they are like the POLFILE.DAT except there is only one form set per file.

See also ooooooooo1.DAT and the [GenWIP Program on page 494](#).

AFP

Advanced Function Printing (AFP), developed by IBM, is a print server language that generates data streams of objects. The data streams merge with print controls and system commands to generate Intelligent Printer Data Stream (IPDS). Your system then sends the IPDS to the AFP printer for printing. The GenPrint program can create spool files for AFP printers.

ARCHIVE.CAR File

See [.CAR Files on page 488](#).

ARCHIVE.DBF File

The ARCHIVE.DBF file is created by the GenArc program and contains records about the archive information stored in the ARCHIVE.CAR file.

ARCHIVE.DFD File

The GenData program creates this file to tell the GenArc program how to store data and forms to be archived. The actual information is stored in ARCHIVE.CAR files.

Base Product

Any executable modules, source code, resource libraries, and documentation and help delivered to you from the Support Services Group within the normal system release and update cycle are considered part of the base product. All executable modules, source code, resource libraries, and documentation and help which were changed or customized by your internal development team, a third-party development team, or the Professional Services staff, are custom solutions.

.BCH Files

The GenData program creates files with the extension BCH, called batch files, which list the transactions to be included in each batch. Batches are specified in your FSISYS.INI file settings. Batch files are used as trigger files by the GenPrint and GenWIP programs. Batch files indicate which transactions should be printed in a given batch job. The GenPrint program uses batch files to print completed forms. The GenData program also creates manual batch files which record incomplete transactions. These manual batch files are used by the GenWIP program. Error batch files contain transactions which cannot be processed by the system. Batch files are comma-delimited TEXT files.

See also [MANUAL.BCH File on page 496](#).

Batch Files

See [.BCH Files on page 488](#).

.CAR Files

The GenArc program creates CAR files in which it stores archived forms and data. An example of a generated CAR file is ARCHIVE.CAR. You can have multiple CAR files. The GenArc program also creates DBF files which serve as an index to the archived information stored in the CAR file.

Custom Solution

All executable modules, source code, resource libraries, and documentation and help which were changed or customized by your internal development team, a third-party development team, or Oracle Insurance's Professional Services staff, are considered to be a custom solution. Any executable modules, source code, resource libraries, and documentation and help delivered to you from the Support Services Group, within the normal system release cycle, are considered part of the base product.

DAL

Document Automation Language (DAL) is the language you use when you tell the system how to calculate variable fields. This calculation is also called a script. When you select calculation options for a variable field, you can choose one of the following:

DAL CALC. Recalculates the value of all fields each time a user tabs to a new field in the section.

DAL SCRIPT. Recalculates the value of the fields to which you assign the script only when a user tabs out of that field

NOTE: You can find detailed information about DAL in the DAL Reference.

.DAT Files

Data table (DAT) files define various information the system uses as it processes information. All DAT are text files which have the extension *DAT*. Some DAT files are comma-delimited text files. You can edit DAT files using a text editor.

In many cases, there are graphical tools, such as Form Set Manager, which you can use to edit specific DAT files. For example, the FORM.DAT file tells the system how the various forms are organized in the form set. The SETRCPTB.DAT file contains information about the recipients of a form and the conditions which determine whether or not a form is included in a form set or sent to a recipient. You can edit these files using the Form Set Manager.

The NAFI.DAT file contains the variable data generated by the GenData program. This file, along with the POLFILE.DAT file, tell the GenPrint program what to print. This file also tells the GenWIP and GenArc programs what to place into WIP and what to archive. These files can only be edited with a text editor.

The GenWIP program also creates DAT files for each incomplete transaction it must process. These files are numbered sequentially and for each file there is a corresponding POL file which contains information about the forms to use.

.DBF Files

Database files (DBF) are used in several places in the system. For each DBF file, there is a corresponding MDX file which serves as its index. Examples of DBF files are FDB.DBF, which is created by the Field Database Editor; ARCHIVE.DBF, which is created by the GenArc program; and WIP.DBF, which is created by the GenWIP program.

See also [Field Database Editor on page 492](#) and [External Database Editor on page 491](#).

DDT Files

The data definition table (DDT) file tells the GenData program what rules it should use as it processes the data. You can edit DDT files using a text editor or by using the Image Editor.

In the DDT file you store comma-delimited information which defines the source and target fields, field length and offset, rules to apply to the field, and optional parameters for the rules.

See also [.JDT Files on page 495](#).

DESKJET.FXR File

This font cross reference file provides information about internal HP fonts for HP Deskjet and compatible printers.

.DFD Files

Data field definition (DFD) files define to the system the file formats of the files generated by the system.

An example of a DFD file is the TRNDFDFL file which the GenTrn program creates. The GenData program uses this file to read the TRNFILE which contains the actual transactions GenTrn creates.

Distributed Resource Library

A *Distributed Resource Library* provides a decentralized repository into which you can place compiled items you select from your master resource library. A distributed resource library provides a unique and customized library of reusable resources for specific users at various locations in your organization. A distributed resource library contains a section (image) library, a variable data dictionary library, a rules library, and a system library.

See also [Master Resource Library on page 496](#).

Duplex

A form printed on both the front and back sides of a sheet of paper is printed in duplex mode.

See also [Simplex on page 498](#).

ERRFILE.DAT

The GenTrn program creates this file to note any transactions it could not process correctly. The other programs, such as GenData, GenPrint, GenWIP, and GenArc, update this file as they perform their processing activities. This file will help you discover and correct any processing errors you may encounter. Common errors are caused by incorrect or missing data. The system records error information by transaction. You can view this file using a text editor.

Error Batch

The GenData program creates error batch files if it spots an error. In contrast to manual batch files, you cannot correct these errors using the GenWIP program. Instead, you must, for instance, correct the error in the extract file, change the flag to operator required, or change the FAP file and then process the transaction again.

Error Files

See [ERRFILE.DAT on page 490](#).

External Database Editor

The External Database Editor provides you with an easy-to-use tool for creating and maintaining information about the extract file being used. The data in the file can be automatically merged onto a form's variable fields using the External Database Editor. The External Database Editor can import DDF, DFD, DBF, and COBOL copy book files. The tool defines customer data or transaction file data, which provides you with a greater ability to apply and modify data, and increase the ease of reusing resources.

See also [Field Database Editor on page 492](#).

Extract Files

Extract files are typically text files which contain the data the system processes. Extract files are created by another program, typically a database program, in a format the system can read. The text file format provides a standard interface into the system. For example, your data may be stored in a DB/2 or VSAM database from which you extract the data you want to process in the system in text format.

You can customize the system to read almost any type of file layout. The GenTrn program first reads the extract file and, using that extract data and TRNDFDFL.DFD file, creates transaction files (TRN files) the GenData program can use as it applies the processing rules and creates batch files, the NAFILE.DAT, and the POLFILE.DAT file.

NOTE: For use on an MVS platform, the extract file must be converted to EBCDIC format if the file contains international characters.

The system includes a base extract file, called EXTRFILE.DAT, which serves as an example of the type of file the base system can read. You can use this file to experiment with the base system and determine how you want to set up your system.

.FAP Files

The information which defines each section is stored in a FAP file. FAP files are text files with the extension *FAP*. You can edit FAP files using a text editor, but they are most commonly created and edited using Image Editor. The FAP file defines the section while the FORM.DAT file defines the sections which comprise a form and form set.

See also [Image Editor on page 495](#).

FDB.DBF File

The FDB.DBF file is a database file created by the Field Database Editor which contains a record for each unique variable field you create in the Image Editor. You can add records (variable fields) using the Field Database Editor or as you create sections using Image Editor. The FDB.MDX file serves as an index to this file.

See also [Field Database Editor on page 492](#).

fetype

An *fetype* defines the field format type. You can have an input and an output fetype. For example, an input fetype with the fmtnum rule tells the system where the decimal goes in the number. The output fetype tells the system how to format the output amount. An fetype can consist of either one or four characters.

For more information, see the [Rules Reference](#).

Field Database Editor

The Field Database Editor provides you with an easy way to store common variable field information to make setting up and creating FAP files faster and more consistent. When you use this tool, you create a file named FDB.DBF. This file contains a record for each unique variable field name, and is indexed in ascending order.

Use this tool to ensure consistency in forms sets. For example, if you have a Name variable field on a form, you can pull the attributes for that field into the form from the Field Database Editor. The database contains such information as the name of the field, font, type of variable field, and so on.

See also [External Database Editor on page 491](#).

Fixed Data

Fixed data is the same on every copy of the form. This includes items such as logos, headers and titles. This information remains constant regardless of the data entry.

Font Manager

Font Manager is used to organize fonts and font sets. A font is a collection of letters, symbols, and numbers that share a particular design. A font set is a collection of fonts you choose to group together for your section and printing needs. The font set information is stored in the font cross reference file (FXR file) which is created by Font Manager. Font Manager lets you make sure your documents print the same way on different printers.

A well organized font set makes section creation quick and efficient. Forms composers need a variety of fonts for text and field creation. Font Manager does not change the actual printer fonts. This tool is used for defining the appropriate characteristics (bold, size, and so on) about the font so the fonts used to create a particular form set are consistent and easily accessible to the forms composers.

Form

A form is a single document containing one or more pages or sections. Most forms contain multiple pages that are usually printed on both sides of a single sheet (duplex). Some forms are printed only on one side (simplex). Typical forms include insurance policies, tax returns, and mortgage documents.

A form includes two types of data: fixed and variable.

- Fixed data is the same on every copy of the form. This includes items such as logos, headers and titles. This information remains constant regardless of the data entry.
- Variable data may differ from form to form. This includes items such as individuals' names, addresses, and policy numbers. This information relates to the specific data processed on each form.

Form Set

A form set is a group of logically related forms required to process a single transaction. A form set may contain one or many forms. You can group forms any way you want as you create form sets.

Form Set Manager

This tool helps you group the individual sections and forms you create using Image Editor into a set of related forms. This information is stored in the form set definition table. The Entry module uses the form set definition table to control the data entry, print, work-in-process, and archive/retrieval functions for related forms and sections.

The system stores the form set information in a semi-colon delimited file named FORM.DAT. This file includes information about the company, line of business, forms, each section in the form, and the names of the person, organization, or entity who receives a copy of each section of the form. Specific information about the recipients of a form is stored in the Set Recipient table which is stored in the SETRCPTB.DAT file.

FORM.DAT File

This file, also known as the Form Set Definition table, contains information about the key fields, such as company, line of business, and policy number, plus information about each section in the form, its recipients, and the form set itself. The information is stored in semicolon-delimited format and you can edit this file using Form Set Manager or a text editor. The information that comprises the individual sections is stored in a FAP file.

FSISYS.INI File

The FSISYS.INI file is a one of the initialization (INI) files used by the system to set system parameters and to enable or disable system features. For example, the FSISYS.INI file contains information the GenTrn program uses to determine when a new record starts and other information about the extract files the GenTrn program processes.

NOTE: The Data control group in the FSISYS.INI file lets you specify many of the file names you want to use in Documaker Server. For instance, by modifying the settings in this group, you can change the name of the error file (ERRFILE.DAT) to any file name you want. In this manual, we refer to the default names for these files.

FSIUSER.INI File

The FSIUSER.INI file is one of the initialization (INI) files used by the system to set system parameters. For example, the FSIUSER.INI file contains information specific to each user, such as the location of files and so on.

.FXR Files

Font cross-reference (FXR) files are used by the system so you can make sure your documents print the same way, regardless of which printer you choose. These files contain information about the various fonts you use and their equivalents on various printers.

The system includes several font cross-reference files. You can edit and create font cross-reference files using the Font Manager.

GenArc Program

The GenArc program archives forms and data so you can store the information efficiently and retrieve it quickly. This program receives information stored in the APPIDX.DFD file from the GenData program. Using this information, the GenArc program creates CAR files to store the information and forms and DBF files which serve as an index to the data in the CAR files. The GenArc program can create multiple CAR files, as needed.

Depending on the operating system you use, this program has various names such as genacw32.exe for 32-bit Windows environments.

GenData Program

The GenData program takes information created by the GenTrn program and applies processing rules to those transactions and data. The GenData program creates batch files, the NAFILE.DAT, and the POLFILE.DAT file for the GenPrint program. It also creates a manual batch file for the GenWIP program. The output from the GenData program is also used by the GenArc program to archive forms and data.

Depending on the operating system you use, this program has various names such as gendaw32.exe for 32-bit Windows environments.

GenPrint Program

The GenPrint program takes information produced by the GenData program and creates a printer spool file for use with PCL, AFP, Metacode, and PostScript printers. Specifically, the GenData program produces batch files, an NAFILE.DAT, and a POLFILE.DAT file which the GenPrint program uses to create printed forms.

Depending on the operating system you use, this program has various names such as genptw32.exe for 32-bit Windows environments.

GenTrn Program

The GenTrn program reads data in an extract file and creates transaction records which in turn are processed by the GenData program. The main file created by the GenTrn program is the TRN file which, along with the TRNDFDFL.DFD file, tells the GenData program which transactions to process.

Depending on the operating system you use, this program has various names such as gentnw32.exe for 32-bit Windows environments.

GenWIP Program

The GenWIP program receives information about incomplete transactions from the GenData program. This information is stored in manual batch files. The GenWIP program then creates separate files for each incomplete transaction. The data for these incomplete transactions is stored in a file with the extension *DAT*, such as 00000001.DAT. The corresponding form set information is stored in a file with the extension *POL*, such as 00000001.POL.

The GenWIP program also creates the WIP.DBF file, a database file which contains records of all the incomplete transactions extracted from the NAFILE.DAT file produced by the GenData program. The WIP.MDX file, also created by the GenWIP program, serves as an index to the WIP.DBF file.

This gives the Entry module the information it needs to display the form so you can fill in the missing information and complete the form. Once completed, you can resubmit the form for processing by the GenData program.

Depending on the operating system you use, this program has various names such as genwpw32.exe for 32-bit Windows environments.

Help Editor

The Help Editor is a tool you can use to create user specific help records that are accessible from a variable field during form entry time. You can easily create a help file which contains records for the variable fields on a form. Each help record usually contains an explanation of a description for entering correct data into a variable field. Help files reside in the resource library.

Image (Section) <hr/>	<p>A section is a group of text or graphics or both that make up a form or a section of a form. You create sections using Documaker Studio or Image Editor. Each section is stored in a separate file, so you can reuse sections in several forms and form sets. Multiple sections can comprise a single form. For instance, a three-page form with text and graphics, printed on both sides of each page, could contain a total of six sections. Some examples of sections include an insurance policy declaration page, the return portion of a bill, and page one of a 1040 Federal tax return form.</p> <p>You may choose to create a single page containing multiple sections, especially if you develop a page with graphics.</p>
Image Editor <hr/>	<p>The Image Editor lets you create documents, forms, and sections that become part of an electronic form or document. It is a full-featured design tool with an easy to learn and use graphical user interface. With Image Editor, you have complete control and flexibility in managing and creating your section. The section and objects that you create are stored in the resource library's section library. Each section is stored in a file with the extension, <i>FAP</i>. Sections are also referred to as <i>FAPs</i>.</p>
.INI Files <hr/>	<p>Initialization (INI) files are used by the system to set system parameters and to enable or disable system features. Some examples of system INI files are: <i>FSISYS.INI</i> and <i>FSIUSER.INI</i>. For example, the <i>FSISYS.INI</i> file contains information the GenTrn program uses to determine when a new record starts and other information about the extract files the GenTrn program processes. The <i>FSIUSER.INI</i> file contains information specific to each user, such as the location of files and so on.</p>
INTL.FXR <hr/>	<p>This font cross reference file includes international characters for producing forms in languages other than English.</p>
INTLSM.FXR <hr/>	<p>A smaller version of the <i>INTL.FXR</i> font cross reference file, this file includes international characters for producing forms in languages other than English.</p>
.JDT Files <hr/>	<p>Job Definition Table (JDT) files tell the system which rules to use as it processes a specific job. Rules defined in the JDT file are run before the system runs rules assigned to specific fields. An example of a JDT file is the <i>AFGJOB.JDT</i> file.</p> <p>See also DDT Files on page 489.</p>
Library Manager <hr/>	<p>The Library Manager lets you manage documents and logos while maintaining the versions, revisions, and integrity of the sections you are developing. You may want to set up a library for a specific client or form set. You can store all sections and logos in a resource library. The storage consists of a listing of the section or logo, as well as a snap shot of the section.</p> <p>When you set up a library, you must define the locations of the library and storage files. Entries made during library setup are automatically saved back to the <i>FSIUSER.INI</i> file when you exit the setup window.</p>

Log Files

When you run GenTrn, the program creates log files which record by transaction each transaction the program processes. You can review these log files using any editor.

.LOG Files

Logos and other graphics, such as scanned signatures, are stored as LOG files in the system. You use Logo Manager to manage and manipulate LOG files. You can view these files using Logo Manager.

Logo Manager

Once you create a graphic object such as a logo or a scanned signature, you can edit it using Logo Manager. This tool lets you resize, reverse, rotate, crop, and otherwise manipulate an section to fit your needs. The system stores these graphic files as LOG files.

MANUAL.BCH File

The GenData program creates this file if it is unable to complete the processing of a form set. Typically, this occurs because the forms are missing information. This file is then used by the GenWIP program so a data entry operator can manually complete the form and resubmit it for processing.

See also [Batch Files on page 488](#) and the [GenWIP Program on page 494](#).

Master Resource Library

Master resource libraries provide a central repository into which you can place all reusable resources such as sections, fonts, graphic files, data definitions, processing rules, and processing procedures. A master resource library contains an section library, a variable data dictionary library, a rules library, and a system library.

See also [Distributed Resource Library on page 490](#).

Metacode

A printer definition language developed by Xerox. Metacode is the native language of Xerox's Centralized Printing Systems. The GenPrint program can create spool files for Metacode printers.

.MDX Files

The various system programs create MDX files which serve as indexes to the database files (.DBF files). For example, the GenWIP program creates the WIP.DBF file and the corresponding WIP.MDX file to record the incomplete transactions which were not printed.

The Field Database Editor creates the FDB.MDX file to serve as an index to the FDB.DBF file which contains common variable field definitions.

NAFILE.DAT File

The GenData program creates an NAFILE.DAT file, commonly referred to as the NA file, in which it stores section and variable field information. The GenPrint program uses this file, along with the POLFILE.DAT file, which is also produced by the GenData program to print the forms.

If the data is incomplete and GenData cannot complete the form, it creates a manual batch file. The GenWIP program then creates separate DAT and POL files for each incomplete transaction. These files provide the entry system with the information it needs to open the form so a data entry operator can add the missing data. This is a comma-delimited text file.

NEWTRN.DAT File

The GenData program creates the NEWTRN.DAT file. This file tells the GenArc and GenWIP programs where to find data in the NAFILE.DAT file and which forms to use in the POLFILE.DAT file. This is a comma-delimited text file.

Objects

Objects are the individual items which comprise your section. Examples of objects are boxes, bar codes, lines, graphics, and text. All objects have unique attributes within the section. Attributes include items such as position, size, font type, and color. Documaker Studio and Image Editor let you easily create the various objects which comprise an section.

Overflow

Overflow refers to a situation where there is not enough room on the form for all of the data you need to enter. In this situation, you want to have the system automatically place the additional data onto another form or another copy of the same form. The system includes features which let you do this.

For instance, suppose you have a form which records automobiles and the drivers of the automobiles. The form has room to record four different automobiles and drivers. In most cases this will suffice but, in some situations, you need to include information about additional automobiles and drivers. Using the overflow features, you can handle this situation automatically.

Page

Pages are the printed result of an section or a group of sections. You can have one section per page, several sections per page, or even an section that spans several pages. You determine the size of a page based on the size of your printed output. With Documaker Studio or Image Editor, you can design forms for any size page your printer can print.

PCL

PCL (Printer Control Language) is a printer definition language developed by the Hewlett-Packard company. The GenPrint program can create spool files for PCL printers.

POLFILE.DAT File

The POLFILE.DAT file, commonly referred to as the POL file, defines the form set used for a specific transaction. The GenData program creates this file which is used by the GenPrint, GenWIP, and GenArc programs. For instance, if the data is complete, GenData creates an NA file and a POL file. These files are used by GenPrint, along with the batch files, to produce the print-ready file.

If the data is incomplete and GenData cannot prepare the form for printing, it creates a manual batch file. The GenWIP program then creates separate files for each transaction to provide the entry system with the information it needs to open the form so a data entry operator can add the missing data. This is a semicolon-delimited text file.

PostScript

PostScript is a printer definition language developed by Adobe Systems which you can use on various printers. The GenPrint program can create spool files for PostScript printers.

Section

A section (image) is a group of text or graphics or both that make up a form or a section of a form. You create sections using Documaker Studio or Image Editor. Each section is stored in a separate file, so you can reuse sections in several forms and form sets. Multiple sections can comprise a single form. For instance, a three-page form with text and graphics, printed on both sides of each page, could contain a total of six sections. Some examples of sections include an insurance policy declaration page, the return portion of a bill, and page one of a 1040 Federal tax return form.

You may choose to create a single page containing multiple sections, especially if you develop a page with graphics.

SETRCPTB.DAT File

This file, also known as the Form Set Trigger table, contains information which tells the GenData program which recipients receive which forms or sections.

This file also contains the information the GenData program needs to determine whether or not to include or exclude a form. You can define conditions using Form Set Manager or by editing the SETRCPTB file in a text editor.

Simplex

A form printed on only one side of a sheet of paper is printed in simplex mode.

See also [Duplex on page 490](#).

System Releases

To continually improve and support the product, software enhancements and corrections are organized into regularly scheduled system releases. Releases are noted with a major and minor version number, such as 10.3 or 11.0.

System Patches

In certain situations, and on a case by case basis, a correction to the current system release can be made available as a system patch. Corrections to the prior release are handled on a case by case basis, and are made available only as system patches.

Table Editor

The Table Editor lets you create a table of data used to automatically fill a variable field during form data entry. Tables make the entry process quicker and more efficient for the end user. Users can choose from data options within a table format rather than keying information. Using tables reduces data entry errors and increases speed. In the Table Editor, you can create and edit table files, tables, and table entries. Tables are stored in the resource library.

Transaction List

The GenTrn program creates the transaction list which is used by the GenData program as an index to the data in the extract file. The transaction list is stored in the TRN File.

.TRN Files

The GenTrn program creates transaction or TRN files which contains a record for each transaction. The record format for the TRN file can vary to meet your needs. This format is defined in the TRNDFDL.DFD file. The GenData program uses the TRNDFDL.DFD file to read the information in the TRN file as it processes the information.

Each record in a TRN file contains a series of offsets or pointers. These offsets define the location of the transaction data. For instance, the offsets in a TRN file tell the GenData program where the transaction begins in the extract file, where the data for the transaction is stored in the NAFILE.DAT file, and where the form set for the transaction is stored in the POLFILE.DAT file.

TRNDFDFL.DFD File

The TRNDFDFL.DFD file tells the GenData program how to read the TRN file. If necessary, you can edit this text file in a text editor.

UFSTSM.FXR File

This is a font cross reference file which provides Times (Roman), Courier, and Univers(al) fonts for Xerox, AFP, PostScript, and PCL printers. This font cross reference file is automatically installed when you install Docucreate.

UNIQUE.DBF File

The UNIQUE.DBF file contains the last number for the WIP file that was created. Whenever a WIP file is created, a number is generated to uniquely identify it to make sure no WIP file is overwritten. You should not modify, rename, or delete this file. The highest number it will generate for WIP files is FFFFFFFF, which is 4,294,967,295. After this number, the counter resets to 00000001.

The GenWIP and GenArc programs use this information to create separate data and form information files for the incomplete transactions received from the GenData program and for the individual forms stored in archive.

See also [00000001.DAT File on page 487](#) and [00000001.POL File on page 487](#).

Variable Data

Variable data may differ from form to form. This includes items such as individuals' names, addresses, and policy numbers. This information relates to the specific data processed on each form.

WIP.DBF File

The WIP.DBF file contains information about the incomplete transactions which the GenWIP and GenArc programs extracted from the NAFILE.DAT and POLFILE.DAT file created by the GenData program. The WIP.MDX file serves as an index to this file.

See also the [GenWIP Program on page 494](#).

WIP.MDX

This file serves as an index to the WIP.DBF file.

xBase

A generic term for industry-standard dBase IV file format.

Index

SYMBOLS

& (ampersand) 377
.BCH files 464
.CAR files 464
.DAT files 464
.DBF files 464
.DDT files 464, 474
.DFD files 465
.FAP files 466
.INI files 466
.JDT files 466, 475
.LOG files 466
.MDX files 466
~Encrypted 113
~GetEnv function 111
~OS function 112
~Platform function 112
~WIPField built-in function 115

NUMERICS

00000001.DAT file 484
00000001.POL file 484
2-up printing
 overview 68
 rule order 74

A

A4

- PaperSize option 248, 273

ABNORMAL statements 284

Access databases 411

Acrobat Reader

- included fonts 208

AddBlankPages function 96

AddComment function 299, 300

AddDocusaveComment function 359

Added_Fonts control group 71

AddedOn option 398

AddForm function

- banner forms 16

adding

- fonts to the font set 214

- printer fonts to the FXR file 189

- tables of contents and indexes 99

AdditionalDJE option 279

AddLine rule 72

ADDPAGES utility 269

AddTextLabel rule 72

AdjLeftMargin option 290

Adobe Acrobat 2

AFEMAIN program

- defined 384

- viewing archives 410

AfeProcedures control group 413

AFEW32 413

AFEWIP2ArchiveRecord 413

AfeWIP2ArchiveRecord control group 413

AFG2WIP control group 89

AFGJOB.JDT file 466, 475

- and 2-up printing 74

AFM files 174

AFP

- comment records 72

- fonts 188, 234

- record list and the AddTextLabel rule 72

- using custom fonts 195

AFP control group 250

AFP printers

- fonts 258

- form-level comments 262

- handling multiple trays 363

- highlight color printing 251

- INI options 246

- overlays 258

- page segments 258

- paper size 248

- resources 258

- setting up 246

- TLE records 362

- troubleshooting 259

AFPFMDEF utility 258

AIX

- archive/retrieval scenarios 387

AliasPrintBatches option 103

AllowInput option 322

AlwaysSQLPrepare option 393

ampersands

- in messages 378

ancestor 123

ANSI code page

- for PC platforms 201

APPIDX file

- defined 384

APPIDX.DBF file 485

APPIDX.DFD file 473, 485

APPIDX.MDX file 485

application index file 473

archive

- creating print streams for Docusave 358

- features 386

- field names 422

- retrieval 416

- TLE records 362

- transaction log 392

archive index file

- and WIP 413

- Archive rule 51, 52
- ARCHIVE.CAR file 485
- ArchiveMem option 392, 398
- archiving
 - transactions 31
- ArcRet control group 392, 417
- ARCRET utility 424
- ARCVIEW utility 410
- ASCII
 - code pages 179
- Asian languages
 - PCL 6 293
- Auto-size option 259

B

- banner form processing
 - multi-file print 17
- banner forms
 - groups 16
- banner processing
 - custom callback function 15
 - overview 15
- BARR
 - format 288
 - interface attachment 288
- BARR SPOOL
 - OutMode option 266
- BARRWRAP utility 288
- BaseErrors option 34
- baseline 220
- BaseRuleTime option 105
- Batch control group 52
- batch files 482
 - and single-step processing 46
 - grouping 103
 - page statistics 37

- BatchBannerBeginForm option 16
- BatchBannerBeginScript option 16
- BatchBannerEndForm option 16
- BatchBannerEndScript option 16
- BatchByPageCount rule 52
- BatchingByRecip control group 52
- BatchingByRecipINI rule 52
- BatchPrint control group 321
- BatchTable option 103
- bitmap compression
 - PCL print driver 298
 - PostScript printer driver 305
- bitmap fonts
 - defined 175
 - inserting 234
- bitmaps
 - compression for Metacode printers 266
 - highlight color printers 298
 - Metacode LGO files 282
 - scaling 250
 - Xerox images 282
- black rectangles 259
- blank pages 269
- boxes, WriteFrame option 322
- BreakBatch function 81
- BuildMasterFormList rule 52
- built-in functions 111
- business envelopes 296
- byte-serving 320

C

- Cabinet option 398
- CacheFAPFiles option 277
- CacheFiles option 277
- CacheMethod option 277
- callback functions
 - InitPageBatchedJob rule 72
- CallbackFunc option 85
 - RTF driver 343

- CARData control group 409
- CARFILE
 - defined 384
- CARFILE.DFD file 405
- case toggles 265
- CaseSensitiveKeys option 414
- CATALOG file
 - defined 385
- CD/IG 286
- Character Set field 252
- character sets
 - defined 182
 - determining characters used in a printer font 182
- Character Width field 221
- ChartResolution option
 - AFP printers 247
 - Metacode printers 271
- charts
 - BARRWRAP utility 288
 - compression for Metacode printers 266
 - printing on Metacode printers 266
 - rendering on Metacode printers 271
 - using the Metacode loader 278
- CheckCount option 33
- CheckImageLoaded rule
 - rotated variable fields 260
- CheckZeroFontID option 102
- child 123
- Class option 411
 - AFP printers 248
 - GDI driver 314
 - Metacode printers 278
 - PCL printers 291
 - PostScript printers 303
- class recipient 65
- ClearMsgFile option 370
- CMY palette 297
- Code Page Font field 252
- code pages
 - ASCII code pages 179
 - code page 1004 178, 180
 - code page 37 181
 - code page 437 178
 - code page 850 178, 179
 - code page names 184
 - converting text files from one code page to another 204
 - EBCDIC code pages 181
 - for EBCDIC platforms 202
 - using the ANSI code page for PC platforms 201
- CODE statement 265
- CODEPAGE.INI file
 - and PostScript fonts 189, 236
 - and the Char Set ID field 224
 - and the CPCNV utility 195, 196
 - defined 208
- ColorCharts option 274
- colors
 - for charts 274
 - PCL support for 289
 - printing 251
 - simple color mode 297
 - specifying ink for Metacode printers 271
 - troubleshooting for Metacode printers 283
- column names 433
- COMM_RECS column
 - restarting GenArc 396
- commas
 - in search masks 164
- comment records 18
- commit
 - defined 384

- CommitEvery option 396
- CommonFonts control group 273
- CompileInStream option 267
- Compression option 305
- CompressMode option 266
- concepts
 - setting recipients and copy counts 138
- configuring
 - the message system 369
 - the system 132
- console
 - logging information 104
- console messages 274
- controlling the message translation process 372
- converting
 - fonts from other vendors 229
 - system fonts 227
 - text files from one code page to another 204
- ConvertWIP rule 51, 53
- copy counts
 - DAL and GVM variables 152
 - example 152
 - setting 137, 138
- copying
 - fonts 225
- Counter field 141, 471
- counter search mask 155
- CounterDFD option 70
- CounterTbl option 52, 53, 70
- CreateIndex option 408
- CreateTable option 408
- CreateTime field 88
- CreateTime option 398
- creating
 - print spool files (multi-step processing) 27
 - transaction records (multi-step processing) 21
 - transaction records (single-step processing) 45

- creating messages 374
- CRYRU utility 113
- CSTSetMailRecip function 344
- CUSSetMailRecipGVM function 344
- custom callback function in banner processing 15
- custom fonts 194
- Custom Rule field 141
- custom rules
 - field 472

D

- DAL
 - analyzing performance 106
- DAL functions
 - manipulating file names 82
- DAL scripts
 - and extract files 477
 - banner processing 16
 - creating print streams for Docusave 359
 - splitting print streams 79
- DALFile option 406
- DALLibraries control group 16, 97
- DALRUN built-in function 113
- DALVAR built-in function 113
- data
 - length validation 287
- Data control group
 - print batches 103
- data definition table
 - defined 464
 - file format 474
- data format definition files 465
- data table files 464
- database
 - archiving to 392
- database files 464, 472
- DataPath option 103
 - and message files 370
 - and the TRANSLAT.INI file 371
- date stamps
 - turning off 372

- DB Field Name values 422
- DB2
 - databases 411
- dBase 499
- DBErrors option 395
- DBHandler option 407, 408, 412
- DBLib tracing 373
- DBLogFile option 373
- DBTable option 407
- DCD files
 - mapping fonts 244
- DDTFile option 407
- Debug option 408, 412
- Debug_If_Rule option 101
- Debug_Switches control group 101
- DefaultTag option 393
- defining
 - output message files 370
- DefLib option
 - and the TRANSLAT.INI file 371
 - PostScript printers 302, 310
- DelBlankPages function 97
- deleting
 - fonts 231
- descendant 123
- Description tab
 - fonts 216, 218
- Deselect All option
 - fonts 213
- DestField token 381
- Device field 318
- Device option 268
 - AFP printers 246
 - GDI driver 313
 - PCL printers 289
 - PostScript printers 301
- DeviceName function 81
- DFD file
 - defined 385
- DFD files 472
 - and 2-up printing 71
 - format 477
- Dimensions tab
 - fonts 219
- DisplayCodedFont option 250, 252
- DJDE command 279
- DJDE statements
 - user-defined 279
- DJDECarrControl option 279
- DJDEForceOffsetEnd option 270
- DJDElden option 265
- DJDELevel option 276
- DJDEOffset option 265
- DJDESkip option 265
- DlgTitles control group 417
- DocSetNames control group 50
- Documaker Server
 - resource files 467
 - running via IDS 57
 - system benefits 8
 - system overview 2
 - understanding the system 9
- Documanager
 - categorizing documents 421
 - data types 420
 - Extended Document Properties 425
 - mapping Documaker archive fields 422
 - Next/Retrieve cursor 424
 - using resources in GenData and GenPrint 13
 - using with GenArc 397
 - viewing archives 410
- Document Type Number 166
- Docupresentment 2
 - PDF support 320
- Docusave
 - creating print streams 358
 - retrieving form sets 416
- DocusaveScript option 249, 359
- dots per inch
 - Resolution option 246
- DoubleOutputRes option
 - AFP printers 250
- DownloadFAP option 99, 305
 - and the CompileInStream option 267

- DownloadFonts option 300
 - emailing forms 345
 - GDI driver 313
 - PCL printer resources 300
 - PCL printers 290
 - PostScript printers 302, 310
- DPA files
 - viewing 410
- DPASSWD command line option 393
- DPRAddBlankPages rule 98
- DPRDelBlankPages rule 98
- DSCHeaderComment option
 - PostScript printers 301
- duplex
 - adding and removing pages 96
 - and simplex on Metacode printers 269
 - compressed PCL files 298
 - printing multi-page FAP files 285
 - switching modes 276
- DUSERID command line option 394

E

- EBCDIC 465
- EBCDIC platforms
 - and code pages 181
 - using Code Page 37 202
- editing
 - fonts 226
- EjectPage rule
 - multi-page FAP files 285
- ElapsedTimeStamp option 102
- email
 - aliases 345
 - GenWIP 90
 - sending a print-ready file 342

- Email Application Servers 345
- embedded hex values 265
- embedding fonts 208
- EMIT_ERROR type 374
- EMIT_MESSAGE type 374
- EMIT_WARNING type 374
- EmptyFooters option 323
- EmptyHeaders option 323
- Enable_Debug_Options option 101, 104, 373
- EnableEmailNotification option 90
- EnableTransBanner option 97
- encrypted values 113
- end of report conditions 269
- envelope feeders 296
- EPTLIB 342
- EPTSetRecipFunc function 344
- ERRFile option 370
- ERRFILE.DAT file 368
 - and the ImmediateTranslate option 372
- error batch 482
- error codes 369
- error files 465, 480
 - turning off the date stamp 372
- error messages
 - configuring 369
 - creating 374
 - defining the output file 370
 - delaying the translation process 372
 - determining where a message originates 380
 - disabling 369
 - formatting 378
 - initializing output files 371
 - message tokens 376
 - negative left offsets 259
 - overview 368
 - setting up static text 377
- ErrorFileDateStamp option 102
- ErrorFileOpenMode option 371
- errors
 - correcting 46
 - using GenArc with Documange 409

European paper 469
examples
 copy counts and sections 152
 of form set definition files and transaction trigger tables 151
 RECIPIF rule 157
 search mask and sections 155
 setting search masks and recipients 162
 transaction code 154
Excel spreadsheet databases 411
executive
 PaperSize option 248, 273
executive paper 469
ExportIndex option 392
Expression option 103
Ext option
 Metacode printers 278
EXT_Length option 71, 409
Extended Binary Coded Decimal Interchange Code 181
Extended Document Properties (XDPs) 421
extract files
 and code pages 204
 defined 465
 guidelines for 475
 layout of 476
 NoGenTrnTransactionProc rule 53
 offset limits 108
 XML files 121
EXTRACT.DAT file 465

F

FAP files
 adding and removing 96
 mapping fonts 244

FAP2CFA utility 283
FAP2FRM utility 282, 287
FAP2MET utility 267, 277, 282, 283
FAP2OVL utility 258
FAPAddBlankPages 96
FAPCOMP.INI file 467
 mapping fonts 244
 Metacode loader 278
FAPDelBlankPages 96
FAX drivers 316
fax, drivers 311
FEED command 286
FIELD
 BatchName control group 103
Field Description control group 478
FieldErrors option 34
FieldFuncTime option 105
FieldList option 103
fields
 mapping with XPath 50
Fields control group 477
 grouping print batches 103
file names
 DAL functions 82
File option
 INIFiles control group 112
file summary
 GenArc program 32
 GenData program (multi-step processing) 25
 GenData program (single-step processing) 48
 GenPrint program (multi-step processing) 28
 GenTrn program (multi-step processing) 22
 GenWIP program (multi-step processing) 30

- FileDrive function 82
- FileExt function 82
- FileName function 82
- FilePath function 82
- files
 - .CAR files 464
 - .DAT files 464
 - .DBF files 464
 - .DDT files 464, 474
 - .DFD files 465
 - .FAP files 466
 - .JDT files 466
 - .LOG files 466
 - .MDX files 466
 - ooooooo1.DAT file 484
 - ooooooo1.POL file 484
 - APPIDX.DBF file 485
 - APPIDX.DFD file 473, 485
 - APPIDX.MDX file 485
 - ARCHIVE.CAR file 485
 - batch files 482
 - BCH files 464
 - created by the GenData program 481
 - created by the GenTrn program 480
 - created by the GenWIP program 484
 - DFD file format 477
 - DFD files 472
 - error batch files 482
 - error files 465, 480
 - extract files 465, 475
 - FORM.DAT file 139, 467
 - formats of 461
 - FSISYS.INI file 467
 - FSIUSER.INI file 467
 - initialization files 466
 - JDT files 475
 - log files 466, 480
 - MANUAL.BCH file 482
 - NAFILE.DAT file 481
 - NEWTRAN.DAT file 482
 - POLFILE.DAT file 481
 - RCBDFDFL.DFD file 473
 - recipient and copy count files 139
 - resource files 467
 - SETRCPTB.DAT file 470

- system files 461
- transaction files 466, 480
- TRNDFDFL.DFD file 473
- types of 464
- UNIQUE.DBF file 484
- updated log and error files 482, 483
- used by the GenArc program 485
- WIP.DBF file 484
- WIP.MDX file 484
- FileType option 398
- filter list 213
- FinalPrinter option
 - and 2-up printing 69
- FitToWidth option
 - GDI driver 313
 - PCL printers 290
 - PostScript printers 302
- floating section limitations 259
- FolderBy option 397
- folders
 - updating 399
- FON files 221
- font cross-reference files
 - adding printer fonts 189
 - AFP printer resolution 261
 - choosing 199
 - for Monotype fonts 191
 - GDI drivers 312
 - generating 241
 - inserting 234
- Font List window 234
- Font Manager
 - choosing screen fonts 238
 - generating files 240
 - starting 199, 211
 - using 211
 - using FXR files 199
- Font Properties window 199
 - using Like to select screen fonts 238
- FontFamilyMatching control group 244
- FontLib option
 - PCL printers 300
 - PostScript printers 310

fonts

- adding system fonts 214
- AFP 188
- available font types 232
- bitmap fonts 175
- character width 221
- common font lists 272
- converting 227
- converting fonts from other vendors 229
- copying 225
- custom fonts 194
- deleting 231
- Description tab 216, 218
- Deselect All option 213
- Dimensions tab 219
- editing 226
- embedding 300
- families 218
- filter list 213
- fixed 219
- FON files 221
- font IDs 218
- Font List window 212
- font substitution in Windows 187
- FXR files 221
- FXR files for Monotype fonts 191
- generating FNT files 240
- generating PFM files 241
- generating XRF files 241
- how computers and printers use fonts 176
- IDs equal to zero 102
- inserting 232
- inserting bitmap fonts 234
- inserting FXR files 234
- inserting PostScript fonts 235
- installing screen fonts in Windows 188
- Metacode 188
- Monotype fonts 190
- naming conventions 210
- PCL 189
- point size 218
- PostScript 176, 189
- PostScript printers 310
- printer fonts 188
- proportional 219

- resetting 216
- scalable fonts 175
- screen fonts 187
- selecting 212
- setting up 171
- styles 219
- terminology 172
- True Type 176
- typefaces 218
- using the Font Manager 211

footer 469

footers

- in RTF files 323

ForceFolderUpdate option 399

ForceNoImages rule 72

form level triggers 139, 146

Form Name field 471

Form name field 140

Form option 58

form set definition table 139, 467

- examples 151

- summary 168

Form Set Manager 138, 275

form set trigger table 470

form sets

- adding and removing pages 96

- PrintFormset rule 54

FORM.DAT file 139, 467

- banner processing 15, 19

- examples 151

- marking forms printer resident 288

- single-step processing 52

format

- DFD files 477

- trigger table record 140

FormDef, AFP resources 258

FormFile option 407

form-level comments 262

FormLib option

- PostScript printers 302

- pre-compiled MET files 267, 277

FormMaker II XRF files

- generating 241

- FormNameCR option 262
- forms
 - background 268
 - marking master forms 150
 - requirements 132
 - triggering in XML files 122
- FormSetID field 87
- FormSetRuleTime option 105
- frames
 - WriteFrame option 322
- FRM files
 - CompileInStream option 268
- FRMFile option 288
- FSIFileName taken 379
- FSIFileName token 381
- FSILineNumber token 379, 381
- FSIPATH environment variable 410
- FSISYS.INI file 467
 - and 2-up printing 69
 - banner processing 19, 20
 - grouping print batches 103
 - single-step processing 46
- FSIUSER.INI file 467
 - INIFiles control group 112
 - single-step processing 46
- FSRSetFileAttachment API 345
- FudgeWidth option
 - AFP printers 247
- FullFileName function 82
- FullSupport option 316, 317
- functions
 - built-in INI functions 111

- FXR File field 221
- FXR files
 - affect on display and print quality 198
 - choosing FXR files 199
 - choosing screen fonts 238
 - insert 234
 - inserting 234

G

- GDI driver
 - handling multiple trays 363
 - INI options 313
 - Netware Client 32 for Windows 95 318
 - troubleshooting 318
- GDIDevice option 314, 317
 - and the Device option 315
- GEN_DEBUG_DebugSwitchSet function 101
- Gen_TabUtil_LoadListFromTable function 101
- GenArc program
 - .CAR files 464
 - and Documanager 425
 - APPIDX.DBF file 485
 - APPIDX.DFD file 485
 - APPIDX.MDX file 485
 - ARCHIVE.CAR file 485
 - archiving transactions 31
 - command line options 393
 - description 12
 - file summary 32
 - files used 485
 - output files 24
 - running 392
 - single-step processing 51
 - system scenarios 386
 - using with Documanager 397

- GenData program
 - .BCH files 464
 - .DDT files 464
 - batch files 482
 - command line options 100
 - description 11
 - error batch files 465, 482
 - file summary (multi-step processing) 25
 - file summary (single-step processing) 48
 - files created 481
 - MANUAL.BCH file 482
 - NAFILE.DAT file 464, 481
 - NEWTRAN.DAT file 482
 - processing transactions (multi-step processing) 23
 - restarting 33
 - TRNDFDFL.DFD file 473
 - updated log and error files 482, 483
- GenDataStopOn control group 34
- Generate Font Download File window 240
- Generate FormMaker II XRF File window 241
- GenPrint program
 - accessing batch totals 38
 - banner processing 15
 - command line options 100
 - creating print spool files (multi-step processing) 27
 - creating print streams for Docusave 358
 - description 12
 - embedding fonts 300
 - file summary (multi-step processing) 28
 - output files from GenData (multi-step processing) 24
- GenTranStopOn control group 43
- GenTrn
 - controlling processing 43
- GenTrn program
 - and single-step processing 45
 - command line options 101
 - creating transaction records for multi-step processing 21
 - description 11
 - error files 465, 480
 - file summary (multi-step processing) 22
 - files created 480
 - initializing message files 371
 - log files 466, 480
 - transaction files 466, 480
 - TRNDFDFL file 473
- GenWIP program
 - .DAT files 464
 - oooooooo1.DAT file 484
 - oooooooo1.POL file 484
 - description 12
 - field assignments 87
 - file summary (multi-step processing) 30
 - files created 484
 - generating emails 90
 - output files from GenData (multi-step processing) 24
 - sending incomplete transactions to WIP 29
 - UNIQUE.DBF file 484
 - WIP.DBF file 484
 - WIP.MDX file 484
- GETENV INI function 88
- GetEnv INI function 415
- GetRCBRec rule 72
- GHO hardware 266
- GOCA charts support 247
- going live 132
- graphics
 - compression for Metacode printers 266
 - orientation 284
 - rendering 274
 - using the Metacode loader 278
- Graphics Device Interface (GDI) print driver 311
- GraphicSupport option
 - AFP printers 247

GroupName1 field 140, 471
GroupName2 field 140, 471
GVG hardware card 266, 282

H

H2 strings 265
H6 strings 265
header 469
header records
 and extract files 477
headers
 in RTF files 323
hidden 469
highlight color printing
 AFP 251
HighlightBlackCmd option 298
HighlightColor option 290
HighlightColorCmd option 298
horizontal motion index 282
HPINTL.FXR file 191
HPINTLSM.FXR file 191
HP-UX
 archive/retrieval scenarios 387

I

IBMXREF.TBL file 252
IDEN statement 265
IDS
 running Documaker Server 57
 trace file 373
image level triggers 139, 144
Image Name field 140, 471
Image option 58
ImageErrors option 34
ImageFuncTime option 105
ImageOpt option
 Metacode printers 266

ImageRuleTime option 105
imaging systems 299
 adding PJI comments 299
ImmediateTranslate option 372
 and ERRFILE.DAT 372
implementation methodologies 132
implementing your system 131
indexes
 adding 99
InfoPak 269
INI built-in functions 111
INI command line option 393
INI files
 changes for 2-up printing 69
 logging 370
 using multiple 112
INI options
 logging 370
INIFiles control group 112
INIGroup control group 113
INILib option 104
InitArchive rule 51, 53
InitConvertWIP rule 51, 53
InitFunc option 300
 RTF driver 343
initialization files 466
InitMerge rule 72
InitPageBatchedJob rule 72
InitPrint rule 53
 and the NoGenTranTransactionProc rule 53
InitSetrecipCache rule 53
ink color 271
inkjet printers 311
inline graphics
 and the CompressMode option 266
 BARRWRAP utility 288
 LOG files 250
Insert Fonts window 232
inserting
 bitmap fonts 234
 fonts 232
 FXR files 234
 PostScript fonts 235

installable functions 268
installing
 screen fonts in Windows 188
 the system 132
INT_LENGTH option 71
INT_Length option 409
international language support 201
Internet Document Server (IDS)
 compressed PCF files 298
 paper size 248, 273
InUse field 87
ISI.INI file 243

J

JDName option 264
JDLCode option 265
JDLData option
 defined 265
 Metacode printers 287
JDLHost option 265
JDLName option 264
JDLRPage option 269
JDLRStack option 269
JDLs
 setting up Metacode printers 263
JES2 format 288, 358
job definition table 466, 475
JOBID command line option 393
JOBID parameter
 restarting GenArc 395
jogging pages 270
JSLs
 setting up Metacode printers 263
jump to new sheet condition 269

K

KEY
 BatchName control group 103

key fields
 and extract files 477
Key1
 CaseSensitiveKeys option 414
KeyID
 CaseSensitiveKeys option 414

L

landscape 469
 AFP limitations 259
 graphic orientation 284
Landscape option
 GDI driver 314
LandscapeSupport option
 AFP printers 247
language
 international language support 201
 national language terminology 173
 using international characters 203
LanguageLevel option 303
LASTERRORTOKEN token 376, 379
LASTREC column
 restarting GenArc 396
LBYD option 408
LBYI option 408
LBYLOG option 408
LBYLOGFile option 407
legal
 PaperSize option 248, 273
letter
 PaperSize option 248, 273
letter size paper 469
Like button 216
limitations
 floating sections 259
 multi-page FAPs 260
line density errors 286
LINE statement 265
lists of figures
 adding 99

Load PostScript Font File window 236

Loader:Met control group 278

LoadFAPBitmap option 284

LoadListFromTable option 101

log files

- configuring 369

- creating log messages 374

- defined 466

- defining the output file 370

- delaying the translation process 372

- determining where a message originates 380

- disabling 369

- formatting 378

- GenTrn program 480

- initializing output files 371

- message tokens 376

- of archived transactions 392

- overview 368

- setting up static text 377

- turning off the date stamp 372

LOG2PSEG utility 258

LogCaching option 277

LOGFile option 370

LOGFILE.DAT file 368

LogFileDateStamp option 102

logging messages 373

logical printers 85

LogINIFileNames option 370

LogINIOptions option 370

Logo Manager 274, 282

LOGO.DAT file

- printing MET files 278

LOGOFile option 407

LogToConsole option 104, 392

LogTransactions option 369

LookUp rule

- and extract files 477

LRECL values 287

M

Mail control group 345

MailAttachment option 90

MailID option 90

MailMessageBody option 90

MailSubject option 90

MailType option 345

MANUAL.BCH file 482

Map Coded Font (MCF) fields 252

MapByDBName option 422

margins

- added by PCL printers 290

- setting minimum 323

marking

- master forms 150

- subordinate sections 149

Master and Subordinate Sections 149

master flag

- and performance 163

master forms

- marking 150

master resource libraries

- implementation 132

MasterDDTNotInLibrary option 407

MasterResource control group

- PCL resources 300

- pre-compiled MET files 267

MaxFonts option 273

MaxPolLineLength option 481

MergeAFP rule 73

message information 342

Message option

- RTF driver 343

message token file

- using 380

message token files

- defining the output file 370

- overview 369

messages

- assigning message numbers 375
- clearing 370
- configuring 369
- creating 374
- defining output message files 370
- determining where the originated 379
- formatting 378
- initializing output message files 371
- message numbers and static text 377
- types 374
- using tokens 376

MET files

- and multi-page FAP files 285

Metacode

- fonts 188

Metacode printers

- creating print streams for Docusave 358
- data length validation 287
- end of report conditions 269
- handling multiple trays 363
- JSL INI options 263
- resources 282
- setting up 263
- troubleshooting 283

METDUMP utility 281

methodologies for implementation 132

METOPT utility

- common font lists 273

Mixed Object Document Content Architecture data streams 246

Mobius

- InfoPak 269
- ViewDirect APIs 281

ModifyTime field 87

Module option

- AFP printers 246
- GDI driver 313
- PCL printers 289
- PostScript printers 301
- RTF driver 343

Monotype fonts 258

- FXR files 191
- using system fonts 190

MRG2FAP utility

- paper size 248, 273

MRG4 format 358

MSGFile option 370

MSGFILE.DAT file 368, 380

msgNO_MORE_IMAGES message 72

MTCLoadFormset rule 281

multi-file print callback method 79

MultiFileLog option

- RTF driver 343

MultiFileLogRecord option 109

MultiFilePrint callback function 49, 166

MultiFilePrint option

- controlling the log 109

MultiLinesPerCommand option

- AFP printers 250

multi-mail transaction

- and the EXT_LENGTH option 71

multi-mail transactions

- PageBatchStage1InitTerm rule 53

multi-page FAP files

- and pre-compiled MET files 285
- creating multiple FRM files 288
- limitations 260

multi-page forms

- and 2-up printing 69

MVS

- archive/retrieval scenarios 386

MVS file format 465

N

NAFILE.DAT file 464, 481

- and the WriteNAFile rule 54
- rotated variable fields 260

NamedColors option 249, 251

NameDocBy INI option 397

NameDocBy option 398

negative left offset 259

New option

- fonts 214

NEWTRAN.DAT file 482

NEWTRN file

Restart option 394

NEWTRN.DAT file

and the WriteNAFile rule 54

next/retrieve cursor 424

NoBatchSupport option 321

NoGenTrnTransactionProc rule 53

and the WriteNAFile rule 54

mapping fields 50

non-stapled forms

and stapled forms 295

NOT conditions

in search masks 165

NUBACK statements 269

NUFRONT statements 269

O

objects

negative left offset 259

Occurrence flag 141, 471

occurs clauses 476

Octal strings 265

ODBC

archive/retrieval scenarios 386

multiple connections 411

ODBC_FieldConvert control group 411

ODBC_FileConvert control group 411

offset, negative left 259

OMR marks

and the AddLine rule 72

OnDemand command records 248

OnDemandScript option 248

OPASSWD command line option 394

OpSystem option 465

Opt option 58, 59

Optimize option 273

OR conditions

in search masks 165

Oracle

archive/retrieval scenarios 386, 387

ODBC driver 404

ORDER BY clause 393

OT_Docs table 397, 398

OTextString option 274

OUSERID command line option 394

OutBuff token 376

OutMode option

AFP printers 359

Metacode printers 266

Mobius 281

print streams for Docusave 358

output files

for the GenArc program (Docusave) 24

for the GenPrint program (multi-step processing) 24

for the GenWIP program (multi-step processing) 24

OutputBin option 292, 295

OutputFunc option 298

OutputHalfRes option 250

OutputMod option 298

overflow

and class recipients 65

defined 476

XML files 121

Overflow flag 141, 471

OverlayExt option

GDI driver 313

PCL printers 290

PostScript printers 302

OverlayPath option

GDI driver 313

PCL printers 290, 300

PostScript printers 302, 310

overlays

AFP resources 258

landscape pages 259

multi-page FAP files 260

OVLCOMP utility

and PCL resources 300

and PostScript resources 310

P

page segments 258

page-at-a-time downloading 320

PageBatchStage1InitTerm rule 53

PageNumbers option

AFP printers 247

GDI driver 313

PCL printers 290

PostScript printers 302

PageRange option 52

and 2-up printing 70

pages

adding and removing 96

jogging 270

numbering 313

starting new pages 269

total 37

paper size

overriding commands 296

paper sizes

changing on Metacode printers 282

paper trays

Metacode printers 274

on HP 5si printers 294

PCL support for 289

switching 286

PaperSize option 248, 273

PaperStockID option 279

parent 123

parentheses

in search masks 165

pass-through printing 318, 319

Passwd option 408

PCL

custom fonts 196

fonts 189

simple color mode 297

PCL printers

adding PJI comments 299

bitmap fonts 300

compressed PCL 298

handling multiple trays 363

INI options 289

mixing simplex and duplex 298

overlays 300

PCL version 5, 5c, and 5e 289

PCL version 6 292

resources 300

setting up 289

simple color mode 297

using a staple attachment 295

PCO interface

OutMode option 266

PDF

incompatibilities 194

PDF files

creating 320

fonts 208

PDF format 2

PDF417 fonts 193

PDS members

caching 276

performance

caching PDS members 276

reducing job throughput 372

SplitPercent option 261

PFM files

generating 241

PJLComment option 299

PJLCommentOn option 291

PJLCommentScript option 291, 299

platforms

multiple INI files 112

PMetLib option

and the CompileInStream option 267

Metacode printers 277

- PMETLIB PDS 267
- PO Handler 397
- PODocument2Field control group 399
- POField2Document control group 399
- point sizes 218
- POLFILE.DAT file 481
 - and the WriteNAFile rule 54
- Port option 316
- portrait graphic
 - orientation 284
- PostScript
 - custom fonts 196
 - fonts 176, 189
 - inserting fonts 235
- PostScript fonts
 - included with Acrobat Reader 208
- PostScript printers
 - handling multiple trays 363
 - INI options 301
 - PPD files 302, 310
 - resources 310
 - setting up 301
 - Type 1 fonts 310
- PreLoadRequired option 321
- PrePrintedPaper option
 - AFP printers 247
 - GDI driver 314
 - PCL printers 291
 - PostScript printers 303
- print 321
- print batches
 - banner processing 15
 - grouping 103
- Print Services Facility 246
- print spool files
 - creating (multi-step processing) 27
- print streams
 - splitting recipient batch 79
- Print window
 - and the Device field (GDI printing) 318, 319
 - and the PrePrintedPaper option 303
 - and the PrePrintedPaper option (AFP) 247
 - and the PrePrintedPaper option (GDI) 314
 - and the PrePrintedPaper option (PCL) 291
 - and the PrePrintedPaper option (PostScript) 303
 - and the SelectRecipients option 291, 314
 - and the SendColor option 290, 302, 313
 - suppressing 314
- Print_Batches control group 103
 - banner forms 16
- printer console messages 274
- printer drivers
 - banner processing 15
- Printer Job Language (PCL) comments 299
- Printer option
 - and 2-up printing 69, 70
- Printer Resident field 287
- PrinterInk option
 - and the ColorCharts option 274
 - spot colors 271
 - troubleshooting 284
- PrinterModel option 310
 - Metacode printers 274
 - PostScript printers 302, 310
- printers
 - adding fonts to the FXR file 189
 - AFP fonts 188
 - configuring trays 363
 - default printer 314
 - determining characters used in a printer font 182
 - Metacode fonts 188
 - PCL bitmap fonts 189
 - PostScript fonts 189
 - using custom fonts 195
 - using printer fonts 188
- PrintFormset rule 54, 120
 - and the NoGenTranTransactionProc rule 53
 - splitting recipient batch print streams 79

PrintFunc option

- AFP printers 246
- GDI driver 313
- PCL printers 289
- PostScript printers 301
- RTF driver 343

printing

- 2-up 68
- PrintFormset rule 54
- under Windows NT 293

PrintTimeStamp option 105, 372

PrintToFile option 316

PrintViewOnly option

- AFP printers 247, 291
- GDI driver 314
- Metacode printers 276
- PostScript printers 303

ProcessID built-in INI function 114

processing

- transactions (multi-step processing) 23
- transactions (single-step processing) 45

processing overview 11

ProcessQueue rule 54

proportional fonts 219

PRTLIB data 53

PrtType option 85

- RTF driver 343

PrtType:AFP control group 246

PrtType:XER control group

- installable functions 268
- required options 263

PRTZCompressOutPutFunc function 298

Q

Qualifier option 408

queues

- ProcessQueue rule 54

R

RCBDFDFL.DAT file

- and 2-up printing 71

RCBDFDFL.DFD file 473

- and the WriteRCBWithPageCount rule 54
- grouping print batches 103

RCBStatDtIDFD option 38

RCBStats option 38

RCBStatsDtl option 38

RCBStatsTot option 38

RCBStatsTotDFD option 38

RCBTotals option 38

RecipBatch function 19

RecipFunc option

- RTF driver 343

recipient batch (RCB) transaction fields 87

recipient batch DFD file

- and 2-up printing 71

recipient batch file 85, 473

recipient batch records

- PageBatchStage1InitTerm rule 53
- unique data 58

Recipient copy count field 141

Recipient list field 141

Recipient option

- and email aliases 345
- RTF driver 343

recipients

- class recipients 65
- Copy Count field 472
- key files 139
- List field 471
- mapping information 113
- selecting 138
- setting 137

RECIPIF rule

- and extract files 477
- and performance 163
- example 157

- RecipMap2GVM control group 58
- RecipMap2GVM INI control group 66
- RecipMod option
 - RTF driver 343
- RecipName function 19
- records
 - maximum number (Metacode) 284
- Records per first image field 141, 471
- Records per overflow image field 141, 471
- RecordType option 89
- REL112.FXR 193
- REL112SM.FXR 193
- REL95.FXR file 191
- REL95SM.FXR file 191
- RelativeScan option 268
- repeat counts 265
- ReplaceBitmap option 249, 251
- Req option 58, 59
- requirements definition 132
- reserved message ranges 375
- resetting
 - fonts 216
- Resolution option
 - AFP printers 246
 - GDI driver 313
 - Metacode printers 274
 - PCL printers 289
 - PostScript printers 301
 - rounding errors 261
- resource files 467
- resources
 - for single-step processing 46
- Restart control group 33
- restart file 33
- Restart option 394, 395
- Restart table
 - defined 385
- RestartJob rule 34
- RetainTransBeginInitForm option 17, 18
- Retrieval
 - options 417

- Retrieval Options window 417
- Retrieve Document window 416
- RightFax 299
- rollback
 - defined 384
 - restarting GenArc 396
- rotated variable fields 260
- rounding errors
 - SplitPercent option 260
- RP Struct 374
- RPAGE command 279
- RPErrProc function 374
- RPLogProc function 374
- RSTACK command 279
- RstFile option 33
- RTF
 - margins 323
 - print driver 321, 342
 - separate files 321
 - WriteFrames option 322
- RTF files
 - mapping fonts 244
- RTFFontMAP control group 244
- RULCheckTransaction rule 33
- RuleFilePool option 277
- rules
 - for 2-up printing 72
 - for single-step processing 52
 - listing those executed 105
 - order for 2-up printing 74
 - used in multi-step processing 32
- rules processing
 - using international characters 203
- Rules Processor
 - trace file 373
- Rules Publishing Solution
 - system overview 3
- RULStandardProc rule
 - and the WriteNAFile rule 54

Run Length Encoding (RLE) compression 305
RunMode control group
 checking font IDs 102
 DownloadFAP option 99
 grouping print batches 103
 mapping fields with XPath 50
RunSetRcpTbl rule
 and the BuildMasterFormList rule 52

S

scalable fonts 175
scaling output 311
screen fonts
 choosing 238
 GDI drivers 312
 installing in Windows 188
 using 187
Search Mask 1 field 141, 471
Search Mask 2 field 141, 472
search masks
 and recipients 162
 example 155
 formatting 164
 RECIPIF rule 157
section level triggers 139, 144
sections
 marking subordinate sections 149
 master and subordinate 149
 tokens 376, 381
 triggering in XML files 121, 122
selecting
 fonts 212
SelectRecipients option
 GDI driver 314
 PCL printers 291
 PostScript printers 303

self 123
SendColor option 251
 AFP printers 249
 and the ColorCharts option 274
 and the PrinterInk option 271
 emailing forms 345
 GDI driver 313
 PCL printers 290
 PostScript printers 302
 troubleshooting 283
SendOverlays option
 AFP printers 246
 GDI driver 313
 PCL printers 289, 300
 PostScript printers 301, 310
sequence numbers
 and extract files 476
Server option 408, 411
set recipient table
 and performance 163
SetDeviceName function 81
SetOrigin rule
 floating sections 259
SetOverprint option 304
SETRCPTB.DAT file 470
 and the StandardFieldProc rule 54
 and the StandardImageProc rule 54
 examples 151
SETRECIP table
 defined 139
 specifying 142
SetState rule
 and extract files 477
setting
 fonts 171
setting up
 error messages and log files 367
 message text 377
 printers 245
 recipients and copy counts 137
 transaction trigger tables 142
Setup Data field
 example 199

- short binding 469
- Show_Debug_Options option 101
- sibling 123
- SIDE statements 269
- simple color mode 290, 297
- simplex
 - adding and removing pages 96
 - and duplex on Metacode printers 269
 - compressed PCL files 298
 - switching modes 276
- single-page forms
 - and 2-up printing 68
- singles-step processing
 - example 55
- single-step processing
 - clearing messages 370
 - overview 45
 - WriteOutput rule 54
- SkipChartColorChange option 249
- skipping batch message 72
- SortFormsForRecip callback function 166
- sorting records 393
- SplitPercent option
 - 240 dpi print problems 260
 - defined 247
- SplitText option
 - 240 dpi print problems 260
 - defined 247
- SQL Server
 - archive/retrieval scenarios 386
- SQLID command line option 394
- StandardFieldProc rule 54
 - and the WriteNAFile rule 54
- StandardImageProc rule 54
- StandardJobProc rule 67
- staple attachments
 - and PCL printers 295
- StapleBin option 291, 295
- StapleJDEName option 275
- StapleOff option 303, 308
- StapleOn option 303, 308
- stapling forms
 - Metacode 275
 - PostScript 308
- start new page 269
- statistics processing 38
- Status column
 - restarting GenArc 395, 396
- StatusCode option 89
- STOPREC command line option 394
- SUB INK commands 283
- subject information 342
- Subject option
 - RTF driver 343
- subordinate flags
 - and performance 163
- subordinate sections
 - marking 149
 - overview 149
- SuppressBanner function 19
- SuppressDialog option 317
 - and the SuppressDlg option 315
- SuppressDlg option
 - and the SuppressDialog option 315
 - GDI print driver 314
- SuppressLogoUnload option 249
- SuppressZeroData option
 - AFP printers 250
 - and the MultiLinesPerCommand option 250
- Sybase
 - archive/retrieval scenarios 386
- system files 461
- system implementation methodology 132
- system overview 11
- system resource files
 - uploading 204
- system scenarios
 - GenArc 386
- system settings
 - multi-step processing 46

T

table names 433

tables

defined 384

Tag Logical Element (TLE) records 362

TblLkUp rule

and extract files 477

TEMPIDX file

defined 384

TemplateFields option

GDI driver 313

PCL printers 290

PostScript printers 302

TermFunc option 300

RTF driver 343

terminology

fonts 172

testing

the system 132

text files

converting from one code page to another 204

TEXTCommentOn option 300

TEXTScript option 299

TicketJobProc rule 67

tildes

in search masks 164

TL/DL buffers 283

TLEEveryPage option 248, 362

TLEScript option 248, 362

TLESeparator option 248, 362

token-data pairs 376, 378, 380

trace files

ProcessID built-in INI function 114

TraceFile option 373

transaction codes 471

example 154

Transaction codes field 141

transaction files 466, 480

transaction records

creating for multi-step processing 21

creating for single-step processing 45

transaction trigger table

defined 139

examples 151

how it works 143

specifying 142

summary 168

TransactionErrors option 34

GenTrn processing 43

transactions

archiving 31

log of archived 392

logging 104

processing (multi-step processing) 23

TransBannerBeginForm option 16

TransBannerBeginScript option 16, 97

TransBannerEndForm option 16

TransBannerEndScript option 16

transferring files

from Xerox format disks 288

TRANSLAT utility 368, 372

TRANSLAT.INI file

defining the output message file 371

determining where messages originate 379

formatting messages 378

message numbers 375

message tokens 376

setting up message text 377

translating messages 372

TranslationFile option 370

trays

configuring printer trays 363

for the HP 5SI printer 294

Metacode printers 274

overriding commands 296

selecting 365

troubleshooting 286

trigger levels

defined 139

trigger records

levels 139

- Trigger Table Record Format 140
- Trigger2Archive control group 51, 53, 414
- Trigger2WIP control group 87, 120
- triggering logic 168
- triggers
 - and performance 163
 - form level 146
 - section level 144
- TrimWhiteSpace option
 - AFP printers 250
- TRN files 466, 480
- Trn_Fields control group 50
- TRNDFDFL.DFD file 473
- true/false search mask 155
- TrueType fonts 176
 - Asian languages 293
 - description 189
 - inserting 235
- TWOUP control group 52, 53
- TwoUp control group 70
- TwoUpStart option 70

U

- Unicode 292
- unique data
 - adding 58
- UNIQUE.DBF file 484
- UniqueString function 82
- UniqueTag option 407, 408
- UNIX
 - archive/retrieval scenarios 387
- updated log and error files 482, 483
- UpdatePOLFile rule
 - and the WriteOutput rule 54

- uppercase 433
- UseRestartTable option 398
- UserID option 89, 113, 408
- UseXMLeXtract rule 121
- using
 - ANSI code page for PC platforms 201
 - custom fonts 194
 - Font Manager 211
 - printer fonts 188
 - screen fonts 187

V

- value-added processes 276
- variable fields
 - in text areas 260
 - rotated 260
- VB datasets 287
- VBPrOptions control group 314
- ViewDirect APIs 281
- Virtual Storage Access Method 276
- VSAM control group 276

W

- white outlines 304
- white space
 - suppressing 250
- Windows
 - archive/retrieval scenarios 386
 - font substitution 187
 - installing screen fonts 188
 - PostScript printers 304
 - printer ports 293
 - selecting screen fonts 238
 - using the ANSI code page 201
- WIP
 - and the archive index file 413
 - transaction fields 120

- WIP Edit plug-in
 - WIPField built-in function 115
- WIP RecType field 89
- WIP StatusCD field 89
- WIP.DBF file 484
- WIP.DFD files 87
- WIP.MDX file 484
- WordDateFormats control group 322
- WordTimeFormats control group 322
- WriteFrames option 322
- WriteNAFile rule
 - and the StandardFieldProc rule 54
 - described 54
- WriteOutput rule 54

X

- Xbase 499
 - archive/retrieval scenarios 386
 - DFD files 465, 472
 - maximum length 478
- XDPS 425
- XERDNLD utility 288
- XERLoadDocuMerge loader function 281
- Xerox
 - 3700 printers 274
 - 4000 printers 263
 - 4050 printers 286
 - 4135 printers 286
 - 4235 printers 266, 282, 286
 - 4635 printers 286
 - 4850 printers 286
 - 9000 printers 263
 - 9700 printers 283
 - 9790 printers 283, 286
 - fonts 282
 - format floppies 288
 - forms 282, 287
 - forms and memory 283
 - highlight color printers 271
 - images 282
 - JSL INI options 263
 - Laser Printing Systems 263
 - line drawing font 286
 - logos 282
 - setting up Metacode printers 263
 - using custom fonts 196
- XML 50
 - job tickets 67
 - path locator 123
- XML files
 - as extract files 121
- XML print driver 120
- XMLExtract option 50
- XMLFileExtract rule 121
- XMLTrnFields option 50
- XPath 123
 - mapping fields 50
- XPATHW32 utility 123, 126
- XRF files
 - generating 241
 - inserting 234

Y

Year 2000 compliance
and extract files 477

Z

z/OS
generating PostScript output 305

