

JDF Specification Draft Spiral 4.0



Copyright © 2000 by Adobe Systems Inc., AGFA-Gevaert N.V., Heidelberger Druckmaschinen AG, MAN Roland Druckmaschinen AG,
All Rights Reserved

Under the terms of this agreement the user shall be allowed to use the manual as well as the software described therein.

Hereby Adobe Systems Incorporated, Agfa, Heidelberger Druckmaschinen, and MAN Roland decline any warranty for any mistake in the manual and in the software described therein as well as for the correction of such mistake and for any computer virus that might occur in the manual and/or in the software described therein and for any other components. The corporations mentioned before shall not be held liable for any violation of a right of any third party which might be caused through the manual or the software described therein. Hereby any claim for damages shall be expressly excluded. This shall also apply to damages caused to any other object and to damages that might result from the loss of use, information, or profit.

Any references to company names in sample templates are for demonstration purposes only and are not intended to refer to any actual organization.

Adobe, the Adobe logo, Acrobat, Adobe, FrameMaker, Illustrator, InDesign, PageMaker, Photoshop, and PostScript are trademarks of Adobe Systems Incorporated.

Apple and Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries.

Microsoft, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

CIP3 is a Trademark of the International Cooperation for Integration of Prepress, Press, and Postpress.

CIP3 c/o Fraunhofer Institute for Computer Graphics, Rundeturmstrasse 6, 64283 Darmstadt, Germany

All other trademarks are the property of their respective owners.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA

Agfa Gevaert N.V., Septestraat 27, B2640 Mortsel, Belgium

Heidelberger Druckmaschinen Aktiengesellschaft, Kurfürsten-Anlage 52-60, 69115 Heidelberg, Germany

MAN Roland Druckmaschinen AG, Mühlheimer Straße 341, D-63075 Offenbach am Main, Germany

Disclaimer

This document is provided to the public as a technology preview. It is non-normative and subject to change. Although the Working Group does not anticipate further major changes to the functionality described here, this is still a working draft, subject to change. The JDF Working Group will not allow early implementation to constrain its ability to make changes to this specification prior to final release.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION	21
1.1 Document References.....	21
1.2 Conventions Used in This Specification	22
1.2.1 Text Styles.....	22
1.2.2 Specification of Cardinality	23
1.3 Terminology	23
1.4 Data Structures	24
1.5 Units	26
CHAPTER 2 OVERVIEW OF JDF	27
2.1 System Components	27
2.1.1 Job Components	27
2.1.1.1 Jobs and Nodes.....	27
2.1.1.2 Elements.....	27
2.1.1.3 Attributes.....	28
2.1.1.4 Relationships	28
2.1.1.5 Links	28
2.1.2 Workflow Component Roles	28
2.1.2.1 Machines	29
2.1.2.2 Devices	29
2.1.2.3 Agents.....	29
2.1.2.4 Controllers	29
2.1.2.5 Management Information Systems—MIS.....	29
2.1.2.6 System Interaction	30
2.2 JDF Workflow.....	31
2.2.1 Job Structure	32
2.3 Hierarchical Tree Structure and Networks in JDF	33
2.4 Role of Messaging in JDF	35
CHAPTER 3 STRUCTURE OF JDF NODES AND JOBS.....	36
3.1 JDF nodes.....	38
3.1.1 Generic Contents of JDF Elements	38
3.1.2 Fundamental JDF Attributes and Elements.....	39
3.2 Common Node Types	42
3.2.1 Product Intent Nodes.....	43
3.2.2 Process Group Nodes	43
3.2.3 Combined Process Nodes.....	44
3.2.4 Process Nodes	45
3.3 AncestorPool.....	45
3.4 Customer Information.....	46
3.5 Process and Node Information.....	47
3.6 Resources	48
3.6.1 Resource Classes	50
3.6.1.1 Parameter Resources.....	50
3.6.1.2 Intent Resources.....	51

3.6.1.3	Implementation Resources	51
3.6.1.4	Physical Resources (Consumable, Quantity, Handling)	51
3.6.1.5	Placeholder Resources	52
3.6.1.6	Selector Resources	52
3.6.2	Position of Resources within JDF Nodes	53
3.6.3	Pipe Resources	53
3.7	Resource Links	53
3.7.1	Links to Parameter Resources	57
3.7.2	Links to Implementation Resources	57
3.7.3	Links to Physical Resources	58
3.7.4	Links to Placeholder Resources	58
3.7.5	Links to Selector Resources	59
3.7.6	Links to Intent Resources	59
3.7.7	Inter-Resource Linking	59
3.8	Subsets of Resources	59
3.8.1	Resource Amount	60
3.8.2	Description of Partitionable Resources	60
3.8.3	Locations of Physical Resources	63
3.8.4	RunIndex	64
3.8.5	Linking to Subsets of Resources	64
3.8.6	Splitting and Combining Resources	65
3.9	AuditPool	67
3.9.1	Audit Elements	69
3.9.1.1	ProcessRun	69
3.9.1.2	Notification	70
3.9.1.3	PhaseTime	70
3.9.1.4	ResourceAudit	72
3.9.1.5	Logging Machine Data by Using the ResourceAudit	73
3.9.1.6	Created	74
3.9.1.7	Modified	74
3.9.1.8	Spawned	74
3.9.1.9	Merged	75
3.10	JDF Extensibility	76
3.10.1	Namespaces in XML	76
3.10.2	Extending Process Types	76
3.10.3	Extending existing Resources	77
3.10.4	Creating New Resources	77
3.10.5	Future JDF Extensions	77
3.10.6	Maintaining Extensions	77
3.10.7	Processing Unknown Extensions	78
3.10.8	Derivation of Types in XMLSchema	78
CHAPTER 4	LIFE CYCLE OF JDF	79
4.1	Creation and Modification	79
4.1.1	Product Intent Constructs	79
4.1.1.1	Representation of Product Intent	80
4.1.1.2	Representation of Product Binding	80
4.1.2	Quote Generation Using Intent Resources	80
4.1.3	Specification of Delivery of End Products	82
4.2	Process Routing	84
4.2.1	Determining Executable Nodes	84

4.2.2	Distributing Processing to Work Centers or Devices	84
4.2.3	Device / Controller Selection	85
4.3	Execution Model	85
4.3.1	Serial Processing	85
4.3.2	Overlapping Processing Using Pipes	87
4.3.2.1	Pipes of Partionable Resources	89
4.3.2.2	Dynamic Pipes	89
4.3.2.3	Comparison of Non-Dynamic and Dynamic Pipes	90
4.3.3	Parallel Processing	90
4.3.4	Iterative Processing	91
4.3.4.1	Informal Iterative Processing	92
4.3.4.2	Formal Iterative Processing	92
4.3.5	Proofing and Verification	92
4.4	Spawning and Merging	92
4.4.1	Case 1: Standard Spawning and Merging	94
4.4.2	Case 2: Spawning and Merging with resource copying	95
4.4.3	Case 3: Parallel Spawning and Merging of Partitioned Resources	96
4.4.4	Case 4: Nested Spawning and Merging in Reverse Sequence	96
	Case 5: Spawning and Merging of Independent Jobs	98
4.4.6	Simultaneous Spawning and Merging of Multiple nodes	99
4.5	Node and Resource IDs	99
4.6	Error Handling	100
4.6.1	Classification of Notifications	100
4.6.2	Event Description	101
4.6.3	Error Logging in the JDF file	101
4.6.4	Error Handling via Messaging (JMF)	101
4.7	Test Running	101
4.7.1	Resource Status During Testrun	102
 CHAPTER 5 JDF MESSAGING WITH THE JOB MESSAGING FORMAT (JMF)		103
5.1	JMF Root	103
5.2	JMF Semantics	104
5.2.1	Message Families	105
5.2.1.1	Query	105
5.2.1.2	Response	105
5.2.1.3	Signal	106
5.2.1.4	Command	108
5.2.1.5	Acknowledge	108
5.2.2	JMF Handshaking	109
5.2.2.1	Single Query/Command Response Communication	109
5.2.2.2	Signal	109
5.2.2.3	Persistent Channels	109
5.3	JMF Messaging Levels	110
5.4	Error and Event Messages	111
5.5	Standard Messages	112
5.5.1	Controller Registration and Communication Messages	113
5.5.1.1	Events	113
5.5.1.2	KnownControllers	114
5.5.1.3	KnownDevices	114
5.5.1.4	KnownJDFServices	115
5.5.1.5	KnownMessages	116

5.5.1.6 RepeatMessages.....	117
5.5.1.7 StopPersistentChannel.....	118
5.5.2 Device/Operator Status and Job Progress Messages.....	119
5.5.2.1 Occupation.....	120
5.5.2.2 Resource.....	121
5.5.2.3 Status.....	125
5.5.2.4 Track.....	130
5.5.3 Pipe Control.....	131
5.5.3.1 PipeClose.....	131
5.5.3.2 PipePull.....	132
5.5.3.3 PipePush.....	134
5.5.3.4 PipePause.....	134
5.6 Queue Support.....	135
5.6.1 Queue Entry ID Generation.....	135
5.6.2 Queue Entry Handling Commands.....	135
5.6.2.1 AbortQueueEntry.....	136
5.6.2.2 HoldQueueEntry.....	136
5.6.2.3 RemoveQueueEntry.....	137
5.6.2.4 ResubmitQueueEntry.....	137
5.6.2.5 ResumeQueueEntry.....	137
5.6.2.6 SetQueueEntryPosition.....	138
5.6.2.7 SetQueueEntryPriority.....	138
5.6.2.8 SubmitQueueEntry.....	139
5.6.3 Global Queue Handling.....	141
5.6.3.1 CloseQueue.....	141
5.6.3.2 FlushQueue.....	141
5.6.3.3 HoldQueue.....	142
5.6.3.4 OpenQueue.....	142
5.6.3.5 QueueEntryStatus.....	142
5.6.3.6 QueueStatus.....	143
5.6.3.7 ResumeQueue.....	143
5.6.3.8 SubmissionMethods.....	143
5.6.4 Queue-Handling Elements.....	144
5.7 Extending Messages.....	146
5.7.1 IFRATrack Support.....	146

CHAPTER 6 PROCESSES..... 147

6.1 Process Template.....	147
6.2 General Processes.....	147
6.2.1 Approval.....	147
6.2.2 Combine.....	148
6.2.3 Delivery.....	148
6.2.4 Ordering.....	149
6.2.5 ResourceDefinition.....	149
6.2.6 Split.....	150
6.2.7 Verification.....	150
6.3 Prepress Processes.....	150
6.3.1 Scanning.....	150
6.3.2 LayoutElementProduction.....	151
6.3.3 DBDocTemplateLayout.....	151
6.3.4 DBTemplateMerging.....	152
6.3.5 ColorSpaceConversion.....	152

6.3.6	ColorCorrection	153
6.3.7	Preflight	154
6.3.8	ImageReplacement	155
6.3.9	Separation	155
6.3.10	Trapping	156
6.3.11	Imposition	157
6.3.12	PDFToPSConversion	158
6.3.13	PSToPDFConversion	158
6.3.14	RIPping	159
6.3.15	Interpreting	159
6.3.16	Rendering	160
6.3.17	ContoneCalibration	161
6.3.18	Screening	161
6.3.19	SoftProofing	162
6.3.20	Proofing	163
6.3.21	PreviewGeneration	164
6.3.22	InkZoneCalculation	166
6.3.23	Tiling	167
6.3.24	ImageSetting	167
6.3.25	FilmToPlateCopying	168
6.4	Press Processes	168
6.4.1	ConventionalPrinting	168
6.4.2	DigitalPrinting	170
6.4.3	IDPrinting	171
6.5	Postpress Processes	172
6.5.1	Web Processes	172
6.5.1.1	Dividing	172
6.5.1.2	LongitudinalRibbonOperations	173
6.5.2	HoleMaking	173
6.5.3	Tip-on/in	174
6.5.3.1	EndSheetGluing	174
6.5.3.2	Inserting	174
6.5.4	Block Production	175
6.5.4.1	Block Compiling	175
6.5.4.1.1	Collecting	175
6.5.4.1.2	Gathering	176
6.5.4.2	Block Joining	176
6.5.4.2.1	AdhesiveBinding	176
6.5.4.2.2	SaddleStitching	177
6.5.4.2.3	SideSewing	177
6.5.4.2.4	Stitching	177
6.5.4.2.5	ThreadSewing	178
6.5.4.2.6	Single Leaf Binding Methods	178
6.5.4.2.6.1	Loose Leaf Binding Method	178
6.5.4.2.6.1.1	RingBinding	179
6.5.4.2.6.2	Mechanical Binding Methods	179
6.5.4.2.6.2.1	ChannelBinding	179
6.5.4.2.6.2.2	CoilBinding	180
6.5.4.2.6.2.3	PlasticCombBinding	180
6.5.4.2.6.2.4	VeloBinding	181
6.5.4.2.6.2.5	WireCombBinding	181
6.5.5	Numbering	181
6.5.6	Sheet Processes	182
6.5.6.1	Cutting	182

6.5.6.2	Folding	183
6.5.7	Trimming	183
CHAPTER 7	RESOURCES	185
7.1	Intent Resources	185
7.1.1	Span Resource Sub-elements	185
7.1.1.1	Structure of Abstract Span Elements	186
7.1.1.2	Structure of the Span-Element Type IntegerSpan	187
7.1.1.3	Structure of the Span-Element Type NameSpan	187
7.1.1.6	Structure of the Span-Element Type StringSpan	189
7.1.1.7	Structure of the TimeSpan Sub-element	190
7.1.2	Named Span resources	190
7.1.3	ArtDeliveryIntent	191
7.1.4	BindingIntent	192
7.1.5	ColorIntent	195
7.1.6	DeliveryIntent	196
7.1.7	FoldingIntent	197
7.1.8	HoleMakingIntent	198
7.1.9	InsertingIntent	198
7.1.10	LaminatingIntent	199
7.1.11	MediaIntent	200
7.1.12	Numbering Intent	201
7.1.13	PackingIntent	201
7.1.14	PocketingIntent	202
7.1.15	ProofingIntent	202
7.1.16	ScanningIntent	203
7.1.17	ScreeningIntent	204
7.1.18	ShapelIntent	205
7.1.19	SizeIntent	206
7.1.20	StampingIntent	206
7.2	Process Resources	207
7.2.1	Process Resource Template	207
7.2.2	Address	208
7.2.3	AdhesiveBindingParams	208
7.2.4	ApprovalParams	213
7.2.5	ApprovalSuccess	214
7.2.6	ByteMap	214
7.2.7	ChannelBindingParams	215
7.2.8	CIELABMeasuringField	216
7.2.9	CoilBindingParams	217
7.2.10	CollectingParams	218
7.2.11	Color	218
7.2.12	ColorantControl	221
7.2.13	ColorControlStrip	223
7.2.14	ColorCorrectionParams	224
7.2.15	ColorPool	225
7.2.16	ColorSpaceConversionParams	225
7.2.17	ComChannel	227
7.2.18	Company	228
7.2.19	Component	228
7.2.20	Contact	233

7.2.21	ConventionalPrintingParams	233
7.2.22	CostCenter	236
7.2.23	CutBlock.....	236
7.2.24	CutMark.....	237
7.2.25	DBMergeParams.....	238
7.2.26	DBRules	239
7.2.27	DBSchema	239
7.2.28	DBSelection.....	240
7.2.29	DeliveryParams	240
7.2.30	DensityMeasuringField	241
7.2.31	Device	242
7.2.32	DigitalPrintingParams	242
7.2.33	Disjointing.....	243
7.2.34	DividingParams	244
7.2.35	Employee	245
7.2.36	EndSheetGluingParams	245
7.2.37	ExposedMedia.....	246
7.2.38	FileSpec	247
7.2.39	FoldingParams	250
7.2.40	FontParams.....	256
7.2.41	FontPolicy	257
7.2.42	GatheringParams	258
7.2.43	GlueLine.....	258
7.2.44	HoleMakingParams	259
7.2.45	IdentificationField.....	260
7.2.46	IDPrintingParams	261
	Resource Properties.....	261
	Resource Structure.....	261
7.2.47	ImageCompressionParams	268
7.2.48	ImageReplacementParams	271
7.2.49	ImageSetterParams.....	272
7.2.50	Ink	273
7.2.51	InkZoneCalculationParams.....	274
7.2.52	InkZoneProfile	274
7.2.53	InsertingParams	275
7.2.54	InsertSheet.....	276
7.2.55	InterpretedPDLData.....	277
7.2.56	InterpretingParams	277
7.2.57	Layout	278
7.2.58	LayoutElement	279
7.2.59	LongitudinalRibbonOperationParams	280
7.2.60	Media	282
7.2.61	NumberingParams.....	283
7.2.62	OrderingParams	284
7.2.63	PDFToPSConversionParams	284
7.2.64	PDLResourceAlias	288
7.2.65	Person.....	288
7.2.66	PlaceHolderResource.....	289
7.2.67	PlasticCombBindingParams	289
7.2.68	PlateCopyParams.....	290
7.2.69	PreflightAnalysis.....	290

7.2.70	PreflightInventory.....	292
7.2.71	PreflightProfile	293
7.2.72	Preview	294
7.2.73	PreviewGenerationParams.....	295
7.2.74	ProofingParams.....	296
7.2.75	PSToPDFConversionParams	296
7.2.76	RegisterMark	300
7.2.77	RenderingParams.....	300
7.2.78	RingBindingParams.....	301
7.2.79	RunList.....	302
7.2.80	SaddleStitchingParams	305
7.2.81	ScanParams.....	306
7.2.82	ScreeningParams.....	308
7.2.83	SeparationControlParams	309
7.2.84	SeparationSpec.....	311
7.2.85	Sheet.....	311
7.2.86	SideSewingParams	312
7.2.87	StitchingParams	313
7.2.88	Surface.....	315
7.2.89	ThreadSewingParams	318
7.2.90	Tile	319
7.2.91	TransferCurvePool	320
7.2.92	TrappingDetails	321
7.2.93	TrappingParams.....	322
7.2.94	TrapRegion.....	326
7.2.95	TrimmingParams	326
7.2.96	VeloBindingParams	327
7.2.97	VerificationParams	328
7.2.98	WireCombBindingParams	329
CHAPTER 8 BUILDING A SYSTEM AROUND JDF		330
8.1	Implementation Considerations and Guidelines	330
8.2	JDF and JMF Interchange Protocol.....	330
8.2.1	File-Based Protocol (JDF only).....	330
8.2.2	HTTP-Based Protocol (JDF + JMF).....	330
8.2.3	Protocol Implementation Details	330
8.2.4	Mime Types and File Extensions.....	331
8.3	MIS Requirements.....	331
APPENDIX A ENCODING		332
A.1	XML Schema Data Types	332
A.2	JDF Data Types	333
A.2.1	CMYKColor	333
A.2.2	IntegerList	333
A.2.3	IntegerRange.....	333
A.2.4	IntegerRangeList.....	334
A.2.5	LabColor.....	334
A.2.6	Matrix	334
A.2.7	NamedColor	335

A.2.8	NameRange	335
A.2.9	NameRangeList.....	336
A.2.10	NumberList.....	336
A.2.11	NumberRange	336
A.2.12	NumberRangeList.....	336
A.2.13	Path.....	337
A.2.14	Rectangle	337
A.2.15	sRGBColor	337
A.2.16	TimeRange.....	338
A.2.17	TransferFunctions.....	338
A.2.18	XYPair	338
A.3	JDF Data Structures.....	339
A.3.1	Links.....	339
A.4	JDF File Formats.....	339
A.4.1	MIME File Packaging.....	339
A.4.1.1	MIME Basics	340
A.4.1.2	MIME Fields	340
A.4.1.3	CID URL scheme	340
A.4.1.4	JDF Agent Requirements.....	341
A.4.2	HTTP 1.0 Field	341
A.4.3	PNG Image Format	341
APPENDIX B SCHEMA		343
B.1	Schema of the JDF-node	343
APPENDIX C CONVERTING PJTF TO JDF		344
C.1	PJTF Object Conversion	344
C.1.1	Accounting	344
C.1.2	Address	344
C.1.3	Analysis.....	344
C.1.4	AuditObject.....	344
C.1.5	ColorantAlias	344
C.1.6	ColorantControl	345
C.1.7	ColorantDetails.....	345
C.1.8	ColorantZoneDetails.....	345
C.1.9	ColorSpaceSubstitute.....	345
C.1.10	Delivery	345
C.1.11	DeviceColorant.....	345
C.1.12	Document.....	345
C.1.13	Finishing.....	346
C.1.14	FontPolicy	347
C.1.15	InsertPage.....	347
C.1.16	InsertSheet.....	347
C.1.17	Inventory	347
C.1.18	JobTicket.....	347
C.1.19	JobTicketContents.....	347
C.1.20	JTFile	349
C.1.21	Layout	349
C.1.22	Media	349
C.1.23	MediaSource	349

C.1.24	MediaUsage	349
C.1.25	PageRange	349
C.1.26	PlacedObject	351
C.1.27	PlaneOrder	351
C.1.28	Preflight	351
C.1.29	PreflightConstraint	351
C.1.30	PreflightDetail	351
C.1.31	PreflightInstance	351
C.1.32	PreflightInstanceDetail	351
C.1.33	PreflightResults	351
C.1.34	PrintLayout	352
C.1.35	Profile	352
C.1.36	Rendering	352
C.1.37	ResourceAlias	352
C.1.38	Scheduling	352
C.1.39	Signature	353
C.1.40	Sheet	353
C.1.41	SlipSheet	353
C.1.42	Surface	353
C.1.43	Tile	353
C.1.44	Trapping	353
C.1.45	TrappingDetails	353
C.1.46	TrappingParameters	353
C.1.47	TrapRegion	353
C.2	Translating Values	354
C.3	Translating the Contents Hierarchy	354
C.4	Representing Pages	355
C.5	Representing Pre-separated Documents	355
C.6	Representing Inherited Characteristics	355
C.7	Translating Layout	355
C.8	Translating PrintLayout	356
C.9	Translating Trapping	356

APPENDIX D CONVERTING PPF TO JDF..... 357

D.1	Converting PPF Data Types	358
D.2	PPF Product Definitions	358
D.2.1	Comparison of the PPF Component to the JDF Component	359
D.2.2	Collecting	360
D.2.3	Gathering	360
D.2.4	ThreadSewing	360
D.2.5	SaddleStitching	360
D.2.6	Stiching	360
D.2.7	SideSewing	360
D.2.8	EndSheetGluing	361
D.2.9	AdhesiveBinding	361
D.2.10	Trimming	362
D.2.11	GluingIn	362
D.2.12	Folding	363
D.3	PPF Sheet Structure	364

D.3.1	Administration Data	365
D.3.2	Preview Images.....	367
D.3.3	Transfer Curves.....	368
D.3.4	Register Marks	368
D.3.5	Color and Ink Control.....	368
D.3.6	Cutting Data	369
D.3.7	Folding Data.....	370
D.3.8	Comments and Annotations	370
D.3.9	Private Data and Content	370
APPENDIX E MODELLING IFRATRACK IN JDF		371
E.1	IFRA Objects and JDF Nodes	371
E.1.1	Object Identification	371
E.1.2	IFRA Object Hierarchy.....	371
E.1.3	Object States.....	371
E.1.4	Deadlines and Scheduling.....	372
E.2	JMF Messages that Translate IFRATRACK Messages	372
E.2.1	JMF Phase Message.....	372
E.2.2	JMF Progress Message.....	372
APPENDIX F STATUSDETAILS SUPPORTED STRINGS		374
APPENDIX G MODULETYPE SUPPORTED STRINGS.....		376
APPENDIX H SUPPORTED ERROR CODES IN JMF.....		377
APPENDIX I EVENT TYPES AND VALUES		378
APPENDIX J EXAMPLES		379
Brief Example.....		379
J.1.1	Before Processing	379
J.1.2	After Processing.....	379
J.2	Product JDF to Process	380
J.3	16-Page 4-up Brochure—Layout, RunList, Cut, Fold	381
J.4	Spawning and Merging.....	381
J.4.1	Example 2 Component JDF before Spawning	382
J.4.2	Example 2 Component JDF Parent after spawning the cover node	383
J.4.3	Example 2 Component JDF spawned node.....	384
J.4.4	Example 2 Component JDF after merging.....	384
J.5	Conversion of PJTF to JDF	385
J.5.1	PJTF input.....	385
J.5.2	JDF output.....	388
J.6	Conversion of PPF to JDF.....	389
J.7	Messages.....	389

TABLE OF FIGURES

Figure 2.1 Example of JDF and JMF workflow interactions	30
Figure 2.2 JDF tree structure	32
Figure 2.3 Example of a hierarchical tree structure of JDF nodes	34
Figure 2.4 Example of a process chain linked by input and output resources	34
Figure 3.1 Structure of the JDF node type	37
Figure 3.2 Structure of JDF Generic Contents	39
Figure 3.3 Job hierarchy with process, process group, and product intent nodes	43
Figure 3.4 Structure of the Abstract Resource Types	50
Figure 3.5 Nodes linked by a resource	54
Figure 3.6 Structure of the abstract ResourceLink types	55
Figure 3.7 Splitting and combining physical resources	66
Figure 3.8 Structure of Audit element types derived from the abstract Audit type	68
Figure 4.1 Example of a simple process chain linked by resources	86
Figure 4.2 Example of a Pipe resource linking two processes	87
Figure 4.3 Example of status transitions in case of overlapping processing	88
Figure 4.4 Spawning and merging mechanism and its phases	93
Figure 4.5 JDF node structure that requires resource copying during spawning and merging	95
Figure 4.6 Example for a JDF node structure with nested spawning	97
Figure 4.7 Example of the spawning and merging of independent jobs	98
Figure 5.1 Contents of a JMF root element and the message families	104
Figure 5.2 Mechanism of a PipePull message	132
Figure 5.3 Mechanism of a PipePush message	134
Figure 6.1 Worst-case scenario for area coverage calculation	165
Figure 7.1 Parameters and coordinate systems for back-preparation process	210
Figure 7.2 Parameters and coordinate system for glue application	211
Figure 7.3 Parameters and coordinate system for the spine-taping process	212

Figure 7.4 Parameters and coordinate system for cover application	213
Figure 7.5 Coordinate Systems Used for Collecting.....	218
Figure 7.6 Terms and Definitions for Components.....	231
Figure 7.7 Cut Mark Types	238
Figure 7.8 Parameters and coordinate system used for end-sheet gluing.....	246
Figure 7.9 Names of the reference edges of a sheet in the FoldingParams resource.....	250
Figure 7.10 FoldCatalog part 1.....	252
Figure 7.11 FoldCatalog part 2.....	253
Figure 7.12 Coordinate system used for gathering	258
Figure 7.13 Parameters and coordinate system used for inserting.....	275
Figure 7.14 Staple shapes.....	306
Figure 7.15 Parameters and coordinate system used for saddle stitching.....	306
Figure 7.16 Parameters and coordinate system used for side sewing.....	312
Figure 7.17 Parameters and coordinate system used for stitching	314
Figure 7.18 Parameters and coordinate system used for thread sewing.....	318
Figure 7.19 Parameters and coordinate system used for trimming.....	327
Figure D.1 JDF node of a CIP3 product structure	358
Figure D.8.2 JDF representation of sheets.....	365

 **TABLE OF TABLES**^[DH1]

Table 3.1	Generic Contents of elements	38
Table 3.2	Contents of the Comment element.....	38
Table 3.3	Contents of a JDF node.....	39
Table 3.4	Contents of the AncestorPool element	45
Table 3.5	Attributes of the Ancestor element.....	46
Table 3.6	Contents of the CustomerInfo element.....	46
Table 3.7	Contents of the NodeInfo element.....	47
Table 3.8	Contents of the ResourcePool element.....	48
Table 3.9	Contents of the abstract Resource element	49
Table 3.10	Additional contents of the abstract physical Resource elements	51
Table 3.11	Contents of the Location element.....	52
Table 3.12	Contents of the ResourceLinkPool element	55
Table 3.13	Contents of the abstract ResourceLink element.....	56
Table 3.14	Contents of the abstract ParameterLink element	57
Table 3.15	Contents of the abstract ImplementationLink element	57
Table 3.16	Additional contents of the abstract physical ResourceLink element	58
Table 3.17	Contents of the partitionable Resource element	61
Table 3.18	Contents of the Part element.....	62
Table 3.19	Contents of the Selector resource	64
Table 3.20	Contents of the AuditPool element	68
Table 3.21	Contents of the abstract Audit type.....	69
Table 3.22	Contents of the ProcessRun element.....	69
Table 3.23	Contents of the Notification element.....	70
Table 3.24	Contents of the PhaseTime element	71
Table 3.25	Contents of the ModulePhase element.....	71
Table 3.26	Contents of the ResourceAudit element.....	73

Table 3.27 Contents of the Created element.....	74
Table 3.28 Contents of the Modified element.....	74
Table 3.29 Contents of the Spawned element.....	75
Table 3.30 Contents of the Merged element	75
Table 4.1 Examples of resource and process states in the case of simple process routing.....	86
Table 4.2 Actions generated when a dynamic-pipe buffer passes various levels	89
Table 5.1 Contents of the JMF root	103
Table 5.2 Contents of the abstract Message element	104
Table 5.3 Contents of the Query message element	105
Table 5.4 Contents of the Response message element.....	105
Table 5.5 Contents of the Signal message element.....	107
Table 5.6 Contents of the Command message element	108
Table 5.7 Contents of the Acknowledge message element	108
Table 5.8 Contents of the Subscription element.....	109
Table 5.9 Messaging table template.....	112
Table 5.10 Process registration and communication messages	113
Table 5.11 Contents of the Events element.....	113
Table 5.12 Contents of the Known Controllers element.....	114
Table 5.13 Contents of the JDFController	114
Table 5.14 Contents of the KnownDevices element.....	114
Table 5.15 Contents of the DeviceFilter element.....	115
Table 5.16 Contents of the KnownJDFServices element	115
Table 5.17 Contents of the JDFService element.....	116
Table 5.18 Contents of the KnownMessages element	116
Table 5.19 Contents of the KnownMsgQuParams element	116
Table 5.20 Contents of the MessageService element.....	117
Table 5.21 Contents of the RepeatMessages element	117
Table 5.22 Contents of the MsgFilter element.....	117

Table 5.23	Contents of the StopPersistentChannel element.....	118
Table 5.24	Contents of the StopPersChParams element.....	119
Table 5.25	Status and progress messages	119
Table 5.26	Contents of the Occupation element	120
Table 5.27	Contents of the EmployeeDef element.....	120
Table 5.28	Contents of the Occupation element	120
Table 5.29	Contents of the Resource query message element	121
Table 5.30	Contents of the ResourceQuParams element.....	121
Table 5.31	Contents of the Resource command message element.....	122
Table 5.32	Contents of the ResourceCmdParams element.....	123
Table 5.33	Contents of the ResourceInfo element	123
Table 5.34	Contents of the Status element	125
Table 5.35	Contents of the StatusQuParams element.....	125
Table 5.36	Contents of the DeviceInfo element.....	126
Table 5.37	Contents of the JobPhase element.....	128
Table 5.38	Contents of the ModuleStatus element.....	129
Table 5.39	Contents of the Track element.....	130
Table 5.40	Contents of the TrackFilter element.....	130
Table 5.41	Contents of the TrackResult element	131
Table 5.42	Dynamic pipe messages.....	131
Table 5.43	Contents of the PipeClose element	131
Table 5.44	Contents of the PipePull element	132
Table 5.45	Contents of the PipeParams element.....	133
Table 5.46	Contents of the PipeCmdResult element	134
Table 5.47	Contents of the PipePush element	134
Table 5.48	Contents of the PipePause element	134
Table 5.49	QueueEntry handling messages.....	135
Table 5.50	Contents of the AbortQueueEntry element.....	136

Table 5.51	Contents of the HoldQueueEntry element.....	136
Table 5.52	Contents of the RemoveQueueEntry element.....	137
Table 5.53	Contents of the ResubmitQueueEntry element.....	137
Table 5.54	Contents of the ResubmissionParams element.....	137
Table 5.55	Contents of the ResumeQueueEntry element.....	137
Table 5.56	Contents of the SetQueueEntry element.....	138
Table 5.57	Contents of the QueueEntryPosParams element.....	138
Table 5.58	Contents of the QueueEntryPriParams element.....	139
Table 5.59	Contents of the SubmitQueueEntry element.....	139
Table 5.60	Contents of the QueueSubmissionParams element.....	139
Table 5.61	Global queue-handling commands.....	141
Table 5.62	Contents of the CloseQueue element.....	141
Table 5.63	Contents of the FlushQueue element.....	141
Table 5.64	Contents of the HoldQueue element.....	142
Table 5.65	Contents of the OpenQueue element.....	142
Table 5.66	Contents of the QueueEntryStatus element.....	142
Table 5.67	Contents of the QueueStatus element.....	143
Table 5.68	Contents of the SubmissionMethods element.....	143
Table 5.69	Contents of the SubmissionMethods element.....	144
Table 5.70	Contents of the Queue element.....	144
Table 5.71	Contents of the QueueEntry element.....	145
Table 5.72	Contents of the QueueEntryDef element.....	145
Table 7.1	Matrices used to change the orientation of a Component.....	231
Table 7.2	Terms and Definitions for Components.....	231
Table 7.3	Predefined variables used in FileTemplate.....	249
Table A.1	Mapping of named colors to sRGB colors.....	335
Table D.2	Conversion of PPF Data Types.....	358
Table D.8.3	JDF Representation of a product definition step.....	359

Table D.8.4	Converting a PPF Component	359
Table D.8.5	Converting the PPF EndSheetGluing operation to JDF	361
Table D.8.6	Converting the PPF AdhesiveBinding operation to JDF	361
Table D.8.7	Converting the PPF AdhesiveBinding sub-operation Lining	361
Table D.8.8	Converting the PPF AdhesiveBinding sub-operation CoverApplication	362
Table D.8.9	Converting the PPF GluingIn operation to JDF	362
Table D.8.10	Converting the PPF Folding operation to JDF	363
Table D.8.11	Converting the PPF Folding sub-operation of type Fold	363
Table D.8.12	Converting the PPF Folding sub-operation of type Lime	363
Table D.8.13	Converting the PPF Folding sub-operation of all other types	364
Table D.8.14	Converting administration data	365
Table D.8.15	PPF preview representation as PNG	367
Table D.8.16	Converting the parameter of the CIP3PlaceRegisterMark command	368
Table D.8.17	Converting PPF color-measuring data	368
Table D.8.18	Converting PPF density-measuring data	369
Table D.8.19	Converting the parameter of the CIP3PlaceColorControlStrip command	369
Table D.8.20	Converting the Cutting Data structure	369
Table D.8.21	Converting the parameter of the CIP3PlaceCutMark command	370
Table E.8.22	Contents of the Phase message	372

Job Definition Format

Chapter 1 Introduction

Welcome to the specification of the Job Definition Format (JDF) and its counterpart, the Job Messaging Format (JMF).

Four companies prominent in the graphic arts industry—Adobe, Agfa, HEIDELBERG, and MAN Roland—have united to create an extensible, XML-based format built upon the existing technologies of CIP3's Print Production Format (PPF) and Adobe's Portable Job Ticket Format (PJTF). JDF provides three primary benefits to the printing industry. Unlike any previous format, it has the ability to unify the pre-press, press, and post-press aspects of any printing job. It also provides the means to bridge the communication gap between production services and Management Information Systems (MIS). And finally, it is able to carry out both of these functions no matter what system architecture is already in place, and no matter what tools are being used to complete the job. In short, JDF is extremely versatile and comprehensive.

JDF is an interchange data format to be used by a system of administrative and implementation-oriented components, which together produce printed products. It provides the means to describe print jobs in terms of the products eventually to be created, as well as in terms of the processes needed to create those products. The format provides a mechanism to explicitly specify the controls needed by each process, which may be specific to the devices that will execute the processes.

JDF works in tandem with a counterpart format known as the Job Messaging Format, or JMF. JMF provides the means for production components of a JDF workflow to communicate with system controllers and administrative components. It relays information about the progress of JDF jobs and gives MIS the active ability to query devices about the status of processes being executed or getting ready to be executed. JMF will provide the complete job tracking functionality that is defined by IFRATrack messaging standard. Depending on the system architecture, JMF may also provide the means to control certain aspects of these processes directly.

This document describes components of JDF, both internal and external, and explains how to integrate the format components to create a viable workflow. Ancillary aspects are also introduced, such as how to convert PJTF or PPF to JDF, and how JDF relates to IFRATrack.

1.1 Document References

This specification assumes that the reader has a basic awareness of, or access to, the following documents:

Portable Job Ticket Format

Version 1.1

Date: 2-April-1999

Produced by Adobe Systems Inc.

Available at; <http://partners.adobe.com/asn/developer/PDFS/TN/5620.pdf>

Print Production Format

Version 3.0

Date: 2-June-1998

Produced by the International Cooperation for Integration of Prepress, Press, and Postpress

Available at: http://www.cip3.org/documents/technical_info/cip3v3_0.pdf

*XML Specification**Version 1.0*

Date: 10-February-1998

Produced by: World Wide Web Consortium (W3C)

Available at: <http://www.w3.org/TR/REC-xml>*XML Schema Part 1+2: Structures and Datatypes*

Version (currently in working draft)

Date: 25-February-2000

Produced by: World Wide Web Consortium (W3C) XML Schema working group

Available at: <http://www.w3.org/TR/xmlschema-1/> and<http://www.w3.org/TR/xmlschema-2/>*IFRATrack Specification*

Version 2.0

Date: June-1998

IFRA Special Report 6.21.2

Produced by IFRA

Available at: <http://www.ifra.com/>*Spec ICC.1:1998-09*

File Format for Color Profiles

Version 3.5

Date: 1998

Produced by: International Color Consortium

Available at: http://www.color.org/ICC-1_1998-09.PDF

1.2 Conventions Used in This Specification

This section contains conventions and notations used within this document.

1.2.1 Text Styles

The following text styles are used to identify the components of a JDF job:

- Elements are written in sans serif. Examples are: **Comment**, **CustomerInfo**, and **ResourceLinks**.
- Attributes are written in italic sans serif. Examples are: *Status*, *ResourceID*, and *ID*.
- Resources are written in bold sans serif. Examples are **ImpositionProof**, **Toner**, and **ExposedMedia**.
- Processes are written in bold-italic sans serif. Examples are ***ColorSpaceConversion***, ***Rendering***, and ***Scanning***.
- Enumerative and boolean values of attributes are written in italics. Examples are: *true*, *waiting*, *completed*, and *stopped*.
- Standard bold text is used for the following purposes:
 - to highlight glossary items. Examples are **device**, **element**, and **job**.
 - to highlight defined items inside a table. An example is the data type **timeDuration** in the table in section 1.4 Data Structures.
 - to highlight definitions of local terms. These are terms that are of local importance for a certain chapter, or some sections inside a chapter. An example is a **spawned job** in section 4.4 Spawning and Merging.
 - to designate PPF objects in Appendix D, Converting PPF to JDF. Examples are **CIP3ProductName** and **CIP3ProductComponent**.

- For the benefit of those who are reading this document in PDF or online, cross-reference links are denoted by gray text. Examples are [Chapter 6 Processes](#), and [section 1.2 Conventions Used in This Specification](#). To follow a link, click the highlighted text. The examples provided are not actual links.
- Also for the benefit of online readers, external hyperlinks are graphically designated. An example is <http://URL.com>. To follow a link, click the highlighted text. The example provided is not an actual link.

1.2.2 Specification of Cardinality

The cardinality of JDF Data Types is expressed using a simple Extended Backus-Naur Form (EBNF) notation. The symbols in this notation may be combined to indicate both simple and complex patterns, as demonstrated in the following table. A and B represent simple expressions.

Notation	Description
(expression)	Expression is treated as a unit and may be combined as described in this list.
A ?	Matches A or nothing. A is optional, or is required only in the circumstances explained in the description field.
A +	Matches one or more occurrences of A.
A *	Matches zero or more occurrences of A.
A B	Matches A followed by B.
A B	Matches A or B but not both.
A - B	Matches any string that matches A but does not match B.

1.3 Terminology

The following terms are used throughout this specification. For more detail, see section 2.1 System Components.

Agent: The component of a JDF-based workflow that writes JDF.

Attribute: An XML-based syntactic construct describing an unstructured characteristic of a JDF node or element.

Big job: The job that independent jobs are merged into in the case of independent spawning and merging.

Class: Denotes a set of complex data types with common content in an object-oriented sense. A complex data type may consist of elements and attributes.

Controller: The component of a JDF-based workflow that initiates devices, routes JDF, and communicates status information.

Device: The component of a JDF workflow part that interprets JDF and executes the instructions. Devices control machines in a proprietary manner.

Element: An XML-based syntactic construct. Used in JDF to describe structured data.

Instance document: A document that is part of the output of a job. This generally refers to personalized printing jobs. Each of the individual documents produced from the same input template is referred to as an instance document. For example, in a credit card statement run, each statement is an instance document.

JMF: Job Messaging Format. A messenger format with multi-level messaging capabilities. Communicates information between MIS and controllers.

Job: A hierarchical tree structure comprised of nodes. Describes the output that is desired by a customer.

Job Part: One or more nodes which comprise the smallest level of control of interest to MIS.

Link: A pointer to information that is located elsewhere in a JDF document or that is located in another document.

Machine: The part of a device that does not know JDF and is controlled by a JDF device in a proprietary manner.

MIS: Management Information Systems. The part of a JDF workflow that oversees all processes and communication between system components and system control.

Node: The JDF element type detailing the resources and process specification required to produce a final or intermediate product or resource.

Partitioned Resource: Structured resource that represents multiple physical or logical entities such as separated plates.

PDL: Page Description Language. A generic term for any language that describes pages, which may be printed. Examples are PDF®, PostScript® or PCL®.

Process: An individual step in the workflow.

Queue: Entity that accepts job entries via a JMF messaging system.

Resource: A physical or conceptual entity that is modified or used by a node. Examples include paper, images, or process parameters.

Small job: An independent job that is merged into a *big job*.

Tag: A syntactical construct that marks the start or end of an element.

Work Center: An organizational unit, such as a department or a subcontracting company, that can accomplish a task.

1.4 Data Structures

The following table describes the data structures used in the specification. For more details, see [Appendix A Encoding](#).

Data Type	Description
boolean	Binary-valued logic: (true false).
CMYKColor	Represents a CMYK color specification.
date	Represents a time period that starts at midnight of a specified day and lasts for 24 hours.
double	Corresponds to IEEE double-precision 64-bit floating point type
element	Structured data. The specific data type is defined by the element name.
enumeration	Limited set of NMTOKEN (see below).

enumerations	Whitespace-separated list of enumeration data types.
ID	Unique identifier as defined by [XML Specification 1.0] (see section 1.1 Document References). Must be unique within the scope of the JDF-document.
IDREF	Reference to an element holding the unique identifier as defined by [XML Specification 1.0].
IDREFS	List of references (IDREFs) separated by white spaces as defined by [XML Specification 1.0].
integer	Represents numerical integer values.
IntegerList	Whitespace-separated list of integers .
IntegerRange	Two integers separated by a “~” character that define a closed interval .
IntegerRangeList	Whitespace-separated list of IntegerRanges .
LabColor	Represents a Lab color specification.
language	Represents a language and country code (for example, en-US) for a natural language.
matrix	Whitespace-separated list of 6 numbers representing a coordinate transformation matrix.
NMTOKEN	A continuous sequence of special characters as defined by the [XML Specification 1.0].
NMTOKENS	Whitespace-separated list of NMTOKEN .
number	double or integer
NumberList	Whitespace separated list of numbers .
NumberRange	Two numbers separated by a “~” (tilde) character that defines the closed interval of the two.
NumberRangeList	Whitespace-separated list of NumberRanges
path	Whitespace-separated list of path operators as defined in PDF.
rectangle	Whitespace-separated list of 4 numbers representing a rectangle.
shape	Whitespace-separated list of 3 numbers representing a 3-dimensional shape consisting of a height, width and length.
sRGBColor	Represents an sRGB color specification.
string	Character strings without line feed.
telem	Text elements that contain larger chunks of character data and may include line feeds.
text	Text data contained in an telem-element.
time	Represents an instant of time that recurs daily.
timeInstant	Represents a combination of date and time values denoting a specific instant of time.
timeDuration	Represents a duration of time.
TimeRange	Two timeInstants separated by a “~” (tilde) character that defines the closed interval of the two.
TransferFunction	Whitespace separated list of an even number of numbers representing a set of XY coordinates of a transfer function.
URI	Short for URI-reference . Represents a Uniform Resource Identifier (URI)

	Reference as defined in Section 4 of [RFC 2396] .
URL	Short for URL-reference . Represents a Uniform Resource Locator (URL) Reference as defined in Section 4 of [RFC 2396] .
XYPair	Whitespace-separated list of 2 numbers .

1.5 Units

JDF specifies most values in default units. That means there is no freedom to use alternate units instead of the defined default units. All measurable quantities are stated in double precision.

Processors should only specify a *Unit* if no default exists, such as when new resources are defined. Then the units must be based on metric units. Overriding the default units that are defined in this table is non-standard and may lead to undefined behavior. Any exceptions are specified in the appropriate descriptive tables.

The following table lists the units used in JDF. The representation column specifies the XML representation in the *Unit* attribute of resources.

Measurement	Unit	Representation	Remarks
Length	point (1/72 inch)	pt	-
Length	meter	m	used for lengths of paper rolls in web printing.
Volume	liter	l	-
Weight	gram	g	-
Area	m ²	m2	-
Resolution	dpi or lpi	dpi or lpi	-
Paper weight	g/m ²	g/m2	-
Speed	units/hour	*/h	replace the "*" in the representation with the appropriate unit
Temperature	C° (Celsius)	C	degree centigrade
Angle	degrees°	degree	-
Countable Objects	1	-	Countable objects, such as sheets, have no unit specification.

Chapter 2 Overview of JDF

Introduction

This chapter explains the basic aspects of JDF. It outlines the vocabulary used and recognized by the format, and the components of a workflow necessary to execute a printing job using JDF. Also provided is a brief discussion of JDF architecture, as well as an overview of the execution model of a JDF job.

2.1 System Components

The following sections provide definitions for the principal vocabulary terms used within this specification. The first set of terms describe the components of JDF, while the second set describes how JDF identifies the aspects of the workflow system necessary to carry out a JDF job.

2.1.1 Job Components

The terminology explained below describe the way in which JDF is divided conceptually and hierarchically.

2.1.1.1 Jobs and Nodes

The term **job** describes the entirety of a JDF project. Each job is organized in a tree structure containing all of the information required to complete the intended project. The information is collected logically into what is called a **node**. Each node in the tree structure represents an aspect of the job to be executed.

The nodes in a job are organized in a hierarchical structure that resembles a pyramid. The node at the top of the pyramid describes the overall intention of the job. The intermediate nodes describe increasingly process-oriented aspects of the job, until the nodes at the bottom of the pyramid each describe a single, simple process. Depending on where in the job structure it resides, a node can represent a portion of the product to be created, one or many processing steps, or other job parts. For more information about jobs and nodes, see [Chapter 3 Structure of JDF Nodes and Jobs](#).

2.1.1.2 Elements

The term **element** describes an XML syntactic construct. Within this document, the term refers to the structured sub-parts of a JDF node. Technically, JDF nodes are themselves XML elements. However, within this specification, the term “node” is used to distinguish between the independent JDF aspect and its sub-parts. Furthermore, elements that are sub-parts of other elements are often referred to as sub-elements. There is no structural distinction between nodes, elements and sub-elements; rather, the different terminology is intended to describe the hierarchical relationships.

JDF elements are represented by two kinds of data types: element and text element. The latter is abbreviated as telem.

For more information about elements, see [section 3.1.2 Fundamental JDF Attributes and Elements](#).

2.1.1.3 Attributes

The term **attribute** also describes an XML syntactic construct. Within this document, the term refers to characteristics of elements. For instance, each node has an *ID* attribute that contains a unique identifier. Attributes contain parameters of different data types, such as string, enumeration, and time.

For more information about attributes, see [section 3.1.2 Fundamental JDF Attributes and Elements](#).

2.1.1.4 Relationships

The hierarchical JDF structure implies relationships between nodes and elements within a JDF tree structure. The terms used in this document to describe these relationships are defined below, and, in some cases, include a brief representation of the encoding that would express them.

- **Parent:** An element that directly contains a child element.
`<Parent><Child/></Parent>`
- **Child:** An element that resides directly in the parent element.
- **Sibling:** An element that resides in the same parent element as another child element.
`<Any><Sibling/><Sibling/></Any>`
- **Descendent:** An element that is a child or a child of a child, etc.
- **Ancestor:** An element that is a parent or a parent's parent, etc.
`<Ancestor>
 <Any>
 <Descendent/>
 <MoreAnys>
 <Descendent/>
 </MoreAnys>
 </Any>
 </Ancestor>`
- **Root:** The single element that contains all other elements as descendents.
- **Leaf:** Node without further children.
- **Branch:** An intermediate node in a hierarchy that contains at least one child node. A branch is never a leaf.

2.1.1.5 Links

JDF makes extensive use of links in order to reuse information that is relevant in more than one context of the job. The same target may be referenced by multiple links. However, no link references more than one target.

There are two kinds of links in JDF: internal links and external links. Internal links are pointers to information that is located elsewhere in a JDF document. The data that is referenced by the link is located in a target element. External links are used to reference objects that are outside of the JDF document itself, such as content files or color profiles. These objects are linked using standard URLs (Uniform Resource Locators).

2.1.2 Workflow Component Roles

The four components required to create, modify, route, interpret and execute a JDF job are known as agents, controllers, devices and machines. Overseeing the workflow created by these components is MIS, or Management Information Systems. These five aspects of a JDF workflow are described in the sections that follow.

By defining these terms, this specification does not intend to dictate to manufacturers how a JDF/JMF system should be designed, built, or implemented. The intention is to name the component mechanisms required for the interaction of actual components in a workflow during the course of a JDF job. In practice, it is very likely that individual system components will include a mixture of the capabilities described in the following sections. For example, many controllers are also agents.

2.1.2.1 Machines

A **machine** is any part of the workflow system designed to execute a process. Most often, this term refers to a piece of physical equipment, such as a press or a binder, but it can also refer to the software components used to run a particular machine. Computerized workstations, whether run through automated batch files or whether controlled by a human worker, are also considered machines if they have no JDF interface.

2.1.2.2 Devices

The most basic function of a **device** is to execute the information specified by an agent and routed by a controller. Devices must be able to execute JDF nodes and initiate machines that can perform the physical execution.

The communication between machines and devices is not defined in this specification. Devices may, however, support JMF messaging in order to interact dynamically with controllers.

2.1.2.3 Agents

Agents in a JDF workflow are responsible for writing JDF. An agent has the ability to create a job, to add nodes to an existing job, and to modify existing nodes. Agents may be software processes, automated tools, or even text editors. Anything that can be used in composing JDF can be considered an agent.

Actual implementations of devices or controllers will most often be able to modify JDF. These system components have agent properties in the terms of this specification.

2.1.2.4 Controllers

Whereas agents are responsible for creating and modifying JDF information, **controllers** are responsible for routing it to the appropriate devices. The minimum requirement of a controller is that it be in a position to initiate processes on at least one device, or at least one other slave controller that itself will initiate processes on a device. In other words, a controller is not a controller if it has nothing to control. In some cases, a pyramid-like hierarchy of controllers can be built, with controllers at the top of the pyramid controlling a series of lower-level controllers at the bottom. The lowest-level controllers in the pyramid, however, must have device capability. Therefore, controllers must be able to work in collaboration with other controllers. In order to communicate with one another, as well as to communicate with devices, controllers must support the JDF file-exchange protocol and may support the Job Messaging Format (JMF).

Controllers can also determine process planning and scheduling data, such as process times and planned production amounts.

2.1.2.5 Management Information Systems—MIS

The overseer of the relationships between all of the units in a workflow is known as Management Information Systems, or **MIS**. MIS is, in effect, a macrocosmic controller. It is responsible for dictating and monitoring the execution of all of the diverse aspects of the workflow. To do this, it must remain in

contact with the actual production facilities. This can be accomplished either in real time using JMF messaging or post-facto using the audit records within JDF.

To allow MIS to communicate effectively with the other workflow components, JDF supplies what is essentially a messenger service, in the form of JMF, to run between MIS and production. This format is equipped with a variety of message types, ranging from simple, uni-directional notification through queries and even commands. System designers have a great deal of flexibility in terms of how they choose to use the messaging architecture, so that they can tailor the processes to the capabilities of the existing workflow mechanism. Figure 2.1 depicts how various communication threads can run between MIS and production.

JDF also provides system components the ability to collect performance data for each node, which can then be passed on to a job-tracking system for use by the MIS system. These data may be derived from the messages that the controller receives or from the audit records in the job (for more information on audits, see section 3.9.1 *Audit Elements*). Alternatively, the completed job may be passed to the job accounting system, which examines the audit records to determine the costs of all the processes in the job.

2.1.2.6 System Interaction

An example of the interaction and hierarchical structure of the components considered in the preceding sections is shown in the following figure. Single arrows indicate uni-directional communication channels and double arrows indicate bi-directional communication.

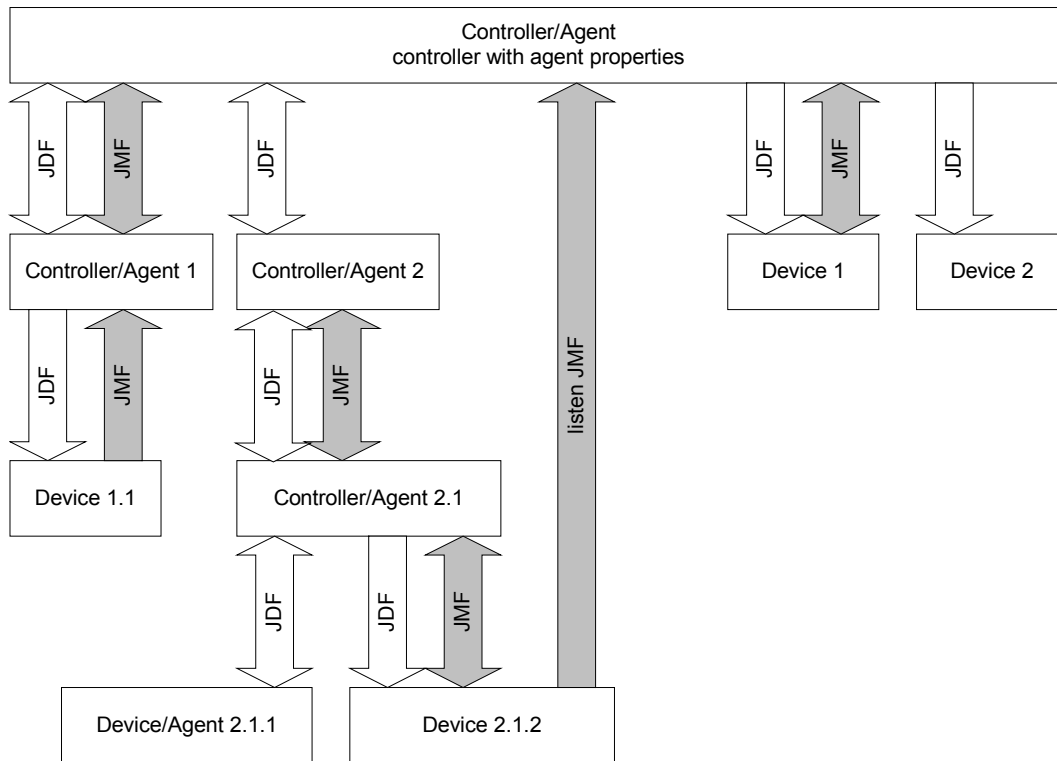


Figure 2.1 Example of JDF and JMF workflow interactions

2.2 JDF Workflow

JDF does not dictate that any particular workflow must be constructed in order for it to be usable. On the contrary, its flexibility allows JDF to model the myriad custom solutions that already exist and that can be imagined for the graphic-art world. JDF is equally as effective with a simple system using a single controller-agent that controls one device as it is with a completely automated industrial press workflow with integrated pre- and post-press operations.

Because of the way in which workflow systems are currently constructed, the procedures of the principal subsections of a printing job—pre-press, press, and post-press—remain largely disconnected from one another. JDF provides an extremely flexible and thorough compensation. With JDF, a print job becomes an interconnected workflow, from job submission through trapping, ripping, filmmaking, platemaking, inking, printing, cutting, binding, and sometimes even through shipping. JDF constructs an architecture that defines each process necessary to produce an intended result, and identifies the elements necessary to complete the processes. Each process is separated into a node, and the entire job is represented by a tree of these nodes. All of the nodes taken together delineate the processes necessary to produce the desired printed product.

Each individual node is defined in terms of inputs and outputs. The inputs for a node consist of the resources it uses and the parameters that control it. Inputs in a node describing the process parameters for imaging the cover of a brochure, for example, might include requirements for trapping, ripping, and imposing the image. The output of this node will be a raster image.

Resources produced by one node, however, are modified or consumed by subsequent nodes. Therefore, the output described above—a raster image—becomes one of the input resources for a node describing the printing process for the brochure. Other inputs in this node would include the inks, the press sheets, the plates, and a set of parameters that indicate how many sheets should be produced. The output will be a set of printed press sheets that in turn will become the input resource for post-press operations such as folding and cutting. And so on until the brochure is completed.

This system of interlinked nodes effectively unites the pre-press, press, and post-press processes, and even extends the notion of where a job begins. A JDF job, like any printing job, is defined by the original intent for the end product. The difference between a JDF job and a generic printing job, however, is that JDF allows the entire job, from pre-press through post-press, to be defined up front. All of the resources and processes necessary to produce an entire printed product can be identified and organized into nodes before the first pre-press process is set in motion. Furthermore, the product intent specification can be extremely broad *or* extremely detailed, or anywhere along the spectrum in between. This means that a job may be so well defined before production begins that the system administrator only has to set the wheels in motion and let the job run its course. It may also mean that the person submitting the job has only a general idea of what the final product will look like, and that modifications to the intent will be made along the way, depending on the course of the job.

For example, the person submitting the job specification for the brochure described above may know that she wants 400 copies, that she wants it done on a four-color press with no spot colors, that the cover will be on a particular paper stock and the contents on another, that the binding will be stapled, and that she requires the job in two weeks. Another person might know only that he wants the pages she's designed to be put into some sort of brochure form, although she doesn't know exactly what. Either person's request can be translated into a JDF product intent node that will eventually branch into a tree structure describing each process required to complete the brochure. In the first example, the pre-press, press, and post-press processes will be well-defined from the start. In the second example, information will be included as it is gathered.

The following sections describe the way in which nodes can combine to form a job.

2.2.1 Job Structure

As was mentioned above, JDF jobs consist of a set of nodes that specify the production steps needed to create the desired end product. The nodes, in addition to being connected through inputs and outputs, are arranged in a hierarchical tree structure. Figure 2.3, below, shows a simple example of a tree of nodes.

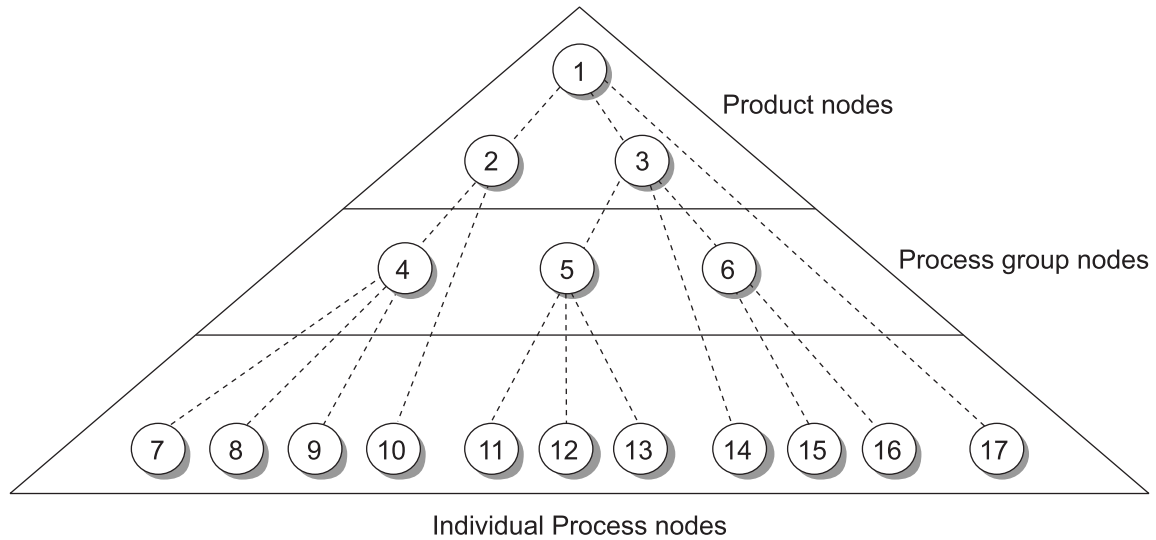


Figure 2.2 JDF tree structure

The following table provides a hypothetical breakdown of the nodes in the tree structure shown above:

Table 2.1 Information contained in JDF nodes, arranged numerically

Node #	Meaning
1	Entire book
2	Cover
3	Contents
4	Production of cover
5	Production of all color pages
6	Production of all black & white pages
7	Cover production process 1
8	Cover production process 2
9	Cover production process 3
10	Cover Finishing process
11	RIPping for color pages
12	Plate making for color pages
13	Printing for color pages
14	Color page finishing process
15	RIPping for black & white pages
16	Printing for black & white pages on a digital press
17	Binding process for entire book

The uppermost nodes (1, 2, & 3) represent the product intent in general terms. These nodes describe the desired end product and the components of that product, which, in this case, are the cover and the content pages. As the tree branches, the information contained within the nodes gets more specific. Each sub-node defines a component of the product that has a unique set of characteristic, such as different media, different physical size, or different color requirements. The nodes that occur in the middle of the tree (4, 5, & 6) represent the groups of processes needed to produce each component of the product. The nodes that occur closest to the bottom of the tree (7 – 17) each represent individual processes.

In this example, there are two sub-components of the job, the cover and the contents, each with distinct requirements. Therefore, two nodes—nodes 2 and 3—are required to describe the elements of the job in broad terms. Within the content pages there are some black & white pages, and some color pages. Since fabricating each requires a different set of processes, further branching is necessary. The following table arranges the nodes in groups according to the processes they'll be executing:

Table 2.2 Information contained in JDF nodes, arranged by group

Process Group	Node #	Meaning
Entire Book	1	Entire book
	17	Assemble Book
Cover	2	Cover
	4	Cover assembly processes
	7	Cover production process 1
	8	Cover production process 2
	9	Cover production process 3
	10	Finishing process for cover
	3	Contents
Color Pages	5	Production of all color pages
	11	RIPping for color pages
	12	Plate making for color pages
	13	Printing for color pages
	14	Color page finishing
Black & White Pages	6	Production of all black & white pages
	15	RIPping for black & white pages
	16	Printing for black & white pages on a digital press

This hierarchical structure is discussed in more detail in the following section.

2.3 Hierarchical Tree Structure and Networks in JDF

As has been described, many output resources of JDF nodes are the input resources for other JDF nodes. Many nodes cannot begin executing until all of their resources are complete and ready, which means that the nodes execute in a well defined sequence. One process follows the next. For example, a process for making plates will produce, as output resources, press plates that are required by a printing process.

In the hierarchical organization of a JDF job, nodes that occur higher in the tree represent higher-level, more abstract operations, while lower nodes represent more detailed, specific process operations. More specifically, nodes near the top of the tree may represent only intent regarding the components or

assemblies that comprise the product, while the leaf nodes provide specific, detailed instructions to a device to perform some operation. Figure 2.3 shows an example of a hierarchical structure.

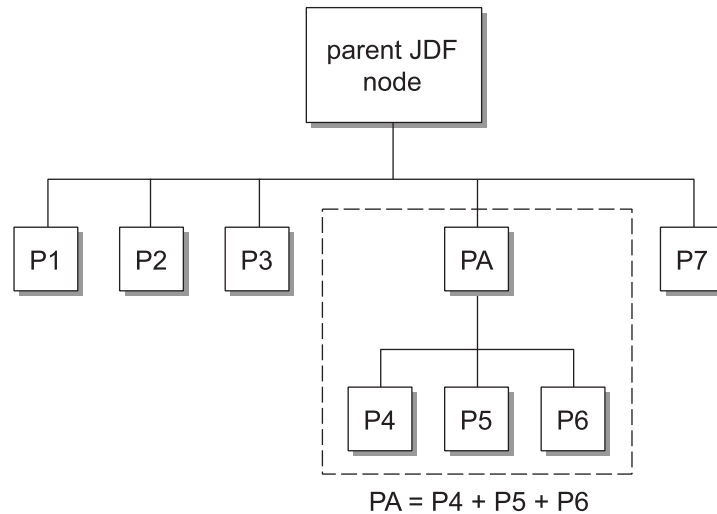


Figure 2.3 Example of a hierarchical tree structure of JDF nodes

In addition to the hierarchical structure of the node tree, sibling nodes are linked in a process chain by their respective resources. In other words, an output resource of one node ends up representing the input resource of the following node (as represented in Figure 2.4). This interrelationship is known as resource linking.

With resource linking, complex networks of processes can be formed. Figure 2.4, below, displays an alternate representation of the process described in Figure 2.3. Whereas Figure 2.3 represents a hierarchical structure, Figure 2.4 shows the linking mechanism of the same job.

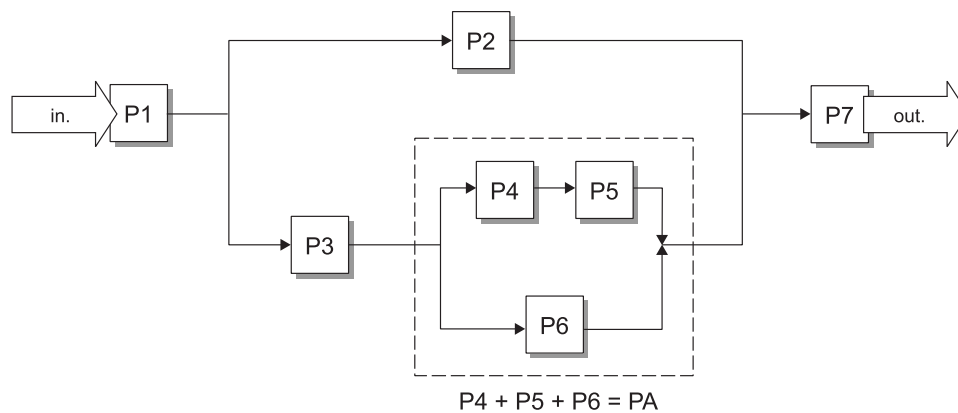


Figure 2.4 Example of a process chain linked by input and output resources

In JDF, the linking of processes is not explicitly specified. In other words, nodes are not arranged in an abstract chronology, dictating, for example, that the trapping node must come before the ripping node. Rather, the links are implicitly defined in the exchange of inputs and outputs. Resource dependencies form a network of processes, and the sequence of process execution—that is, the routing of processes—can be derived from these dependencies. One resource dependency might have multiple possible process routing scenarios, and it is up to MIS to define what will be a proper solution with respect to the local constraints.

The agent or set of agents employed by MIS to write the JDF job must therefore be familiar with these local constraints. They must take into account factors such as the control abilities of the applications that complete the pre-press processes, the transport distance between the pre-press facility and the press itself,

the load capabilities of the press, and the time requirements for the job, to name a few issues. All of the factors taken together construct a process network representing the workflow of production. To aid agents in defining the workflow, JDF provides the following four different and fundamental types of process routing mechanisms, which may be combined in any way:

1. **Serial processing** that is subsequent production and consumption of resources as a whole, represented by a simple process chain.
2. **Overlapping processing** that is simultaneous production and consumption of resources by pipes.
3. **Parallel processing** that involves the splitting and sharing of resources.
4. **Iterative processing** that is somehow a circular or back and forward processing for developing resources by iteration.

These mechanisms are discussed in greater detail in section 4.3 Execution Model.

2.4 Role of Messaging in JDF

Whereas JDF provides a container to define a job, JMF messaging, defined in Chapter 5, provides a method to generate snapshots of a job status and to interactively manipulate elements of a workflow system.

JMF is specifically designed for communication between the production system controller and the work centers or devices with which it interacts. It provides a series of queries and commands to check the status of processes and, in some cases, to dictate the next course of action. For example, the **KnownDevices** and **KnownJDFServices** queries allow the controller to determine what processes can be executed by a particular device or workcenter. These processes are likely to be determined at system initialization time. The **QueueEntry** messages provide a means for the controller to submit a job ticket to individual work centers or devices. And the **Status**, **Progress** and **Phase** messages allow the device or workcenter to communicate quasi real-time¹ processing status to a controller. Depending on the system configuration, the message handler may choose to record status changes in the history logs. The progress message allows the controller to request status updates from the controller.

JDF also provides mechanisms to define recipients for individual messages on a node-by-node basis, thus enabling controllers to define the aspects and the parts of jobs that they are interested in tracking.

For more information about messaging, see Chapter 5 JDF Messaging with the Job Messaging Format (JMF)

¹ Real-Time in the time-scale typically associated with macro-cosmic production control systems. JMF is not intended for real-time, low-level machine control.

Chapter 3 Structure of JDF Nodes and Jobs

Introduction

This chapter describes the structure of JDF nodes and how they interrelate to form a job. As was described in section 2.1.1 *Job Components*, a node is a construct, encoded as an XML element, that describes a particular part of a JDF job. Each node represents an aspect of the job, either in terms of a process necessary to produce the end result, such as imposing, printing, or binding; in terms of a product that contributes to the end result, such as a brochure; or in terms of some combination of the two. In short, a node describes a product or a process.

In addition to describing the structure of an individual JDF node, this chapter examines in what way those nodes interact to form a coherent job structure. The interrelation of nodes can be divided into two categories: hierarchical and lateral. Hierarchical interrelation is the nested structure of parent nodes that contain child nodes. The visual correlative of this structure resembles a family tree, with a single node describing the entire job at the top, and a number of nodes at the bottom that each describe only one specific process. JDF-supported leaf-level processes are described in Chapter 6 *Processes*.

Lateral interrelation, on the other hand, is the interrelation that occurs between nodes as a result of resource linking. Resource linking is the result of the transformation of inputs into outputs, which in turn may become inputs of other nodes. It also occurs when nodes share the same resource. The combination of hierarchical nesting of nodes and lateral linking allows complex process networks to be constructed. In a very simple case, however, a JDF file may contain only one node.

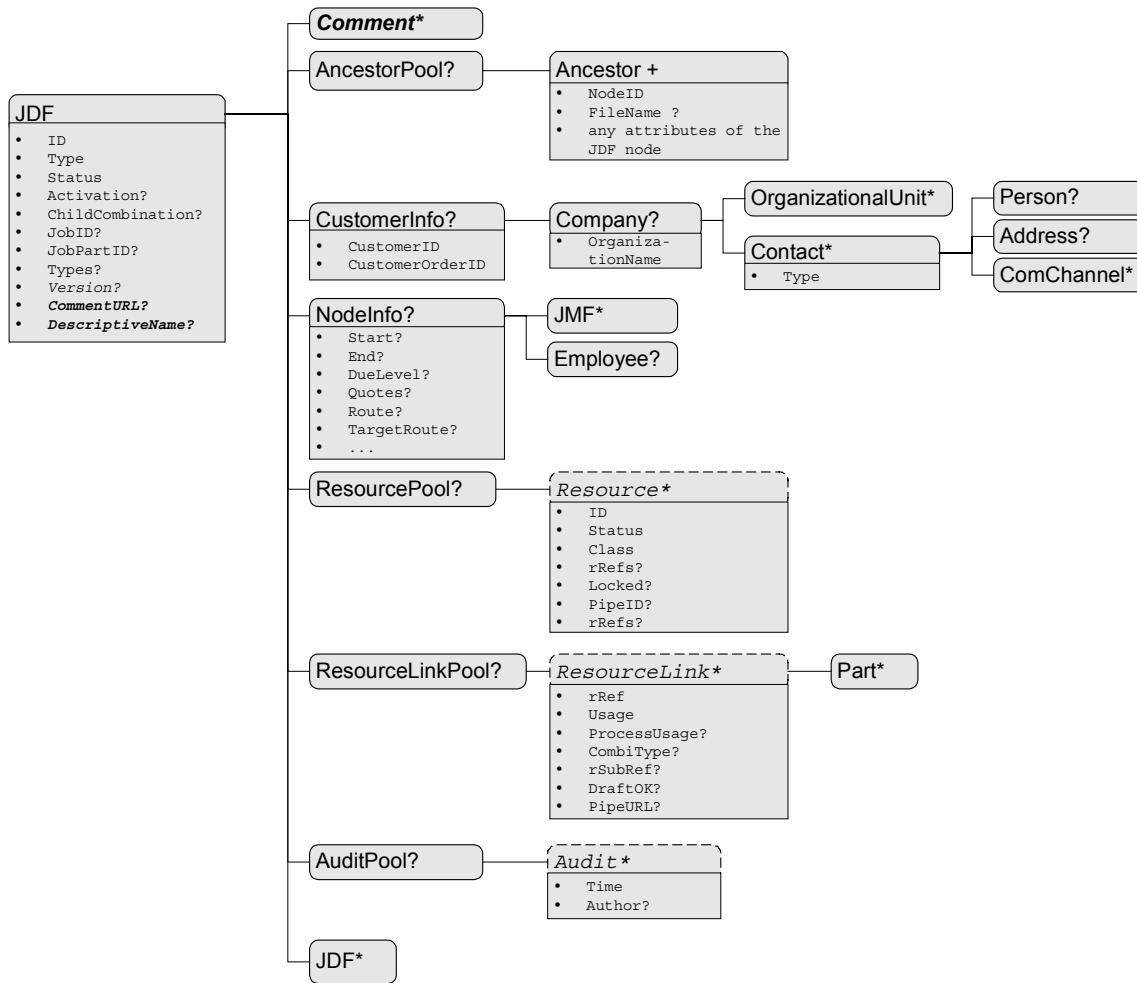
The hierarchical structure of a JDF job achieves a functional grouping of processes. For example, a job may be split into a prepress node, a press node, and a finishing node that contain the respective process nodes. Each and every node in turn contains attributes that represent various characteristics of that node. Nodes also contain sub-elements of certain types, such as resources, process information, customer information, audits, logging information, and other JDF nodes. Some elements, such as those that deal with customer information, generally occur only in the root structure, while other elements, such as resources, may occur anywhere in the tree. Where the elements can reside depends on their type and their usage scope.

This chapter describes the elements, sub-elements, and attributes commonly found in JDF nodes, and provides the characteristics necessary to understand where each belongs and how it is used. Many of these characteristics are presented in tables, and each of these tables includes the following three columns:

- **Name**—Identifies the element being discussed.
- **Data Type**—Refers to the data type, all of which are described in **Error! Reference source not found.** Only the data types **element** or **telem** (which is short for text element) are applied to elements. All other types are attributes.
- **Description**—Provides detail about the element or attribute being discussed.

The JDF workflow model is based on a resource/consumer model. JDF nodes are the consumers that are linked by input resources and output resources. The ordering of siblings within a node, however, has no effect on the execution of a node. All chronological and logical dependencies are specified using **ResourceLinks**, which are defined in section 3.7 *Resource Links*.

Figure 3.1 shows schematically the structure of the JDF node type. In this figure, generic attributes and elements (see section 3.1.1 *Generic Contents of JDF Elements*) are inserted only in the JDF root node. The element types that are displayed in this figure are described in the subsequent sections. Abstract data types are surrounded by a dashed line. Types derived from the abstract data type **Resource** are shown schematically in Figure 3.4.



Attributes:

JDF:

Type = Product | ProcessGroup | Combined | *any process name*

Status = waiting | quoted | ready | failed_testrun | setup | in_progress | cleanup | spawned | stopped | completed | aborted

Activation = inactive | RFQ | testrun | testrun_and_go | **active**

Resource:

Status = unavailable | draft | available | in_use

SpawnStatus = not_spawned | spawned_RO | spawned_RW

Locked = **false** | true ; (volatile or persistent)

ResourceLink:

Usage = input | output

Figure 3.1 Structure of the JDF node type

3.1 JDF nodes

JDF nodes are encoded as XML elements. Nodes, in turn, contain various attributes and further sub-elements including nested JDF nodes.

Many of the tables in this particular section contain a fourth column that provides further details about the valid range of the attribute/element content, how the content is inherited by descendants (children, grandchildren, etc.), and where the attribute/element may reside in the JDF tree. The heading for this column is “Scope,” which is short for “Scope and Position.” The following abbreviations are defined:

- D)** Descendent: The content is valid locally within its node and in all descendent nodes, unless a descendent contains an identical attribute that overrides the content.
- L)** Local: The content is only valid locally, within the node where the content is defined.
- R)** Root: The attribute may only be specified in the root node. An exception from the localization only in the root node occurs if the spawning and merging mechanism for independent job tickets is applied as described in section 4.4 *Spawning and Merging*.

All attributes and elements listed in subsequent chapters should be considered local, unless otherwise noted.

3.1.1 Generic Contents of JDF Elements

JDF contains a set of generic structures that may occur in any element of a JDF or JMF document. These are provided as containers for human-readable comments and descriptions, and are described below.

Table 3.1 *Generic Contents of elements*

Name	Data Type	Description
<i>CommentURL</i> ?	URL	URL to an external, human-readable description of the element.
<i>DescriptiveName</i> ?	string	Human-readable descriptive name of, for example, a resource, process or product.
Comment *	telem	Any human-readable text.

The comment fields may contain a language attribute to support internationalization.

Table 3.2 *Contents of the Comment element*

Name	Data Type	Description
<i>Box</i> ?	rectangle	The rectangle that is associated with the comment. The coordinate system of the rectangle is the same as the coordinate system defined in the <i>Path</i> attribute.
<i>Language</i> ?	language	Possible values are defined in IETF RFC 1766. If none is specified, the system default is assumed.

<i>Name ?</i>	NMTOKEN	A name that defines the usage of a comment. For example, it may determine whether two comments should fill two distinct fields of a user interface. Default = <i>description</i> , which is required if the Comment element may become mandatory, as is the case in the Notification element (see Table 3.23).
<i>Path ?</i>	path	Description of the area that the comment is associated with in the coordinate system of the element where the path resides. For example, the path refers to the plate coordinate system, if the comment is inserted in an ExposeMedia resource that describes a plate.
	text	Body of the comment.

The following figure shows the structure of the generic content defined above.

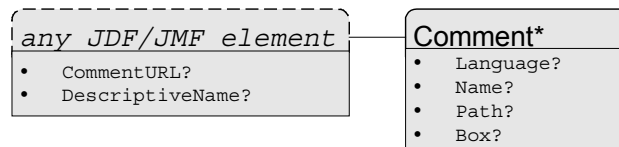


Figure 3.2 Structure of JDF Generic Contents

3.1.2 Fundamental JDF Attributes and Elements

The following table presents the attributes and elements likely to be found in any given JDF node. Three of the attributes in Table 3.3, below, are required, and must appear in every JDF node. Although the rest are designated as optional, they are optional only in the sense that they are required only under certain circumstances, not that they may be left out if desired. The circumstances under which they are required are described in the Description column.

The most important of the attributes is the *Type* attribute, which defines the node type. The value of the *Type* attribute defines the product or process the JDF node represents. As is detailed in section 3.2 Common Node Types, all nodes fall into one of the following four general categories: process, process group, combined processes and product intent. Each node is identified as belonging to one of these categories by the value of its *Type* attribute, as described in the table below. For example, if *Type* = *Product*, the node is a product intent node. Each of these categories is described in greater detail in the sections that follow.

Table 3.3 Contents of a JDF node

Name	Data Type	Scope	Description
<i>Activation ?</i>	enumeration	D ¹	Describes the activation status of the node. Allows for a range of activity, including deactivation and testrunning. Possible values, in order of involvement from least to most active, are: <i>inactive</i> – The node and all its descendants shall not be executed or tested. This value is set if only certain parts of a JDF job should be executed or tested or if the node contains information required by other processes (as is the case with independent spawning and merging,

¹ The inheritance of activation is non-trivial and specified in the description field.

described in section 4.4.5).

RFQ – The node specifies a request for a pricing quote and shall not be executed.

testrun – The node requests a test run check by an controller or a device. This does not imply that the node should be automatically executed when the check is completed. Descendants of a node that is being test run not to be considered *active*.

testrun_and_go – Similar to *testrun*, but requests a subsequent automatic start if the *testrun* has been completed successfully.

active – Default. The node maybe executed as soon as all inputs are available.

A child node inherits the value of the *Activation* attribute from its parent. The value of *Activation* corresponds to the least active value of *Activation* of any ancestor, including itself. Therefore, if any ancestor has an *Activation* of *inactive*, the node itself is *inactive*. If no ancestor is *inactive* but any ancestor is *testrun*, the node is *testrun* unless the node itself is *inactive*. If no ancestor has a value of *inactive* or *testrun* and any ancestor has a value of *testrun_and_go*, the node has a value of *testrun_and_go* unless that node is *inactive* or *testrun*, and so on.

<i>ChildCombination ?</i>	enumeration	L	<p>Specifies how the child nodes node should be combined to define the job. Possible values are:</p> <p><i>and</i> – The child nodes shall be combined and all of them shall be executed according to their <i>Activation</i>. The default.</p> <p><i>or</i> – The child nodes are alternatives of which one must be selected.</p> <p>Note that <i>ChildCombination</i> should be specified only for Product nodes.</p>
<i>ID</i>	ID	L	<p>Unique identifier of a JDF node. This ID is used to refer to the JDF node.</p>
<i>JobID ?</i>	string	D	<p>Job identification used by the application that created the JDF job. Typically, a job is identified by the internal order number of the MIS system that created the job.</p>
<i>JobPartID ?</i>	string	D	<p>Identification of a part of a job, used by the application that created the job. Typically, this is internal to the MIS system that created the job and coincides with a process or set of processes.</p>
<i>Status</i>	enumeration	L	<p>Identifies the status of the node. Possible values are:</p> <p><i>waiting</i> – The node may be executed, but it has not completed a <i>testrun</i>.</p> <p><i>quoted</i> – The node specifies the result of a price estimate and shall not be executed.</p> <p><i>ready</i> – As indicated by the successful completion of a</p>

testrun, all **ResourceLinks** are correct, required resources are available, and the parameters of resources are valid. The node is ready to start.

failed testrun – An error occurred during the test run. Error information is logged in the **Notification** element, which is an optional sub-element of the **AuditPool** element described in section 3.9.

setup –The process represented by this node is currently being set up.

in progress – The node is currently executing.

cleanup – The process represented by this node is currently being cleaned up.

spawned – The node is spawned in the form of a separate spawned JDF.

The status *spawned* can only be assigned to the original instance of the spawned job. For details, see section 4.4.

stopped – Execution has been stopped. If a job is *stopped*, running may be resumed later. This status may indicate a break, a pause, maintenance, or a breakdown—in short, any pause that does not lead the job to be aborted.

completed – Indicates that the node has been executed correctly, and is finished.

aborted – Indicates that the process executing the node has been aborted, which means that execution will not be resumed again.

Derivation of the **Status** of a parent node from the **Status** of child nodes is non-trivial and implementation-dependent.

<i>Type</i>	NMTOKEN	L	<p>Identifies the type of the node. Any JDF process name is a valid type. The processes that have been pre-defined are listed in Chapter 6, although the flexibility of JDF allows anyone to create processes. Besides these values, there are three values, which are described in greater detail in the sections that follow. These values are:</p> <p><i>Product</i></p> <p><i>ProcessGroup</i></p> <p><i>Combined</i></p>
<i>Types ?</i>	NMTOKENS	L	<p>List of the Type attributes of the nodes that are combined to create this node. This attribute is mandatory if Type = <i>Combined</i>, and is ignored if Type equals any other value. For details on using <i>Combined</i> nodes, see section 3.2.3.</p>
<i>Version ?</i>	string	RD	<p>Text that identifies the version of the JDF node. The current version of this specification is “1.0”. The Version attribute is mandatory in the JDF root node, but</p>

			not in child nodes.
AncestorPool ?	element	R	If this element is present, the current JDF node has been spawned, and this element contains a list of all ancestors prior to spawning. See section 3.3.
AuditPool ?	element	L	List of elements that contains all relevant audit information. Audits are intended to serve the requirements of MIS for evaluation and invoicing. See section 3.9.
CustomerInfo ?	element	D	Container element for customer-specific information. See section 3.4.
JDF *	element	L	Child JDF nodes. The nesting of JDF nodes defines the JDF tree. In contrast to the elements above, JDF child nodes are not contained in a list element.
NodeInfo ?	element	L	Container element for process-specific information such as scheduling and messaging setup. Scheduling affects the planned times when a node should be executed. Actual times are saved in the AuditPool . See section 3.9 for more details.
ResourceLinkPool ?	element	L	List element for ResourceLink elements, which describe the input and output resources of the node. See section 3.7 for more details.
ResourcePool ?	element	L ²	List element for resources. See section 3.6 for more details.

3.2 Common Node Types

As was noted in preceding section, the *Type* of a node can fall into four categories. The first is comprised of the specific processes of the kind delineated in Chapter 6, known simply as process nodes. The other categories are made up of three enumerative values of the *Type* attribute: *ProcessGroup*, *Combined*, and *Product*, which is also known as product intent. These three node types are described in this section.

The figure below, which was also presented as an illustration in Chapter 2, represents a theoretical job hierarchy comprised of *Product* nodes, *ProcessGroup* nodes, and nodes that represent individual processes. The diagram is divided into three levels to help illustrate the difference between the three kinds of nodes, but these levels do not dictate the hierarchical nesting mechanism of a job. Note, however, that an individual process node may be the child of a product intent node without first being the child of a process group node. Likewise, a process group node may have child nodes that are also process groups.

² Resources are unique and cannot be overwritten by descendants. Rather, they can only be used by descendants. An exception to this is described in section 4.4.5 Case 5: Spawning and Merging of Independent Jobs. In this case, resources may also be used by a parent node.

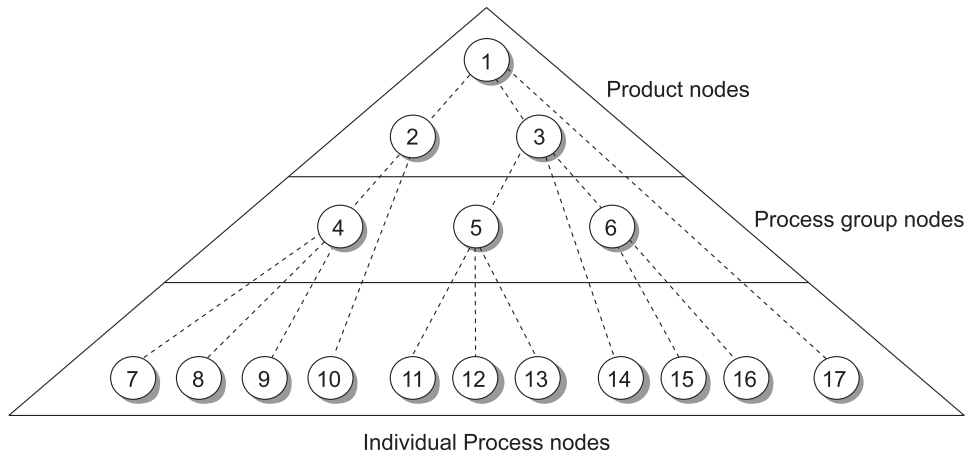


Figure 3.3 Job hierarchy with process, process group, and product intent nodes

3.2.1 Product Intent Nodes

Except in certain specific circumstances, the agent assigned to begin writing a JDF job will very likely not know every process detail needed to produce the desired results. For example, an agent that is a job-estimating or job-submission tool may not know what devices can execute various steps, or even which steps will be required.

If this is the case, the initiating agent creates a set of top-level nodes to specify the product intent, without providing any of the processing details. Subsequent agents then add nodes below these top-level nodes to provide the processing details needed to fulfill the intent specified.

These top-level nodes have a *Type* attribute value of *Product* to indicate that they do not specify any processing. All processing needed to produce the products described in these nodes must be specified in *Process* nodes, which exist lower in the job hierarchy.

Product nodes include intent resources that describe the end results the customer is requesting. The intent resources that have already been defined for JDF are easily recognizable, as they contain the word “intent” in their titles. Examples include **FinishingIntent** and **ColorIntent**. All intent resources share a set of common sub-elements, which are described in section 7.1.1 *Span Resource Sub-elements*. These resources do not attempt to define the processing needed to achieve the desired results; rather they provide a forum to define a range of acceptable possibilities for executing a job.

For more information about product intent, see section 4.1.1 *Product Intent Constructs*.

3.2.2 Process Group Nodes

Intermediate nodes in the JDF job hierarchy—that is, nodes 4, 5, and 6 in Figure 3.3—describe groups of processes. The *Type* attribute value of these kinds of nodes is *ProcessGroup*. These nodes are used to describe multiple steps in a process chain that have common resources or scheduling data. In essence, any process node that is not a leaf node is a *ProcessGroup* node.

Since the agent writing the job has the option of grouping processes in any way that seems logical, custom workflows can be modeled flexibly. Process group nodes may contain further process group nodes, individual process nodes, or a mixture of both node types. Sequencing of process group nodes should be defined by linking resources of the appropriate leaves or, if the nature of the interchange resources is unknown, by linking **PlaceHolder** resources.

The higher the level of the process group nodes within the hierarchy, the larger the number of processes the group contains. A high-level process group node might include, for example, prepress, finishing, or printing processes. Low-level process groups, on the other hand, define a set of individual steps that are executed as a group of steps in the individual workflow hierarchy. For example, all steps performed by one designated individual may be grouped in a low-level process group.

The following example shows the `ResourceLink` structure for a `ProcessGroup` inline-finishing node. Note the presence of intermediate components that are exchanged by the individual processes.

```
<JDF Type = "ProcessGroup" ID = "J1">
  <JDF Type = "DigitalPrinting" ID = "J2">
    <ResourceLinkPool>
<!-- digital printing parameters -->
    <DigitalPrintingParamsLink Usage="input" rRef="L1"/>
<!-- input sheets -->
    <MediaLink Usage="input" rRef="L2"/>
<!-- printed output components -->
    <ComponentLink Usage="output" rRef="L3"/>
  </ResourceLinkPool>
</JDF>
  <JDF Type = "Gathering" ID = "J3">
    <ResourceLinkPool>
<!-- gathering parameters -->
    <GatheringParamsLink Usage="input" rRef="L4"/>
<!-- printed output components -->
    <ComponentLink Usage="input" rRef="L3"/>
<!-- gathered output components -->
    <ComponentLink Usage="output" rRef="L5"/>
  </ResourceLinkPool>
</JDF>
  <JDF Type = "Stitching" ID = "J4">
    <ResourceLinkPool>
<!-- Stitching parameters -->
    <StitchinParamsLink Usage="input" rRef="L6"/>
<!-- gathered output components -->
    <ComponentLink Usage="input" rRef="L5"/>
<!-- stitched output components -->
    <ComponentLink Usage="output" rRef="L7"/>
  </ResourceLinkPool>
</JDF>
</JDF>
```

3.2.3 Combined Process Nodes

The processes described in Chapter 6 `Processes` define individual workflow steps that are assumed to be executed by a single-purpose device. Many devices, however, are able to combine the functionality of multiple single-purpose devices and execute more than one process. For example, a digital printer may be able to execute the *Interpreting*, *Rendering*, and *DigitalPrinting* processes. To accommodate such devices, JDF allows processes to be grouped within a node whose `Type = Combined`. Such a node must also contain a `Types` attribute, which in turn contains an ordered list of the `Type` values of each of processes that the node specifies. Furthermore, `ResourceLink` elements in `Combined` nodes should specify a `CombinedProcessType` attribute in order to define the sub-process to which the resource belongs.

A device with multiple processing capabilities is able to recognize the `Combined` node as a single unit of work that it can execute. Therefore, all resources for each of the sub-tasks that define the `Combined` node

must be available before the node can be executed. In addition, all input and output resources that are consumed and produced externally by the process must be specified in the **ResourceLinkPool** element of the node. This includes all required Parameter resources as well as the initial input resources and final output resources. Intermediate resources that are internally produced and consumed, on the other hand, need not be specified.

The following example of the **ResourceLinkPool** of a JDF node describes digital printing with inline-finishing and includes the same processes as the previous **ProcessGroup** example. The node requires the parameter resources and consumable resources of all three processes as inputs, and produces a completed booklet as output. The intermediate printed sheets and gathered piles are not declared, since they exist only internally within the device and cannot be accessed or manipulated by an external controller.

```
<JDF Type = "Combined" Types = "DigitalPrinting Gathering Stitching" ID
= "J1">
  <ResourceLinkPool>
<!-- digital printing parameters -->
  <DigitalPrintingParamsLink Usage="input"
CombinedProcessType="DigitalPrinting" rRef="L1"/>
<!-- gathering parameters -->
  <GatheringParamsLink Usage="input" CombinedProcessType="Gathering"
rRef="L4"/>
<!-- Stitching parameters -->
  <StitchingParamsLink Usage="input" CombinedProcessType="Stitching"
rRef="L6"/>
<!-- input sheets -->
  <MediaLink Usage="input" CombinedProcessType="DigitalPrinting"
rRef="L2"/>
<!-- stitched output components -->
  <ComponentLink Usage="output" CombinedProcessType="Stitching"
rRef="L7"/>
  </ResourceLinkPool>
</JDF>
```

3.2.4 Process Nodes

Process nodes represent the very lowest level in a job hierarchy. They may not contain further nested JDF nodes, as every process node is a leaf node. These nodes define the smallest work unit that may be scheduled and executed individually within the JDF workflow model. In figure 3.6, nodes 7-17 represent process nodes.

The various individual process node types are specified in Chapter 6 Processes.

3.3 AncestorPool

The **AncestorPool** element is only required in the root of a spawned job. Spawning and merging is described in section 4.4. The **AncestorPool** element contains an ordered list of one or more **Ancestor** elements, which reflect the family tree of a spawned job. Each **Ancestor** element identifies exactly one ancestor node. The ancestor nodes reside in the original job where the job with the **AncestorPool** has been spawned off. The position of the **Ancestor** element in the ordered list defines the position in the family tree. The first element in the list is the original root element, the last element in the list is the parent, the last but one the grandparent, and so on.

The following table lists the contents of an **AncestorPool** element.

Table 3.4 Contents of the **AncestorPool** element

Name	Data Type	Description
------	-----------	-------------

Name	Data Type	Description
Ancestor +	element	Ordered list of one or more Ancestor elements, which reflect the family tree of a spawned job.

An **Ancestor** element may contain copies of all the attributes of the node that it represents with the exception of the *ID* attribute, which must be copied to the *NodeID* attribute of that **Ancestor** element. **Ancestor** elements cannot, however, contain further sub-elements. The attributes of **Ancestor** elements are described in Table 3.5.

Table 3.5 Attributes of the Ancestor element

Name	Data Type	Description
<i>Activation</i> ?	enumeration	Copy of the <i>Activation</i> attribute from the ancestor node. For details, see Table 3.3.
<i>FileName</i> ?	URL	The URL of the JDF file where the ancestor node resided prior to spawning.
<i>JobID</i> ?	string	Copy of the <i>JobID</i> attribute from the ancestor node. For details, see Table 3.3.
<i>JobPartID</i> ?	string	Copy of the <i>JobPartID</i> attribute from the original ancestor node. For details, see Table 3.3.
<i>NodeID</i>	NMTOKEN ³	Copy of the ID attribute of the ancestor node.
<i>Status</i> ?	enumeration	Copy of the <i>Status</i> attribute from the original ancestor node. For details, see Table 3.3.
<i>Type</i> ?	NMTOKEN	Copy of the <i>Type</i> attribute from the original ancestor node. For details, see Table 3.3.
<i>Types</i> ?	NMTOKENS	Copy of the <i>Types</i> attribute from the original ancestor node. For details, see Table 3.3.
<i>Version</i> ?	string	Copy of the <i>Version</i> attribute from the original ancestor node. For details, see Table 3.3.

3.4 Customer Information

The **CustomerInfo** element contains information about the customer who orders the job. Usually, this element is specified in the uppermost node of a job (that is, the root node), although it is also valid in lower nodes in situations such as model subcontracting. Table 3.6, below, describes the contents of this element.

Table 3.6 Contents of the CustomerInfo element

Name	Data Type	Description
<i>CustomerID</i>	string	Customer identification used by the application that created the job. This is usually the internal customer number of the MIS system that created the job.
<i>CustomerJobName</i>	string	The name that the customer uses to refer to the job.
<i>CustomerOrderID</i>	string	The internal order number for the customer. This number is usually provided when the order is placed and then referenced on the order confirmation or the bill.
<i>Company</i> ?	element	Resource element describing the business or organization of

³ The data type is NMTOKEN and not IDREF because the ID does not reside in the spawned job. The corresponding ID element resides in the original job.

Name	Data Type	Description
		the contact.

3.5 Process and Node Information

The `NodeInfo` element contains information about planned scheduling and message routing. It allows MIS to plan, schedule and invoice jobs or job parts.

Table 3.7 Contents of the `NodeInfo` element

Name	Data Type	Description
<i>CleanupDuration</i> ?	timeDuration	Estimated duration of the clean-up phase of the process.
<i>Currency</i> ?	NMTOKEN	Three digit currency definition according to ISO 4217.
<i>DueLevel</i> ?	enumeration	Description of the severity of a missed deadline. Possible values are: <i>Unknown</i> – Default value. Consequences of missing the deadline are not known. <i>Trivial</i> – Missing the deadline has minor or no consequences. <i>Penalty</i> – Missing the deadline incurs a penalty. <i>JobCancelled</i> – The job is cancelled if the deadline is missed.
<i>End</i> ?	timeInstant	Date and time at which the process is scheduled to end.
<i>FirstEnd</i> ?	timeInstant	Earliest date and time at which the process may end.
<i>FirstStart</i> ?	timeInstant	Earliest date and time at which the process may begin.
<i>LastEnd</i> ?	timeInstant	Latest date and time at which the process may end. This is the deadline to which <i>DueLevel</i> refers.
<i>LastStart</i> ?	timeInstant	Latest date and time at which the process may begin.
<i>Notification-Classes</i> ?	enumerations	Defines the set of notification classes to be logged in the <code>AuditPool</code> of this node. Possible values are: <i>event</i> <i>information</i> <i>warning</i> <i>error</i> <i>fatal</i> For details on <code>Notification</code> elements and classes, see section 3.9.1.2 <code>Notification</code> .
<i>QuoteOption</i> ?	integer	Index of the selected quote option defined in the <code>Quotes</code> attribute and the various <code>Selection</code> elements of <code>Intent</code> resources. Defaults = -1, which means that no option is selected.
<i>Quotes</i> ?	NumberSpan	Pricing of the execution of the node and all its children for a given option.
<i>Route</i> ?	URL	The URL of the controller or device that should execute this node. If URL is not specified, the routing controller must

Name	Data Type	Description
		determine a potential controller or device independently. For details, see section 4.2.
<i>SetupDuration</i> ?	timeDuration	Estimated duration of the setup phase of the process.
<i>Start</i> ?	timeInstant	Date and time of the planned process start.
<i>TargetRoute</i> ?	URL	The URL where the JDF should be sent after completion. If <i>TargetRoute</i> is not specified, it defaults to the input <i>Route</i> attribute of the subsequent node in the process chain.
<i>MergeTarget</i> ?	boolean	If <i>MergeTarget</i> = <i>true</i> and this node has been spawned, it must be merged with its direct ancestor by the controller that executes this node. The path of the ancestor is specified in the last <i>Ancestor</i> element located in the <i>AncestorPool</i> of this node. It is an error to specify both <i>MergeTarget</i> and <i>TargetRoute</i> in one node. Default = <i>false</i> , which means that some other controller will take care of merging.
<i>TotalDuration</i> ?	timeDuration	Estimated total duration of the process, including setup and cleanup.
<i>Employee</i> ?	element	The internal administrator or supervisor that is responsible for the product or process defined in this node.
<i>JMF</i> *	element	Represents JMF query messages that set up a <i>persistent channel</i> , as described in section 5.2.2.3 <i>Persistent Channels</i> . These message elements define the receiver that is designated to track jobs via JMF messages. These message elements should be honored by any JMF-capable controller or device that executes this node. When these messages are honored, a persistent communication channel is established that allows devices to transmit, for example, the status of the job.

3.6 Resources

Resources represent inputs and outputs, the ‘things’ that are produced or consumed by processes. They may be physical, corporeal items such as inks, plates, or glue; electronic items such as files or images; or conceptual items such as parameters and device settings. No matter what their composition, however, they are the tools that JDF uses to link processes to one another, and are contained in the *ResourcePool* element of a node. The *ResourcePool* element is described in the following table.

Table 3.8 Contents of the *ResourcePool* element

Name	Data Type	Description
<i>Resource</i> *	element	List of <i>Resource</i> elements. The <i>Resource</i> elements are abstract and serve as placeholders for any resource.

Like the *Type* attribute in abstract JDF nodes, the *Class* attribute in *Resource* elements helps to identify how particular resources should be used. This attribute contains seven values, and all resources fall under one of these seven classifications. For example, all resources whose *Class* = *Consumable* are physical resources that will be consumed over the course of the process. These values are listed in Table 3.9, below, and are described in greater detail in the sections that follow.

Table 3.9 Contents of the abstract Resource element

Name	Data Type	Description
<i>Class</i>	enumeration	<p>Defines the abstract resource type. For details, see the sections that follow. Possible values are:</p> <p><i>Consumable</i></p> <p><i>Handling</i></p> <p><i>Implementation</i></p> <p><i>Intent</i></p> <p><i>Parameter</i></p> <p><i>Placeholder</i></p> <p><i>Quantity</i></p> <p><i>Selector</i></p>
<i>ID</i>	ID	Unique identifier of a resource.
<i>Locked ?</i>	boolean	<p>If <i>true</i>, the resource is referenced by an <i>Audit</i> and cannot be modified without invalidating the <i>Audit</i>.</p> <p>Default = <i>false</i></p>
<i>PipeID ?</i>	string	If this attribute exists, the resource is a pipe. The <i>PipeID</i> is used by JMF pipe-control messages to identify the pipe. For more information, see section 4.3.2.
<i>rRefs ?</i>	IDREFS	Array of <i>IDs</i> of internally referenced resources.
<i>SpawnStatus ?</i>	enumeration	<p>The spawn status of a node indicates whether or not a node has been spawned, and under what circumstances. Possible values are:</p> <p><i>not_spawned</i> – Default value. Indicates that the resource has not been copied to another process.</p> <p><i>spawned_RO</i> – Indicates that the resource has been copied to another process where it cannot be modified. RO stands for read-only.</p> <p><i>spawned_RW</i> – Indicates that the resource has been copied to another process where it can be modified. RW stands for read/write.</p>
<i>Status</i>	enumeration	<p>The status of a node indicates under what circumstances it may be processed or modified. Possible values are:</p> <p><i>unavailable</i> – Indicates that the resource is not ready to be used or that the resource in the real world represented by the physical resource in JDF is not available for processing.</p> <p><i>draft</i> – Indicates that the resource exists in a state that is sufficient for setting up the next process but not for production.</p> <p><i>available</i> – Indicates that the whole resource is available for usage.</p> <p><i>in_use</i> – Indicates that the resource exists, but is in use by another process. Also used for active pipes (see sections 3.6.3 and 4.3.2).</p>

Figure 3.4 shows the structure of the abstract resource classes defined above.

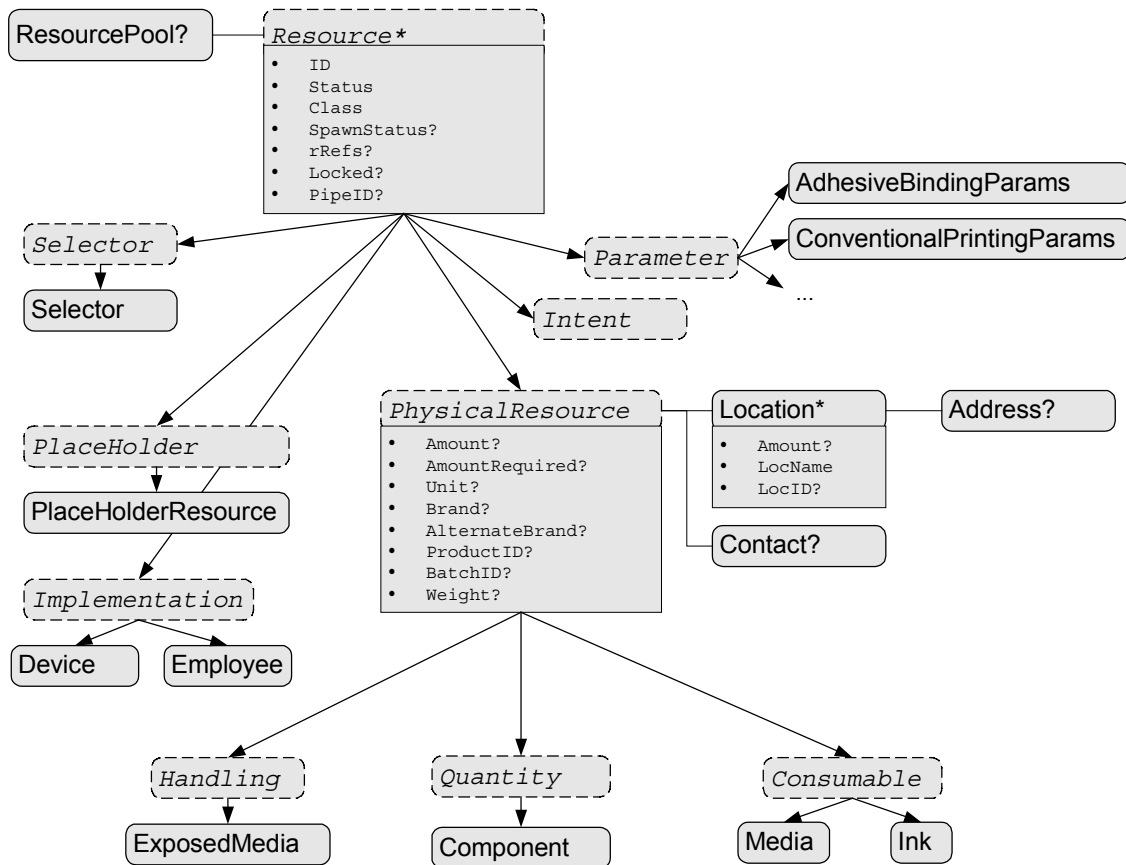


Figure 3.4 Structure of the Abstract Resource Types

3.6.1 Resource Classes

The following sections describe the functions of each of the seven values of the *Class* attribute. All resources fall into one of these classes. In Chapter 7 Resources, the class of each resource is indicated in the Resource Properties sub-heading.

3.6.1.1 Parameter Resources

Parameter resources define the details of processes, as well as any non-physical computer data such as files used by a process. They are usually associated with a specific process. For example, a required input resource of the *ColorSpaceConversion* process is the *ColorSpaceConversionParams* resource. All predefined parameter resources contain the moniker “Params” in their titles. Other examples of *Parameter* resources include **AdhesiveBindingParams** and **ConventionalPrintingParams**.

The abstract *Parameter* resource element contains no attributes or elements besides those contained in the abstract *Resource* element.

3.6.1.2 Intent Resources

Intent resources define the details of products to be produced without defining the process to produce them. In addition, they provide structures to define sets of allowable options and to match these selections with prices. The details of all intent resources are described in section 7.1 *Intent Resources*.

The abstract *Intent* resource element contains no attributes or elements besides those contained in the abstract **Resource** element.

3.6.1.3 Implementation Resources

Implementation resources define the devices and operators that execute a given node. Only two implementation resource types are defined: **Employee** and **Device**, each of which is described in greater detail in the Chapter 7.

Implementation resources can only be used as input resources and may be linked to any process. The abstract *Implementation* resource element contains no attributes or elements besides those contained in the abstract **Resource** element. An example demonstrating how to use implementation resources is provided in section 3.7.2 *Links to Implementation Resources*.

3.6.1.4 Physical Resources (Consumable, Quantity, Handling)

Any resource whose *Class* is *Consumable*, *Quantity*, or *Handling* is considered a physical resource. They are defined as follows:

- *Consumable* resources are resources that are consumed during a process. Examples include **Ink** and **Media**. They are the unmodified inputs in a process chain.
- *Quantity* resources are resources that have been created by a process from either a *Consumable* resource or an earlier *Quantity* resource. For example, printed sheets are cut and a pile of cut blocks is created. **Component** resources are an example of *Quantity* resources.
- A *Handling* resource is used during a process, but is not destroyed by that process. **ExposedMedia** is the only example of such a resource, although it does describe various kinds of items such as film and plates. A *Handling* resource may be created from a *Consumable* resource.

The Table 3.10, below, defines the additional attributes and elements that may be defined for physical resources. The processes that consume physical resources—any kind of physical resource—have the option of using these attributes and elements to determine in what way the resources should be consumed. Table 3.11 then describes the contents of the **Location** sub-element of physical resource elements.

Table 3.10 Additional contents of the abstract physical Resource elements

Name	Data Type	Description
<i>AlternateBrand</i> ?	string	Information, such as the manufacturer or type, about a resource compatible to that specified by the <i>Brand</i> attribute, which is described below.
<i>Amount</i> ?	number	Actual amount of the resource that is available. Note that the amount of consumption and production of a node is specified in the corresponding resource links.
<i>AmountRequired</i> ?	number	Total amount of the resource that is referenced by all nodes that will consume this resource. This corresponds to the sum of all <i>Amount</i> values of input resource links that reference this

Name	Data Type	Description
		resource.
<i>BatchID</i> ?	string	ID of a specific batch of the physical resource
<i>Brand</i> ?	string	Information, such as the manufacturer or type, about the resource being used.
<i>ProductID</i> ?	string	An ID of the resource as defined in the MIS system
<i>Unit</i> ?	NMTOKEN	Unit of measurement for the values of <i>Amount</i> and <i>AmountRequired</i> .
<i>Weight</i> ?	double	Weight of a single component of the resource in grams.
<i>Contact</i> ?	element	If this element is specified, it describes the owner of the resource.
<i>Location</i> *	element	Descriptions of resource locations.

Structure of Location Sub-element

Table 3.11 Contents of the Location element

Name	Data Type	Description
<i>Amount</i> ?	number	Actual amount of the resource at the location.
<i>LocID</i> ?	string	Location identifier within a warehouse system.
<i>LocName</i>	string	Name of the location in MIS, referred by the <i>Location</i> attribute of links to physical resources (see Table 3.16).
<i>Address</i> ?	element	Address of the storage facility. For more information, see section 7.2.2.

3.6.1.5 Placeholder Resources

Placeholder resources, unlike physical resources, do not describe any logical or physical entity. Rather, they define process linking and help to define process ordering when the exact nature of interchange resources is still unknown. In essence, they serve as placeholders that stand in for defined resources. Using *Placeholder* resources, a processing skeleton can be constructed that gives a basic shape to a job. The appropriate resources can be substituted for *Placeholder* resources when they become known.

This kind of resource should only be used to link nodes of *Type* = *ProcessGroup*, since process leaf nodes have well-defined resources that should be used in preference. The only resource whose *Class* = *Placeholder* is called **PlaceholderResource**.

Like *Parameter* and *Implementation* resources, *Placeholder* resources contain no attributes besides those contained in the abstract *Resource* element.

3.6.1.6 Selector Resources

Resources of class *Selector* allow the definition of subsets of resources. The **Selector** resource is the only resource of this class. The way in which they may be used is described in section 3.8.5 Linking to Subsets of Resources.

3.6.2 Position of Resources within JDF Nodes

Resources may exist in any JDF node, but JDF nodes may only reference local or global resources. In other words, JDF nodes may only reference resources in the two kinds of locations: in the node's own **ResourcePool** element or in JDF nodes that are hierarchically closer to the JDF root. An exception to this rule, however, occurs if two independent jobs are merged for a process step and are to be separated afterwards, as is the case when two independent jobs are printed on the same web-fed press. For further details on independent job merging, see section 4.4.5 *Case 5: Spawning and Merging of Independent Jobs*.

It is good practice to put resources into the highest-level node that references the resource. For example, the **RenderingParams** resource should be located in the **Rendering** node, unless it is used by multiple **Rendering** processes, in which case it should be located in the **ProcessGroup** node that contains the **Rendering** process nodes. Resources that link more than one node should be placed in the parent node of the siblings that are linked by the resource.

A process that needs additional detailed process information specifying the creation of a resource must infer this information by explicitly linking to the appropriate parameter resource.

3.6.3 Pipe Resources

Pipes describe links between processes that do not exist as a complete entity at any moment in time. For example, a data stream that is consumed by the subsequent process while it is being written by the previous process is never a wholly complete process in the way that a physical process is complete.

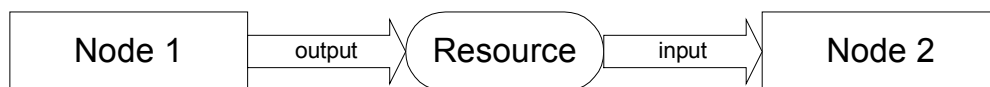
Pipes may also exist for a quantifiable resource, such as an in-line finishing operation on a press that produces sheets. Additional description of pipes and process communication via pipes is provided in section 4.3.2 *Overlapping Processing Using Pipes*.

Resources may contain a string attribute called *PipeID* that declares the resource to be a pipe, and identifies it in a dynamic-pipe messaging environment. A pipe that is also controlled by JMF pipe messages is called **dynamic pipe**. For more information about dynamic pipes, see section 4.3.2.2 *Dynamic Pipes*.

3.7 Resource Links

Not only may a JDF node contain resources that it uses itself, it may also contain resources that are used by its child nodes. Resource links define the resources that are required explicitly by the JDF node in whose **ResourceLinkPool** element the **ResourceLink** elements reside. In other words, a JDF node contains a **ResourceLinkPool** element that in turn contains all of the **ResourceLink** elements that link the node to the resources it uses. They also define whether the resources are inputs or outputs. These inputs and outputs provide conceptual links between the execution elements of JDF nodes. Outputs of one node may in turn become inputs in another node, and a given node may not be executed before all required input resources are available.⁴

Figure 3.4 shows two processes that are linked by a resource. The resource represents the output of Node 1, which in turn becomes an input for Node 2.



⁴ The availability of a resource that is consumed as a whole is given by the **Resource** attribute *Status=available*. In the case of pipe resources, the availability depends on the individual parameter defining the dynamics of a pipe (for details see section 4.3.2 *Overlapping Processing Using Pipes*).

Figure 3.5 Nodes linked by a resource

ResourceLink elements also may contain optional attributes to select a part of a resource, such as only one separation. A detailed description of resource partitioning is given in section 3.8 Subsets of Resources.

ProcessGroup and *Product* nodes may be defined without knowledge of the individual process nodes that define a specific workflow. In this case these intermediate nodes will contain **ResourceLink** elements to the appropriate resources. For example, a prepress node may be defined that produces a set of plates. When the processes for creating the plates are defined in detail, the agent that writes the nodes may remove the **ResourceLink** elements from the intermediate node. Removing the **ResourceLink** specifies that the intermediate node may execute; that is, it may be sent to the appropriate controller or department, even though the specific resources are not yet available. If the **ResourceLinks** are not removed, the intermediate node may not execute until the input resources that are linked are available.

Resource links may be used for process control. For example, if a proof input resource is required for a print process, a print run may only commence when the proof is signed. The JDF format specification also includes a complete specification of how resources are managed when JDF tickets are spawned and merged.

In some cases, determining whether information should be stored in an input or an output resource may be difficult, as the distinction can be ambiguous. For example, is the definition of the color of a separation in the RIP process a property of the output separation or a parameter that describes the RIP process? In order to reduce this ambiguity, the following rules have been applied for the definition of input and output resources of processes as described in Chapter 6 Processes and Chapter 7 Resources:

- Product intent and process parameters are generally input resources, except when one process defines the parameters of a subsequent process.
- *Consumable* resources are always input resources.
- *Quantity* and *Handling* resources are used both as input and output resources. Their usage is defined by the “natural” process usage. For example, a printing plate is described as an **ExposedMedia** resource that is the output of a *ImageSetting* process and the input of a *ConventionalPrinting* process.
- Printed material is exchanged from node to node using the **Component** resource. Product intent nodes also create **Component** output resources.
- Every detailed process description must be defined as an input parameter of the first process where it is referenced. This means that a device may not imply process parameters from its output resources. For example, paper grammage MAY be defined in the sheet output resource of the printing process but MUST be defined as an input parameter of the printing process.
- Any resource parameter that is used must be referenced explicitly. Resource parameters cannot be inferred by following the chain of nodes backwards. This would make spawning of nodes non-local.
- The last process in a chain of processes defines the output resource of its parent process.
- In case of parallel processing, the sum of the outputs of all parallel sub-nodes defines the output of the parent node.

Like **Resource** elements, **ResourceLink** elements are an abstract data type. The class tree of abstract **ResourceLink** elements is further subdivided into classes defined by the *Class* attribute of the resource that it references. Individual instances of **ResourceLink** elements are named by appending the suffix

“Link” to the name of the referenced resource. For example the link to a **Component** resource is entitled **ComponentLink** and the link to a **ScanParams** resource is entitled **ScanParamsLink**.

The following eight abstract resource link classes exist:

- ParameterLink
- ImplementationLink
- ConsumableLink
- QuantityLink
- HandlingLink
- PlaceholderLink
- SelectorLink
- IntentLink

Each listed class name is described in greater detail in the sections that follow. The following figure shows the abstract resource link types derived from the abstract **ResourceLink** type.

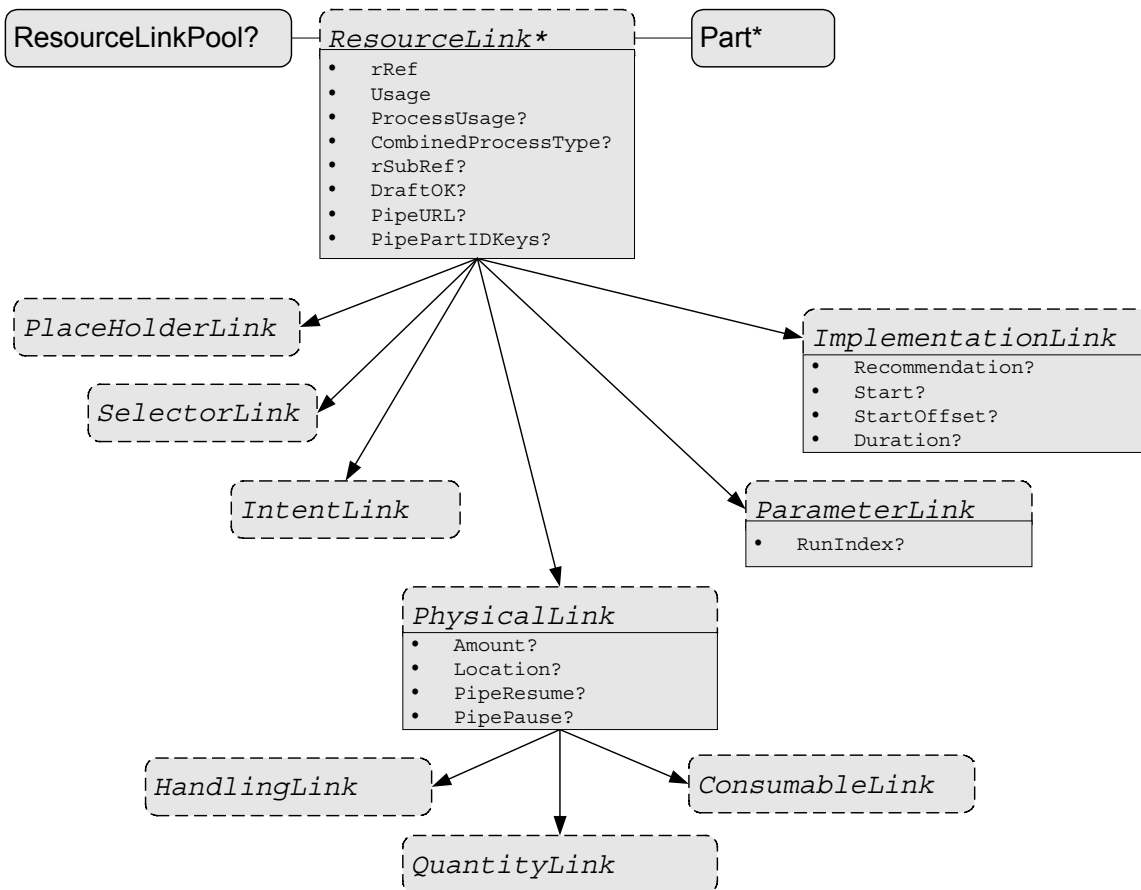


Figure 3.6 Structure of the abstract *ResourceLink* types

The following table lists the contents of a *ResourceLinkPool* element.

Table 3.12 Contents of the *ResourceLinkPool* element

Name	Data Type	Description
------	-----------	-------------

ResourceLink *	element	List of ResourceLink elements. The ResourceLink elements are abstract and are a placeholder for any resource link element.
----------------	---------	--

The following table lists the possible contents of all ResourceLink elements.

Table 3.13 Contents of the abstract ResourceLink element

Name	Data Type	Description
<i>CombinedProcessType</i> ?	NMTOKEN	<i>Combined</i> nodes contain input resources from multiple process nodes. The <i>CombinedProcessType</i> attribute specifies the individual process to which a ResourceLink in a <i>Combined</i> node belongs.
<i>DraftOK</i> ?	boolean	If <i>true</i> , the process may commence with a draft resource. Default = <i>false</i>
<i>PipePartIDKeys</i> ?	NMTOKENS	Defines the granularity of a dynamic pipe for a partitioned resource. For instance, a resource may be partitioned by sheet, surface and separation (resource attribute <i>PartIDKeys</i> = <i>SheetName Side Separation</i>), but pipe requests should only be issued once per surface (resource link attribute <i>PipePartIDKeys</i> = <i>SheetName Side</i>). The contents of <i>PipePartIDKeys</i> must be a subset of the <i>PartIDKeys</i> attribute of the resource that is linked by this ResourceLink. If <i>PipePartIDKeys</i> is not specified, it defaults to <i>PartIDKeys</i> , i.e. maximum granularity. For details on partitioned resources, see section 3.8.2.
<i>PipeURL</i> ?	URL	Pipe request URL. Dynamic pipe requests from this end of a pipe should be made <u>to</u> this URL. ⁵ Note that this URL is only used for initiating pipe requests. Responses to a pipe request are issued to the URL that is defined in the PipePush or PipePull message. For details on using <i>PipeURL</i> , see section 4.3.2.
<i>ProcessUsage</i> ?	string	Identifies the resource usage in the process if multiple resources of the same type are required. For example, this attribute appears when two components—one Cover and one BookBlock—are used in <i>AdhesiveBinding</i> . The allowed values of <i>ProcessUsage</i> are defined in the appropriate process descriptions in Chapter 6 Processes.
<i>rRef</i>	IDREF	Link to the target resource.
<i>rSubRef</i> ?	IDREF	Link to a sub-element within the resource.
<i>Usage</i>	enumeration	Resource usage within this JDF node. Possible values are: <i>input</i> – The resource is an input. <i>output</i> – The resource is an output.
Part *	element	The Part elements identify the parts of a partitioned resource that are referenced by the ResourceLink. The structure of the Part element is defined in Table 3.18.

⁵ Note that in most cases this is the URL of the controller of the *other end* of the pipe. This may seem counterintuitive, but it allows parallel spawning and merging of processes that represent a dynamic pipe without having to include the node that describes the other end in the spawned file.

For details on partitioned resources, see section 3.8.2.

3.7.1 Links to Parameter Resources

Parameter resources are linked by a `ParameterLink` element. If the parameter resource is a **RunList**, an additional subsetting method, indicated by the following attribute, is allowed.

Table 3.14 Contents of the abstract `ParameterLink` element

Name	Data Type	Description
<i>RunIndex</i> ?	IntegerRangeList	Selects a set of runs from a RunList resource.

3.7.2 Links to Implementation Resources

Implementation resources are linked by an `ImplementationLink` element. Using the resource attributes, the link may specify whether the implementation is a recommendation that may be ignored or a request that must be fulfilled. For example, the job may contain a request that the job be run by a specific, experienced operator. If the value of the *Recommendation* is *true* and that operator is ill, he may be replaced by a less experienced operator. If, on the other hand, a product could be created on a device that theoretically can do the job but does not produce sufficient quality, and if it is certain that customer will reject inferior quality, *Recommendation* should be set to *false*.

Since implementation `ResourceLinks` define the usage of a specific device during the course of a job, situations can arise where that resource is not required during the whole processing time. For instance, a forklift that only has to transport the completed components is not required to be available during the entire process run, only during the times when it is needed. This means that, contrary to the general rule that all resources must be *available* for node execution to commence, a node may commence when implementation resources are still *in_use* by other processes if *Start* or *StartOffset* are specified.

`ImplementationLink` elements always have a *Usage* of *input*.

Table 3.15 Contents of the abstract `ImplementationLink` element

Name	Data Type	Description
<i>Duration</i> ?	timeDuration	Estimated duration during which the resource will be used.
<i>Recommendation</i> ?	boolean	If <i>true</i> and the request cannot be fulfilled, the change may be logged as a Modified Audit and the job may continue. If <i>false</i> , an error occurs if the request is not fulfilled. Default = <i>false</i>
<i>Start</i> ?	timeInstant	Time and date when the usage of the implementation resource starts.
<i>StartOffset</i> ?	timeDuration	Offset time when the resource is required after processing has begun. If both <i>Start</i> and <i>StartOffset</i> are specified, <i>Start</i> has precedence.

The following example shows how the operator Smith is linked to a `ConventionalPrinting` process as the only valid operator:

```
<ResourcePool>
```

```

    <Employee PersonalID="007" ID="L1" Class="Implementation">
      <Person FamilyName="Smith" JobTitle="Press Operator">
    </Employee>
  </ResourcePool>
  ...
  <ResourceLinkPool>
    <EmployeeLink Recommendation="false" rRef="L1"/>
  </ResourceLinkPool>

```

3.7.3 Links to Physical Resources

The physical resources that fall into the *Consumable*, *Quantity*, and *Handling* classes are linked, predictably, by the appropriate *ConsumableLink*, *QuantityLink*, or *HandlingLink* resource link elements. Just as physical resources inherit the contents of the abstract resource element, physical resource links inherit the contents of the abstract resource link element. They may, however, contain additional contents. These optional attributes are described in Table 3.16, below.

Table 3.16 Additional contents of the abstract physical *ResourceLink* element

Name	Data Type	Description
<i>Amount</i> ?	number	Amount of the resource that is required by the process, in units as defined in the resource. Allows resources to be only partially consumed or produced (see section <i>Resource Amount</i>).
<i>Location</i> ?	string	Refers to a definite location of the resource (see <i>LocName</i> in Table 3.11).
<i>PipePause</i> ?	number	Parameter for controlling the pausing of a process if the resource amount in the pipe buffer passes the specified value. For details on using <i>PipePause</i> , see 4.3.2.
<i>PipeResume</i> ?	number	Parameter for controlling the resumption of a process if the resource amount in the pipe buffer passes the specified value. For details on using <i>PipeResume</i> , see section 4.3.2.
<i>RemotePipeEnd-Pause</i> ?	number	Parameter for controlling the pausing of a process at the other end of the pipe if the resource amount in the pipe buffer passes the specified value. For details on using <i>RemotePipeEndPause</i> , see section 4.3.2.
<i>RemotePipeEnd-Resume</i> ?	number	Parameter for controlling the resumption of a process at the other end of the pipe if the resource amount in the pipe buffer passes the specified value. For details on using <i>RemotePipeEndResume</i> , see section 4.3.2.

3.7.4 Links to Placeholder Resources

Placeholder resources are linked by a *PlaceholderLink* element. *Placeholder* links, used together with the **PlaceholderResource** resource, can be employed to predefine a skeleton of a processing network consisting of process group nodes without knowing the exact nature of the interchange resources. For instance, although the deadlines for the job may be known, it may not be known whether a press run shall be defined for a digital press or a conventional press.

3.7.5 Links to Selector Resources

Selector resources are linked by a **SelectorLink** element. Note that the name of the abstract resource link class **SelectorLink** is identical to the name of the resource link to the **Selector** resource. **Selector** resources are described together with subsets of resources in section 3.8.5 *Linking to Subsets of Resources*.

3.7.6 Links to Intent Resources

Intent resources are linked by a **IntentLink** element. They have no additional parameters.

3.7.7 Inter-Resource Linking

In some cases, it is necessary to reference resources directly from other resources in order to reuse information. The linking element retains its name but has the syntax of a **ResourceLink** with a mandatory *rSubRef* attribute to define the target element.

In order to enable spawning and merging without having to scan every single resource, inter-resource links must be specified in an *rRefs* attribute of the resource. In the case of a link to a resource subset, the *rRefs* attribute contains a reference to the atomic resource. Even if a resource is linked more than once, one occurrence of that resource in the *rRefs* array is sufficient.

Elements within a resource may also contain an *ID* attribute. These elements may be explicitly referenced by a **ResourceLink**. The **ResourceLink** element has an optional *rSubRef* attribute that contains an IDREF to the sub-element of the resource.

The following example demonstrates inter-resource linking.

Example:

```
<Layout rRefs="res1 res2">
  ...
  <Surface rRef="res1" rSubRef="surf1"/>
  <Surface rRef="res2" rSubRef="surf2"/>
  <Surface rRef="res1" rSubRef="surf1"/>
  <!-- another link to the same resource -->
</Layout>
<Sheet ID="res1">
  <Surface ID="surf1" ... />
</Sheet>
<Sheet ID="res2">
  <Surface ID="surf2" ... />
</Sheet>
```

3.8 Subsets of Resources

In many cases, a set of similar resources—such as separation films, plates, or **RunList** resources—is produced by one process and consumed by another. When this occurs, it is convenient to define one resource element that describes the complete set and allows individual sub-sets to be referenced. This mechanism also removes process ambiguity if multiple input resource links and multiple output resource links exist that must be unambiguously correlated.

Resource elements and **ResourceLink** elements have optional attributes that enable an agent to specify an explicit part of a structured resource.

3.8.1 Resource Amount

Yet another flexible feature of resources is that they may be only partially consumed. For example, in a scenario in which various versions of a product share identical parts—such as versioned books that all have the same cover—each version will only use as many copies of the cover as it needs to fulfill its job requirement, even though all of the covers can be printed in one step for all versions. This feature is specified in the *Amount* attribute of the resource links and allows multiple JDF nodes to share resources. It allows both the sharing of output resources (as when a binding process consumes identical sheets from multiple press lines) and the sharing of input resources (as when the covers for multiple jobs are identical and are all printed in one press run).

The *Amount* attribute of a physical resource element contains the actual amount of a given resource. It is adjusted by the production or consumption amount of every process that is executed, and refers to that amount in the corresponding physical resource link element. Thus the value of the *Amount* attribute of a resource that is consumed as an input should be reduced by the amount that is consumed. It is up to the agent that writes a JDF job to ensure that the *Amount* attributes of resources and the resource links that reference them are consistent. The units used in the *Amount* attribute of a physical resource link element is defined by the unit of the resource element the to which the link refers.

The definition of *Amount* for partitioned resources is explained in detail in section 3.8.2 Description of Partitionable Resources.

3.8.2 Description of Partitionable Resources

Printing workflows contain a number of processes that are repeated over a potentially large number of individual sheets, surfaces or separations. In order to define a partitioned resource in a concise manner without having to create a large number of individual nodes and resources, these resources may be defined as one resource with individual nested parts.

A partitionable resource contains nested elements, each with the same name as the resource. The part-independent resource elements and attributes are located in the root of the resource, while the partition-dependent elements are located in the nested elements. Thus one individual part is defined by the convolution of the partition-independent elements and attributes, with the elements and attributes contained in the appropriate nested elements. The attributes of nested part elements may be overwritten by the equivalent attributes in descendent parts.

The *Amount* attribute of a partitioned resource is treated formally exactly in the same manner as any other attribute. This implies that the amount specified refers to the amount defined by one leaf and not to the amount defined by the sum of leaves in a branch. The *Amount* attribute defined in the example below is, therefore, two, even though 24 physical plates are described.

The following example defines two sets of 12 plates for 2 sheets with 3 surfaces. Each has a common brand attribute called “Gooley.” Each individual separation has its own *ProductID*. Furthermore, the *Status* attribute varies from part to part. For example, if a yellow plate breaks, only it will need to be remade and therefore set to *unavailable*; the others, meanwhile, may remain *available*.

```
<ExposedMedia Class="Handling" Type="plate" Brand="Gooley" ID="L1"
Status="available" PartIDKeys="SheetName Side Separation" Parts="12"
Amount="2" >
  <ExposedMedia SheetName="S1">
    <ExposedMedia Side="Front">
      <ExposedMedia Separation="Cyan" ProductID="S1FCPlateJ42"/>
      <ExposedMedia Separation="Magenta" ProductID="S1FMPlateJ42"/>
      <ExposedMedia Separation="Yellow" ProductID="S1FYPlateJ42"
Status="unavailable"/>
    </ExposedMedia>
  </ExposedMedia>
</ExposedMedia>
```

```

    <ExposedMedia Separation="Black" ProductID="S1FKPlateJ42"/>
  </ExposedMedia>
  <ExposedMedia Side="Back">
    <ExposedMedia Separation="Cyan" ProductID="S1BCPlateJ42"/>
    <ExposedMedia Separation="Magenta" ProductID="S1BMPlateJ42"/>
    <ExposedMedia Separation="Yellow" ProductID="S1BYPlateJ42"/>
    <ExposedMedia Separation="Black" ProductID="S1BKPlateJ42"/>
  </ExposedMedia>
</ExposedMedia>
<ExposedMedia SheetName="S2" Side="Front">
  <ExposedMedia Separation="Cyan" ProductID="S2FCPlateJ42"/>
  <ExposedMedia Separation="Magenta" ProductID="S2FMPlateJ42"/>
  <ExposedMedia Separation="Yellow" ProductID="S2FYPlateJ42"/>
  <ExposedMedia Separation="Black" ProductID="S2FKPlateJ42"/>
</ExposedMedia>
</ExposedMedia>

```

Note that only resources may be partitioned. If a resource contains sub-elements, the sub-elements may **NOT** be individually partitioned.

Two examples are provided below. The first example is valid, the second is invalid. In the first example, the **ExposedMedia** resource is partitioned:

```

<ExposedMedia ID="L1" Status="available" PartIDKeys="Separation" ... >
  <Media MediaType="Film"/>
  <ExposedMedia Separation="Cyan">
    <Media Brand="foo"/>
  </ExposedMedia >
  <ExposedMedia Separation="Magenta">
    <Media Brand="bar"/>
  </ExposedMedia >
</ExposedMedia >

```

In this invalid example, **Media** is a sub-element that may **NOT** be partitioned:

```

<ExposedMedia ID="L1" Status="available" PartIDKeys="Separation" ... >
  <Media MediaType="Film">
    <Media Brand="foo" Separation="Cyan">
    <Media Brand="bar" Separation="Magenta" />
  </Media >
</ExposedMedia >

```

In addition to the usual resource attributes and elements, the partitionable **Resource** element has the following partition-specific attributes and elements in its root:

Table 3.17 Contents of the partitionable Resource element

Name	Data Type	Description
------	-----------	-------------

<i>PartIDKeys</i> ?	NMTOKENS	List of attribute names that are used to separate the individual parts. Possible NMTOKEN values are : <i>PartVersion</i> <i>Separation</i> <i>SheetName</i> <i>Side</i> <i>SignatureName</i> <i>TileID</i> For details, see Table 3.18.
<i>Parts</i> ?	integer	Number of parts that the resource represents.
Resource *	element	Nested resource elements that contain the appropriate part ID(s). These elements must be of the same type as the root Resource element. They represent the individual parts or groups of parts.

Partitionable resources are uniquely identified by the attribute values listed in *PartIDKeys* attributes. The choice of which attributes to use depends on how the agent organizes the job.

The following table lists the content of a **Part** element, which contains a set of pre-defined attributes that have a well-described meaning. Each of the attributes, except *Sorting*, may be used in the nested resource elements of partitionable resources as the part ID key (see example above).

Table 3.18 Contents of the Part element

Name	Data Type	Description
<i>PartVersion</i> ?	string	Version identifier, such as the language version of a catalog.
<i>Separation</i> ?	string	Identifies the separation name. The predefined values are: <i>Composite</i> – Non-separated resource. <i>Separated</i> – The resource is separated, but the separation definition is handled internally by the resource, such as a PDF file that contains <i>SeparationInfo</i> dictionaries. <i>Cyan</i> – Process color. <i>Magenta</i> – Process color. <i>Yellow</i> – Process color. <i>Black</i> – Process color.
<i>SheetName</i> ?	string	A string that uniquely identifies each sheet. The value of this attribute must match the value of the <i>SheetName</i> attribute of a sheet.

<i>Side</i> ?	enumeration	Denotes the side of the sheet. Possible values are: <i>Front</i> <i>Back</i> If <i>Side</i> is specified, the Part element refers to one surface of the sheet. If it is not specified, it refers to both sides.
<i>SignatureName</i> ?	string	A string that uniquely identifies the signature within the partitionable resource.
<i>Sorting</i> ?	IntegerRangeList	Mapping from the implied partitionable resource order to a process order. The indices refer to the elements of the complete partitionable resource, not to the index in the selection of parts defined by the Part element. ⁶
<i>SortAmount</i> ?	boolean	If a sorted resource has an <i>Amount</i> attribute and <i>SortAmount</i> = <i>true</i> , each resource shall be processed completely. If <i>SortAmount</i> = <i>false</i> (the default), each Part element shall be processed the number of times specified in the <i>Amount</i> attribute before starting the next Part .
<i>TileID</i> ?	XYPair	XYPair of integer values that identifies the tile. Tiles are identified by their X and Y indexes. Values are zero-based and expressed in the PS coordinate system. So “0 0” is the lower left tile and “1 0” is the tile next to it on the right. Tile resources are described in detail in the section 7.2.90 Tile.

3.8.3 Locations of Physical Resources

Unlike other kinds of resources, physical resources may be stored at multiple, distributed locations. This is specified by including one or more **Location** elements in the resource element and accessing the location by specifying a *Location* attribute in the respective **ResourceLink**.

The following example describes a set of plates that are distributed over two locations:

```
<ExposedMedia ID="L1" Type="Plate" ... >
  <Location Amount="42" LocName="Desk Drawer 1">
    <Address ... />
  </Location>
  <Location Amount="100" LocName="Desk Drawer 2" LocID="PP_01234">
    <Address ... />
  </Location>
</ExposedMedia>
...
<ExposedMediaLink ResourceID="L1" Location="Desk Drawer 2" Amount="50"
Usage="input"/>
```

⁶ Note that *Sorting* is semantically different from the other attributes in this table, as it implies an ordering of parts, whereas the other attributes define a selection of parts.

3.8.4 RunIndex

The *RunIndex* attribute selects a set of logical pages from a **RunList** in a manner that is independent of the internal structure of the **RunList**. It contains an array of mixed ranges and individual indices separated by whitespace. Each range consists of two indices connected with a tilde (~) and no whitespace. For example, *RunIndex* = "2~5 8 10 22~-1".

Negative numbers reference pages from the back of a file in base 1 counting. In other words, -1 is the last page, -2 the second to last, etc. Thus *RunIndex* = "0~-1" refers to a complete range of pages, from first to last.

3.8.5 Linking to Subsets of Resources

There are two ways in which an agent can select a subset of a resource. The first is to modify the **ResourceLink** element and the second is to add an additional *Selector* resource. In the first method, a **Part** element and/or *RunIndex* attribute is included in a **ResourceLink** element in order to define a specific subset of a resource. In the second method, a JDF node selects a subset of a resource by linking to an optional **Selector** resource. Thus a node that links to a partitionable resource, such as a set of plates, and to a **Selector** resource that, for example, contains only a **Part** element describing a *Yellow* separation is effectively referencing the *Yellow* plate. This is the method of choice if a subset is selected for a whole process network. The subset selection of all processes for a complete network can be changed by locally modifying one selector resource.

Only one selector resource may be linked by a node. **Selector** resources affect only partitionable resources or **RunList** resources.

The following table describes the contents of the **Selector** resource.

Table 3.19 Contents of the Selector resource

Name	Data Type	Description
<i>RunIndex?</i>	IntegerRangeList	Selects a set of runs from a RunList resource.
Part +	element	One or more individual Part elements. Each one selects a part from a partitioned resource. If multiple parts are specified, the Selector resource describes the sum (boolean or) of these parts.

The following example demonstrates how a **Selector** resource can be used with a partitioned resource. The *Cyan* and *Magenta* plate set is described by the combination of the partitioned **ExposedMedia** resource from the previous example and the **Selector** resource.

```
<Selector ID="L2" Class="Selector" Status="available">
  <Part Separation="Cyan"/>
  <Part Separation="Magenta"/>
</Selector>
...
<ResourceLinkPool>
<ExposedMediaLink rRef="L1" Usage="input"/> <!--This links to the complete plate
resources.-->
<SelectorLink rRef="L2" Usage="input"> <!-- This subsets to the Cyan and Magenta plate.-
->
</ResourceLinkPool>
```


Partitionable hierarchies define an implied ordering of the individual parts. In the example in section 3.8.2 Description of Partitionable Resources, the first element has a *ProductID* = *S1FCPlateJ42* and the last has a *ProductID* = *S2FKPlateJ42*. If process ordering of a partitionable resource is important, the *Part* element of the *ResourceLink* or the *Part* element inside of the **Selector** must specify a *Sorting* attribute. If *Sorting* is not specified, process ordering is arbitrary. If *Sorting* is specified multiple times, the resolution of the sorting must be unambiguous.

The *Sorting* attribute maps the implied part ordering to a specified process ordering in a 0-based list. The first entry in *Sorting* defines the first entry to be processed. The following example, using a *ResourceLink* element, describes how the plates described in the previous example could be ordered by separation for the first sheet followed by the complete second sheet, in reverse order (back to front). Each set of two plates, as specified in the *Amount* attribute of the resource, would be processed together.

```
<ExposedMediaLink rRef="L1">
  <Part Sorting="0 4 1 5 2 6 3 7 -1~8" SortAmount="false" />
</ExposedMediaLink>
```

A partitionable resource may also be split into individual resources by an agent. In this case, one resource must be created for each individual part or set of parts. For example, a resource that describes a set of films that are also separated may be split into a set of resources that each describe all separations of a sheet.

3.8.6 Splitting and Combining Resources

Depending on the circumstances, it may be appropriate either to split a resource into multiple new nodes or to specify multiple locations or parts for an individual resource. There are four possible methods for splitting and combining resources, each of which is illustrated in Figure 3.7, below. Both Case A and Case B in Figure 3.7 represent workflows that use the *Amount* attribute of their resource links to share resources. This method is practical when one controller controls all aspects of resource consumption or production. In Case A, the resource amount is split between subsequent processes. In Case B, individual processes produce amounts that are then combined into a unified resource that is, in turn, used by a single process. In both cases, a single, shared resource is employed. To enable independent parallel processing by multiple controllers, however, independent resources are required. To create independent resources from one resource, the **Split** process is used, as shown in Case C (for further details, see section 6.2.6 Split). This process allows multiple processes to be spawned off, after which multiple processes can consume the same resource in parallel and may therefore run in parallel. Case D demonstrates the reverse situation, which occurs if resources have been produced by multiple processes and are then consumed, as a unified entity, by a single subsequent process. To accomplish this, the **Combine** process (described in section 0

Combine) combines multiple resources to create the single resource.

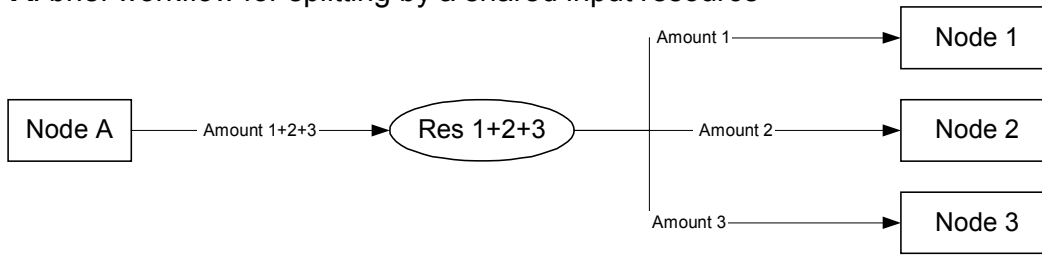
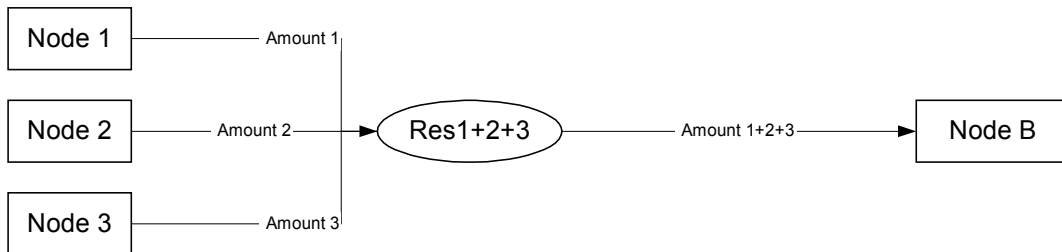
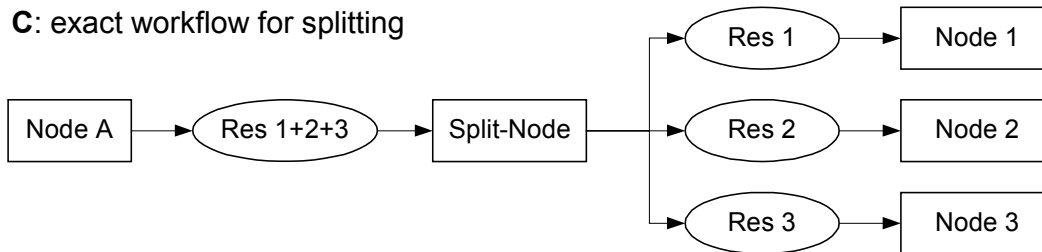
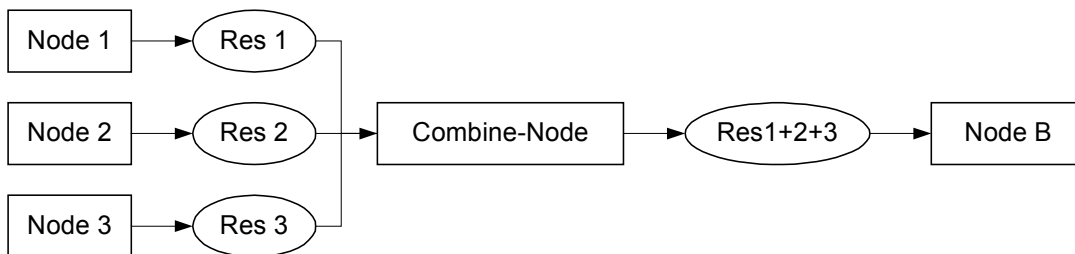
A: brief workflow for splitting by a shared input resource**B: brief workflow for combining by a shared output resource****C: exact workflow for splitting****D: exact workflow for combining**

Figure 3.7 Splitting and combining physical resources

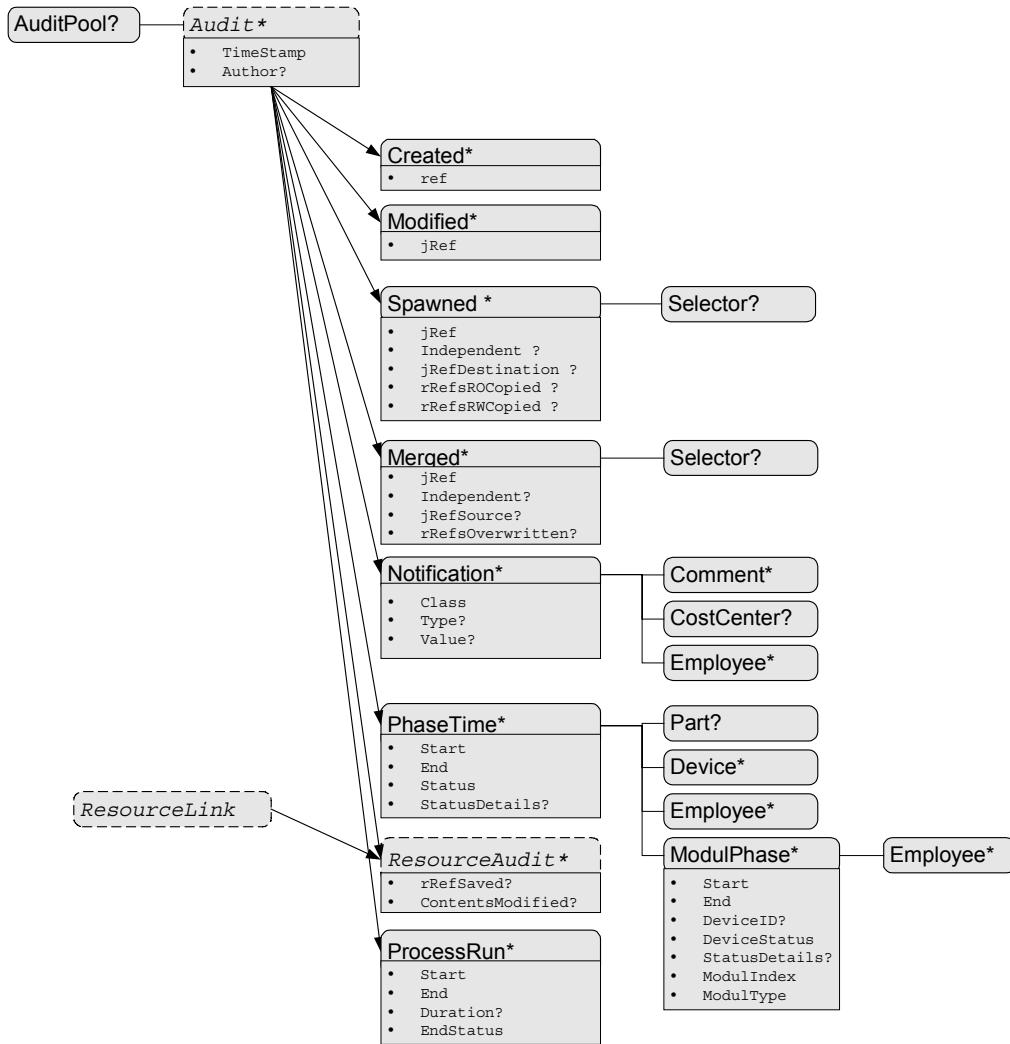
3.9 AuditPool

Audit elements contain the post-facto recorded results of a process. Audit elements are static and cannot ever be modified after the process has been started. Therefore, if Audit elements link to resources, those resources should be locked in order to inhibit accidental modification of audited information, which is why JDF includes a locking mechanism for resources.

Audit elements record any event related to the following situations:

1. The creation of a JDF node by a **Created** element.
2. Spawning and merging, including resource copying by spawned and merged elements.
3. Errors such as unnecessary **ResourceLink** elements, wrongly linked resources, missing resources, or missing links, which may be detected by agents during a test run or by a **Notification** element.
4. Actual data about the production and resource consumption by a **ResourceAudit** element.
5. Any process phase times. Examples include setting up a device, maintenance, and washing, as well as down-times as a result of failure, breaks, or pauses. Changes of implementation resource usage, such as a change of operators by a **PhaseTime** element, would also constitute an example of a phase time.
6. Actual process scheduling data. For example, the process start and end times, as well as the final process state, as determined by a **ProcessRun** element.
7. Any modification of a JDF node not covered by the preceding items, as recorded by a **Modified** element.

Audit information may be used by MIS for operations such as evaluation or invoicing. Figure 3.8 depicts the structure of the **AuditPool** and **Audit** element types derived from the abstract audit type.



Attributes:
 ref = reference via ID to a resource or a JDF-node
 jRef = reference via ID to a JDF-node
 Notification:
 Class = event | information | warning | error | fatal

Figure 3.8 Structure of Audit element types derived from the abstract Audit type

Audit entries are ordered chronologically, with the last entry in the AuditPool representing the newest. A ProcessRun element containing the scheduling data finalizes each process run. All subsequent entries belong to the next run. The following table defines the contents of the AuditPool element.

Table 3.20 Contents of the AuditPool element

Name	Data Type	Description
Audit *	element	Chronologically ordered list of Audit elements. The Audit elements are abstract and serve as placeholders for any audit. Audit elements are described in the sections that follow.

Audit elements are described in greater detail in the following sections.

3.9.1 Audit Elements

All Audit elements inherit the content from the abstract Audit data type, described in the following table.

Table 3.21 Contents of the abstract Audit type

Name	Data Type	Description
<i>Author ?</i>	string	Text that identifies who made the entry. This can describe a person, an agent, or both.
<i>TimeStamp</i>	timeInstant	In case of the audits Created , Modified , Spawned , Merged , and Notification , this attribute records the date and time when the related event occurred. In case of the audits PhaseTime , ProcessRun , and ResourceAudit , the attribute describes the time when the entry was appended to the audit pool.

Listed in the following sections are the elements derived from the abstract Audit type. Following the description of each element is a table outlining the attributes associated with that element.

3.9.1.1 ProcessRun

This element serves two related functions. Its first is to summarize one complete execution run of a node. It contains attributes that record the date and time of the start, the end time, the final process state when the run is finished, and, optionally, the process duration of the process run. These attributes are described in Table 3.20.

Table 3.22 Contents of the ProcessRun element

Name	Data Type	Description
<i>Duration ?</i>	timeDuration	Time span of the effective process runtime without intentional or unintentional breaks. That time span is the sum of all process phases when the status is <i>in_progress</i> , <i>setup</i> or <i>cleanup</i> .
<i>End</i>	timeInstant	Date and time at which the process ends.
<i>EndStatus</i>	enumeration	The Status of the process at the end of the run. For a description of process states, see Table 3.3 Contents of a JDF node. Possible values are: <i>quoted</i> <i>ready</i> <i>failed_testrun</i> <i>completed</i> <i>aborted</i>
<i>Start</i>	timeInstant	Date and time at which the process starts.

The second function of a **ProcessRun** element is to delimit a group of audits for each individual process run. Every group of audits terminates with a **ProcessRun** element, which contains the information described above. If a process must be repeated (as a result of a late change in the order, for example), all audits belonging to the new run will be appended after the last **ProcessRun** element that terminates the

audits of the previous run. The number of **ProcessRun** elements is therefore always equivalent to the number of process runs.

Even if a node describes partitioned resources, only one **ProcessRun** should be specified. Details about the individual part processing times are logged in **PhaseTime** elements.

3.9.1.2 Notification

This element contains information about individual events that occurred during processing. For a detailed discussion of event properties, see section 4.6 *Error Handling*.

Table 3.23 Contents of the Notification element

Name	Data Type	Description
Class	enumeration	Class of the notification. Possible values, in order of severity from lowest to highest, are: <i>event</i> – Indicates that an event due to any activity has occurred, for example, machine events, operator activities, etc. This class is used for the transfer of conventional event messages. In case of Class = <i>event</i> , further event information should be provided by the attributes Type and Value . <i>information</i> – Any information about a process which cannot be expressed by the other classes. No user interaction is required. <i>warning</i> – Indicates that a minor error has occurred and an automatic fix was applied. Execution continues. <i>error</i> – Indicates that an error has occurred that requires user interaction. Execution cannot continue. <i>fatal</i> – Indicates that a fatal error led to abortion of the process.
Type ?	NMTOKEN	Identifies the event type. For a list of supported event types, see Appendix I.
Value ?	string	Contains a value associated with the event. For a list of data types associated to supported event types, see Appendix I.
Comment *	telem	The Notification element may contain Comment elements with a verbose, human-readable description of the event. If the value of the Class attribute is one of <i>information</i> , <i>warning</i> , <i>error</i> , or <i>fatal</i> , it should provide at least one Comment element. In case of Class = <i>event</i> , Comment elements are optional.
CostCenter ?	element	The cost center to which this event should be charged.
Employee *	element	The Employee associated with this event.

3.9.1.3 PhaseTime

This element contains audit information about the start and end times of any process states and sub-states, denoted as phases. Phases may reflect any arbitrary subdivisions of a process, such as maintenance, washing, plate changing, failures, and breaks.

PhaseTime elements may also be used to log the actual time spans when implementation resources are used by a process. For example, the temporary necessity of a fork lift can be logged if a **PhaseTime**

element is added that contains a link to the fork-lift device resource and specifies the actual start and end time of the usage of that fork lift.

The times specified in the **PhaseTime** elements should not overlap with each other and should cover the complete time range defined in the **ProcessRun** element that identifies the end of the run.

Table 3.24 Contents of the *PhaseTime* element

Name	Data Type	Description
<i>End</i>	timeInstant	Date and time of the end of the phase.
<i>Start</i>	timeInstant	Date and time of the beginning of the phase.
<i>Status</i>	enumeration	Status of the phase. Possible values of JDF node states are: <i>setup</i> <i>in_progress</i> <i>cleanup</i> <i>spawned</i> <i>stopped</i> The states listed above are a subset of the possible states of a JDF node. For all possible states of a JDF node see Table 3.3. The remaining set of states— <i>quoted</i> , <i>ready</i> , <i>failed_testrun</i> , <i>aborted</i> and <i>completed</i> —must be logged by the ProcessRun audit element that terminates the list of audits for one process run.
<i>StatusDetails</i> ?	string	Description of the status phase that provides details beyond the enumerative values given by the Status attribute. For a list of supported values, see Appendix F.
Device *	element	Links to Device resources that are working during this phase.
Employee *	element	Links to Employee resources that are working during this phase.
ModulePhase *	element	Additional phase information of individual device modules, such as print units.
Part ?	element	Describes which part of a job is currently being logged. If Part is not specified for a node that modifies partitioned resources, PhaseTime refers to all parts. For example, imagine a print job that should produce 3 different sheets. All sheets are described by one partitioned resource. In order to separate the different print phases for each sheet, the Part element defines, unambiguously, the sheet to which the audit refers.

It is possible to monitor the states of individual modules of a complex device, such as a printer with multiple print units, by defining **ModulePhase** elements. One **PhaseTime** element may contain multiple **ModulePhase** elements and can therefore record the status of multiple units in a device. In contrast to **PhaseTime** audit elements **ModulePhase** elements are allowed to overlap in time with one another. **ModulePhase** elements are defined in the following table.

Table 3.25 Contents of the *ModulePhase* element

Name	Data Type	Description
<i>DeviceID</i>	string	Name of the device. This must be the <i>DeviceID</i> attribute of one of the Device elements specified in the PhaseTime audit.

Name	Data Type	Description
DeviceStatus	enumeration	Status of the device module. Possible values are: <i>idle</i> – The module is not used, for example, a color print module that is inactive during a black-white print. <i>down</i> – The module cannot be used. It may be broken, switched off etc. <i>setup</i> – The module is currently being set up. <i>running</i> – The module is currently executing. <i>cleanup</i> – The module is currently being cleaned. <i>stopped</i> – The module has been stopped, but running may be resumed later. This status may indicate any kind of break, including a pause, maintenance, or a breakdown, as long as running can be easy resumed. These states are analogical to the device states of Table 5.38.
End	timeInstant	Date and time of the end of the module phase.
ModuleIndex	IntegerRange List	0-based indices of the module or modules. If multiple module types are available on one machine, it is device dependent whether the indices of each type restart at 0 or simply continue indexing.
ModuleType	NMTOKEN	Module description. The allowed values depend on the type of device that is described. The predefined values are listed in Appendix G.
Start	timeInstant	Date and time of the beginning of the module phase.
StatusDetails ?	string	Description of the module status phase that provides details beyond the enumerative values given by the DeviceStatus attribute. For a list of supported values, see Appendix F.
Employee *	element	Links to Employee resources that are working during this module phase on this module (the module is specified by the attributes ModuleIndex and ModuleType).

3.9.1.4 ResourceAudit

The abstract **ResourceAudit** element describes the usage of resources during execution of a node. It logs consumption and production amounts of any quantifiable resources, accumulated over one process run or one part of a process run.

A **ResourceAudit** element inherits all of the contents of the **ResourceLink** sub-class to which the resource to be logged belongs, as well as the contents of the abstract **Audit** type. For example, the **QuantityAudit** element may contain an **Amount** attribute that specifies the logged amount and that may also contain an optional **Part** element that defines the part of a partitioned resource that was produced. The element name of the audit depends on the **ResourceLink** sub-class from which it is derived as follows:

- **ParameterAudit** – Inherits from **ParameterLink** and **Audit**.
- **HandlingAudit** – Inherits from **HandlingLink** and **Audit**.
- **QuantityAudit** – Inherits from **QuantityLink** and **Audit**.
- **ConsumableAudit** – Inherits from **ConsumableLink** and **Audit**.
- **ImplementationAudit** – Inherits from **ImplementationLink** and **Audit**.

Note that in contrast to **ResourceLink** elements, where the *Class* attribute of the linked resource defines an abstract **ResourceLink** element, the class of **ResourceAudit** elements defines the actual element name.

If the contents of a resource are modified, an optional *rRefSaved* attribute exists that can reference a copy of the original resource. If the original resource does not need to be saved, a boolean *ContentsModified* attribute should be used to indicate that a change has been made.

Table 3.26 Contents of the ResourceAudit element

Name	Data Type	Description
<i>ContentsModified</i> ?	boolean	Specifies that a modification has occurred but that the original resource has been deleted.
<i>rRefSaved</i> ?	IDREF	<p>If specified, this attribute represents the ID of the resource before modification and before consumption or production. This ID must be generated by the agent when the resource copy is created. The element must reside in the node where the referenced resource resides.</p> <p>If a locked resource should be modified, then the resource must be copied, and the attribute <i>rRefSaved</i> refers to the locked resource instance and the <i>rRef</i> attribute to the modified resource instance.</p> <p>The usage of <i>rRefSaved</i> together with <i>rRef</i> allows to retrace resource modifications. In other words, it enables undo operations.</p>

For details on **ResourceLink** elements and **ResourceLink** sub-classes, see section 3.7 Resource Links. The partitioning of resources using **Part** elements is defined in section Description of Partitionable Resources.

3.9.1.5 Logging Machine Data by Using the ResourceAudit.

If a resource is modified during processing, any nodes that also reference the resource may also be affected. The following logging procedure is recommended in order to track the resource modification and to insure consistency of the job:

1. Create a copy of the original resource with a new ID.
2. Modify the original resource to reflect the changes.
3. Insert a **ResourceAudit** element that references the modified original resource with the *rRef* attribute and the copied resource with the *rRefSaved* attribute.

The following example describes the logging of a modification of the media weight and amount. The JDF document before modification:

```
<JDF ... >
  <ResourceLinkPool>
    <MediaLink rRef="RLink" Usage="input" Amount="400"/>
  </ResourceLinkPool>
  <ResourcePool>
    <Media weight="80" ID="RLink"/>
  </ResourcePool/>
</JDF>
```

The JDF after modification:

```
<JDF ... >
  <ResourceLinkPool>
    <MediaLink rRef="RLink" Usage="input" Amount="400"/>
  <!--note that the ResourceLink has not changed -->
  </ResourceLinkPool>
  <ResourcePool>
    <Media weight="80" ID="RPrev"/> <!--Copy of the original resource-->
    <Media weight="90" ID="RLink"/> <!--modified resource-->
  </ResourcePool/>
  <AuditPool>
    <ConsumableAudit rRef="RLink" rRefSaved="RPrev" Usage="input "
Amount="421"/>
  </AuditPool>
</JDF>
```

3.9.1.6 Created

This element allows the creation of a JDF node or resource to be logged. If the element refers to a JDF node, it can be located in the **AuditPool** element of the node that has been created or in any ancestor node. If the element refers to a resource it must be located in the node where the resource resides so that the spawning and merging mechanism can work effectively.

Table 3.27 Contents of the Created element

Name	Data Type	Description
<i>ref?</i>	IDREF	Represents the ID of the created element.

3.9.1.7 Modified

This element allows any modifications affecting a JDF node, such as changes made to the **NodeInfo** element or **CustomerInfo** element, to be logged. Changes that can be logged by other audit element types, such as resource changes, must not use this common log entry.

The modification can be described textually by adding a generic **Comment** element to the **Modified** element. The location of the element in the node tree is the same as the location of the corresponding **Created** element.

Table 3.28 Contents of the Modified element

Name	Data Type	Description
<i>jRef</i>	IDREF	The ID of the modified node. The modified element resides in the modified node.

3.9.1.8 Spawned

This element allows a job that has been spawned to be logged in the **AuditPool** of the parent node of the spawned job-part. For details about spawning and merging, see section 4.4 **Spawning and Merging**.

Table 3.29 Contents of the Spawned element

Name	Data Type	Description
<i>Independent ?</i>	boolean	Declares that independent jobs that have previously been merged into a big job are spawned. If it is set to <i>true</i> , the attributes <i>jRefDestination</i> , <i>rRefsROCopied</i> and <i>rRefsRWCopied</i> have no meaning and should be omitted. Default = <i>false</i>
<i>jRef</i>	IDREF	ID of the JDF node that has been spawned.
<i>jRefDestination ?</i>	NMTOKEN	ID of the JDF node to which the job has been spawned. ⁷ This attribute must be specified in the parent of the original node if independent jobs are spawned.
<i>rRefsROCopied ?</i>	IDREFS	List of IDs separated by whitespaces. Identifies the resources copied to the ResourcePool element of the spawned job during spawning. These resources should <u>not</u> be modified by the spawned job.
<i>rRefsRWCopied ?</i>	IDREFS	List of IDs separated by white spaces. Identifies the resources copied to the ResourcePool element of the spawned job during spawning. These resources may be modified by the spawned job and must be copied back into their original location by the merging agent. Resource copying is required if resources are referenced simultaneously from spawned nodes and from nodes in the original JDF document.
<i>Selector ?</i>	element	Identifies the parts that were selected for spawning in case of parallel spawning of partitionable resources (see section 4.4.3). Note that this copy may <i>NOT</i> contain an ID attribute.

3.9.1.9 Merged

This element logs a merging event of a spawned job. For more details, see section 4.4 Spawning and Merging.

Table 3.30 Contents of the Merged element

Name	Data Type	Description
<i>Independent ?</i>	boolean	Declares that independent jobs are merged into a big job for common production. If it is set to <i>true</i> , the attributes <i>jRefSource</i> and <i>rRefsOverwritten</i> have no meaning and should be omitted. Default = <i>false</i>
<i>jRef</i>	IDREF	ID of the JDF node that has been returned or merged.
<i>jRefSource ?</i>	NMTOKEN	ID of the JDF root node of the big job from which the spawned structure has been returned. ⁸

⁷ The data type is NMTOKEN and not IDREF because the attribute refers to an external ID.

<i>rRefsOverwritten</i> ?	IDREFS	Identifies the copied resources that have been overwritten during merging. Resources are usually overwritten during return if they have been copied during spawning with read/write access.
Selector ?	element	Specifies the selected parts of the resource that were merged in case of parallel spawning and merging of partitionable resources (see section 4.4.3). Note that this copy may <i>NOT</i> contain an ID attribute.

3.10 JDF Extensibility

JDF is meant to be flexible and therefore useful to any vendor, as each vendor will have specific data to include in the JDF files. JDF is able to provide this kind of versatility by using the XML namespaces. This chapter describes how JDF uses the XML extension mechanisms.

3.10.1 Namespaces in XML

JDF Extensibility is implemented using XML Namespaces. The Namespaces in XML specification is found at <http://www.w3.org/TR/REC-xml-names/>.

XML namespaces are defined by *xmlns* attributes. A general example is provided below. The example illustrates how private namespaces are declared and used to extend an existing JDF resource by adding private attributes and a private element.

```
<JDF xmlns="JDFSchema URI" xmlns:foo="fooschema URI" ... >
...
  <SomeJDFDefinedResource name="abc" foo:specialname="cba">
    ...
    <foo:PrivateStuff type=""/>
    ...
  </SomeJDFDefinedResource>
...
</JDF>
```

Namespaces are inserted in front of attribute and element names. The associated namespace of element names with no prefix is the default namespace defined by the *xmlns* attribute. The associated namespace of attributes with no prefix is that one of the element (see Appendix A.2 XML Namespace Partitions in the specification Namespaces in XML). All namespaces prefixes must be declared by *xmlns:xxx* attributes.

3.10.2 Extending Process Types

JDF defines a basic set of process types. Because JDF allows flexible encoding, however, this list, by definition, will not be complete. Vendors that have specific processes that do not fit in the general JDF processes and that are not combinations of individual JDF processes (see section 3.2.3 Combined Process Nodes) can create JDF process nodes of their own type. Then the content of the *Type* attribute may be specified with a prefix that identifies the organization. The prefix and name should be separated by a single colon (':') as shown in the following example:

```
<JDF Type="myCompaniesNS:MyVeryImportantProcess" xmlns="JDFSchema URI"
xmlns:myCompaniesNS="my companies namespace URI" ... >
```

⁸ The data type is NMTOKEN and not IDREF because the attribute refers to an external ID.

...
</JDF>

If a process is simply an extension of an existing process, it is possible to describe the private data by extending the existing resource types and/or creating new resource types, which are linked to pre-defined JDF Process nodes. This is described in greater detail in the sections below.

Extending the `NodeInfo` and `CustomerInfo` nodes is achieved in a manner analogous to the extension of resources, which is described below. On the other hand, extending the direct contents of JDF nodes by adding new elements or attributes is discouraged.

3.10.3 Extending existing Resources

All resources defined by JDF may be extended by adding attributes and elements using an own namespace for these resource extensions. This is useful when the pre-defined resource types need only a small amount of private data added, or if those resources are the only appropriate place to put the data. The namespace of the resource extended must not be modified. However, the mechanism for creating new resources in a separate namespace is provided in the next section.

This does not mean that duplicate functionality may be added into these resource types. You must make sure to use the JDF-defined attributes and elements where possible and extend them with additional information that cannot be described using JDF-defined constructs. For example, it is not allowed to extend the RIP resource that controls the resolution with a `foo:Resolution` or `foo:Res` attribute that overrides the JDF defined resolution parameter (see attribute *Resolution* of resource **RenderingParams** in section 7.2.77).

3.10.4 Creating New Resources

There are certain process implementations that have functionality that cannot be specified by the pre-defined Resource types. In these cases, it is necessary to create a new Resource-type element, which must be clearly specified using its own namespace. These resource types can be linked to both pre-defined and custom type JDF process nodes.

3.10.5 Future JDF Extensions

In future versions, certain private extensions will become more widely used, even by different vendors. As private extensions become canonical, those extensions will be candidates for inclusion in the next version of the JDF specification.

At that time the specific extensions will have to be described and will be included into the JDF namespace.

3.10.6 Maintaining Extensions

Given the mix of vendors that will use JDF, it is likely that there will be a number of private extensions. Therefore, JDF controllers must be prepared to receive JDF files that have extensions. These controllers can and should ignore all extensions they don't understand, but under no circumstance are they allowed to remove these extensions when making modifications to the JDF. If they do, it will break the extensibility mechanism. For example, imagine that JDF Agent A creates a JDF and inserts private information for Process P. Furthermore, the information is only understood by agent A and the appropriate device D for executing P. If the JDF needs to be processed first by another Agent/Device C, and that process removes all private data for P, Process P will not be able to produce the correct results on device D that were specified by Agent A.

3.10.7 Processing Unknown Extensions

If a node is processed by a controller or device and it encounters an unknown extension in one of its input resources, a Notification element with *Class* = *warning* should be logged.

3.10.8 Derivation of Types in XML Schema

The XML Schema definition <http://www.w3.org/TR/xmlschema-1/> describes a mechanism to create new types by derivation from old types. This is an alternative to extend or create new elements and is described in section 4 of <http://www.w3.org/TR/xmlschema-0/>. This mechanism is not allowed to be applied to any elements defined by JDF because such new element types can only be understood by the agent that made the extension.

The use of the derivation mechanism is allowed only for private extensions but not required.

Chapter 4 Life Cycle of JDF

This chapter describes the life cycle of a JDF job, from creation through modification to processing. Information is provided about the spawning of individual aspects of jobs and in what way they are re-incorporated into the job once the process is completed. Ancillary aspects of the life cycle, such as test running and error handling, are also discussed.

4.1 Creation and Modification

The life cycle of a JDF job will likely follow one of two scenarios. In the first scenario, a job is created all at once, by a single agent, and then is consumed by a set of devices. More often, however, a job is created by one agent and is then transformed, or modified, over time by a series of other agents. This process may require specification of product intent, which is defined in section 4.1.1, below.

Jobs can be modified in a variety of ways. In essence, any job is modified as it is executed, since information about the execution is logged. The most common instance of modification of a JDF job, however, occurs during processing, when more detailed information is learned or understood and then added along the way. This information may be added because an agent knows more about the processing needed to achieve some result specified in a JDF node than the original, creating agent knew. For example, one agent may create a product node that specifies the product intent of a series of pages. This product node may include information about the number of pages and the paper properties. Another node may then be inserted that includes a resource describing how the pages should be RIPped. Later, another agent may provide more detail about the RIPping process by appending optional information to the RIP parameter resource.

Regardless of where in the life cycle they are written, nodes and their required resources must be valid and include all mandatory information in order to have a *Status* of *ready* or *available*. This restriction allows for the definition of incomplete output resources. For example, a URL resource without a file name may be completed by a process. On the other hand, it is impossible to define a valid and executable node with insufficient input parameters.

Once all of the inputs and parameters for the process requested by a node are completely specified, a controller can route the JDF job containing this node to a device that can execute the process. When the process is completed, the agent/controller in charge of the device will modify the node to record the results of the process.

4.1.1 Product Intent Constructs

JDF jobs, in essence, are requests made by customers for the production of quantities of some product or products. In other words, a job begins with a particular goal in mind. In JDF, product goals are often specified by using a construct known as product intent, represented by intent resources. In contrast to process resources that define precise values, intent resources allow ranges or sets of preferred values to be specified. Resources of this kind include **FinishingIntent**, **ColorIntent**, **MediaIntent**, and **ScreeningIntent**, all of which are described in Chapter 7 Resources.

The product intent of a job is like a plan of action. The plan may be extremely vague, detailing only the general goal, or it may be very specific, stipulating the specific requirements inherent in meeting that goal. Product intent may be defined for an end product about which little is known or about which the processing details for the job are entirely unknown. Product intent constructs also allow agents to describe jobs that comprise multiple product components, and that may share some parts.

Product intent is defined by the initiating agent of a job. It is not mandatory, however. Many JDF jobs are written with full knowledge of the necessary processes, and are therefore comprised entirely of the various kinds of process nodes described in sections 3.2.1, 3.2.2, and 3.2.3. Any job that specifies product intent, however, must include nodes whose *Type* = *Product*. This representation is described in the following section.

4.1.1.1 Representation of Product Intent

The product description of a job is a hierarchy of *Product* nodes, and the bottom-most level of the product hierarchy represents portions of the product that are each homogeneous in terms of their materials and formats. All nodes below these *Product* nodes begin specifying the processes required to produce the products.

Product nodes are required to contain only one thing, and that is a resource that represents the physical result specified by the node. This resource is generally a **Component**. In addition, somewhere in the hierarchy of product nodes, it is a good idea to include an intent resource to describe the characteristics of the intended product. Although these are the only resources that should occur, *Product* nodes can contain multiple resources. For example, some *ResourceTypes*, such as **MediaIntent** and **SizeIntent**, are defined to provide more general mechanisms to specify product intent.

In some cases, more than one higher-level product node will use the output of a product node. These higher-level nodes represent the combination of homogeneous product parts. In this case, the *Amount* attribute of the *ResourceLinks* that connect the nodes will identify how the low-level product is shared.

4.1.1.2 Representation of Product Binding

Some product intent nodes, such as **BindingIntent**, define how to combine multiple products. To accomplish this, the respective **Component** resources must be labeled according to their usage. For example, the *Cover* and *Insert* attributes use the *ProcessUsage* attribute of the respective resource links.

For more information about product intent, see section 3.2.1 Product Intent.

4.1.2 Note Generation [RP2] Using Intent Resources

One potential use for defining product intent is to help in estimating cost quotes for customers. The customer may not know any more than the broad strokes of what he is looking to accomplish, but may want a cost estimate based on what he does know. For example, a customer may want a 4-color brochure with stapled bindings, but he may not know the size it should be or the kind of paper to use. A product intent node can be constructed that fills in potential details and provides an estimate for that job, and the customer can be given a set of quotes based on specific intent. Options are labeled by an integer index and inserted into the appropriate intent resources as **Selection** elements, and the actual price is inserted into a **Selection** element of the **NodeInfo** element. This allows multiple quotes to be specified within one JDF node.

The following example shows an RFQ for a simple job¹ and a set of three quote options—one for each colormodel—as a response. The dimensions of the product is selected to be 16 pages 8.5*15 paper for all quotes (specified in **black bold**). The returned quote JDF specifies one quote for each *ColorModel* (BW, Process4, Process6) defined in the range.

Request for Quote

¹ Note that the job is incomplete and that the resource examples were chosen only to illustrate the method of defining multiple quote options within a job.


```

<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="L00" Type="Product" JobID="JobID" Status="waiting"
Version="0.9" Activation="RFQ">
  <NodeInfo Currency="DEM"/>
  <ResourcePool>
    <Component ID="Link0002" Class="Quantity" Status="unavailable"
DescriptiveName="complete 16-page Brochure"/>
    <SizeIntent ID="Link0003" Class="Intent" Status="available">
<Height Range="720~864" DataType="NumberSpan" Preferred="792"/>
      <Width Range="612~720" DataType="NumberSpan"/>
      <Pages DataType="IntegerSpan" Preferred="16"/>
    </SizeIntent>
    <ColorIntent ID="Link0004" Class="Intent" Status="available">
      <ColorModel Range="Process4 BW Process6" DataType="NameSpan"/>
    </ColorIntent>
  </ResourcePool>
  <ResourceLinkPool>
    <ComponentLink rRef="Link0003" Usage="output" Amount="10000"/>
    <SizeIntentLink rRef="Link0004" Usage="input"/>
    <ColorIntentLink rRef="Link0005" Usage="input"/>
  </ResourceLinkPool>
</JDF>

```

Returned Quote

```

<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="L00" Type="Product" JobID="JobID" Status="quoted"
Version="0.9" Activation="RFQ">
  <NodeInfo Currency="DEM">
    <Quotes DataType="NumberSpan">
      <NumberSelection Index="0" Actual="2000"/>
      <NumberSelection Index="1" Actual="4000"/>
      <NumberSelection Index="2" Actual="6000"/>
    </Quotes>
  </NodeInfo>
  <ResourcePool>
    <Component ID="Link0002" Class="Quantity" Amount="10000"
Status="unavailable" DescriptiveName="complete 16-page Brochure"/>
    <SizeIntent ID="Link0003" Class="Intent" Status="available">
      <Height Range="720~864" DataType="NumberSpan" Preferred="792">
        <NumberSelection Index="0~-1" Actual="792"/>
      </Height>
      <Width Range="612~720" DataType="NumberSpan">
        <NumberSelection Index="0~-1" Actual="612"/>
      </Width>
      <Pages DataType="IntegerSpan" Preferred="16">
        <IntegerSelection Index="0~-1" Actual="16"/>
      </Pages>
    </SizeIntent>
    <ColorIntent ID="Link0004" Class="Intent" Status="available">
      <ColorModel Range="Process4 BW Process6" DataType="NameSpan">
        <NameSelection Index="0" Actual="BW"/>
        <NameSelection Index="1" Actual="Process4"/>
        <NameSelection Index="2" Actual="Process6"/>
      </ColorModel>
    </ColorIntent>
  </ResourcePool>
</JDF>

```

```

</ResourcePool>
<ResourceLinkPool>
  <ComponentLink rRef="Link0002" Usage="output"/>
  <SizeIntentLink rRef="Link0003" Usage="input"/>
  <ColorIntentLink rRef="Link0004" Usage="input"/>
</ResourceLinkPool>
</JDF>

```

Selection of the Quote

The following JDF is returned to the printer by the buyer and specifies, that he wants option 2 selected. The original quote is truncated, since the information already resides with the printer.

```

<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="HDM20000919091206" Type="Product" JobID="JobID"
Status="quoted" Version="0.9" Truncated="NodeInfo AuditPool"
Activation="RFQ">
  <NodeInfo LastEnd="2000-09-20T00:38:01+02:00" Currency="DEM"
QuoteSelection="2">
    <Quotes DataType="NumberSpan">
      <NumberSelection Index="0" Actual="2000"/>
      <NumberSelection Index="1" Actual="4000"/>
      <NumberSelection Index="2" Actual="6000"/>
    </Quotes>
  </NodeInfo>
  <AuditPool>
    <Created Author="Rainer's JDFWriter 0.2000" TimeStamp="2000-09-
19T09:12:06+02:00"/>
  </AuditPool>
</JDF>

```

4.1.3 Specification of Delivery of End Products

A job may define one or more products and specify a set of deliveries of those end products. To accomplish this, a node of *Type = Product* is created to define each delivery mode to be made. A delivery contains a set of drops, which in turn contain a set of packages. Each drop has a common delivery address and each package contains the amount of an individual **Component** that is to be delivered to this address.

The following example defines a fairly complex delivery scenario of three pre-manufactured components with two mutually exclusive delivery options. The **blue** components (ID=Link0004-Link0006) describe the actual components that should be delivered. These would be the output of high level product nodes in a complete JDF. The **green** node (ID=Link0002) describes a simple delivery option where all components are picked up at one drop. The **red** node (ID=Link0003) defines a delivery mode with three drops, each containing varying amounts of the individual components.

```

<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="Deliv" Type="Product" JobID="Delivery" Status="waiting" Version="0.9"
ChildCombination="or">
  <NodeInfo/>
  <ResourcePool>
    <Component ID="Link0004" Class="Quantity" Amount="10000" Status="available"
DescriptiveName="First Product"/>
    <Component ID="Link0005" Class="Quantity" Amount="10000" Status="available"
DescriptiveName="Second Product"/>
    <Component ID="Link0006" Class="Quantity" Amount="10000" Status="available"
DescriptiveName="Third Product"/>
  </ResourcePool>

```

```

<JDF ID="Link0002" Type="Product" Status="waiting" DescriptiveName="First Delivery
Option" >
  <NodeInfo/>
  <ResourcePool>
    <DeliveryIntent ID="Link0007" Class="Intent" rRefs="Link0004 Link0005 Link0006"
Status="available">
      <NameSpan Name="Method" Preferred="PickUp"/>
      <!--Simple Drop that links to all three components-->
      <DropIntent Earliest="2000-09-19T01:17:32+02:00">
        <Company>
          <Contact>
            <Address City="Kiel" Country="Germany" PostalCode="24113"/>
          </Contact>
        </Company>
        <PackageIntent rRef="Link0004"/>
        <PackageIntent rRef="Link0005"/>
        <PackageIntent rRef="Link0006"/>
      </DropIntent>
    </DeliveryIntent>
  </ResourcePool>
  <ResourceLinkPool>
    <DeliveryIntentLink rRef="Link0007" Usage="input"/>
  </ResourceLinkPool>
</JDF>
<JDF ID="Link0003" Type="Product" Status="waiting" DescriptiveName="Second Delivery
Option">
  <NodeInfo/>
  <ResourcePool>
    <DeliveryIntent ID="Link0008" Class="Intent" rRefs="Link0004 Link0005 Link0006"
Status="available">
      <NameSpan Name="Method" Range="SurfaceMail"/>
      <!--Drop that links to all of the first component-->
      <DropIntent Earliest="2000-09-21T08:50:52+02:00">
        <Company>
          <Contact>
            <Address City="Hamburg" Country="Germany" PostalCode="24000"/>
          </Contact>
        </Company>
        <PackageIntent rRef="Link0004">
          <IntegerSpan Name="Amount" Preferred="10000"/>
        </PackageIntent>
      </DropIntent>
      <!--Drop that links to a part of the last two components-->
      <DropIntent Earliest="2000-09-21T08:50:52+02:00">
        <Company>
          <Contact>
            <Address City="Paris" Country="France" PostalCode="75008"/>
          </Contact>
        </Company>
        <PackageIntent rRef="Link0005">
          <IntegerSpan Name="Amount" Preferred="4000"/>
        </PackageIntent>
        <PackageIntent rRef="Link0006">
          <IntegerSpan Name="Amount" Preferred="3000"/>
        </PackageIntent>
      </DropIntent>
      <!--Drop that links to the rest of the last two components-->
      <DropIntent Earliest="2000-09-14T10:10:52+02:00">
        <NameSpan Name="Method" Range="AirMail"/>
        <Company>
          <Contact>
            <Address City="San Francisco" Country="US" PostalCode="77777"/>
          </Contact>
        </Company>
        <PackageIntent rRef="Link0005">
          <IntegerSpan Name="Amount" Preferred="6000"/>
        </PackageIntent>
        <PackageIntent rRef="Link0006">
          <IntegerSpan Name="Amount" Preferred="7000"/>
        </PackageIntent>
      </DropIntent>
    </DeliveryIntent>
  </ResourcePool>
  <ResourceLinkPool>
    <DeliveryIntentLink rRef="Link0008" Usage="input"/>
  </ResourceLinkPool>
</JDF>

```

```

    </DeliveryIntent>
  </ResourcePool>
</ResourceLinkPool>
  <DeliveryIntentLink rRef="Link0008" Usage="input"/>
</ResourceLinkPool>
</JDF>
</JDF>

```

For more information, see section 6.2.3 Delivery.

4.2 Process Routing

A controller in a JDF workflow system has two tasks. The first is to determine which of the nodes in a JDF document are executable, and the second is to route these nodes to a device that is capable of executing them. Both of these procedures are explained in the sections that follow.

In a distributed environment with multiple controllers and devices, finding the right device or controller to execute a specific node may be a non-trivial task. Systems with a centralized, smart master controller may want to route jobs dynamically by sending them to the appropriate locations. Simple systems, on the other hand, may have a static, well-defined routing path. Such a system may, for example, pass the job from hot-folder to hot-folder. Both of these extremes are valid examples of JDF systems that have no need for additional routing meta-data.

In order to accommodate systems between these extremes, the `NodeInfo` element of a node contains optional *Route* and *TargetRoute* attributes that let an agent define a static process route on a node-by-node basis. If no *Route* or *TargetRoute* attribute is specified, and if a controller has multiple options where to route a job, it is up to the implementation to decide which route to use.

The controller or device reading the JDF job is responsible for processing the nodes. A device examines the job and attempts to execute those nodes that it knows how to execute, whereas a controller routes the job to the next controller or device that has the appropriate capabilities.

4.2.1 Determining Executable Nodes

In order to determine which node should be executed, the controller/device uses the following procedures:

1. First, it searches the JDF document for node types it can execute by comparing the *Type* attribute of the node to its own capabilities, and by determining the *Activation* of the nodes. It should also verify that the *Status* of the node is either *waiting* or *ready*.
2. The controller/device may then determine whether all of the input resources of the respective nodes have a *Status* of *available* and not a *SpawnStatus* of *spawned_RW*, and that all processes that are attached through pipes are ready to execute. A controller may optionally skip this check and expect the lower-level controller or device that it controls to perform this step and return with an error if it fails.
3. Finally, if scheduling information is provided in the `NodeInfo` element, the specified start and/or end time must be taken into account by the executing device. If no process times are specified, it is up to the device in charge of queue handling to execute the process node.

4.2.2 Distributing Processing to Work Centers or Devices

JDF syntax supports two means of distributing processes to work centers or devices. Its first option is to use a so-called “smart” controller that has the ability to parse a JDF job and identify individual processes or process groups that may be distributed to a particular work center or device. This smart controller may use

spawning and merging facilities to sub-divide the job ticket and pass specific instructions to a work center or device.

The second option, which is applicable when the controller being used isn't "smart," is to employ a simple controller implementation that routes the entire job to each workcenter or device, thus leaving it up to the recipient to determine which processing it can accomplish. For this option to work, each JDF-capable device must be able to identify process nodes it is capable of executing. Furthermore, each device must have sufficient JDF-handling capabilities to identify processes that are ready to run.

4.2.3 Device / Controller Selection

The method used to determine which is the appropriate device or lower-level controller to use to execute a given node depends greatly on the implemented workflow being used. Although JDF provides a method for storing routing information in the *Route* attribute of the *NodeInfo* element of a node, it does not prescribe any specific routing methods. However, some of the tools available to figure out alternative workflows are described below.

Knowledge of the capabilities of lower level controllers/devices either may be hard-wired into the system or gained using the *KnownJDFServices* message. Since JDF does not yet provide mechanisms to determine whether a given device is capable of processing a node without actually performing a test run, a controller must either have a priori knowledge of the detailed capabilities of devices that it controls or it must perform a test run to determine whether a device is capable of executing a node. Furthermore, in addition to the explicit routing information in the *Route* attribute of the *NodeInfo* element of a node, JDF may contain implicit routing information in the form of **Device** implementation resources.

JMF defines the *KnownControllers* query to find controllers and the *KnownDevices* query to find devices that are controlled by a controller. The information provided by these queries can be used by a controller to infer the appropriate routing for a node. In a system that does not support messaging, this information must be provided outside of JDF.

4.3 Execution Model

JDF provides a range of options that help controllers tailor a processing system to the needs of the workflow and of the job itself. The following sections explain the ways in which controllers execute processes using these various options.

The processing model of JDF is based on a producer/consumer model, which means that the sequencing of events is controlled by the availability of input resources. As has been described, nodes act both as producers and consumers of resources. When all necessary inputs are available in a given node, and not before, the process may execute. The sequence of processing, therefore, is implied by the chain of resources in which the output resources of one node become the input resources of a subsequent node.

JDF supports four kinds of process sequences: serial processing, overlapping processing, parallel processing, and iterative processing. All four are described in the following sections.

4.3.1 Serial Processing

The simplest kind of process routing, known as serial processing, executes nodes sequentially and with no overlap. In other words, no nodes are executed simultaneously. Once the process has acted upon the resource in some way, the resource availability is described by the *Status* attribute of the resource, as described above. When the process state is *ready* or *waiting*, the process can begin executing.

In a workflow using serial processing, the controller is responsible for comparing the actual amount available with the specified amount in the corresponding **PhysicalLink** element to determine whether or not the input resource can be considered available. If no amount is specified in the **PhysicalLink**, the process is assumed to consume the entire resource.

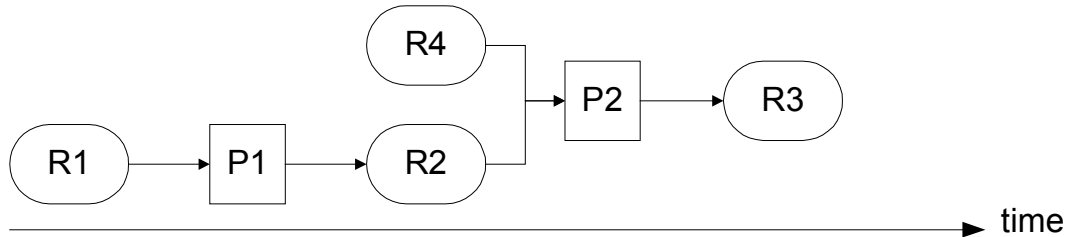


Figure 4.1 Example of a simple process chain linked by resources

Figure 4.1 depicts a simple process chain that produces and consumes *Quantity* resources and uses an implementation resource. The resources R1, R2, and R3 represent *Quantity* resources. Process P1 consumes resource R1 and produces resource R2. R2 is then completely consumed by P2, which also requires the implementation resource R4 for processing. Process P2 uses these two resources and produces resource R3. All of this is accomplished along a linear time axis.

Table 4.1, which follows, shows the value of the **Status** attribute of each of the resources and processes used in Figure 4.1. The time axis runs from left to right both in Figure 4.1 and in Table 4.1. Note that no process may execute until all resources leading up to that process are in place. In other words, the job executes serially and sequentially. For more information about the values of the **Status** attribute of resources, see Table 3.9. For more information about the values of the **Status** attribute of processes, see Table 3.3.

Table 4.1 Examples of resource and process states in the case of simple process routing

Object Status	before running P1	during running P1	after running P1, before P2	during P2	after P2
resource R1	<i>available</i>	<i>in_use</i>	<i>unavailable</i>	<i>unavailable</i>	<i>unavailable</i>
resource R2	<i>unavailable</i>	<i>unavailable</i>	<i>available</i>	<i>in_use</i>	<i>unavailable</i>
resource R3	<i>unavailable</i>	<i>unavailable</i>	<i>unavailable</i>	<i>unavailable</i>	<i>available</i>
resource R4	<i>available</i>	<i>available</i>	<i>available</i>	<i>in_use</i>	<i>available</i>
process P1	<i>waiting or ready</i>	<i>in_progress</i>	<i>completed</i>	<i>completed</i>	<i>completed</i>
process P2	<i>waiting or ready</i>	<i>waiting or ready</i>	<i>waiting or ready</i>	<i>in_progress</i>	<i>completed</i>

When the attribute **Amount** is used in connection with the quantifiable resources R1, R2, or R3 and their links, then the controller must decide whether or not a resource is available by comparing the individual values. If the amounts are used to define the availability, then the resource **Status** may be set to *available* for all *Quantity* resources. Note that when the value of the **Status** attribute of the resource is *unavailable* the resource is not available, even if a sufficient amount is specified.

If amounts are specified in the resource element, they represent the actual available amount. If they are not specified, the actual amount is unknown, and it is assumed that the process will consume the entire resource. Amounts of **PhysicalLink** elements must be specified for output resources that represent the intended production amount. The specification of the **Amount** attribute for input resources is not required, although it can be specified. If the controller cannot determine the amounts, this constitutes a JDF content error, which is logged by error handling. This process is described in section 4.6 Error Handling.

If a process in a serial processing run does not finish successfully, the final process status is designated as *aborted*. In an aborted job, only a part of the intended production may be available. If this occurs, the actual produced amount is logged into the audit pool by a resource audit element.

4.3.2 Overlapping Processing Using Pipes

Whereas pipes themselves are identified in the resource that represents the pipe, pipe dynamics are declared in the resource links that reference the pipe. This allows multiple nodes to access one pipe, each of them with its own pipe buffering parameters.

In some situations, resource linking is a continuous process rather than a chronological one. In other words, one process may require the output resources of another process before that process has completely finished producing them. The ability to accomplish this kind of resource transfer is known as overlapping processing, and it is accomplished with the use of a mechanism known as pipes. Pipes are considered to be **active** if any process linking to the pipe simultaneously consumes or produces that pipe resource.

Any resource may be transferred into a pipe resource. All that is required is that the *PipeID* attribute be specified in the resource. Pipes of quantifiable resources resemble reservoir tanks that hang between processes. Processes connected to the pipe via output links fill the tank with necessary resources, while processes connected via input links deplete it (see Figure 4.2). The level is controlled by the *PhysicalLink* attributes *PipeResume*, *PipePause*, *RemotePipeEndPause*, and *RemotePipeEndResume* (see Table 3.16). If none of them are specified, any produced *Quantity* may be immediately consumed by the consuming end of the pipe. The unit of the buffers is defined by the *Unit* attribute of the resource.

The two following diagrams show the ways in which pipes mediate between the process producing the resource and the process consuming the resource. The following optional attribute values are defined for pipes: *PipePartIDKeys*, *PipePause*, *PipeResume*, *RemotePipeEndPause*, and *RemotePipeEndResume*. The latter two—*RemotePipeEndPause* and *RemotePipeEndResume*—are used to control the level in context with pipe command messages which will be described in section 4.3.2.2 *Dynamic Pipes*. The specified value of each of these attributes in any given node dictates the levels at which a pipe should resume or pause execution. Figure 4.3 gives an example of a view on the dynamics of a pipe resource. The available level of the pipe resource, represented as R2, and the availability status of two entity resources, represented as R1 and R3, are changing along a consistent time line. Below the progressions of these resources is the status of two processes—P1 and P2. P1 represents the process producing the pipe resource and P2 represents the process consuming that resource. The resource status of an active pipe (here R2) is defined to be *Status* = *in_use* (see also Table 3.9).

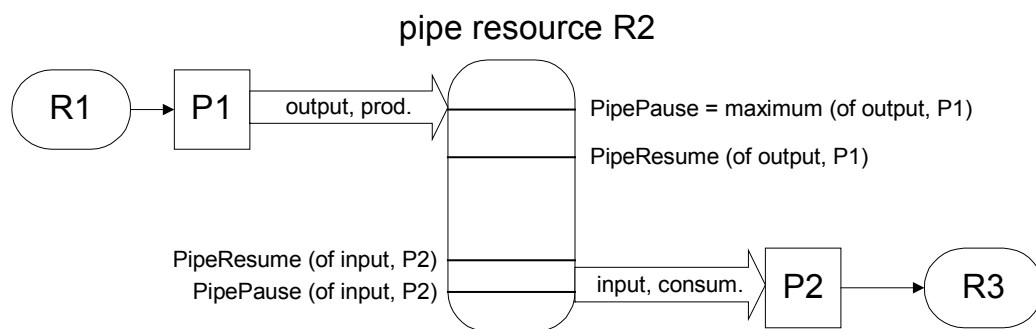


Figure 4.2 Example of a Pipe resource linking two processes

Figure 4.2 is a view on the structure and Figure 4.3 a view on the dynamics of the pipe example considered here. R1 represents an input resource for P1, which feeds into the intermediate pipe resource R2. Once the

tank R2 is filled to the predetermined level, it is used as the input resource for P2, which in turn produces output resource R3.

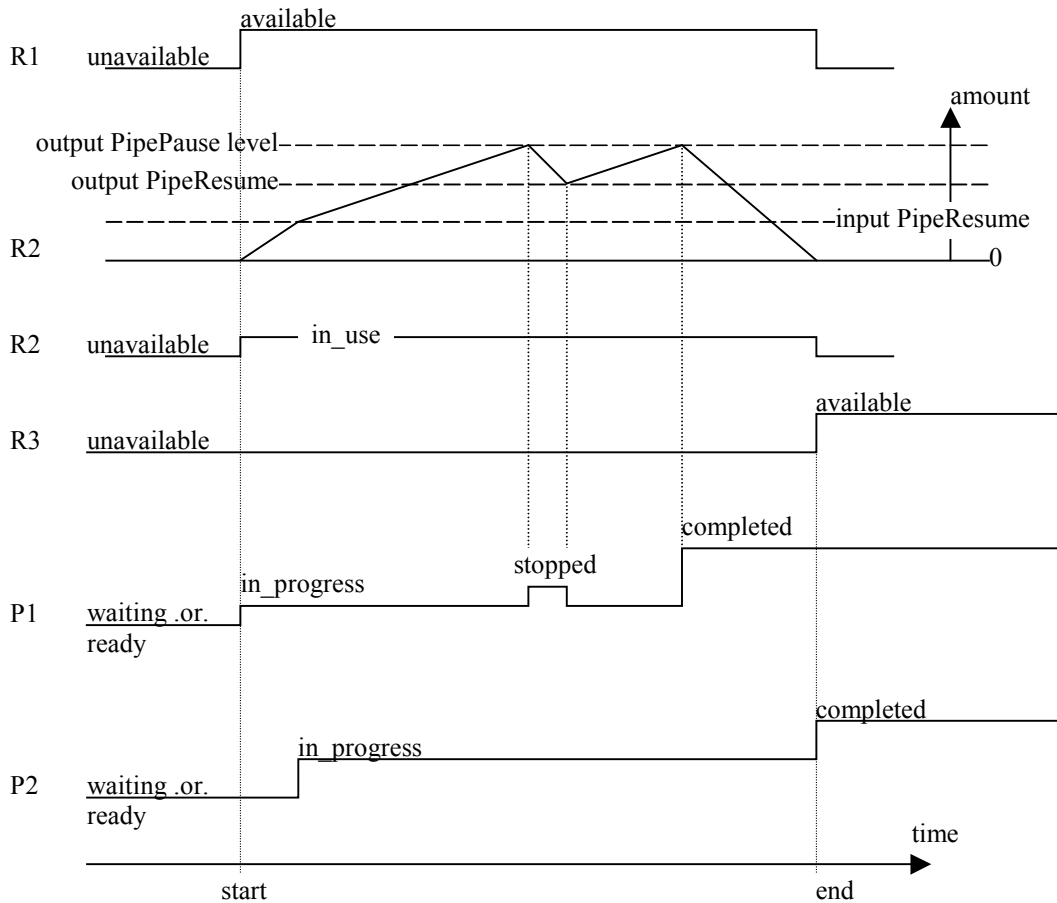


Figure 4.3 Example of status transitions in case of overlapping processing

Resource linking through pipes is controlled through the specification of the *PipePause* and *PipeResume* attributes. The intended amount of a resource must be specified in advance in the output link. Whenever the level representing the available quantity of the pipe resource exceeds the *PipePause* level of the output link, the process P1 is halted (*Status = stopped*) so that the process does not overproduce. Once the level falls below the *PipeResume* value, the process P1 resumes execution. P1 is completed when it has produced the intended amount. Once P1 has performed its task, the resources still in the pipe are consumed by the subsequent process without level control. In other words, after a process filling a pipe buffer has completed, pipe buffering becomes disabled.

Conversely, if the level representing the actual amount exceeds the *PipeResume* level of the input link, P2 can start or resume execution. If it falls below the *PipePause* level, P2 is halted (*Status = stopped*) unless the intended amount of the pipe resource R2 has already been produced. Then the *PipePause* level is ignored and the pipe resource is completely consumed.

In the case of output links, the *PipeResume* value must be smaller than the *PipePause* value, whereas in the case of input links, the *PipeResume* value must be greater than the *PipePause* value. If *PipePause* is specified for an input or an output link and *PipeResume* is not specified, the related process may run into a deadlock state. In other words, the process stops and cannot resume execution automatically. Once a process is stopped under these circumstances it can only be resumed manually or by sending a pipe control message for resumption that allows interconnected execution control (halting and resumption of processes

by pipe control messages is described in section 5.5.3 Pipe Control). If the attributes *PipeResume* or *PipePause* of links to pipe resources are not specified, the controller is responsible when the linked processes start and stop in dependence of the level.

4.3.2.1 Pipes of Partitionable Resources

Pipes of partitionable resources may also define the granularity of the resources that are considered to be one part. To accomplish this, the *PipePartIDKeys* attribute must be specified in the appropriate *ResourceLink* element. For instance, a partitioned *ImageSetting* process may be defined for multiple sheet separations, but a complete set containing all separations of both sides of a single sheet should be sent to the pressroom as one pipe request. In this case, the value of the *PartIDKeys* attribute of the *ExposedMedia* resource would be *SheetName Side Separation* and the value of the *PipePartIDKeys* attribute of the resource link to the pipe would be *SheetName*.

4.3.2.2 Dynamic Pipes

In addition to abstractly declaring pipe properties, JMF provides pipe messages that allow dynamic control of pipes. Dynamic pipes can be used to model situations where the required amount of resources is not known beforehand but becomes known during processing. An example of this behavior is a long press run where new plates are required during a press run because of quality deterioration. The exact point in time where quality becomes unacceptable is not predetermined and may even vary from separation to separation. Dynamic pipes provide the flexibility to adjust to changing situations of this nature.

Dynamic pipes provide a *PipeURL* attribute that allows dynamic requests for a status change of the pipe while a process is executing. Dynamic requests use JMF pipe control messages (see section 5.5.3 Pipe Control) sent to another controller whose URL address is specified by the *PipeURL* attribute of the respective resource link. Depending on the values of the resource link's *Usage* attribute, the following actions are possible:

- *Input* – The consumer sends a *PipePull* message to its *PipeURL* in order to request additional resources or a *PipePause* to halt production by the creator. The consumer sends a *PipeClose* message to the producer if the consumer does not require any further resources.
- *Output* – The creator sends a *PipePush* message to its *PipeURL* in order to deliver additional resources or a *PipePause* to halt consumption by the consumer.

When dynamic pipes are used—that is, when the *PipeURL* attribute is specified—the pipe buffering parameters *RemotePipeEndResume* and *RemotePipeEndPause* define the buffering parameters of the remote (controlled) end. *PipeResume* and *PipePause*, meanwhile, define the buffering parameters of the local node as described in section 4.3.2. The buffering parameters of a non-dynamic pipe may control the process that contains the resource link, whereas the buffering parameters of a dynamic pipe control the process at the other end of the pipe. The pipe control messages described later in section 5.5.3 Pipe Control are designed to establish communication between processes at both ends of dynamic pipe, even if the corresponding processes are spawned separately.

The following table summarizes the actions to be taken when the buffer in a dynamic pipe reaches a certain level L:

Table 4.2 Actions generated when a dynamic-pipe buffer passes various levels

Controlling Pipe End	Situation	Message	Description
----------------------	-----------	---------	-------------

output (creator)	$L > RemotePipeEndResume$	PipePush	Sufficient resources have been produced by the creator and are ready for delivery to the consumer.
output (creator)	$L < RemotePipeEndPause$	PipePause	The consumer has consumed to the low water mark and must pause until a sufficient amount of resources have been produced.
input (consumer)	$L < RemotePipeEndResume$	PipePull	More resources are requested from the creator and processing may continue by the consumer.
input (consumer)	$L > RemotePipeEndPause$	PipePause	The creator has produced to the high water mark and must wait until a sufficient amount of resources have been consumed.

Dynamic pipes are initially dormant, and must be activated by an explicit request. Dynamic pipe requests may be initiated by both ends of the pipe. For example, a print process may notify an off-line finishing process when a certain amount is ready by sending a PipePush message, or the printing process may request a new plate by sending a PipePull message.

4.3.2.3 Comparison of Non-Dynamic and Dynamic Pipes

The resource link between non-dynamic pipes provides the buffering parameters for the process to which the link belongs. Therefore, many processes can link to the same pipe resource. Furthermore, each process has its own buffering parameters, whether it is a consumer or a producer. In order to control non-dynamic pipes, one master-controller must control all processes linked to the pipe resource.

In contrast, dynamic pipes provide a URL address to control a process at the other pipe end. Then the buffering parameters of the resource link control the process at the other end. In the case of dynamic pipes, no master-controller is required in order to control the pipe. Control is accomplished by sending pipe messages.

If pipe resources are linked to multiple consumers or producers, such as two finishing lines that consume the output of one press one palette at a time, it is up to implementation to ensure consistency of the processes.

When using pipe resources, it is recommended that scheduling data for the process be specified only in the NodeInfo element of the parent node of the processes linked by pipe resources in order to avoid scheduling deadlocks. In Figure 4.3 for instance, the actual start and end time of the corresponding parent of P1 and P2 are marked on the time axis.

4.3.3 Parallel Processing

While serial processing assumes that all resources will be produced and consumed in a linear fashion, and while overlapping processing uses multiple processes that work together to use and create resources, there are times when it makes sense to run more than one process simultaneously, creating a more multi-pronged workflow. This kind of process routing is known as parallel processing. Subsections of jobs are spawned off so that nodes may be executed individually and simultaneously by the appropriate devices. Once the

processes are complete, the spawned nodes are merged back into the original job. The output resources of the merged nodes become inputs for later processes. For example, an insert may be produced independently of a cover, and both will be bound together later.

In parallel processing, processes can be run in a coordinated parallel fashion by using independent resources. An independent resource is a resource that is not shared between multiple processes. Implementation resources, for example, cannot be shared and are therefore always independent, and *Consumable* and *Quantity* resources can each be split to function as independent resources. Individual partitions of partitionable resources are independent and may be processed in parallel. Read-only resources, such as parameters, can be shared without any restrictions, and can therefore be used in read-only mode for parallel processing. Process chains created using independent resources are known as independent process chains.

Parallel processing can proceed in one of two ways. Either a controller may organize the JDF nodes in a way that allows it to initiate parallel processing or it can use the spawning-and-merging mechanism to field out chunks of the job to execute simultaneously. If a controller chooses the latter method, parent nodes that contain independent process chains can be spawned off and processed independently. For example, in order to improve production capacity, an agent may split consumable resources and create independent process chains in which each chain consumes its own resource part. Afterwards, the agent can submit one of the created job parts to a subcontractor and process the other part with its own facilities.

When splitting resources, it is not enough simply to insert an additional **Location** element into the resource element and divide the amount between the **Location** elements, as the parallel running processes would then link simultaneously to the same resource element. To split resource elements correctly, a separate resource must be created with its own **Location** element, and the amount must be divided between the **Location** elements of the independent resource instances.

Parallel processing is used only to process multiple aspects of a job simultaneously; it is not used to process multiple copies of a JDF job. In other words, a job must not be copied and sent to different controllers for parallel processing.

For more information about spawning of jobs, see section 4.4 *Spawning and Merging*.

4.3.4 Iterative Processing

Some processes, especially in the prepress area of production, cannot be described as a serial or parallel set of process steps. Instead, a set of interdependent processes is iterated in a non-deterministic order. These processes are known as iterative processes. For example, an advertisement is laid out that requires a photographic image. During the layout phase, changes must be made to the color settings of the image, which is the reinserted to the layout. Changes such as these can be described in a high-level fashion by defining a resource *Status* attribute of *draft*. As long as an input resource to a process has a status of *draft*, the *Status* of the output resource may not be *available*.

The **ResourceLink** that links to a draft input resource must include a *DraftOK* attribute to state that a draft input resource is acceptable for a process. Thus a prepress layout process can be abstractly defined to work on draft resources until an acceptable output has been achieved, but the output PDL-file may not be used for printing until it is *available* and no longer designated as a *draft*.

Iterative processes may be set up in a formal fashion using dynamic pipes to convey parameter change requests or in an informal way that assumes that the operators of the various processes have an informal communication channel. Both are described in greater detail below.

4.3.4.1 Informal Iterative Processing

Informal iterative processing does not require a complete redefinition of the required resources at every iteration. This kind of processing is generally used in a creative workflow, where a job is defined and gets refined in a series of steps until it is completed. The information about the changes is transferred through channels that bypass JDF. Nonetheless, the description of these processes in JDF is useful for accounting purposes, as the status of each process may be monitored individually.

The **ResourceLink** elements for informal processing contain an additional *DraftOK* attribute, but in all other ways they are identical to the **ResourceLink** elements used in simple sequential processing. Furthermore, the nodes run through the same set of phases as they would in sequential processing. Nodes are designated only as *stopped* and not as *completed* after being processed for an iterative cycle. They are marked as completed after their output resources lose their *Status* of *draft*.

4.3.4.2 Formal Iterative Processing

In formal iterative processing, all **ResourceLink** elements between interacting processes are dynamic pipes. Every request for a new resource is initiated by a **PipePush** or **PipePull** message that contains at least one **Resource** element with the updated parameters. This resource is used by the process, and the resulting new output resource can be consumed by the requesting process. The *Status* of *draft* can be removed from a resource by sending the creator a **PipeClose** message that has the optional *UpdatedStatus* attribute set to *available*. A node can only reach a *Status* of *completed* if it has no remaining draft resources. Another method to remove the draft status is to define a node for an **Approval** process that accepts draft resources as inputs and has non-draft resources representing the same entities as outputs.

4.3.5 Proofing and Verification

In many cases, it is desirable to ensure that an executed process or set of processes have been executed correctly. In the graphic arts industry this is verified by generating approvals and signing them. JDF allows modeling of the proof process and modeling of the verification processes by allowing an optional **ApprovalSuccess** input resource in any process. An **ApprovalSuccess** resource may only be set as *available* if it has been signed by an authorized person.

If an approval fails and one or more processes that create the approved resource must be rerun, an agent must modify the job appropriately and resubmit it to the corresponding controllers and devices. All interchange resources from the first unsuccessful process to the approval must be designated as *unavailable*.

4.4 Spawning and Merging

JDF spawning is the process of extracting a JDF sub-node from a job and creating a new, complete JDF document that contains all of the information needed to process the sub-node in the original job. Merging is the process of recombining the information from a spawned job part with the original JDF job, even after both documents have evolved independently. By using the mechanism for spawning and merging different parts of a job, it is possible to submit job parts to distributed controllers, devices, other work areas, or other work centers.

The JDF spawning-and-merging mechanism can be applied recursively. In other words, sub-jobs that have already been spawned may in turn spawn other sub-sub-jobs, and so on.² No matter how many job parts

² A respawning of already spawned nodes is not allowed. If a node should be spawned a second time, the previously submitted version must be deleted first and the procedure must be applied again to the original node.

have been spawned, however, merging is realized by copying nodes back to their original location and synchronizing the appropriate resources. Therefore, each spawning must be logged in the job by the agent performing the actions that result in a spawned job. Furthermore, in order to avoid inconsistent JDF states after merging, each merging should be logged, or the appropriate spawn audit must be removed from the AuditPool element.

Figure 4.4, below, shows, schematically, the spawning and merging of a sub-job, designated as P.b. The following three phases are defined on a time scale:

1. The first phase occurs before the sub-job is spawned off.
2. The second phase occurs during the spawn phase, when the spawned sub-job is executed separately.
3. The third phase occurs after the spawned job has been merged back into the original job.

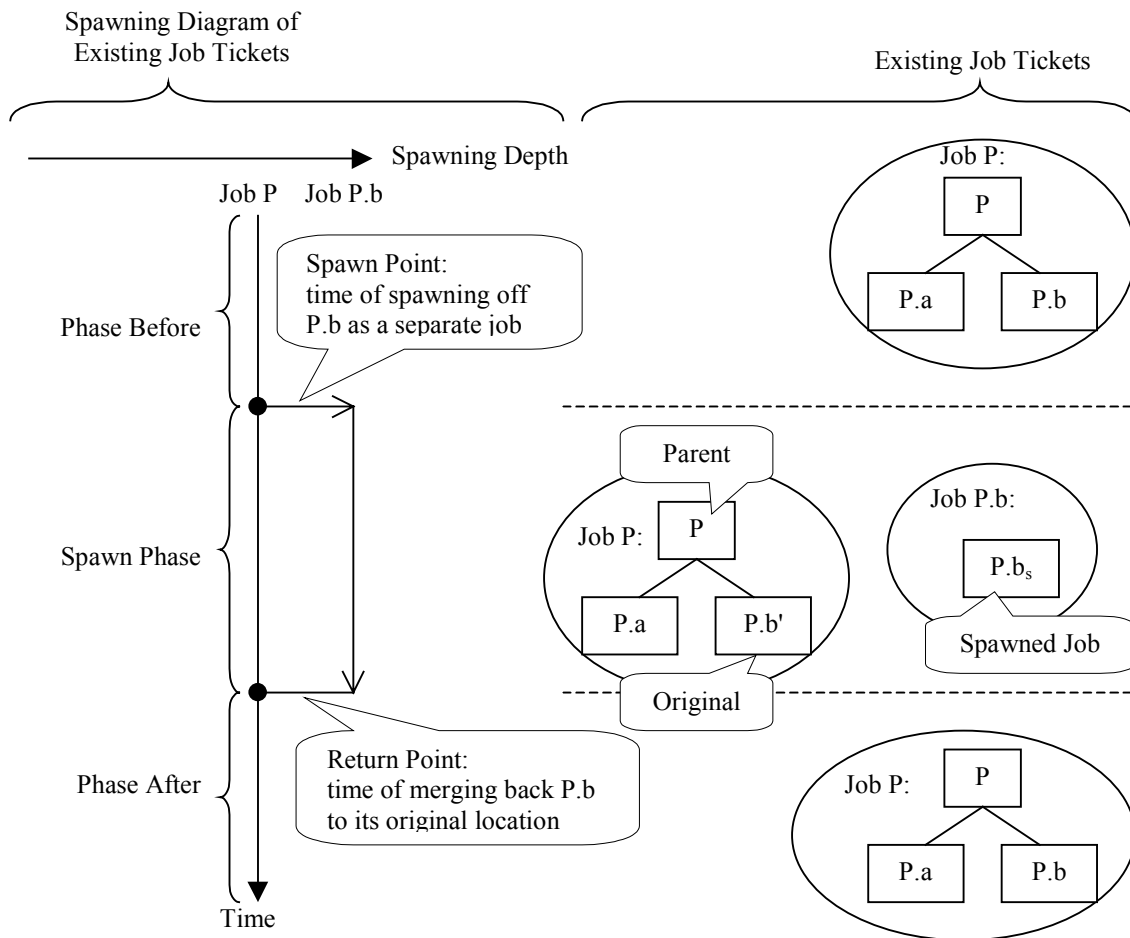


Figure 4.4 Spawning and merging mechanism and its phases

The three phases of the job part are bordered by the spawning point and the merging point. On a job scale, denoted as spawning depth in Figure 4.4, one job ticket exists during the phases before and after spawning, and the following two job tickets exist during the spawning phase: The job with the **parent** (P) of the **original** job part (P.b', also denoted as a sub-job) that has been spawned; and the **spawned job** (P.b_s) itself.

This section provides examples that outline the various ways in which spawning and merging can be applied. The six following cases are considered in the next six sections:

1. Standard spawning and merging
2. Spawning and merging with resource copying
3. Parallel spawning and merging of partitioned resources
4. Nested spawning and merging in reverse sequence
5. Spawning and merging of independent job tickets
6. Simultaneous Spawning and Merging of Multiple nodes

JDF can support any combination of the cases described, but these four represent a cross-section of likely scenarios. Case one is the simplest of all of the cases, and is required in every instance of spawning and merging, regardless of the circumstances surrounding the process. After that, each case requires additional processing that builds upon the processing described in the cases that precede it.

4.4.1 Case 1: Standard Spawning and Merging

The actions described in this case must be applied in every spawning and merging process. All cases described in this chapter, as well as any other that may be invented, begin with these procedures.

Spawning

When spawning a JDF sub-node, the JDF elements `CustomerInfo` and `NodeInfo` elements of the spawned job may be created and/or filled with the appropriate information (for details, see sections 3.4 *Customer Information* and 3.5 *Process and Node Information*). All resources that are referenced in the spawned node and its sub-nodes are located in the `ResourcePool` containers of the respective node(s).

To indicate that a process has been spawned, the *Status* attribute of the original JDF node must be set to the value *spawned* (see Table 3.3). The *Status* attribute of the spawned node remains unchanged.

In order to identify all of the ancestors of job that has been spawned, an `AncestorPool` element is included in the root node every spawned job. This element contains an `Ancestor` element that identifies every parent, grandparent, great-grandparent, and so on of the spawned sub-node. In this way, the family tree of every spawned node is tracked in an ordered sequence that allows an unbroken trace back through all predecessors. Consequently, the elements that comprise the `AncestorPool` of a spawned job must be copied into the `AncestorPool` element of the newly spawned job before the ancestor information of the previously spawned job is appended to the `AncestorPool` element of the newly spawned job. The last `Ancestor` element in each `AncestorPool` is the parent, the second-to-last the grandparent, and so on. The following code is an example of a family tree:

```
<AncestorPool>
  <Ancestor NodeID="p_01" FileName="file://grandparent.jdf"/>
  <Ancestor NodeID="p_02" FileName="file://parent.jdf"/>
</AncestorPool>
```

The complete ancestor information is required in order to merge back semi-finished jobs with nested spawned jobs. If the last spawn is always merged first (LIFO) then knowing the direct parent is sufficient, as each parent will in turn know its own parent back to the original and a complete ancestor line may be inferred.

When a job is spawned, the action must be logged in the parent node of the spawned node in the original job. This is accomplished by creating a `Spawned` element with the *jrefSpawned* attribute set to the ID of the spawned JDF node. This `Spawned` element must be appended to the `AuditPool` container of the

original parent node. If no **AuditPool** container exists in the parent node, one must be created for the purpose.

After a node has been spawned, it is legal, although not necessary, to remove all contents of the spawned node in the original node except for the *ID* attribute. It is not, however, possible to undo the spawning operation without accessing the spawned node once the contents of the spawned node have been removed.

Merging

After processing, the spawned job must be merged back to its original location. Before this can occur, however, duplicate information contained in any elements that are not required for further processing (such as **CustomerInfo** or **NodeInfo**) may optionally be deleted by the agent executing the spawning and merging. Once this has been accomplished, the spawned node is copied to the location of the original node, completely overwriting the original node. The *Status* of the original node is then overwritten with the result.

To complete the merging process, the merging agent must add a **Log** element of *Type = Merged* to the **AuditPool** (see section 3.9 **AuditPool**). Furthermore, the **AncestorPool** container with all child elements must be removed.

4.4.2 Case 2: Spawning and Merging with resource copying

Figure 4.5, shown below, represents an example of a job that requires that resources be copied during spawning. In this job, the nodes B_1 and B_2 are linked to the same resource, which is localized in the resource pool of an ancestor node, denoted as node A. This node is the parent node.

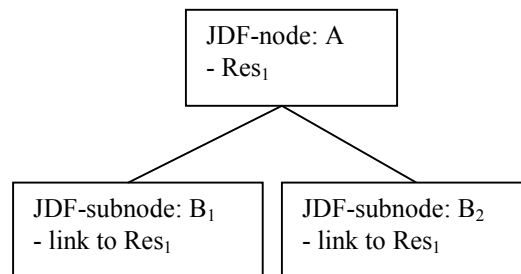


Figure 4.5 JDF node structure that requires resource copying during spawning and merging

When node B_1 is spawned, its resources must also be duplicated. To accomplish this, the affected resources must be copied to the spawned job and purged during merging, a process that is described below.

Spawning

Spawning begins as it did in case 1. The affected resources must then be copied to the resource pool of the spawned job. The copied resources retains the same *ID* values as the original resources. These resources can be spawned for read-only access, which allows multiple simultaneous spawning of one resource, or for read/write access, in which case a resource may only be spawned one time. The read/write spawning of a resource locks the resource in the original file in order to avoid conflicts that result from simultaneous modification or reading and modification of a resource. The *SpawnStatus* attribute of the original resource must be set to *spawned RW* (which stands for “spawned read/write”) or *spawned RO* (which stands for “spawned read-only”) to indicate that the resource is spawned. In other words, a copy of the resource is spawned together with the spawned job. Read/write access effectively locks the original

resources, just as if the attribute *Locked = true*³ were present. If a resource is spawned as read-only, it is not a good idea to modify the original resource that remains in the parent job ticket, as this may lead to inconsistencies. The *Locked* attribute of spawned resources that are copied read-only should also be set *true*. Furthermore, the value of the *ID* attribute of each copied resource must be appended to the appropriate *rRefsROCopied* or *rRefsRWCopied* values of the *Spawned* element that resides in the *AuditPool* of the parent node.

Merging

Merging begins as it did in Case 1. Then, if resources have been copied for spawning, they must be purged after merging. Read-only resources may simply be deleted in the spawned node before merging. If the original resource and the spawned resource are not identical, however, a JDF content error should be logged by a *Notification* element of *Class = error* (see section 4.6 Error Handling). Read/write resources must be copied into their original location, completely overwriting the original resource. The *ID* attributes of the overwritten resources must be specified in the *rRefsOverwritten* attribute of the *Merged* element. The *Merged* element is then inserted into the *AuditPool* container of the parent during the usual merging procedure, which is shown as the return point in the spawning diagram.

4.4.3 Case 3: Parallel Spawning and Merging of Partitioned Resources

In many cases, it is desirable to define a parallel workflow for partitioned resources. This is modeled by spawning a node that defines the process for each part that is to be processed individually using *Selector* resources.

Spawning

Spawning begins as it did in case 1. Then, a *Selector* resource is added to the spawned node and the spawned node, along with all the other sub-nodes, is linked to that *Selector* resource. If any *Selector* resource exists prior to spawning, it must be merged with the newly created *Selector* resource to create one, unified *Selector* resource. This resulting *Selector* selects those elements that would be selected if both selectors had been applied sequentially. This is equivalent to a boolean (logical, mathematical) AND operation. In addition, a copy of the spawned *Selector* resource (without the contents of the prior *Selector*) is appended to the *Spawned* audit element. This copy may *NOT* contain an *ID* attribute. The *Status* of any partitioned resource is defined individually for each partition.

The spawning procedure described in this section can be performed iteratively for multiple parts, effectively generating one *Spawned* audit element per part.

Merging

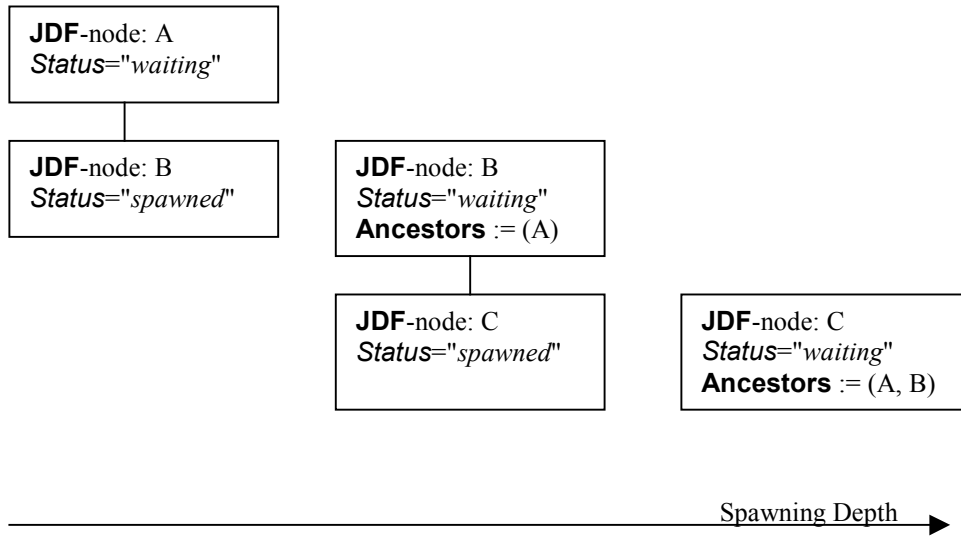
After an individual partitioned spawned node has been processed, it is merged back to the parent as was described in Case 1. In addition, a copy of the *Selector* resource is appended to the *Merged* element (again, this copy may *NOT* contain an *ID* attribute), and any read/write resources are merged into their appropriate parts. The selector resources are either removed, assuming no prior selector exists, or unfolded to the original *Selector* resource using the information in the copied *Selector* resource contained in the *Spawned* audit.

4.4.4 Case 4: Nested Spawning and Merging in Reverse Sequence

Figure 4.6 shows an example of nested spawning and merging in reverse sequence. Process A spawns node B, and node B spawns node C. Even if B is merged back to A for any reason before C is merged back to B,

³ Usually resources become locked (*Locked = true*) if they are referenced by audit elements (see also section 3.9 *AuditPool*).

C still contains the information of its grandparent in the AncestorPool element. In this way, C can trace back its ancestors and find the localization of its parent, node B, in node A, even though the spawned job, with B as root node, has already been deleted.



Spawning Diagrams

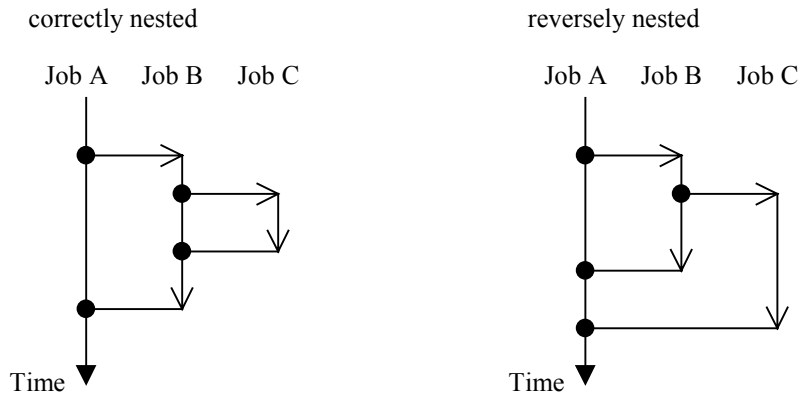


Figure 4.6 Example for a JDF node structure with nested spawning

4.4.5 Case 5: Spawning and Merging of Independent Jobs

It is useful to spawn and merge independent jobs in situations where the execution of separate, independent small jobs is not efficient in a commercial sense. Business cards for individual customers that are printed on one set of sheets and subsequently cut are an example of this kind of situation. In cases such as these, small jobs can be collected in order to form a big job that may then be executed as a whole. This allows job aspects such as production, equipment load, and balancing of implementation resources to be performed more efficiently.

In this example, diagrammed in Figure 4.7, nodes C and E represent small jobs of identical type. Node bigA represents a big job, which may exist already or which may have been created for the purposes of this spawning-and-merging process. Once nodes C and E are gathered beneath node bigA, as described below, big job may then be executed as a whole for the sake of efficiency.. When the big job is executed, the small jobs are effectively executed simultaneously. Nodes A, B, and D are provided to demonstrate that spawned nodes in this example may be related to other nodes in various ways.

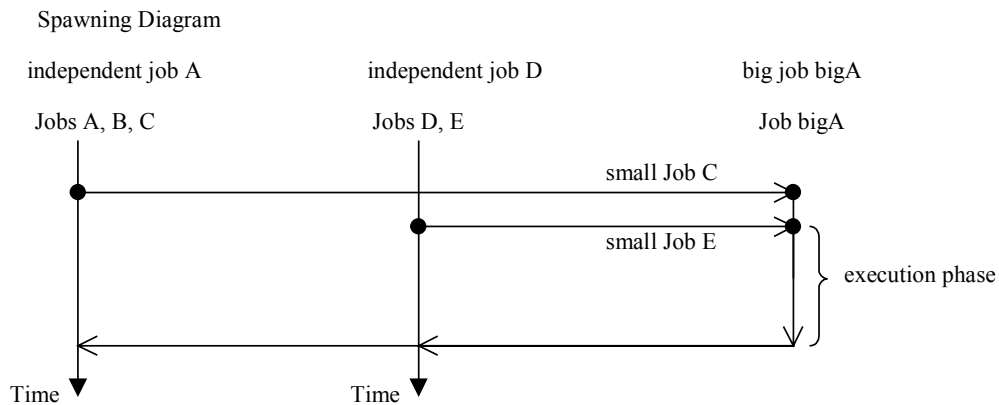
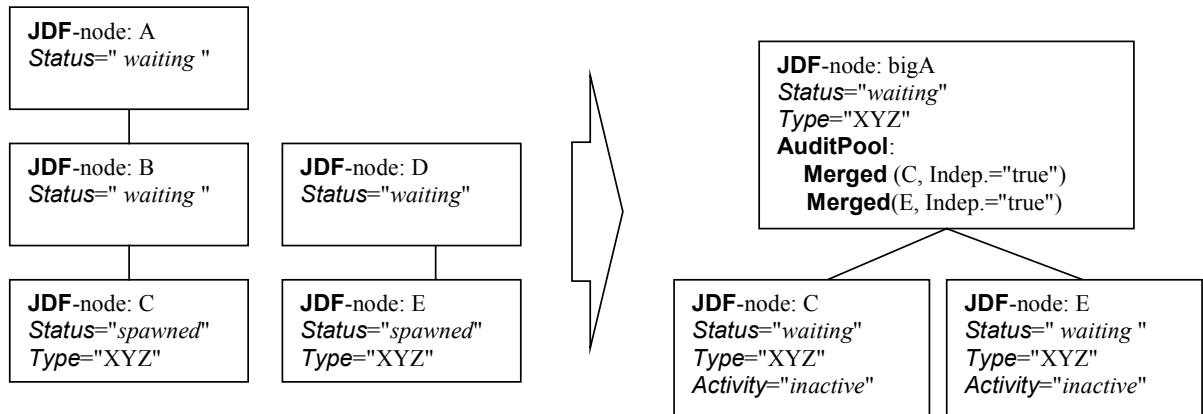


Figure 4.7 Example of the spawning and merging of independent jobs

Spawning

Spawning begins as it did in case 1. Then, the process to be spawned (job C in Figure 4.7) is copied into a newly created, or already existing, big job (job E in Figure 4.7). The process type of the root node of the big job must be identical to that of the spawned processes. The *Activation* state of the spawned processes

is set to *inactive*, and an *AncestorPool* element is added to the inactive spawned job to define the ancestry (as was described above). A *Merged* element containing information about the spawned independent jobs and when they have been received is added to the big job.

In the original jobs, the *Status* of the process is designated as *spawned*, and a *Spawned* element with the optional attribute *jrefDestinationJDF* specified is added to the parent of the original job. The attribute *jrefDestinationJDF* contains the ID of the big job beneath which the spawned process has been placed. The changes in the parent are the equivalent of those described in Case 1, except for the specification of the attribute *jrefDestinationJDF* in the *Spawned* element.

Where necessary, resource instances must be copied and logged by appending the IDs to the appropriate attribute (*rRefsROCopied* or *rRefsRWCopied*) of the *Spawned* element in the parent of the original job. This is required in single spawning and merging. Furthermore, the *ResourceLink* elements of the spawned process must be transferred in content to the *ResourceLinkPool* of the active, big process node. In this way, the input resources and the resources to be produced are linked to the big job.

Merging

For each of the spawned small jobs, the return procedure is performed as it was in the preceding cases. Once the process explained in Case 1 is performed, the completed job is copied back to its original location and the attribute *Activation* is restored by setting it to the activation of the big-job node after completion.

Eventually, copied resources must be purged and handled just as they were in Case 2. Then, the merging must be logged by appending the *Merged* element to the *AuditPool* container of the parent of the original node. In independent spawning and merging, the attribute *jrefSourceJDF* must be specified in the appropriate *Merged* element.

If the big job is retained, a *Spawned* element with the attribute *Independent = true* must be appended to the *AuditPool* of the big job. For instance, saving the finished big job may be desirable if the audit information contained in the big job should be available for an individual invoicing. Finally, the newly created big JDF should be deleted to avoid the double existence of nodes.

4.4.6 Simultaneous Spawning and Merging of Multiple nodes

It is not possible to explicitly spawn multiple nodes simultaneously. The nodes must be grouped into a single *ProcessGroup* node, and this node can then be spawned and merged as described in the previous sections.

4.5 Node and Resource IDs

All nodes and resources must contain a unique identifier, not only because it is important to be able to identify individual components of a job, but because JDF uses these IDs for internal linking purposes. Each agent that creates resources and sub-nodes or that performs spawning and merging is responsible for providing IDs that are unique in the scope of the file, taking into account all of the phases of a job's life cycle.

IDs come in two flavors: pure and composite. A **pure ID** is an ID that does not contain the character period “.” A **composite ID** is made up of pure IDs delimited by periods. For example:

```
pureID :: = ID -{'.'}
compositeID :: = pureID ['.'pureID]+
ID :: = pureID | compositeID
```

IDs are used differently under different circumstances. Several different circumstances are described below.

In case of no spawning:

If an agent inserts new elements requiring IDs into an original job, then the agent assigns pure IDs to the new elements and must guarantee their uniqueness.

In case of single spawning:

If an agent inserts new elements into a spawned job, then the agent creates composite IDs by using the ID of the root node and appending a unique pure ID delimited by a period. For example:

- ID of spawned root node: *ID* = "Job_01234.Proc1"
- ID used for new element: *ID* = "Job_01234.Proc1.newpureID"

In case of independent spawning:

The agent that merges the independent jobs beneath a big job inserts a unique, pure ID (delimited by a period) in front of all IDs of each small job it receives. That means that the agent must replace all IDs of each job it receives whenever it encounters an ID collision. If an agent inserts new elements into a spawned job, then the agent creates composite IDs by using the ID of the respective root node of the small job and appends unique pureID, delimited by a period. For example:

- ID of the big job with node *ID* = "A"
- Receives small job A₁ with some IDs: *ID* = "A" *ID* = "A.A" *ID* = "A.B" where the first is the ID of the root node.
- Receives small job A₂ with some IDs: *ID* = "A" *ID* = "A.A" *ID* = "anything" ...
- The agent creates locally unique pure IDs: *ID* = "A1" and *ID* = "A2" each prepended to all IDs of each received small job; the IDs of the small job A₁ become: *ID* = "A1.A" *ID* = "A1.A.A" *ID* = "A1.A.B" and the IDs of the small job A₂ become: *ID* = "A2.A" *ID* = "A2.A.A" *ID* = "A2.anything". All IDs in the big job are unique.
- The agent creates a new element added to the small job A₁ with ID: *ID* = "A1.A.C". Here the agent must resolve the possible conflict if it would append the pure ID = "A" to the root ID = "A1.A". That means the agent has to check the uniqueness of each created ID.
- Before merging the jobs back to its original location the agent must remove the prepended pure IDs of all IDs, here "A1", "A2" respectively. Then the newly created element will be merged back with the *ID* = "A.C".

4.6 Error Handling

Error handling is an implementation-dependent feature of JDF-based systems. The **AuditPool** element provides a container where errors that occur during the execution of a JDF may be logged using **Notification** elements. **Notification** elements may also be sent in **JMF Signal** messages. The content of the **Notification** element is described in Table 3.23.

Further details about error handling are provided in the next four sections.

4.6.1 Classification of Notifications

Notification elements are classified by the attribute *Class*. Every workflow implementation must associate a class with all events on an event-by-event basis. The following list shows the possible values for *Class*:

- *event* Indicates an event which occurred due to a certain operation-related action, for example, machine events, operator activities, etc. This class is used for messaging.

- *information* Indicates not an error, but rather any information about a process that cannot be expressed by the other classes, for example, the beginning of execution.
- *warning* Indicates that a minor error has occurred and an automatic fix was applied. Execution continues. The node's *Status* is unchanged. Appears in situations such as A4-Letter substitutions, when toner is low, or when unknown extensions are encountered in a required resource
- *error* Indicates that an error has occurred that requires user interaction. Execution cannot continue until the problem has been fixed. The node's *Status* is *stopped*. This value appears in situations such as when resources are missing, when major incompatibilities are detected, or when the toner is empty.
- *fatal* Execution must be aborted. The node's *Status* is *aborted*. Seen with most protocol errors or when major device malfunction has occurred.

4.6.2 Event Description

A description of the event is given by a generic *Comment* element, which is mandatory for the notification classes *information*, *warning*, *error*, or *fatal*. For example, after a process is aborted, error information describing a device error may be logged in the *Comment* element of the *Notification* element. If phase times are logged, the *PhaseTime* element that logged the transition to the *aborted* state may also contain a local *Comment* element that describes the cause of the process abortion. *PhaseTime* and *Notification* elements are optional sub-elements of the *AuditPool*, which is described in section 3.9.

4.6.3 Error Logging in the JDF file

A JDF-compliant controller/agent should log an error by inserting a *Notification* element in the *AuditPool* of the node that generated the error. The *NodeInfo* element may contain an *NotificationClasses* attribute to define the minimum severity of events (or, more specifically, errors) that should be logged.

4.6.4 Error Handling via Messaging (JMF)

A JMF *Signal* message with a *Notification* element in the message body should be sent through all persistent channels that are defined for events (or, more specifically, errors with an event severity that is greater or equal to the event severity specified in the *JMF* element of the *NodeInfo* element. Note that this is different from the *NotificationClasses* attribute of the *NodeInfo* element, which is defined for logging events by *Notification* elements to the *AuditPool*. The mechanism described in this section is instead an appropriate message definition.

4.7 Test Running

In JDF, the notion of a test run is similar to the press notion of preflight. The goal is to detect JDF content errors and inconsistencies in a job before the job is executed.

The ability to perform a test run may be built into individual devices or controllers. Alternatively, a controller implementation may perform test runs on behalf of its devices. A test run may be routed through all of the different devices and controllers in a workflow, just as if the test run were a standard execution run. For the routing of jobs and nodes through different devices and controllers for a test, the spawning and merging mechanism may also be applied. The devices/controllers receiving a job read it and analyze without initiating execution. Rather, they investigate the content of the node they would execute. A

device/controller with agent capabilities may record results into the audit pool associated with a given process.

During test running, the requirements of the processes specified are compared to the capabilities of the devices targeted. A device or controller explicitly tests whether the inputs that have been specified as required are actually the inputs that are required, and that none are missing or in error. For example, an input requirement may be a URL that, when a test run is performed, is found to point to an item that no longer exists in that location. Test running is meant to prevent errors as a result of that kind of misinformation. It is particularly useful when running expensive or time-consuming jobs.

It is also possible to test run specific parts of a workflow, or even individual nodes. An agent may request a test of certain nodes by setting the JDF attribute *Activation* to *testrun* (see Table 3.3), which is inherited by all descendant nodes that are not inactive (*Activation = inactive*). If a device or controller⁴ detects an error in a node a **Notification** element containing a textual description should be appended to the **AuditPool** element of the node in which the error occurred, and, if messaging is supported, the error should be also communicated to the connected listeners via messaging (for more information see section 5.4 Error and Event Messages). If an error has been detected the agent can modify the job in order to correct the error. Once a test run has been completed successfully, the device/controller with agent capabilities changes the *Status* attribute of the tested node to *ready*. If a test run fails, the device/controller is required to record the process status as *failed_testrun*. After the test run has finished the agent should log the result by appending a **ProcessRun** element to the **AuditPool** element. For more information about audits, see section 3.9 AuditPool.

In principle, execution and test runs may be run simultaneously. For example, one job part may be executed while another part requests only a test.

JDF also defines an *Activation* value of *testrun_and_go* that requests a test run and, upon successful completion, automatically initiates processing.

4.7.1 Resource Status During Testrun

In order to test run a complete set of nodes, it is sometimes necessary to imply the *Status* of resources that are produced by prior nodes. Successful test running does *not* set the *Status* attribute of a resource to *available* unless the resource actually is available. Nodes that require an output resource of a node that has completed test running for purposes of test running may assume that these resources have a *Status* of *available* for the purpose of test running as long as the producing node has a *Status* of *ready*.

⁴ Note that only devices and controllers with agent capabilities can write in a JDF-document.

Chapter 5 JDF Messaging with the Job Messaging Format (JMF)

A workflow system is a dynamic set of interacting processes, devices and MIS systems, and for the workflow to run efficiently, these processes and devices must communicate and interact in a well-defined manner. Messaging is a simple but powerful way to establish this kind of dynamic interaction. The JDF-based Job Messaging Format, or JMF, provides a wide range of capabilities to facilitate interaction between the various aspects of a workflow, from simple uni-directional notification through the issuing of direct commands.

This chapter outlines the way in which JMF, accomplishes these interactions. The following list of use cases is considered:

- System setup
- Dynamic status and error tracking for jobs and devices
- Pipe control
- Device setup and job changes
- Queue handling and job submission

JMF messages are most often encoded in pure XML, without an additional MIME/Multipart wrapper. Only controllers that support JDF job submission via the message channel must support MIME for messages.

5.1 JMF Root

JMF and JDF have an inherently different structure. In order to allow immediate identification of messages, JMF uses the unique name **JMF** as its own root-element name.

The root element of the XML fragment that encodes a message, like the root element of a JDF fragment, contains a series of predictable attributes and instances of **Message** elements. These contents are defined in the tables that follow, and illustrated in Figure 5.1. **Message** elements are abstract, as is indicated by the dashed line surrounding the **Message** element in Figure 5.1.

Table 5.1 Contents of the JMF root

Name	Data Type	Description
<i>DeviceID</i> ?	string	Identifies the recipient device or controller. The envelope of the message contains the URL address of the controller that receives the message via HTTP. Therefore, if <i>DeviceID</i> does not specify a recipient, that controller is assumed to be the recipient.
<i>SenderID</i>	string	String that identifies the sender device, controller or agent.
<i>TimeStamp</i>	timeInstant	Time stamp that identifies when the message was created.
<i>Version</i> ?	string	JMF version. The current and default version is "1.0".
Message +	element	Abstract message element.

The following table describes the contents of the abstract **Message** element. All messages contain an *ID* and a *Type* attribute.

Table 5.2 Contents of the abstract Message element

Name	Data Type	Description
<i>ID</i>	ID	Identifies the message.
<i>Time ?</i>	timeInstant	Time at which the message was generated. This attribute is only required if this time is different from the time specified in the <i>TimeStamp</i> attribute of the JMF element.
<i>Type</i>	NMOKEN	Name that identifies the message type. Message types are described in sections 5.5 and 5.6.

The following figure depicts the basic messaging structure and the message families.

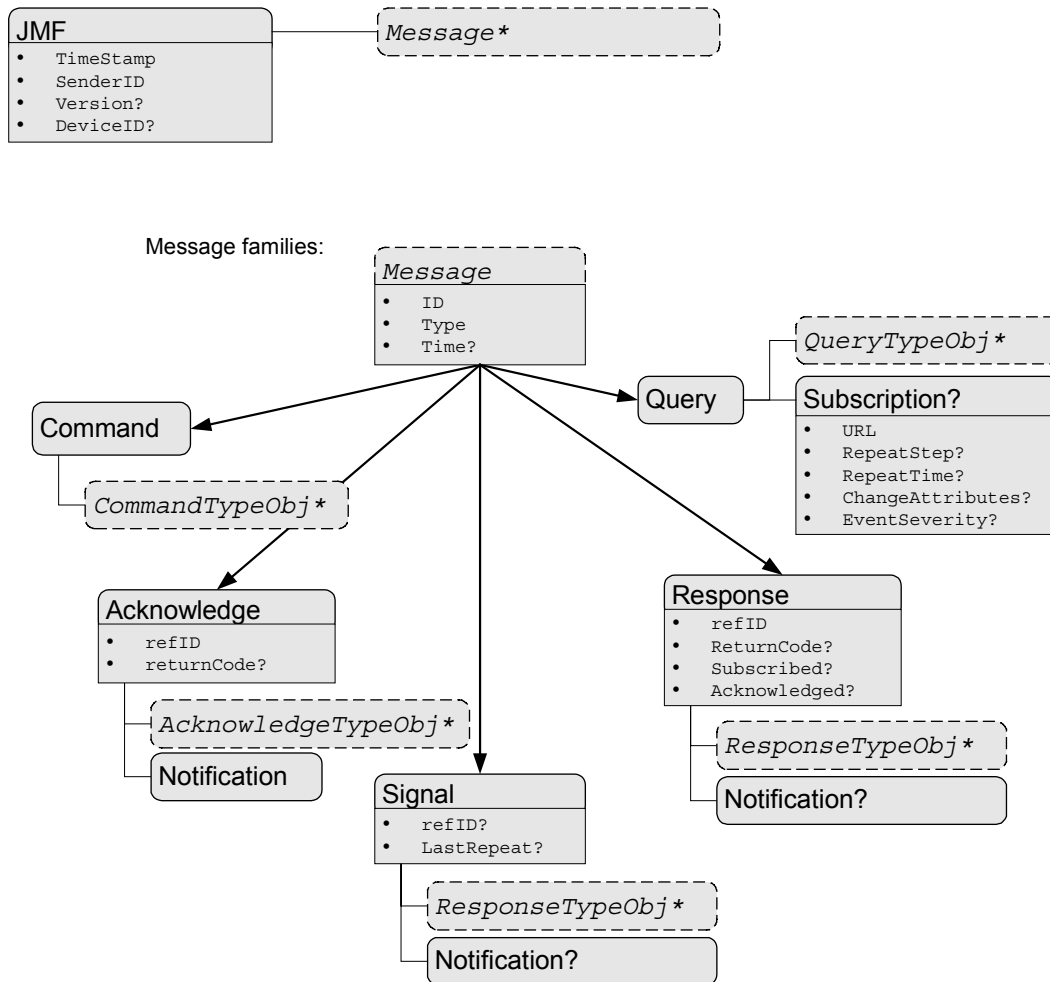


Figure 5.1 Contents of a JMF root element and the message families

5.2 JMF Semantics

JMF encodes messages of several types. The first part of this section describes message elements that contain and convey content, while the second describes the way in which these element types can be used to establish communication.

5.2.1 Message Families

A message contains one or more of the following five high-level elements, referred to as **message families**, in the root node. These families are **Query**, **Command**, **Response**, **Acknowledge**, and **Signal**. An explanation of each family is provided in the following sections, along with an encoding example.

5.2.1.1 Query

A query is a message that retrieves information from a controller without changing the state of that controller. A query is sent to a controller. It contains an *ID* attribute and a *Type* attribute, which it inherits from the abstract message type described in Table 5.2. JMF supports a number of well-defined query types, and each query type can contain additional descriptive elements, which are described in sections 5.5 and 5.6. The following table shows the content of a **Query** message element.

Table 5.3 Contents of the Query message element

Name	Data Type	Description
QueryTypeObj *	element	Abstract element that is a placeholder for any descriptive elements that provide details required for the query. The element type of QueryTypeObj is defined by the <i>Type</i> attribute of the abstract Message element.
Subscription ?	element	If specified creates a persistent channel. For the structure of a <i>Subscription</i> element, see section 5.2.2.3 Persistent Channels.

The following is an example of a query message:

```
<JMF TimeStamp="2000-07-25T11:38:23.3+02:00" SenderID="Controller-1">
  <Query Type="KnownJDFServices" ID="007"/>
</JMF>
```

5.2.1.2 Response

A response to a query or a command is always a direct acknowledgement of a query or a command. A response is returned from a controller to the controller that put the query/command. Responses are not acknowledged themselves.

A command response indicates that the command has been received and interpreted. The response of commands with short latency also includes the information about the execution. Commands with long latency will generate additionally a separate **Acknowledge** message (see section 5.2.1.5 **Acknowledge**) to broadcast the execution of the command. Command responses should comprise a **Notification** element that describes textually the return status.

Responses contain an attribute called *refID*, which identifies the initiating query or command. The following table shows the content of a **Response** message.

Table 5.4 Contents of the Response message element

Name	Data Type	Description
<i>Acknowledged</i> ?	boolean	Used only in responses to command messages. Indicates whether the command will be acknowledged separately. If <i>true</i> , an Acknowledge message will be supplied after command execution. If <i>false</i> , no Acknowledge message

		will be supplied. Default = <i>false</i>
<i>refID</i>	NMTOKEN	Copy of the <i>ID</i> attribute of the initiating query or command message to which the response refers.
<i>ReturnCode</i> ?	integer	Describes the result. 0 indicates success. For all other possible codes see Appendix H . Default = 0
<i>Subscribed</i> ?	boolean	If a Subscription element has been supplied by the corresponding query, this attribute indicates whether the subscription has been refused or accepted. If <i>true</i> , the requested subscription is accepted. If <i>false</i> , the subscription is refused because the controller does not support persistent channels. For details, see section 5.2.2.3 Persistent Channels . Default = <i>true</i>
<i>Notification</i> ?	element	Textual description of the return code. The Notification element should be provided if the <i>ReturnCode</i> is greater than 0, which indicates that an error has occurred, or if the initiating message is a command.
<i>ResponseTypeObj</i> *	element	Abstract element that is a placeholder for any descriptive elements that provide details queried for or details about command execution.

An example of a response on a command is provided in the section 5.2.1.4 [Command](#). The encoding example for the query, shown above, might generate the following response:

```
<JMF TimeStamp="2000-07-25T11:38:25+02:00" SenderID="RIP-1">
  <Response Type="KnownJDFServices" ID="107" refID="007">
    <JDFService Type="RIPping"/>
    <JDFService Type="Imposition"/>
    <JDFService Type="Trapping"/>
  </Response>
</JMF>
```

5.2.1.3 Signal

A signal message, which is syntactically equivalent to a combination of a **Query** message and a **Response** message, is a unidirectional message sent on any events to other controllers. This kind of message is used to automatically broadcast some status changes.

Controllers can get signal messages in one of three ways. The first way is to subscribe for them with an initiating query transmitted via a message channel that includes a **Subscription** element. The second way is to subscribe for them with an initiating query defined in the **NodeInfo** element of a JDF node that includes also a **Subscription** element (see JMF elements in [Table 3.7](#)). The first query is transmitted separately via a mechanism such as HTTP, whereas the second is read together with the corresponding JDF node. Once the subscription has been established, signals are sent to the subscribing controllers via persistent channels. In both cases, however, the **Signal** message contains a *refID* attribute that refers to the persistent channel. The value of the *refID* attribute identifies the persistent channel that initiated the **Signal**.

The third way in which a controller may receive a signal is to have the signal channels hard-wired, for example, by a tool such as a list of controller-URLs read from an initialization file. For example, signals may be generated independently when a service is started, or when sub-controllers that are newly connected to a network want to inform other controllers about their capabilities. Hard-wired signals, however, may not have a *refID* attribute. If no *refID* is specified, the corresponding query parameters must be specified instead.

Table 5.5 Contents of the Signal message element

Name	Data Type	Description
<i>LastRepeat</i> ?	Boolean	If <i>true</i> , the persistent channel is being closed by the controller and no further messages will be generated that fulfill the persistent channel criteria. If <i>false</i> , further signals will be sent. For further details, see section 5.2.2.3 Persistent Channels. Default = <i>false</i>
<i>refID</i> ?	NMTOKEN	Identifies the initiating query message that subscribed this signal message. Note that hard-wired signals may not contain a <i>refID</i> attribute.
Notification ?	element	Textual description of the signal. The Notification element should be provided if the severity of the event that caused this signal is greater than <i>warning</i> , or if pure events have been subscribed. For details about subscribing pure events see section 5.5.1.1 Events.
QueryTypeObj ?	element	If no <i>refID</i> is specified, the corresponding query parameters must be specified instead by providing this element. This element is an abstract element and a placeholder for any descriptive elements that provide details for the virtual Query , which, if sent, would convey the same ResponseTypeObj elements. The element type of QueryTypeObj is defined by the <i>Type</i> attribute of the abstract Message element.
ResponseTypeObj *	element	Abstract element that is a placeholder for any descriptive elements that provide details subscribed. These element types are the same as in the Response message element.

Example of a signal message:

```
<JMF TimeStamp="2000-07-25T12:28:01+02:00" SenderID="Press 45">
  <Signal Type="Progress" ID="s123">
    <ProgressQuParams JobID="42" JobPartID="66"/>
    <Progress JobId="p1234" Unit="Page" Quantity="2000"/>
  </Signal>
</JMF>
```

5.2.1.4 Command

A command is syntactically equivalent to a query, but rather than simply retrieving information, it also causes a state change in the target device. The following table contains the contents of a Command message.

Table 5.6 Contents of the Command message element

Name	Data Type	Description
CommandTypeObj *	element	Abstract element that is a placeholder for any descriptive elements that provide details of the command.

The following example demonstrates how a ResumeQueueEntry command may cause a job in a queue to begin executing:

```
<JMF DeviceID="A3 Printer" TimeStamp="2000-07-25T12:32:48+02:00"
SenderID="MIS master A">
  <Command ID="009" Type="ResumeQueueEntry">
    <QueueEntry EntryID="job-0032"/>
  </Command>
</JMF>
```

The following example shows a possible response to the command example above:

```
<JMF ... SenderID="A3 Printer">
  <Response ID="109" Type="ResumeQueueEntry" refID="009">
    <Notification Class="information">
      <Comment>Start Job successful</Comment>
    </Notification>
    ...
  </Response>
</JMF>
```

5.2.1.5 Acknowledge

An Acknowledge message is a response to a command issued by a controller. Each acknowledge message is unidirectional and syntactically equivalent to a command response, and the *refID* attribute of each refers to the initiating command. Acknowledge messages are generated if commands with long latency have been executed in order to inform the command sender about the results. They are announced in the Response message to the command by the setting the attribute *Acknowledged* = true.

Table 5.7 Contents of the Acknowledge message element

Name	Data Type	Description
<i>refID</i>	NMTOKEN	Identifies the initiating command message the acknowledge refers to.
<i>ReturnCode</i> ?	integer	Describes the result. 0 indicates success. For all other possible codes see Appendix H. Default = 0
AcknowledgeTypeObj *	element	Abstract element that is a placeholder for any descriptive elements that provide details about command execution.
Notification	element	Textual description of the command execution.

The following is an example of an acknowledge message:

```
<JMF ... >
  <Acknowledge ID="109" Type="PipePush" refID="010">
    <Notification Class="information">
      <Comment>Requested pipe resource produced successful
    </Comment>
    </Notification>
    ...
  </Acknowledge>
</JMF>
```

5.2.2 JMF Handshaking

JMF can seek to establish communication between system components in several ways. This section describes the actions and appropriate reactions in a communication using JMF.

5.2.2.1 Single Query/Command Response Communication

The handshaking mechanisms for queries and commands are equivalent. The initiating controller sends a **Query** or **Command** message to the target controller. The target parses the **Query** or **Command** and immediately issues an appropriate **Response** message. If a **Command** with long latency is issued, an additional **Acknowledge** message may be sent to acknowledge when the command has been executed.

5.2.2.2 Signal

JMF signal messages are “Fire and Forget.” In other words, no acknowledgment is sent by the receiver besides the standard protocol HTTP response that is sent when a communication link is sought.

5.2.2.3 Persistent Channels

Queries may be made persistent by including a **Subscription** element that defines the persistent channel-receiving end (see also Figure 5.1). The responding controller should initially send a **Response** to the subscribing controller. Then, the responding controller should send **Signal** messages whenever the condition specified by one of the attributes in the following table is true. This is referred to as a **persistent channel**. The *refID* attribute of the **Signal** is defined by the *ID* attribute of the **Query**. In other words, the *refID* of the signal identifies the persistent channel. Any **Query** may be set up as a persistent channel, although in some cases this may not make sense.

Table 5.8 Contents of the Subscription element

Name	Data Type	Description
<i>ChangeAttributes</i> ?	NMTOKENS	Requests an update signal whenever the value of one of the attributes specified by the <i>ChangeAttributes</i> string is modified. A value of “*”, which is the default value, denotes a message request for any attribute change.

<i>NotificationClasses</i>	enumerations	Defines the set of notification classes to be subscribed for. Possible values are: <i>event</i> <i>information</i> <i>warning</i> <i>error</i> <i>fatal</i> For details on Notification elements and classes, see section 3.9.1.2 Notification.
<i>RepeatStep</i> ?	integer	Requests an update signal whenever the <i>Amount</i> associated with the query is an integer multiple of <i>RepeatStep</i> . Default = 0, which means no repeat.
<i>RepeatTime</i> ?	number	Requests an update signal every <i>RepeatTime</i> seconds. Default = no repeat
<i>URL</i>	URL	URL of the persistent channel receiving end.

If a persistent signal channel has been set up and the device knows that this is the last time that the condition for signaling will be *true*, it should set the *LastRepeat* flag of the corresponding **Signal** message to *true*. In general, this will happen for a **Progress** query, as when the job that has been tracked is completed. It may also happen when a device is shut down and will therefore not send any further updates.

If a controller that does not support persistent channels is queried to set up a persistent channel, it must answer the query regularly with a **Response** and set *Subscribed* to *false*.

Multiple attributes of a **Subscription** element are combined as a Boolean OR operation of these attributes. For instance, if *RepeatStep* and *NotificationClasses* are both specified, messages fulfilling either of the requirements are requested. If the subscription element contains only a URL, it is up to the emitting controller to define when to emit messages.

Creating Persistent Channels in a JDF Node

The **NodeInfo** element of a JDF node may contain JMF elements that contains a set of queries (not commands) that define persistent channels. Executing the node is equivalent to sending the messages in the JMF node to the processor whenever the JDF node is executed.

Deleting Persistent Channels

A persistent channel may be deleted by sending a **StopPersistentChannel** command, as described in section 5.5.1.7 **StopPersistentChannel**.

5.3 JMF Messaging Levels

A JDF-conforming controller may opt to support one of the following messaging compliance levels offered by JMF:

- **No messaging** Controllers have the option of supporting no messaging at all. For this level, JDF includes **Audit** records for each process that allow the results of the process to be recorded.

- **Notification** Most controllers will choose to support some level of messaging capability. Notification is the most basic level of support. Devices that support notification provide uni-directional messaging by sending **Signal** messages. Notification messages inform the controller when they begin and complete execution of some process within a job. They may also provide notice of some error conditions. Setup of the notification channel can be defined in a JDF node or hard-wired. In order to set up notification messages via a **NodeInfo** element, the controller must be able to read JMF query elements from a JDF document.
- **Query support** The next level of communication supports queries. Controllers that support queries respond to requests from other controllers by communicating their status using such tools as current *JobID* attributes, queued *JobID* attributes, or current job progress. Queries require bi-directional communication capabilities.
- **Command support** This level of support provides controllers with the ability to process commands. The controller can receive commands, for instance, to interrupt the current job, to restart a job, or to change the status of jobs in a queue.
- **Submission support** Finally, controllers may accept JDF jobs via an HTTP **post** request to the messaging channel. In this case, the messaging channel must support MIME/Multipart/Related documents. For more details on submission, see section 5.6.3.8 *SubmissionMethods*.

5.4 Error and Event Messages

If a command or a query message is not successfully handled, a processor must reply with a standardized response that may contain a **Notification** element. **Notification** elements, described in detail in section 3.9.1.2 *Notification*, convey a textual description. The information contained in the **Notification** element may be used by a user interface to visualize errors.

The response messages **Response** and **Acknowledge** contain an *ReturnCode* attribute. *ReturnCode* defaults to 0, which indicates that the response is successful. In case of success and in responses to commands an informational **Notification** element (*Class=information*) may be provided. In case of a warning, error or fatal error, the *ReturnCode* is greater than 0 and indicates the kind of error committed. Error codes are defined in *Appendix H*.

The following example uses a **Notification** element to describe an error:

```
<JMF ... >
  <Response ID="109" Type="ResumeQueueEntry" refID="009"
  ReturnCode="5">
    <Notification Class="error">
      <Comment>StartJob unsuccessful - Device does not handle
  commands</Comment>
    </Notification>
    ...
  </Response>
</JMF>
```

Notification elements are also used to signal usual events due to any activities of a device, operator, etc., for example scanning a barcode. Such pure events can be subscribed by the **Events** message described in section 5.5.1.1 *Events*.

5.5 Standard Messages

The previous sections in this chapter provide a description of the overall structure of JMF messages. This section contains a list of the standard messages that are defined within the JDF framework. It is not required that every JDF-compliant application support every one of the signals and queries described in this list. It is, however, possible to discover which messages are supported in a workflow. A controller responds to the `KnownMessages` query by publishing a list of all the messages it supports (see section 5.5.1.3 `KnownDevices`, below).

At the beginning of each section there is a table that lists all of the message types in that category. These tables contain three columns. The first is entitled “Message Type,” and it lists the elements that identify each message type. The second column is entitled “Family.” The values in this column describe the kind of message that is applicable in the circumstance being illustrated. The following abbreviations are used to describe the values:

- Q:** Query
- C:** Command
- R:** Response
- S:** Signal
- A:** Acknowledgement

More than one of these values may be valid simultaneously. If that is the case, then all applicable letters are included in the column. Additionally, there are a few special circumstances indicated by a particular combinations of these letters. The letters “QR” or “CR” indicate that all **Query** and **Command** messages cause a **Response** message to be returned. If the message may occur as a **Signal**, either from a subscription or independently, the “Family” field in the table contains additionally the letter “S”. If an additional “A” appears in the column it indicates that commands with long latency should initiate an additional **Acknowledge** message after command execution.

Finally, the third column provides a description of each element.

At the beginning of each section describing the contents and function of the elements listed in the tables described above is a table containing the instantiation—that is, the type—of all of the abstract sub-elements applicable to the element being described. Each table contains an entry that describes the details of the query or command as well as an additional entry that describes the details of the corresponding response or acknowledgement. The tables resemble the following template:

Table 5.9 Messaging table template

Object Type	Element name	Description
Abstract element name of the query or command :	Name and type of the sub-element that defines specifics of the query or command, followed by a cardinality symbol.	Short description of the sub-element(s), if applicable.
Abstract element name of the response to a query or command:	Name and type of sub-element that contains specific information about the response to the query or command followed by cardinality symbol.	Short description of the sub-element(s), if applicable.

The name of the abstract query element is `QueryTypeObj`, and the name of the abstract command element is `CommandTypeObj`. Abstract response elements are either `ResponseTypeObj` or `AcknowledgeTypeObj`.

5.5.1 Controller Registration and Communication Messages

The message types of the following table are defined in order to exchange meta-data about controller or device abilities and for general communication.

Table 5.10 Process registration and communication messages

Message type	Family	Description
Events	QRS	Used to subscribe pure events occurring randomly like scanning of a barcode, activation of function-keys at a console, etc.
KnownControllers	QRS	Returns a list of JMF capable controllers.
KnownDevices	QRS	Returns information about the devices that are controlled by a controller.
KnownJDFServices	QRS	Returns a list of services (JDF Node Types) that are defined in the JDF specification.
KnownMessages	QRS	Returns a list of all messages that are supported by the controller.
RepeatMessages	QR	Returns a set of previously sent messages that have been stored by the controller.
StopPersistentChannel	CR	Closes a persistent channel.

5.5.1.1 Events

Table 5.11 Contents of the Events element

Object Type	Element name	Description
QueryTypeObj	-	-
ResponseTypeObj	-	-

The **Events** message type is intended to be used only to subscribe for pure events of a device or controller. **Pure events**, as described in section 4.5 *Node and Resource IDs*, occur randomly, and result from any activities of a device or operator, such as scanning of a barcode, activating a function-key of a console etc.

The controller that subscribes for **Events** messages receives **Signal** messages that convey only **Notification** elements containing information about the event. The event type and values of these messages may then be provided by the *Type* and *Value* attributes of the **Notification** element, as described in section 3.9.1.2 *Notification*. Possible event types and values are given in Appendix I.

By specifying *EventSeverity* = *error* in the **Subscription** element during subscription, for example, it is possible to subscribe for pure events together with error messages but without information and warning messages.

Example of a subscription of **Events** and the response:

```
<JMF ... >
  <Query Type="Events" ID="170">
    <Subscription NotificationClasses ="event error fatal"
URL="http://www.anycompany.com/MIS/JMF/JobTracker"/>
  </Query>
```

```

</JMF>

<JMF ... >
  <Response ID="1001" refID="170" Type="Events">
    <Notification Class="information">
      <Comment>Event subscription successful;
Controller will provide the following pure events:
Barcode
FCN-Key
SystemTimeSet
anycompany:AnyPrivateEvent_1
anycompany:AnyPrivateEvent_2
Subscribed notification classes:
error
fatal</Comment>
    </Notification>
  </Response/>
</JMF>

```

5.5.1.2 KnownControllers

Table 5.12 Contents of the Known Controllers element

Object Type	Element name	Description
QueryTypeObj	-	-
ResponseTypeObj	JDFController *	Known controllers.

The **KnownControllers** query requests information about the controllers and devices that are known to the controller and may be directly accessed by JMF messaging. **KnownControllers** is designed to define a registration server. A processor that needs information about its system environment can query a registration server for a list of known controllers. This list can subsequently be iterated using the other process registration queries in this section. The URL of the master registration server must be defined using a method outside of JDF.

JDFController

Table 5.13 Contents of the JDFController

Name	Data Type	Description
<i>URL</i>	URL	URL of the controller.

Example of a response to a **KnownControllers** query:

```

<Response ID="1" refID="Q1" Type="KnownControllers">
  <JDFController URL="http://xyz" />
  ...
</Response>

```

5.5.1.3 KnownDevices

Table 5.14 Contents of the KnownDevices element

Object Type	Element name	Description
QueryTypeObj	DeviceFilter ?	Refines the list of devices queried. Only devices that match the DeviceFilter are listed. The default is to return a list of all known devices.
ResponseTypeObj	Device *	The known devices.

The **KnownDevices** query requests information about the devices that are controlled by a controller. If a higher level controller controls lower level controllers, it should also list the devices that are controlled by these. The response is a list of **Device** resources (see section 7.2.31 **Device**) controlled by the controller that receives the query, as demonstrated in the following example:

```
<Response ID="1" refID="Q1" Type="KnownDevices">
  <Device DeviceID="Joe the SpeedMaster" DeviceType="Heidelberg SM102/6
rev. 47.11" />
  ...
</Response>
```

Structure of the DeviceFilter Element

The **DeviceFilter** element refines the list of devices that should be returned. Only devices that match all parameters of one of the **Device** resources specified in the **DeviceFilter** element are included.

Table 5.15 Contents of the DeviceFilter element

Name	Data Type	Description
Device *	element	Only devices that match the attribute values specified in one of these Device resources are included. Devices match the criteria if the attribute values specified here in the Device resource match the equivalent attribute values of the known devices. Unspecified attributes always match.

5.5.1.4 KnownJDFServices

Table 5.16 Contents of the KnownJDFServices element

Object Type	Element name	Description
QueryTypeObj	-	-
ResponseTypeObj	JDFService *	Processes that the controller or device can execute.

The **KnownJDFServices** query returns a list of services that are defined in the JDF specification, such as **ConventionalPrinting**, **RIPping**, or **EndSheetGluing**. It allows a controller to publish the services that the devices it controls are capable of providing. The response is a list of **JDFService** elements, one for each supported process type.

JDFService

JDFService elements define the node types that can be processed by the controller. A JDF processor should be capable of processing *Combined* nodes of any of the individual **JDFService** elements that are specified. It is therefore not necessary to define every permutation of allowed combinations. It need not be

able to process individual nodes with a type defined in the *Types* attribute of a *Combined JDFService* element.

Table 5.17 Contents of the *JDFService* element

Name	Data Type	Description
<i>Type</i>	NMTOKEN	JDF <i>Type</i> attribute of the supported process.
<i>Types</i> ?	NMTOKENS	If <i>Type</i> = <i>Combined</i> , this attribute represents the list of combined processes. For details, see section 3.2.3.

The following is an example of a response to a *KnownJDFServices* query:

```
<Response ID="1" refID="Q1" Type="KnownJDFServices">
  <JDFService Type="Rendering" />
  <JDFService Type="Folding" />
  <JDFService Type="Combined" Types="Gather Stitch"/>
  ...
</Response>
```

5.5.1.5 KnownMessages

Table 5.18 Contents of the *KnownMessages* element

Object Type	Element name	Description
QueryTypeObj	KnownMsgQuParams ?	Refines the query for known messages. If not specified, list all supported message types.
ResponseTypeObj	MessageService *	Specifies the supported messages.

The *KnownMessages* query returns a list of all message types that are supported by the controller.

KnownMsgQuParams

The flags of the *KnownMsgQuParams* element filter out the types of messages that should be included in the response list. Multiple flags are allowed.

Table 5.19 Contents of the *KnownMsgQuParams* element

Name	Data Type	Description
<i>ListCommands</i> ?	boolean	Lists all supported command types. Default = <i>true</i>
<i>ListQueries</i> ?	boolean	Lists all supported query types. Default = <i>true</i>
<i>ListSignals</i> ?	boolean	Lists all supported signal types. Default = <i>true</i>
<i>Persistent</i> ?	boolean	If <i>true</i> , only lists messages that may use persistent channels. If <i>false</i> , ignores the ability to use persistent channels. Default = <i>false</i>

MessageService

The response is a list of **MessageService** elements, one for each supported message type. The flags of the **MessageService** response element are set in each **MessageService** entry. They define the supported usage of the message by the controller. Multiple flags are allowed.

Table 5.20 Contents of the *MessageService* element

Name	Data Type	Description
<i>Command</i> ?	boolean	If <i>true</i> the message is a command. Default = <i>false</i>
<i>Persistent</i> ?	boolean	If <i>true</i> the message is supported as a persistent channel. Default = <i>false</i>
<i>Query</i> ?	boolean	If <i>true</i> the message is a query. Default = <i>false</i>
<i>Signal</i> ?	boolean	If <i>true</i> the message is a signal. Default = <i>false</i>
<i>Type</i>	NMTOKEN	Type of the supported message.

The following is an example of a response to a **KnownMessages** query:

```
<Response ID="1" refID="Q1" Type="KnownMessages">
  <MessageService Type="KnownMessages" Query="true"/>
  <MessageService Type="Status" Query="true" Signal="true"
  Persistent="true">
  ...
</Response>
```

5.5.1.6 RepeatMessages

Table 5.21 Contents of the *RepeatMessages* element

Object Type	Element name	Description
QueryTypeObj	MsgFilter ?	A filter for the messages to be repeated.
ResponseTypeObj	Message *	The recent messages queried.

The **RepeatMessages** query returns a list of messages that have been previously sent by the controller. The optional **MsgFilter** element allows the list to be filtered. The list of JMF messages that fulfill the filter criteria may be sorted by time, with the most recent listed first. This specification places no requirements on the size of the message buffer of a controller that supports **RepeatMessages**.

Structure of the MsgFilter Element

Table 5.22 Contents of the *MsgFilter* element

Name	Data Type	Description
<i>After</i> ?	timeInstant	Messages sent only after a certain time.
<i>Before</i> ?	timeInstant	Messages sent only before a certain time.

<i>Count</i> ?	integer	Maximum number of messages, most recent first.
<i>DeviceID</i> ?	string	ID of the device whose messages are required.
<i>Family</i> ?	enumeration	Message family. Possible values are: <i>Acknowledge</i> <i>Response</i> <i>Signal</i> <i>All</i> – Default value. <i>Response</i> , <i>Signal</i> , and <i>Acknowledge</i> messages are queried.
<i>MessageRefID</i> ?	string	The <i>refID</i> attribute must match the value of <i>MessageRefID</i> .
<i>MessageID</i> ?	string	The <i>ID</i> attribute must match the value of <i>MessageID</i> .
<i>MessageType</i> ?	string	<i>Type</i> attribute of the requested messages.
<i>ReceiverURL</i> ?	URL	URL for which the messages are intended.

If the list is incomplete because the parameters supplied in the `MsgFilter` element cannot be fulfilled by the application, the *ReturnCode* may be 108 (empty list) or 109 (incomplete list) and should be flagged as a warning.

The following is an example of a response to a `RepeatMessages` query. Note the nesting of `Response` messages, where the first layer is the response to the `RepeatMessages` query and its contents are the repeated messages.

```
<JMF TimeStamp="2000-06-14T12:11+02:00" ... >
  <Response ... >
    <Response Time="2000-06-14T11:00+02:00" ... />
    <Response Time="2000-06-14T10:50+02:00" ... />
    <Signal Time="2000-06-14T08:20+02:00" ... />
    <Signal Time="2000-06-14T03:01+02:00" ... />
    ...
  </Response>
</JMF>
```

5.5.1.7 StopPersistentChannel

Table 5.23 Contents of the `StopPersistentChannel` element

Object Type	Element name	Description
CommandTypeObj	StopPersChParams	Specifies the persistent channel and the message types to be unsubscribed.
ResponseTypeObj	-	-

The `StopPersistentChannel` command unregisters a listening controller from a persistent channel. No more messages are sent to the controller once the command has been issued. A certain subset of signals may be addressed for unsubscription by specifying a `StopPersChParams` element.

Structure of the StopPersChParams Element

If the optional attributes are not specified, those attributes default to match anything. Therefore it may be possible to cancel the persistent channel for messages belonging to a certain type of message or to a certain job.

Table 5.24 Contents of the *StopPersChParams* element

Name	Data Type	Description
<i>ChannelID</i> ?	string	<i>ChannelID</i> of the persistent channel to be deleted. If the channel has been created with a <i>Query</i> message, the <i>ChannelID</i> specifies the <i>ID</i> of the <i>Query</i> message (identical to the <i>refID</i> of the <i>Response</i> message).
<i>MessageType</i> ?	string	Only messages with a matching message type are suppressed. Message types are specified in the <i>Type</i> attribute of each <i>Message</i> element.
<i>DeviceID</i> ?	string	Only messages from devices or controllers with a matching <i>DeviceID</i> attribute are suppressed.
<i>JobID</i> ?	string	Only messages with a matching <i>JobID</i> attribute are suppressed.
<i>JobPartID</i> ?	string	Only messages with a matching <i>JobPartID</i> attribute are suppressed.
<i>URL</i>	URL	URL of the receiving controller. This must be identical to the URL that was used to create the persistent channel. If no <i>ChannelID</i> is specified, all persistent channels to this URL are deleted.

5.5.2 Device/Operator Status and Job Progress Messages

JDF Messaging provides methods to trace the status of individual devices and resources and additional job-dependent job-tracking data. JDF makes a distinction between the *Status* of a device and of a job. The status of a job is described by the *Progress* and *Phase* of that job.

Devices are uniquely identified by a name—that is, by the attribute *DeviceID* of the **Device** resource (see section 7.2.31 *Device*)—while controllers are uniquely identified by their URL. In other words, controllers are implicitly identified as a result of the fact that they are responding to a message.

One controller may control multiple devices.

The following queries and commands are defined for status and progress tracking:

Table 5.25 Status and progress messages

Query	Family	Description
Occupation	QRS	Queries the occupation of an employee.
Resource	QRSC	Queries and/or modifies JDF resources that are used by a device, such as device settings, or by a job. This message can also be used to query the level of consumables in a device.
Status	QRS	Queries the general status of a device, controller or job.
Track	QRS	Queries the location of a given job or job part.

5.5.2.1 Occupation

Table 5.26 Contents of the Occupation element

Object Type	Element name	Description
QueryTypeObj	EmployeeDef *	Defines the employees queried.
ResponseTypeObj	Occupation *	The occupation status of the employees.

Occupation queries the occupation status of an employee. No job context is required to issue an Occupation message.

Structure of the EmployeeDef Element

The Occupation query may be focused to certain employees specifying a EmployeeDef element. If no EmployeeDef element is specified, a list of all known employees is returned.

Table 5.27 Contents of the EmployeeDef element

Name	Data Type	Description
<i>PersonalID</i> ?	string	<i>PersonalID</i> of the employee being tracked.

Structure of the Occupation Element

The response returns a list of Occupation elements for the queried employees. These elements consist of one entry for every job that is currently being executed. The list format accommodates both employees that service multiple jobs or job parts in parallel and multiple employees working on one job.

Table 5.28 Contents of the Occupation element

Name	Data Type	Description
<i>Busy</i> ?	number	Busy state of the employee in percentage. A value of 100, the default, means that the employee is fully occupied with this task. The sum of all <i>Busy</i> values should not exceed 100.
<i>Device</i> *	element	Devices that the employee is currently assigned to.
<i>JobID</i> ?	string	<i>JobID</i> of the JDF node that the employee is assigned to. If no JobID is specified but devices are, the employee is performing non job-related tasks.
<i>JobPartID</i> ?	string	Job part ID of the JDF node that is currently being executed.
Employee	element	Description of the employee being tracked.

The following is an example of response to an Occupation query:

```
<Response ID="1" refID="Q1" Type="Occupation">
  <!--Two jobs on one device with one operator-->
  <Occupation JobID="J1" Busy="30">
    <Employee PersonalID="P1234"/>
    <Device Name="Joe"/>
  </Occupation>
```



```

<Occupation JobID="J2" Busy="70">
  <Employee PersonalID="P1234"/>
  <Device Name="Joe"/>
</Occupation>
<!--Another operator on job j2 -->
<Occupation JobID="J2" Busy="50">
  <Employee PersonalID="P4321"/>
  <Device Name="Joe"/>
</Occupation>
<!--No Job context -->
<Occupation Busy="0">
  <Device Name="John"/>
  <Employee PersonalID="P5678"/>
</Occupation>
</Response>

```

5.5.2.2 Resource

The **Resource** message can be used as a command or a query to modify or to query JDF resources. In both cases (query and command), it is possible to address either global device resources, such as device settings, or job-specific resources. The query simply retrieves information about the resources without modifying them, while the command modifies those settings within the resource that are specified. Settings that are not specified remain unchanged.

Structure of the Resource Query Message

Table 5.29 Contents of the Resource query message element

Object Type	Element Name	Description
QueryTypeObj	ResourceQuParams ?	Specifies the resources queried.
ResponseTypeObj	ResourceInfo *	Contains the amount data of resources and, if requested, the resources itself.

The **Resource** query may be made selective by specifying a **ResourceQuParams** element. The presence of the *JobID* attribute determines whether global device resources or job-related resources are returned. If no **ResourceQuParams** element is specified, only the global device resources are returned.

The query response returns a list of **ResourceInfo** elements that contains the queried information concerning the resources. If the list is empty because the selective query parameters of the **ResourceQuParams** do lead to a null selection of the known device/job resources, then the *ReturnCode* may be 103 (JobID unknown), 104 (JobPartID unknown) or 108 (empty list) and should be flagged as a warning.

Structure of the ResourceQuParams Element

Table 5.30 Contents of the ResourceQuParams element

Name	Data Type	Description
<i>Class</i> ?	enumerations	List of the resource classes to be queried. For example, in order to query the actual level of consumables in a device outside of any job context, specify <i>Class</i> = <i>Consumable</i> in the query without a <i>JobID</i> attribute. Defaults to any class. For possible resource class names, see the <i>Class</i> attribute

		in Table 3.9.
<i>Exact ?</i>	boolean	Requests an exact description of the JDF resource. If <i>true</i> , the response should also return the requested JDF resource. Default = <i>false</i>
<i>Location ?</i>	string	Identifies the location of a resource, such as paper tray, ink container, or thread holder. The name is the same name used in the attribute <i>LocName</i> of the Location sub-element of a physical resource (see also Table 3.11). Default = all locations
<i>ResourceName ?</i>	NMTOKEN	Name of the resource being queried. For possible resource names, see titles in Chapter 7 Resources.
<i>ResourceUsage ?</i>	NMTOKEN	Selects a resource in which the value of the <i>ProcessUsage</i> attribute of the resource link (see Table 3.13) matches the token specified here in this attribute. Only necessary if a resource name is used more than once by one node. For example, the Component output resources of a ConventionalPrinting process can be distinguished by specifying <i>ResourceUsage</i> = <i>good</i> and <i>ResourceUsage</i> = <i>waste</i> , respectively. The <i>ResourceName</i> and <i>ResourceUsage</i> attributes are combined by a logical “and” conjunction to select the resource to be queried.
<i>JobID ?</i>	string	Job ID of the JDF node that is being queried. If no <i>JobID</i> is specified, global device settings are queried.
<i>JobPartID ?</i>	string	Job part ID of the JDF node that is being queried.

Structure of the Resource Command Message

Table 5.31 Contents of the Resource command message element

Object Type	Element name	Description
CommandTypeObj	ResourceCmdParams	Specifies the resources to be modified.
ResponseTypeObj	ResourceInfo *	Contains information about the resources and the resources after modification.

The Resource command may be used to modify either global device settings or a running job. It may be made selective by specifying the optional attributes in the **ResourceCmdParams** element. The presence of the *JobID* attribute determines whether global device resources or job-related resources are modified.

The response contains a list of **ResourceInfo** elements with all resources and private extensions of the device after the changes have been applied. The type of the resource that is given as a response depends on the type of the resource given in the command.

If the resource command was successful, the value of the *ReturnCode* attribute is 0. If it is not successful, the value of *ReturnCode* may be one of those that have been described above in the section about the Resource query message; 200, which means invalid resource parameters; or 201, which means insufficient resource parameters. Partial application of the resource should also be flagged as a warning. If the value of *ReturnCode* is larger than 0, the controller that issued the command can evaluate the returned resource in order to find the setting that could not be applied.

Structure of the ResourceCmdParams Element

Table 5.32 Contents of the ResourceCmdParams element

Name	Data Type	Description
<i>Exact</i> ?	boolean	Requests an exact description of the JDF resource. If <i>true</i> , the response should also return the requested JDF-resource. Default = <i>false</i>
<i>JobID</i> ?	string	Job ID of the JDF node that is being modified. If no <i>JobID</i> is specified, global device settings are modified.
<i>JobPartID</i> ?	string	Job part ID of the JDF node that is being modified.
<i>ResourceName</i> ?	NMTOKEN	Name of the resource whose production amount shall be modified. For possible resource names see titles in Chapter 7 Resources. Defaults to any name.
<i>ResourceUsage</i> ?	NMTOKEN	Selects a resource in which the value of the <i>ProcessUsage</i> attribute of the resource link (see Table 3.13) matches the token specified here in this attribute. Only necessary if a resource name is used more than once by one node. For example, the Component output resources of a ConventionalPrinting process can be distinguished by specifying <i>ResourceUsage</i> = <i>good</i> and <i>ResourceUsage</i> = <i>waste</i> , respectively. The <i>ResourceName</i> and <i>ResourceUsage</i> attributes are combined by a logical “and” conjunction to select the resource to be queried.
<i>ProductionAmount</i> ?	number	New amount of resource production. This value replaces the <i>Amount</i> in the output resource link of the resource specified by the <i>ResourceName</i> attribute.
Resource *	element	Resources to be uploaded to the controller. The resources to be modified are identified by their <i>ID</i> values, which means that the <i>ID</i> attributes must be known to the controller that issued the Resource command.

Structure of the ResourceInfo Element

Table 5.33 Contents of the ResourceInfo element

Name	Data Type	Description
<i>Amount</i> ?	number	Intended amount for consumption or production of a resource in a job context. This corresponds to the value of the <i>Amount</i> attribute in the corresponding resource link of the resource.
<i>AvailableAmount</i> ?	number	Device-specific amount of the <i>Consumable</i> resource that is available in the device.
<i>Level</i> ?	enumeration	This attribute is device dependent. A device may specify the level status that describes a low or empty consumable level. Possible values are: <i>empty</i> – Specification is left to the device manufacturer.

		<i>low</i> – Specification is left to the device manufacturer.
		<i>OK</i> – Default value.
<i>LocName</i>	string	Device-specific string to identify the location of a given consumable, such as paper tray, ink container, or thread holder. The name is the same name used in the Location sub-element of a physical resource. Default = all locations
<i>ResourceName ?</i>	NMTOKEN	Name of the resource if <i>Exact</i> = <i>false</i> in the query. Only one of Resource or <i>ResourceName</i> shall be specified.
<i>ResourceUsage ?</i>	NMTOKEN	Selects a resource in which the value of the <i>ProcessUsage</i> attribute of the resource link (see Table 3.13) matches the token specified here in this attribute. Only necessary if a resource name is used more than once by one node. For example, the Component output resources of a ConventionalPrinting process can be distinguished by specifying <i>ResourceUsage</i> = <i>good</i> and <i>ResourceUsage</i> = <i>waste</i> , respectively. The <i>ResourceName</i> and <i>ResourceUsage</i> attributes are combined by a logical “and” conjunction to select the resource to be queried.
<i>Unit ?</i>	string	Unit of the amount attributes. In a job-context it is strongly discouraged to specify a unit other than the unit defined in the respective JDF resource, although this may be necessary due to technical considerations, such as when Ink is specified in weight (g) and ink measurement is specified in volume (liter).
<i>CostCenter ?</i>	element	Cost center to which the resource consumption is allocated.
<i>Resource ?</i>	element	JDF description of the resource.

The following is an example for retrieving settings:

```
<Query ID="Q1" Type="Resource">
  <ResourceQuParams Class="Consumable" Exact="true"/>
</Query>
```

The following is a possible response to the query above:

```
<Response ID="1" refID="Q1" Type="Resource">
  <ResourceInfo LocName="Paper Tray 1" AvailableAmount="2120" >
    <Media>
      ... <!-- Media resource defined in JDF -->
    </Media>
  </ResourceInfo>
  <ResourceInfo LocName="Ink1" AvailableAmount="0" Unit="l"
Level="Empty">
    <Ink>
      ... <!-- Ink description resource defined in JDF -->
    </Ink>
  </ResourceInfo>
</Response>
```

The following is an example for modifying the production amount of a specific job to produce brochures:

```
<Command ID="C1" Type="Resource">
  <ResourceCmdParams JobID="MakeBrochure 012" ResourceName="Component "
  ProductionAmount="7500"/>
</Command>
```

The following is a possible response to the resource command above:

```
<Response ID="2" refID="C1" Type="Resource">
  <ResourceInfo Amount="7500" ResourceName="Component" />
</Response>
```

5.5.2.3 Status

Table 5.34 Contents of the Status element

Object Type	Element name	Description
QueryTypeObj	StatusQuParams	Refines the query to include various aspects of the device and job states.
ResponseTypeObj	DeviceInfo	Describes the actual device status.
	Queue ?	Provides information about the queue and all its entries. This element will only be provided if the device has queue capabilities. The Queue element is described in section 5.6.4 Queue-Handling Elements.

The **Status** message queries the general status of a device or a controller and the status of jobs associated with this device or controller. No job context is required to issue a **Status** message. The response contains one **DeviceInfo** element, which contains the device specific information and which may contain other **JobPhase** elements that in turn contain the job specific information. The response also provides a **Queue** element when commanded to do so.

Structure of the StatusQuParams Element

The various aspects of the device, queue, and job states may be refined by the **StatusQuParams** element. This element contains three groups of parameters. The first group serves to refine the device-specific status information queried. The parameters *EmployeeInfo* and *ModuleDetails* belong to this group. The second group serves to refine the job specific status information. These are *JobDetails*, *JobID*, and *JobPartID*. And the third determines simply whether a queue element should be appended. This is specified by the attribute *QueueInfo*.

In order to focus on the status of a certain job the job must be uniquely identified using the *JobID* attribute. It may be necessary to define a process or a part of a job as the query target under certain circumstances, such as when a job is processed in parallel. This is accomplished using the *JobPartID* attribute of the **StatusQuParams** element. A value of *JobDetails = full* requests a complete JDF description of a snapshot of the specified job or job part.

If the specified job or job part is unknown, the value of the *ReturnCode* attribute is 103 or 104 (for error codes see Appendix H).

Table 5.35 Contents of the StatusQuParams element

Name	Data Type	Description
<i>EmployeeInfo</i> ?	boolean	If <i>true</i> , Employee elements may be provided in the response.

		Those elements describe the employees which are associated to the device independent on any job. Default = <i>false</i> .
<i>JobDetails ?</i>	enumeration	Refines the provided status information about the jobs associated with the device. Each higher entry includes the values specified in the lower entries. Possible values are: <i>none</i> – Default value. Specify only <i>JobID</i> , <i>JobPartID</i> and <i>Amount</i> and/or <i>PercentCompleted</i> . <i>MIS</i> – Provide business with the relevant information contained in the <i>CostCenter</i> element and the <i>Deadline</i> , <i>DeviceStatus</i> , <i>Status</i> , <i>StatusDetails</i> , and the various <i>Counter</i> attributes. <i>brief</i> – Provide all available status information except for JDF. <i>full</i> – Provide maximum available status information. Includes a actual JDF which represents a snapshot of the current job state.
<i>JobID ?</i>	string	Job ID of the JDF node whose status is being queried. Defaults to list all known jobs.
<i>JobPartID ?</i>	string	JobPart ID of the JDF node whose status is being queried.
<i>ModuleDetails ?</i>	enumeration	Refines the provided status information about the states of device modules independent on any job. Possible values are: <i>none</i> – The default. No <i>ModuleStatus</i> elements are queried. <i>brief</i> – <i>ModuleStatus</i> elements may be provided without status details and without module specific employee information. <i>full</i> – Provide maximum available status information.
<i>QueueInfo ?</i>	boolean	If <i>true</i> , a <i>Queue</i> element may be provided. This is analogous to a <i>QueueStatus</i> query (see section 5.6.3.6 <i>QueueStatus</i>). Default = <i>false</i> .

Structure of the DeviceInfo Element

The response returns a *DeviceInfo* element for the queried device.

Table 5.36 Contents of the *DeviceInfo* element

Name	Data Type	Description
<i>CounterUnit ?</i>	string	The unit of the <i>ProductionCounter</i> , the <i>TotalProductionCounter</i> and nominator unit of <i>Speed</i> . The default unit is the default unit defined by JDF for the output resource of the node executed by the device. For example, in case of a sheet printer it is the number of sheets, in case of a web printer, it is the length of printed web in meters.
<i>DeviceStatus</i>	enumeration	The status of a device. Possible values are: <i>unknown</i> – No device is known or the device cannot

provide a *DeviceStatus*. The default.

idle – No job is being processed and the device is accepting new jobs.

down – If no job is being processed and the device currently cannot execute a job. The device may be broken, switched off, etc.

setup – The device is currently being set up. This state is allowed to occur also during the execution of a job.

running – The device is currently executing a job.

cleanup – The device is currently being cleaned. This state is allowed to occur also during the execution of a job.

stopped – The device has been stopped, but running may be resumed later. This status may indicate any kind of break, including a pause, maintenance, or a breakdown, as long as execution has not been aborted.

<i>HourCounter</i> ?	timeDuration	The total integrated time (life time) of device operation in hours. Defaults to unknown.
<i>PowerOnTime</i> ?	timeInstant	Date and time when the device was switched on. Defaults to unknown.
<i>ProductionCounter</i> ?	number	The current machine production counter. This counter can be reset. Typically, it starts counting at power-on time. The reset of this counter may be signaled by an Events message of <i>Type</i> = <i>CounterReset</i> (see Appendix I). Defaults to unknown.
<i>Speed</i> ?	number	The current machine speed. <i>Speed</i> is defined in the same units as <i>ProductionCounter</i> / hour. Defaults to unknown.
<i>StatusDetails</i> ?	string	String that defines the device state more specifically. For a list of supported values, see Appendix F.
<i>TotalProductionCounter</i> ?	number	The current total machine production counter. Defaults to unknown.
<i>CostCenter</i> ?	element	The cost center that the device is currently being charged to. Defaults to unknown.
<i>Device</i> ?	element	A Device resource that describes details of the device.
<i>Employee</i> *	element	Employee resources that describe which employees are currently working at the device.
<i>JobPhase</i> *	element	Describes the actual status of jobs in the device. For details on using JobPhase elements, see Table 5.37.
<i>ModuleStatus</i> *	element	Status of individual modules. For details on using ModuleStatus elements, see Table 5.38.

Structure of the JobPhase Element

A **Status** response may provide **JobPhase** elements. The **JobPhase** element represents the actual state of a job. The **JobPhase** element is an analogue to the **PhaseTime** audit element described in section

3.9.1.3 **PhaseTime**. The main difference between a **JobPhase** element and a **PhaseTime** audit element is that a **Phase** message reflects a snapshot of the current job status whereas the **PhaseTime** audit reflects a timespan bordered by two (sub-)status transitions.

For exact information about the job phase a **JobPhase** element may embed a copy of the current state of the job described as JDF. If an actual JDF is not supported by the controller, the same rules apply for the **Status** response as those which apply for the **Consumable** response.

Table 5.37 Contents of the *JobPhase* element

Name	Data Type	Description
<i>Amount?</i>	number	Produced amount. The unit is specified in the <i>CounterUnit</i> attribute of the parent element <i>DeviceInfo</i> .
<i>DeadLine?</i>	enumeration	Scheduling state of the job. Possible values are: <i>InTime</i> – The job or job part will probably not miss the deadline. <i>Warning</i> – The job or job part could miss the deadline. <i>Late</i> – The job or job part will miss the deadline. Default = <i>InTime</i> For more details on scheduling, see section 3.5 Process and Node Information.
<i>JobID?</i>	string	Job ID of the JDF node the <i>JobPhase</i> belongs to.
<i>JobPartID?</i>	string	Job part ID of the JDF node the <i>JobPhase</i> belongs to.
<i>PercentCompleted?</i>	number	Node processing progress in % completed.
<i>QueueEntryID?</i>	string	If the job was submitted to a <i>Queue</i> , and the <i>QueueEntryID</i> is known, this attribute should be provided.
<i>Speed?</i>	number	The current job speed. <i>Speed</i> is defined in the same units as <i>ProductionCounter</i> / hour. Defaults to the speed specified in the <i>DeviceInfo</i> element.
<i>Status</i>	enumeration	The status of the JDF node. Possible values are the same as the possible values of a JDF node's <i>Status</i> attribute: <i>waiting</i> <i>quoted</i> <i>ready</i> <i>failed_testrun</i> <i>setup</i> <i>in_progress</i> <i>cleanup</i> <i>spawned</i> <i>stopped</i> <i>completed</i> <i>aborted</i> For details, see Table 3.3 Contents of a JDF node.
<i>StatusDetails?</i>	string	String that defines the job state more specifically. For a list of supported values, see Appendix F.

CostCenter ?	element	The cost center that the job is currently being charged to. Defaults to the cost center specified in the <code>DeviceInfo</code> element.
JDF ?	element	Complete JDF node that represents a snapshot of the job that is currently being processed.
Part ?	element	Describes which part of a job is currently being processed.

Structure of the ModuleStatus Element

The `ModuleStatus` element is identical to the `ModulePhase` element of the `PhaseTime` audit element (see Table 3.25), except that the attributes `Start` and `End` are missing. These attributes specify the time interval in the audit pendant `ModulePhase` and the `DeviceID` attribute, which, here, is unnecessary. The `ModuleStatus` element is described in the following table.

Table 5.38 Contents of the ModuleStatus element

Name	Data Type	Description
<code>DeviceStatus</code>	enumeration	Status of the module. Possible values are: <i>idle</i> – The module is not used. An example is a color print module that is inactive during a black-white print. <i>down</i> – The module cannot be used. It may be broken, switched off etc. <i>setup</i> – The module is currently being set up. <i>running</i> – The module is currently executing. <i>cleanup</i> – The module is currently being cleaned. <i>stopped</i> – The module has been stopped, but running may be resumed later. This status may indicate any kind of break, including a pause, maintenance, or a breakdown, as long as running can be easily resumed.
<code>ModuleIndex</code>	IntegerRange-List	0-based indices of the module or modules. If multiple module types are available on one machine, indices must also be unique.
<code>ModuleType</code>	NMTOKEN	Module description. The allowed values depend on the type of device that is described. The predefined values are listed in Appendix G.
<code>StatusDetails</code> ?	string	Description of the module status phase that provides details beyond the enumerative values given by the <code>DeviceStatus</code> attribute. For a list of supported values, see Appendix F.
<code>Employee</code> *	element	Links to <code>Employee</code> resources that are working at this module (the module is specified by the attributes <code>ModuleIndex</code> and <code>ModuleType</code>).

The following is an example of a response to a `Status` query. The device in this example holds one job and executes another job that is currently printed duplex each side on four-color modules for the front and three-color modules for the back, with one idle:

```
<Response ID="1" refID="Q1" Type="Status">
  <DeviceInfo JobID="678" JobPartID="01" DeviceStatus="running"
  StatusDetails="Waste">
```

```

    <JobPhase Amount="2560" DeadLine="InTime" JobID="678"
JobPartID="01" PercentCompleted="52" QueueEntryID="Job-05"
Status="in_progress" StatusDetails="Waste"/>
    <JobPhase Amount="0" DeadLine="Warning" JobID="679" JobPartID="01"
PercentCompleted="0" QueueEntryID="Job-06" Status="ready"/>
    <ModuleStatus ModuleIndex="0~3 6~8" ModuleType="PrintModule"
DeviceStatus="running"/>
    <ModuleStatus ModuleIndex="4" ModuleType="PrintModule"
DeviceStatus="idle"/>
    <ModuleStatus ModuleIndex="5" ModuleType="PerfectingModule"
DeviceStatus="running"/>
  </DeviceInfo>
</Response>

```

5.5.2.4 Track

Table 5.39 Contents of the Track element

Object Type	Element name	Description
QueryTypeObj	TrackFilter ?	Refines the Track query.
ResponseTypeObj	TrackResult *	Details of the tracked jobs

The Track query requests information about the location of Jobs that are known by a controller. If a higher level controller controls lower level controllers, it should also list the jobs that are controlled by these. The response is a list of TrackResult elements.

Structure of the TrackFilter Element

The TrackFilter element refines the list of **TrackResults** that should be returned. Only jobs that match all parameters specified are included.

Table 5.40 Contents of the TrackFilter element

Name	Data Type	Description
<i>JobID</i> ?	string	Job ID of the JDF node that is being tracked. Defaults to list JobPhase elements of all known nodes.
<i>JobPartID</i> ?	string	JobPart ID of the JDF node that is being tracked.
<i>Status</i>	enumerations	The status of the jobs being tracked. Possible values are a combination of any of the possible values of a JDF node's Status attribute: <i>waiting</i> <i>quoted</i> <i>ready</i> <i>failed_testrun</i> <i>setup</i> <i>in_progress</i> <i>cleanup</i> <i>spawned</i> <i>stopped</i>

*completed**aborted*

For details, see Table 3.3 Contents of a JDF node.

Structure of the TrackResult Element

One **TrackResult** is returned for each known job or spawned job part. **TrackResult** elements contain information about the location of distributed jobs.

Table 5.41 Contents of the TrackResult element

Name	Data Type	Description
<i>JobID</i>	string	Job ID of the JDF node that is being tracked.
<i>JobPartID ?</i>	string	JobPart ID of the highest level node of the JDF node that is being tracked.
<i>URL</i>	URL	URL of the controller that owns this job.
<i>IsDevice</i>	boolean	If <i>true</i> , the controller that emitted this message is the device that has access to the job and may be queried for details of the job.

The following is an example of a response on a Track message:

```
<Response ID="1" refID="Q1" Type="Track">
  <TrackResult URL="http://wherever.controller.de/MyController"
  JobID="1" JobPartID="42" IsDevice="true" />
  ...
</Response>
```

5.5.3 Pipe Control

JDF Messaging provides methods to control dynamic pipes. Dynamic pipes are described in detail in section 4.3.2 Overlapping Processing Using Pipes.

Table 5.42 Dynamic pipe messages

Query	Family	Description
PipeClose	CRA	Closes a pipe because no further resources are required. This is typically used to terminate the producing process.
PipePull	CRA	Requests a new resource from a pipe.
PipePush	CRA	Notifies that a new resource is available in a pipe.
PipePause	CRA	Pauses a process if no further resources can be consumed or produced.

5.5.3.1 PipeClose

Table 5.43 Contents of the PipeClose element

Object Type	Element name	Description
CommandTypeObj	PipeParams	Describes the pipe resource.
ResponseTypeObj	JobPhase	The status of the responding process.
AcknowledgeTypeObj	PipeCmdResult	Describes the execution result of the pipe command (see Table 5.46).

The PipeClose message notifies the process at the other end of a dynamic pipe that the sender of this message needs no further resources or will produce no further resources through the pipe.

The PipeClose command response is equivalent to the PipePull and PipePush command responses described below. The PipeParams and the PipeCmdResult are described in section 5.5.3.2 PipePull.

5.5.3.2 PipePull

Table 5.44 Contents of the PipePull element

Object Type	Element name	Description
CommandTypeObj	PipeParams	Describes the requested pipe resource.
ResponseTypeObj	JobPhase	The status of the responding process.
AcknowledgeTypeObj	PipeCmdResult	Describes the execution result of the pipe command.

The PipePull message requests resources that are described in a JDF dynamic pipe (see sections 3.6.3 Pipe Resources and 4.3.2 Overlapping Processing Using Pipes). PipePull messages are the JMF equivalent of a dynamic input resource link. Figure 5.2, below, depicts the mode of operation of a PipePull message.

The PipePull command response returns a *ReturnCode* of 0 if the command has been accepted by the receiving controller. If not successful the *ReturnCode* may be one of the codes presented in Appendix H. The response may contain a Notification element. The JobPhase element (see section 5.5.2.3 Status) returned should provide only the *Status* attribute that describes the job status of the responding process after receiving the command.

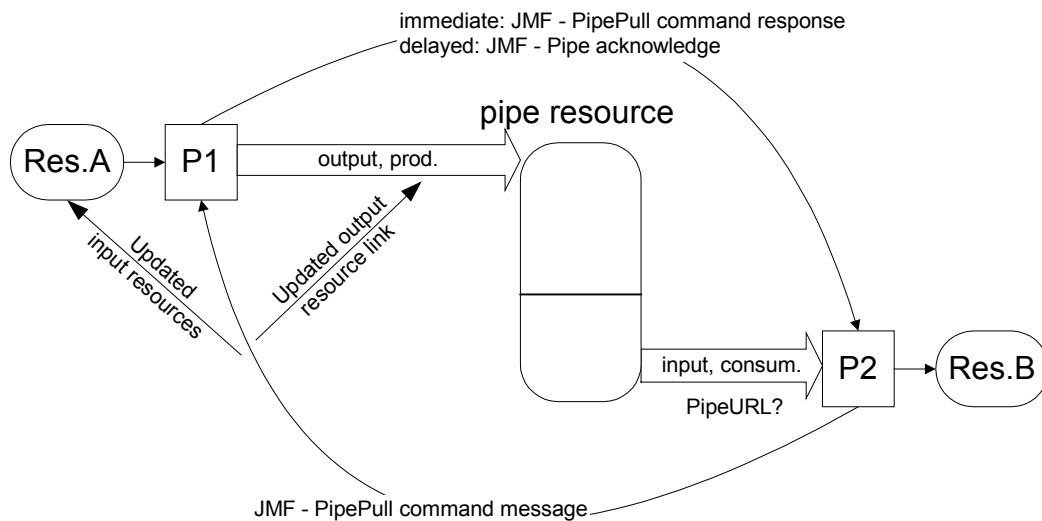


Figure 5.2 Mechanism of a PipePull message

Structure of the PipeParams Element

The PipeParams element is also used by the messages PipeClose, PipePush, and PipePause.

Table 5.45 Contents of the PipeParams element

Name	Data Type	Description
<i>PipeID</i>	string	PipeID of the JDF resource that defines the dynamic pipe.
<i>ReturnURL</i> ?	URL	URL where the Acknowledge should be sent when the pipe command has been executed (PipeClose: when the process has been finished, PipePull: when the resource is available, PipePush: when the resource has been accepted, and PipePause: when the process has been stopped). In general, this is the URL of the controller that is issuing the pipe command.
<i>Status</i> ?	enumeration	Process status after the request. Possible values are defined in Table 3.3. Default = <i>in_progress</i>
<i>Resource</i> *	element	Updated input resources to be used by the process that receives the pipe command (PipePull: the receiver creates the pipe resource, PipePush: the receiver consumes the pipe resource, and PipePause: the receiver only updates the inputs). The resource to be updated is identified by the <i>ID</i> , that means the <i>ID</i> attribute must be known to the controller that issued the pipe command (possible commands are: PipePull, PipePush, or PipePause). In case of the PipeClose command, the resources are ignored.
<i>ResourceLink</i> ?	element	Updated resource link to the pipe resource (PipePull: it is an output link, PipePush: it is an input link, and PipePause: depends on the pipe end). This resource link may be used by the process that links to the pipe resource. The attributes <i>rRef</i> and <i>Usage</i> of a resource link must not be updated (for details see section 3.7 Resource Links). In the context of dynamic pipes the two said attributes have no meaning. In case of the PipeClose command, the resource link is ignored.
<i>UpdatedStatus</i> ?	enumeration	This value represents the actual status of the pipe resource and may be used by the receiving process for process termination control (for details see section 4.3.4.2 Formal Iterative Processing). For possible values of the resource <i>Status</i> attribute see Table 3.9.

Structure of the PipeCmdResult Element

An Acknowledge message may be sent to acknowledge the execution of a previously received pipe command. The *refID* attribute of a Acknowledge message is set to the ID of the issuing pipe message.

Table 5.46 Contents of the PipeCmdResult element

Name	Data Type	Description
<i>PipeID</i>	string	PipeID of the JDF resource that defines the dynamic pipe.
<i>Status</i>	enumeration	The status of the process after execution. Possible values are defined in Table 3.3.

5.5.3.3 PipePush

Table 5.47 Contents of the PipePush element

Object Type	Element name	Description
CommandTypeObj	PipeParams	Describes the produced pipe resource.
ResponseTypeObj	JobPhase	The status of the responding process.
AcknowledgeTypeObj	PipeCmdResult	Describes the execution result of the pipe command (see Table 5.46).

The PipePush message notifies the availability of pipe resources that are described in a JDF dynamic pipe (see sections 3.6.3 Pipe Resources and 4.3.2 Overlapping Processing Using Pipes). PipePush messages are the JMF equivalent of a dynamic output resource link. Figure 5.3 depicts the mode of operation of a PipePush message.

The PipePush command response is equivalent to the PipePull command response described above. The PipeParams and PipeCmdResult messages are also described in section 5.5.3.2 PipePull.

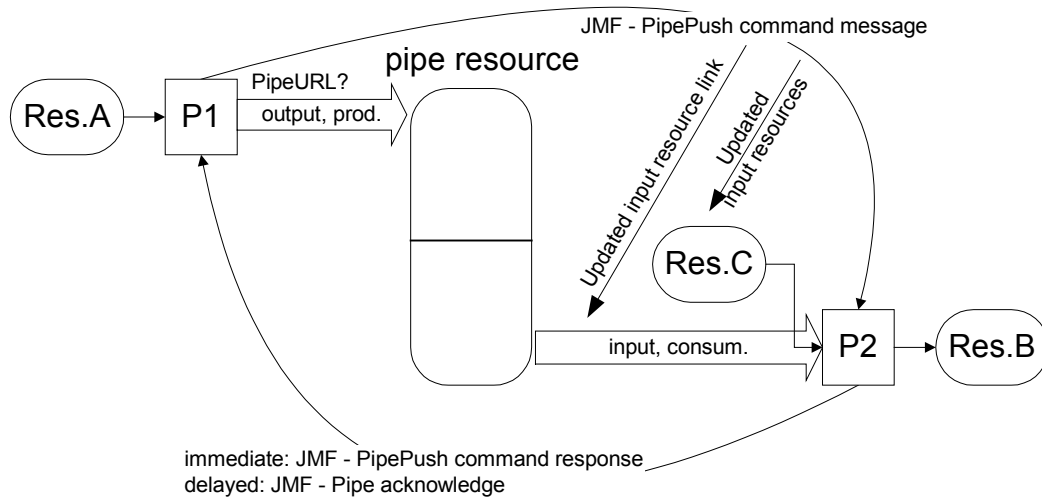


Figure 5.3 Mechanism of a PipePush message

5.5.3.4 PipePause

Table 5.48 Contents of the PipePause element

Object Type	Element name	Description
-------------	--------------	-------------

CommandTypeObj	PipeParams	Describes the pipe resource.
ResponseTypeObj	JobPhase	The status of the responding process.
AcknowledgeTypeObj	PipeCmdResult	Describes the execution result of the pipe command (see Table 5.46).

The `PipePause` message pauses execution of a process that is at the other end of a dynamic pipe.

The `PipePause` command response is equivalent to the `PipePull` command response described above. The `PipeParams` and `PipeCmdResult` messages are also described in section 5.5.3.2 `PipePull`.

5.6 Queue Support

In JMF, a device is assumed to have one input queue that accepts submitted jobs. If a real device supports multiple queues, it is represented by multiple logical devices in JDF. The simple case of a device with no queue can be mapped to a queue with two *Status* states: *waiting* and *full*.

JMF supports simple handling of priority queues. The following assumptions are made:

- Queues support priority. Priority may only be changed for waiting jobs. A queue may round priorities to the number of supported priorities, which may be one, indicating no priority handling.
- Priority is described by an integer from 0 to 100. Priority 100 defines a job that should pause a job that is in progress and commence immediately. If a device does not support pausing running jobs, it should queue a priority-100 job before the last pending priority-100 job.
- A controller may control multiple devices/queues.
- Queue entries can be unambiguously identified by a *QueueEntryID*.

Some conventions used in the following sections have already been introduced in section 5.5 *Standard Messages*. This affects the message families and the descriptive tables at the beginning of each message section that describe the type objects related to the corresponding message. The type objects are `QueryTypeObj`, `CommandTypeObj`, `ResponseTypeObj`, and `AcknowledgeTypeObj` (see also Figure 5.1).

5.6.1 Queue Entry ID Generation

Queue entries are accessed using a *QueueEntryID* attribute, which is generated by the controller of the queue when the job is submitted. This attribute must uniquely identify an entry within the scope of one queue. An implementation is free to choose the algorithm that generates *QueueEntryIDs*.

5.6.2 Queue Entry Handling Commands

Queue-entry handling is provided so that the state of individual jobs within a queue can be changed. Job submission, queue-entry grouping, priorities, and hold/resume of entries are all supported. The individual commands are defined in the table and explained in greater detail in the sections that follow.

Table 5.49 *QueueEntry* handling messages

Command	Family	Description
<code>AbortQueueEntry</code>	CRA	If a job is already running, it is aborted and removed. If it is not already running, it is removed from the queue.

		not already running, it is removed from the queue.
HoldQueueEntry	CR	The entry remains in queue but is never executed.
RemoveQueueEntry	CR	A job is removed from the queue.
ResubmitQueueEntry	CR	Replaces a queue entry without affecting the entry's parameters. The command is used, for example, for late changes to a submitted JDF.
ResumeQueueEntry	CR	A held job is resumed. The job is requeued at the position defined by its current priority. Submission time is set to the current time stamp.
SetQueueEntryPosition	CR	Queues a job behind a given position n, where n represents a numerical value. 0 = pole position. Priority is set to the priority of the job at position n.
SetQueueEntryPriority	CR	Sets the priority of a queued job to a new value. This does not apply to jobs that are already running.
SubmitQueueEntry	CR	A job is submitted to a queue in order to be executed.

5.6.2.1 AbortQueueEntry

Table 5.50 Contents of the AbortQueueEntry element

Object Type	Element name	Description
CommandTypeObj	QueueEntryDef	Defines the queue entry.
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed.

For the definition of the elements listed above, see section 5.6.4.

Once this command is issued, the entry specified by `QueueEntryDef` is removed from the queue. If the device on which the entry is running has already commenced processing, the entry is aborted. `AbortQueueEntry` commands may be terminated by an `Acknowledge` message to indicate the completion of the abortion. Then no predefined `AcknowledgeTypeObj` is provided.

5.6.2.2 HoldQueueEntry

Table 5.51 Contents of the HoldQueueEntry element

Object Type	Element name	Description
CommandTypeObj	QueueEntryDef	Defines the queue entry.
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed.

For the definition of the elements listed above, see section 5.6.4.

The entry specified by `QueueEntryDef` remains in the queue but is never executed. The `HoldQueueEntry` command has no effect on running jobs.

5.6.2.3 RemoveQueueEntry

Table 5.52 Contents of the RemoveQueueEntry element

Object Type	Element name	Description
CommandTypeObj	QueueEntryDef	Defines the queue entry.
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed.

For the definition of the elements listed above see, section 5.6.4.

This command causes the entry specified by `QueueEntryDef` to be removed from the queue. It does not affect running jobs.

5.6.2.4 ResubmitQueueEntry

Table 5.53 Contents of the ResubmitQueueEntry element

Object Type	Element name	Description
CommandTypeObj	ResubmissionParams	Defines the job resubmission.
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed.

For the definition of the `Queue` element, see section 5.6.4.

A job is resubmitted to a queue using the `ResubmitQueueEntry` message. This allows late changes to be made to a job without affecting queue parameters and without exporting the internal structure of a queue. Resubmission overwrites the job specified in the `URL` attribute of the `ResubmissionParams` element. The job must not run. Job resubmission does not affect other queue parameters as specified, for example, resubmission does not affect queue ordering.

Structure of the ResubmissionParams Element

Table 5.54 Contents of the ResubmissionParams element

Name	Data Type	Description
<code>QueueEntryID</code>	string	ID of the queue entry to be replaced.
<code>URL</code>	URI	Location of the JDF to be submitted. May be either a URL or, in the case of MIME/Multipart/Related, a CID.

5.6.2.5 ResumeQueueEntry

Table 5.55 Contents of the ResumeQueueEntry element

Object Type	Element name	Description
CommandTypeObj	QueueEntryDef	Defines the queue entry.
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed.

For the definition of the elements listed above, see section 5.6.4.

The hold status of the queue entry specified by `QueueEntryDef` is removed.

5.6.2.6 SetQueueEntryPosition

Table 5.56 Contents of the `SetQueueEntry` element

Object Type	Element name	Description
CommandTypeObj	QueueEntryPosParams	Defines the queue entry.
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed.

For the definition of the `Queue` element, see section 5.6.4.

The position of the queue entry is modified. The `QueueEntryPosParams` element provides the required parameters.

Structure of the `QueueEntryPosParams` Element

`QueueEntryID` specifies the queue entry to be moved. Jobs may either be set to a specific position within the queue or positioned next to an existing queue entry. The priority of the entry matches the priority of the entry that precedes it, after it has been repositioned.

Table 5.57 Contents of the `QueueEntryPosParams` element

Name	Data Type	Description
<code>NextQueueEntryID?</code>	string	ID of the queue entry that should be ordered directly behind the entry.
<code>QueueEntryID</code>	string	ID of a queue entry. The ID is generated by the queue owner.
<code>PrevQueueEntryID?</code>	string	ID of the queue entry that should be ordered directly in front of the entry.
<code>Position?</code>	integer	Position in the queue. 0 = pole position.

5.6.2.7 SetQueueEntryPriority

Object Type	Element name	Description
CommandTypeObj	QueueEntryPriParams	Defines the queue entry.
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed.

For the definition of the `Queue` element, see section 5.6.4.

The priority of the queue entry is modified. The `QueueEntryPriParams` element provides the required parameters.

Structure of the QueueEntryPriParams Element

QueueEntryID, described in the table below, specifies the queue entry that has its priority modified.

Table 5.58 Contents of the *QueueEntryPriParams* element

Name	Data Type	Description
<i>Priority</i>	integer	Number from 0 to 100, where 0 = lowest priority and 100 = maximum priority.
<i>QueueEntryID</i>	string	ID of a queue entry. The ID is generated by the queue owner.

5.6.2.8 SubmitQueueEntry

Table 5.59 Contents of the *SubmitQueueEntry* element

Object Type	Element name	Description
CommandTypeObj	QueueSubmissionParams	Defines the job submission.
ResponseTypeObj	QueueEntry	Provides the queue entry of the submitted job.
	Queue	Describes the state of the queue after the command has been executed.
	Definition of the <i>QueueEntry</i> and <i>Queue</i> elements, see section 5.6.4.	

The *SubmitQueueEntry* message submits a job to a queue. The *QueueSubmissionParams* element provides the required parameters.

Structure of the QueueSubmissionParams Element

The job submission may contain queue-ordering attributes equivalent to those used by the *SetQueueEntryPriority* and *SetQueueEntryPosition* messages.

The *URL* attribute specifies the location where the JDF file to be submitted can be retrieved by the queue controller. The location type in the *URL* attribute (such as *File*, *http* or *CID*) defines the submission method. The optional *ReturnURL* attribute specifies the location where the modified JDF should be sent after the job is completed or aborted.

Table 5.60 Contents of the *QueueSubmissionParams* element

Name	Data Type	Description
<i>Hold</i> ?	boolean	If <i>true</i> , the entry is submitted as held. Default = <i>false</i>
<i>NextQueueEntryID</i> ?	string	ID of the queue entry that should be ordered directly behind the entry.
<i>PrevQueueEntryID</i> ?	string	ID of the queue entry that should be ordered directly in front of the entry.
<i>Priority</i> ?	integer	Number from 0 to 100, where 0 = lowest priority and 100 = maximum priority. Default = 1
<i>ReturnURL</i> ?	URL	URL where the JDF file should be sent when the job is completed or aborted. If not specified, the JDF should be

		placed in the default output hot-folder of the queue controller.
<i>URL</i>	URL	Location of the JDF to be submitted. In the case of MIME/Multipart/Related, the location may be either a URL or a CID.
<i>WatchURL ?</i>	URL	URL of the controller that should be notified when the status of the <code>QueueEntry</code> changes. Specifying this URL is the equivalent of sending a <code>QueueEntryStatus</code> query with a persistent channel and <i>ChangeAttribute</i> = "*" to this URL.

File Submission

If the URL defines a file, the controller may retrieve the file at the location specified in the *URL* attribute.

The following example declares a file on the network:

```
<Command Type="SubmitQueueEntry" URL=" File:\\AnyDirectory\job1.jdf"/>
```

HTTP External JDF Submission

The following example declares an intranet or internet location. In this example, the queue controller can retrieve the file with a standard http `get` command. Note that the job itself may be a MIME/Multipart entity. It may also be dynamically generated by a CGI script or another such tool.

```
<Command Type="SubmitQueueEntry" URL="HTTP://JobServer.JDF.COM?job1"/>
```

HTTP MIME/Multipart/Related Submission

If a message controller is capable of decoding MIME, it is legal to submit a MIME/Multipart/Related message. The first section of the multipart MIME document must be the JMF submission command. Internal links are defined using the Content-ID (CID) label in MIME. The second section must be the JDF job. Subsequent sections are the linked entities, such as the preview images shown in the following example:

```
MIME-Version: 1.0
Content-Type: multipart/Related; boundary=unique-boundary

--unique-boundary
Content-type: text/xml
...
<JMF TimeStamp="2000-06-12T08:56+02:00" SenderID="JobCreator P_01">
<Command ID="Cmd-0234" Type="SubmitQueueEntry">
<QueueSubmissionParams URL="CID:JDF1"/>
</Command>
</JMF>
...

--unique-boundary
Content-type: text/xml
Content-ID: JDF1

<JDF ... >

--unique-boundary
```

Content-type: image/png
 Content-ID: Yellow-PNG-Page1

png image of a separation may be here

--unique-boundary--

5.6.3 Global Queue Handling

Whereas the commands in the preceding section change the state of an individual queue entry, the commands in this section modify the state of an entire queue. Note that entries that are executing in a device are not affected by the global queue-handling commands and must be accessed individually. An individual queue can be selected by specifying the target device/queue in the *DeviceID* attribute of the JMF root. If no *DeviceID* is specified, the commands or queries are applied to all devices/queues that are controlled by the controller that received the message.

The following individual messages are defined:

Table 5.61 Global queue-handling commands

Command	Family	Description
CloseQueue	CR	The queue is closed. No jobs may be accepted by the queue.
FlushQueue	CR	All entries in the queue are removed.
HoldQueue	CR	The queue is held. No jobs within the queue may be executed.
OpenQueue	CR	The queue is opened. Jobs may be accepted.
QueueEntryStatus	QRS	Returns a QueueEntry element.
QueueStatus	QRS	Returns the Queue elements that describe a queue or set of queues.
ResumeQueue	CR	The queue is activated and queue entries may be executed.
SubmissionMethods	QR	Queries a list of supported submission methods to the queue.

5.6.3.1 CloseQueue

Table 5.62 Contents of the CloseQueue element

Object Type	Element name	Description
CommandTypeObj	-	
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed.

For the definition of the Queue element, see section 5.6.4.

The queue is closed. No further queue entries are accepted by the queue. The status of entries that are already in the queue remains unchanged and prior entries may be executed.

5.6.3.2 FlushQueue

Table 5.63 Contents of the FlushQueue element

Object Type	Element name	Description
CommandTypeObj	-	
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed. For the definition of the Queue element, see section 5.6.4.

All queue entries in the queue are removed. Only pending queue entries may be removed.

5.6.3.3 HoldQueue

Table 5.64 Contents of the HoldQueue element

Object Type	Element name	Description
CommandTypeObj	-	
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed. For the definition of the Queue element, see section 5.6.4.

The queue is held. No entries may be executed. Note that the status of a held entry prior to **HoldQueue** is retained so that held jobs should remain held after a **ResumeQueue**. New entries may, however, still be submitted to a held queue.

5.6.3.4 OpenQueue

Table 5.65 Contents of the OpenQueue element

Object Type	Element name	Description
CommandTypeObj	-	
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed. For the definition of the Queue element, see section 5.6.4.

The queue is opened and new queue entries may be accepted by the queue. A held queue remains held. The **OpenQueue** command is the opposite of a **CloseQueue** command.

5.6.3.5 QueueEntryStatus

Table 5.66 Contents of the QueueEntryStatus element

Object Type	Element name	Description
QueryTypeObj	QueueEntryDef *	Defines the addressed queue entries.
ResponseTypeObj	QueueEntry *	Describes the status of the queried queue entries.

For the definition of the elements above see section 5.6.4.

The `QueueEntryStatus` message returns queue entry descriptions. The `QueueEntryDef` elements specify the queue entries to be queried. If no `QueueEntryDef` element is specified, the query returns a list of `QueueEntry` elements, one for each entry in the queue. If no `QueueEntryDef` is specified and the query defines a persistent channel, a `Signal` is emitted for any entry whose status changes. This includes changes as a result of modifications of the queue status, such as hold or resume.

5.6.3.6 QueueStatus

Table 5.67 Contents of the `QueueStatus` element

Object Type	Element name	Description
QueryTypeObj	-	
ResponseTypeObj	Queue	Describes the status of the queue. For the definition of the <code>Queue</code> element, see section 5.6.4.

Returns a queue description.

5.6.3.7 ResumeQueue

Object Type	Element name	Description
CommandTypeObj	-	
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed. For the definition of the <code>Queue</code> element, see section 5.6.4.

The queue is activated and queue entries may be executed. The `ResumeQueue` command is the opposite of a `HoldQueue` command.

5.6.3.8 SubmissionMethods

Table 5.68 Contents of the `SubmissionMethods` element

Object Type	Element name	Description
QueryTypeObj	-	
ResponseTypeObj	SubmissionMethods ?	Describes the submission methods supported by the queue.

The `SubmissionMethods` message returns the submission methods that are supported by a queue controller.

Structure of the `SubmissionMethods` Element

The response element may contain multiple attributes, as defined below. If an attribute is not specified, the corresponding submission method is not supported.

Table 5.69 Contents of the *SubmissionMethods* element

Name	Data Type	Description
<i>File</i> ?	boolean	Can retrieve a JDF from a File specified in the URL Default = <i>false</i>
<i>HotFolder</i> ?	URL	URL specification of a hot-folder location. Default = <i>no hot-folder</i>
<i>HttpGet</i> ?	boolean	Can retrieve a JDF via HTTP get commands. Default = <i>false</i>
<i>MIME</i> ?	boolean	Accepts MIME/Multipart/Related submission messages via a message post. Default = <i>false</i>

The following is an example of a response to a *SubmissionMethods* query:

```
<Response ID="1" refID="Q1" Type="SubmissionMethods"/>
  <SubmissionMethods File="true"
    HotFolder="File://MyDevice/HotFolder" HttpGet="true" MIME="false"/>
</Response>
```

5.6.4 Queue-Handling Elements

In this section elements used by queue-handling commands are defined.

Structure of the Queue Element

The attributes in the following table are defined for *Queue* message elements.

Table 5.70 Contents of the *Queue* element

Name	Data Type	Description
<i>Status</i>	enumeration	Status of the queue. Possible values are: <i>blocked</i> – Queue is completely inactive. No entries may be added and no entries are executed. The queue is closed and held. <i>closed</i> – Queue entries that are in the queue are executed, but no new entries may be submitted. The lock must be removed explicitly. <i>full</i> – Queue entries that are in the queue are executed but no new entries may be submitted. The lock is removed by the queue controller as soon as it is able to do so. <i>running</i> – An process is executing. Entries may be submitted but will not be executed until they reach their turn in the queue. <i>waiting</i> – Queue accepts new entries and has free resources to immediately commence processing. <i>held</i> – Entries may be submitted but will not be executed until the queue is resumed.
<i>DeviceID</i>	string	Identifies the queue/device.
<i>Device</i> *	element	The devices that execute entries in this queue.

QueueEntry * element Queue entry elements (see Table 5.71, below). The entries are ordered in the sequence they will be executed, beginning with the running entries.

Example of a **Queue** message element:

```
<Queue Status="running" DeviceID="Q12345">
  <QueueEntry QueueEntryId="111-1" Priority="1" Status="running"
JobId="111" JobPartId="1"/>
  <QueueEntry QueueEntryId="111-2" Priority="1" Status="waiting"
JobId="111" JobPartId="2"/>
  <QueueEntry QueueEntryId="112-1" Priority="55" Status="held"
JobId="112" JobPartId="1"/>
</Queue>
```

Structure of the QueueEntry Element

Table 5.71 Contents of the QueueEntry element

Name	Data Type	Description
<i>JobID</i>	string	The Job ID of the JDF process.
<i>JobPartID</i> ?	string	The JobPartID of the JDF process.
<i>Priority</i>	integer	Priority of the QueueEntry . Values are 0-100. 0 = lowest priority, while 100 = highest priority.
<i>QueueEntryID</i>	string	ID of a QueueEntry . This ID is generated by the queue owner.
<i>Status</i>	enumeration	Status of the individual entry. Possible values are: <i>running</i> – The queue entry is running and is no longer represented in the queue. <i>waiting</i> – The queue entry is waiting and will be executed when resources are available. <i>held</i> – The queue entry is held and will not execute until resumed. <i>removed</i> – The queue entry has been removed. This status can only be sent when a persistent channel watches a queue and the queue entry is removed.
<i>SubmissionTime</i> ?	timeInstant	Time when the entry was submitted to the queue.

Structure of the QueueEntryDef Element

The element specifies a queue entry and is used to refer to a certain queue entry.

Table 5.72 Contents of the QueueEntryDef element

Name	Data Type	Description
<i>QueueEntryID</i>	string	ID of the queue entry. The ID is generated by the queue owner.

5.7 Extending Messages

This specification defines a set of predefined messages for general usage. Additional message types may be defined using the standard namespace syntax as described in section 3.10.1 Namespaces in XML. The content of the *Type* attribute may be specified with a prefix that identifies the organization that uses the *Type* extension. The prefix and name should be separated by a single colon (:). Any additional attributes and elements are allowed, and internal elements may be declared with explicit namespaces. The default namespace of JMF elements is xmlns="...TBD...". An example is provided:

```
<JMF ... xmlns="JMFSchema URI" xmlns:Circus="Circus Schema URI">
  <Query Type="Circus:IsClownHappy" ID="Q1">
    <Circus:ClownParams Gender="male"/>
  </Query>
</JMF>
```

The response will also have the "Circus:" namespace identifier. All Circus elements are explicitly declared.

```
<JMF ... xmlns="JMFSchema URI" xmlns:Circus="Circus Schema URI">
  <Response ID="1" refID="Q1" Type="Circus:IsClownHappy">
    <Circus:Clown name="Joe" happy="true">
    <Circus:Clown name="John" happy="false">
  </Response>
</JMF>
```

5.7.1 IFRATrack Support

The extending mechanism can be used to implement compatibility with other XML-based messaging standards, for example version 3.0 of IFRATrack. The *Type* attribute is set to the appropriate namespace, and the foreign message is included, as demonstrated in the following example:

```
<JMF ... xmlns="JMFSchema URI" xmlns:IFRA="IFRATrack URI">
  <Query ID="Q1" Type="IFRA:IMF">
    <IMF xmlns="IFRATrack URI">
      Whatever you want (may be multiple top level elements)
    </IMF>
  </Query>
</JMF>
```

The legal response would be:

```
<JMF ... xmlns="JMFSchema URI" xmlns:IFRA="IFRATrack URI">
  <Response ID="1" refID="Q1" Type="IFRA:IMF">
    <IMF xmlns="IFRATrack URI">
      The appropriate IFRA response(s)
    </IMF>
  </Response>
</JMF>
```

Note that the application is free to select the appropriate response types in order to fulfill its local (IFRATrack) protocol requirements if it uses its own namespace. In the examples above the default namespace associated with the IMF query and response elements has been overwritten by the IFRA-namespace. Additional information on using IFRATrack and JDF is in Appendix E Modelling IFRATrack in JDF.

Chapter 6 Processes

The following chapter describes the processes that are defined in detail for JDF.

6.1 Process Template

Processes are defined by their input and output resources, so all relevant resource information is provided in tables for each process. Furthermore, although they are not listed for each process, additional, optional **ApprovalSuccess** input resources that allow **Approval** processes, as well as any implementation resources are implied for all processes defined in this chapter.

Input Resources

Name	Description
Resource	Represents any input resource.
Res1 (usage1)	A resource of type Res1 with the <i>ProcessUsage</i> attribute <i>usage1</i>
Res1 (usage2)	A resource of type Res1 with the <i>ProcessUsage</i> attribute <i>usage2</i>
ApprovalSuccess *	Any number of ApprovalSuccess resources may be appended to processes in order to model proofing and verification requirements. This is implied and not specified explicitly in the tables in the following section. For more information on the Approval process, see section 6.2.1.
Implementation *	Abstract resource that is a placeholder for any implementation resource (examples are Employee and Device) that is associated with processing this node.

Output Resources

Name	Description
Resource	Represents any output resource.

6.2 General Processes

6.2.1 Approval

The **Approval** process can take place at various steps in a workflow. For example, a resource, such as a printed sheet or a finished book, is used as the input to be proven, and an **ApprovalSuccess** (given, for example, by a customer or foreman) is produced.

Combining the **Approval** process with any other process can be used to represent a request for a receipt.

Input Resources

Name	Description
------	-------------

ApprovalParams	Details of the approval process.
Resource *	The resources to be proofed. The input will most often be a resource of class <i>Handling</i> or <i>Quantity</i> .

Output Resources

Name	Description
ApprovalSuccess	Result of any proofing process given, for example, by a customer or foreman. Note that ApprovalSuccess resources are only available on success.
Resource *	Represents the input resources as outputs that must be accepted for further processing by the approval process. This is typically used to transfer the resource <i>Status</i> of <i>draft</i> to <i>available</i> (see also 4.3.4.2 Formal Iterative Processing).

6.2.2 Combine

The **Combine** process is used to combine multiple physical resources of the same content to form one physical resource. The quantity of the input and output of resources should be equal.

Input Resources

Name	Description
PhysicalResource +	The physical resources to be combined. These may be any resource whose class is Consumable, Handling or <i>Quantity</i> .

Output Resources

Name	Description
PhysicalResource	Result of combining. The physical resource formed as a result of the Combine process. The resulting resource must have a class of Consumable, Handling or <i>Quantity</i> .

6.2.3 Delivery

This process can be used to describe the delivery of a physical resource to or from a location. This delivery may be internal—meaning within the company—or to an external company or customer. The CustomerInfo element of the JDF node can also be used if the delivery to is to be made to only one customer.

Note, that a delivery receipt can be requested by combining the **Delivery** process with an **Approval** process.

Input Resources

Name	Description
------	-------------

DeliveryParams	Necessary information about the item or items to be delivered is stored here.
Resource	Any resource delivered to a location. This can be a physical resource or a Parameter resource that is delivered electronically.

Output Resources

Name	Description
Resource	Any resource picked up from a location. This can be a physical resource or a Parameter resource that is delivered electronically.

6.2.4 Ordering

This process can be used to describe the **Ordering** of a **PhysicalResource** element. Orders can be placed internally—that is, within the company—or externally.

Input Resources

Name	Description
OrderingParams	Necessary information about the items to be ordered, such as the supplier address, item quantity, or unit type.

Output Resources

Name	Description
PhysicalResource	All kinds of physical resources can be ordered.

6.2.5 ResourceDefinition

This process can be used to describe the interactive or automated process of defining resources such as set-up information. This process creates output resources or modifies input resources of the same type as the output resources.

The **ResourceDefinition** process is designed to monitor interactive work such as creating impositioning templates. It can also be used to model a hot-folder process that accepts resources from outside of a JDF based workflow.

Input Resources

Name	Description
Resource ?	Any type of resource.

Output Resources

Name	Description
Resource	The same type of resource as the input.

6.2.6 Split

This process is used for splitting one physical resource into multiple physical resources containing the same content as the original. The quantity of the input and output of resources should be equal.

Input Resources

Name	Description
PhysicalResource	The physical resource to be split.

Output Resources

Name	Description
PhysicalResource +	The resources formed as a result of splitting.

6.2.7 Verification

The **Verification** process is used to confirm that a process has been completely executed. In the case of variable data printing, in which every document is unique and must be validated individually, database access is required. Verification in this situation may involve scanning the physical sheet and interpreting a barcode or alphanumeric characters. The decoded data may then be either recorded in a database to be later cross referenced with a verification list, or cross referenced and validated immediately in real time.

Input Resources

Name	Description
DBSchema ?	Schema description of the cross-reference database.
DBSelection ?	Database link that defines the database that contains cross-reference data.
IdentificationField *	Identifies the position and type of data for an automated, OCR-based verification process.
VerificationParams	Controls the verification requirements.

Output Resources

Name	Description
ApprovalSuccess ?	Signature file that defines verification success.
DBSelection ?	Database link where the verification data should be recorded.

6.3 Prepress Processes

6.3.1 Scanning

Creates bitmaps from analog images using a scanner.

Input Resources

Name	Description
------	-------------

ExposedMedia *	Description of the media to be scanned.
ScanParams	High-level scanner settings. These settings are specifically not intended as a replacement for low-level device interfaces such as TWAIN.

Output Resources

Name	Description
RunList	List of ByteMap resources or LayoutElement resources of <i>Type = image</i> .

6.3.2 LayoutElementProduction

This process describes the creation of page elements. It also explains how to create a layout that can put together all of the necessary page elements, including text, bitmap images, vector graphics, PDL, or application files such as InDesign, PageMaker, and XPress®. The elements might be produced using any of a number of various software tools. This process is often performed several times in a row before the final **LayoutElement**, representing a final layout file, is produced.

Input Resources

Name	Description
LayoutElement *	URL of the PDL or application file, bitmap image file, text file, vector graphics file, etc. Additional information (such as the page number or X, Y-coordinates) might be stored in the Comment element of the LayoutElement resource. Customer information such as the file templates, manuscripts, and sketches are handled via URL or URI.

Output Resources

Name	Description
LayoutElement ?	A URL of the PDL or application file is produced by this process if no RunList is produced. Additional information such as page number or X, Y-coordinates might be stored in the Comment of the LayoutElement .
RunList ?	A RunList of LayoutElement resources of <i>ElementType page</i> or <i>document</i> is produced if this LayoutElementProduction task is the last process of type LayoutElementProduction .

6.3.3 DBDocTemplateLayout

This process specifies the creation of a master document template that is used as an input resource for the **DBTemplateMerging** process. It is similar to the **LayoutElementProduction** process except that the output is a set of document templates. Document template are represented in JDF as **LayoutElement** resources with *Template = true*.

Input Resources

Name	Description
------	-------------

LayoutElement *	Page elements without links to a database.
DBRules	Description of the rules that should be applied to database records in order to generate graphic output.
DBSchema	Database schema that describe the structure of data in the database.

Output Resources

Name	Description
LayoutElement *	The document template is a LayoutElement with links to a database. These links are proprietary to the linking application and are not described in JDF. The <i>Template</i> attribute must be <i>true</i> .

6.3.4 DBTemplateMerging

This process specifies the creation of personalized PDL instance documents by combining a document template and instance data records from a database. The resulting instance documents will generally be consumed by an *Imposition*, a *RIPping*, and ultimately by a *DigitalPrinting* process.

Input Resources

Name	Description
DBMergeParams	Parameters of the merge process.
DBSelection	Instance database records to be merged into the document.
LayoutElement *	Document template page element with internal links to a database.

Output Resources

Name	Description
RunList	Page element without links to a database. This element usually contains a printable LayoutElement resource such as PPML, vPDF or even plain ASCII.

6.3.5 ColorSpaceConversion

ColorSpaceConversion, as the name implies, is the process of converting all colors used in the job to a known colorspace. There are two ways in which a controller can use this process to accomplish the color conversion. It can simply order the colors to be converted by the device assigned to the task, or it can request that the process simply tag the input data for eventual conversion. Additionally, the process may remove all tags from the content.

The parameters of this resource provide the ability to control, selectively, the conversion or tagging of graphical objects based on object class and/or incoming color space.

Like all other color manipulation supported in JDF, the color conversion controls are based on the use of ICC profiles. While the assumed characterization of input data can take many forms, each can internally be represented as an ICC profile. In order to perform the transformations, input profiles must be paired with the identified final target device profile to create the transformation.

In order to avoid the loss of black color fidelity resulting from the transformation from a four-component CMYK to a three-component interchange space, the agent may select a DeviceLink¹ profile as the assumed color space characterization. In these instances, the final target profile is ignored. Since there is no algorithmic way to determine that the output characterization in a device link profile is equivalent to another profile, some of the responsibility to select a sensible combination falls on the agent or end user.

Input Resources

Name	Description
ColorantControl	Identifies the assumed color model for the job.
ColorPool	Identifies the specifics of individual colorants used by the job.
ColorSpaceConversionParams	Parameters that define how colorspace will be converted in the file.
RunList	List of pages on which to perform the selected operation.

Output Resources

Name	Description
ColorantControl	Identifies the assumed color model for the job. The ColorantControl resource may be modified by a ColorSpaceConversion Process.
ColorPool	Identifies the specifics of individual colorants used by the job.
RunList	List of pages on which the selected operation has been performed.

6.3.6 ColorCorrection

ColorCorrection is the process of modifying the specification of colors in documents to achieve some desired visual result. The process may be performed to ensure consistent colors across multiple files of a job, or to achieve a specific design intent (such as, 'Brighten the image up a little').

ColorCorrection is distinct from **ColorSpaceConversion**, which is the process of changing how the colors specified in the job will be produced on paper. Rather, **ColorCorrection** is the process of modifying the desired result, whatever the specified colorspace might be.

Input Resources

Name	Description
ColorantControl	Identifies the assumed color model for the job.
RunList	List of content elements that are to be operated on.

Output Resources

Name	Description
RunList	List of color-corrected pages.

¹ DeviceLink profiles are ICC profiles that map directly from one device color space to another device color space. Therefore it represents a one-way link or connection between devices. Examples for DeviceLink profiles are CMYK to CMYK print process conversions or RGB to CMYK color separations.

6.3.7 Preflight

Preflighting is the process of examining the components of a print job to ensure that the job will print successfully and with the expected results. **Preflight** checks may be performed on each PDL document identified within the associated **RunList** resource.

Preflighting a file is generally a three-step process. First, the pages are inventoried against a preflight **profile**, detailing the expected or hoped-for results. The resulting **inventory** identifies the significant characteristics of all the pages in the job. Next, the characteristics are tested against a set of criteria specified by a series of preflight **constraint** resources. Finally, results and discrepancies are reported in a **PreflightAnalysis** hierarchy log as **analysis**.

Agents record the instructions for, and devices the results of, preflight operations in JDF jobs, using hierarchies headed by three types of resources: Inventory, Profile, and Results. The Inventory hierarchy may be used to record all the information gathered in the first step, although devices need not record this information. The Profile hierarchy is used to record the criteria used to test the file in the second step. And the Results hierarchy is used to record the results of the tests. In all three hierarchies, information is grouped into categories. There are six pre-defined categories in JDF—Colors, Document, Fonts, FileType, Images and Pages—but applications may define other categories if needed.

In a profile hierarchy, each category is populated with **PreflightConstraint** elements. Each **PreflightConstraint** element specifies a test that the application will perform when analyzing the file. In the Inventory and Results hierarchies, each category is populated with two kinds of sub-elements that record information about specific characteristics of the file: **PreflightInstance** and **PreflightDetail**. Such information is recorded in the following two ways:

1. Information that is specific to one instance of some file object is recorded via **PreflightInstance** sub-elements that occur in each of the results pools such as **FontResultsPool** and **ImageResultsPool**). Within each **PreflightInstance** element, **PreflightInstanceDetail** sub-elements provide detailed information about that instance. For example, to record information about each font used in the file, the **FontResultsPool** contains one **PreflightInstance** sub-element, which groups a set of **PreflightInstanceDetail** sub-elements. Each of these sub-elements records one specific characteristic of the font.
2. Information that applies to the file as a whole is recorded via **PreflightDetail** sub-elements, which occur in the various results pools. For example, to record all the page sizes used in the file, the **PagesResultsPool** would contain several **PreflightDetail** sub-elements, one for each page size used in the file.

An Inventory hierarchy may be used to record all information about a file. Preflight tools are not required to create an Inventory hierarchy as part of the preflight information they record. However, tools may find it useful to record this information, allowing them to avoid re-parsing the entire file in order to perform a new Analysis.

Profile hierarchies specify the constraints against which the file is tested. Each Analysis hierarchy reflects the results of evaluating the file characteristics, which may be recorded in an Inventory hierarchy, against a set of tests recorded in a Profile hierarchy.

PreflightConstraint elements record the specific details for the constraints specified in the **PreflightProfile** resource. **PreflightDetail** and **PreflightInstanceDetail** elements record results, while **PreflightInstance** elements group **PreflightInstanceDetail** sub-elements for instances of file objects. The details recorded are PDL-specific.

Applications can define constraints within any of the defined constraint categories for any file type. In addition, applications may add to the set of defined constraints and constraint categories, defining both the new category and the constraint within the category.

Whether constraints are specified for predefined or new constraint categories, the eventual values for those constraints are always expressed as **PreflightConstraint** elements which are part of a **PreflightProfile**. Furthermore, the results are always expressed as either **PreflightDetail** elements or **PreflightInstance** elements, which group **PreflightInstanceDetail** sub-elements for Analysis results.

Input Resources

Name	Description
PreflightInventory ?	Provides an exhaustive list of all items already resolved in a previous preflight.
PreflightProfile	A specified list of constraints against which pages may be tested.
RunList	The list of pages to be preflighted.

Output Resources

Name	Description
PreflightAnalysis ?	Describes the results of a preflight operation. Provides analytical information for the constraints against which the file was tested.
PreflightInventory ?	Provides an exhaustive list of all items considered in preflight.
RunList ?	A list of pages that may or may not have been modified as a result of a fix-up operation.

6.3.8 ImageReplacement

This process provides a mechanism for manipulating documents that contain referenced image data. It allows for the “fattening” of files that simply contain a reference to external data or contain a low-resolution proxy. Additionally, the **ImageReplacementParams** resource can be specified so that this process generates proxy images from referenced data.

ImageReplacement is intentionally neutral of the conventions used to identify the externally referenced image data.

Input Resources

Name	Description
ImageReplacementParams	Describes the controls selected for the manipulation of images.
RunList	List of page contents on which to perform the selected operation.

Output Resources

Name	Description
RunList	List of page contents with images that have been manipulated as indicated by the ImageReplacementParams resource.

6.3.9 Separation

The **Separation** process specifies the controls associated with the generation of color-separated data, and is designed to be flexible enough to allow a variety of possible methods for accomplishing this task. First

of all, it sponsors host-based PDF separating operations, in which a **RunList** of pre-separated PDF data is generated. It can also be combined with a RIP to allow control of In-RIP separations. In this scenario a **RunList** containing **ByteMap** resources is generated as the output. Yet another anticipated combination is with the **ColorCorrection** process to deal with incoming device-dependent data. And finally, it may be combined with an **ImageReplacement** process in order to do image substitution for omitted or proxy images.

Input Resources

Name	Description
ColorantControl	Identifies which colorants in the job are to be output.
ColorPool	Identifies the specifics of individual colorants used by the job.
RunList	List of pages that are to be operated on.
SeparationControlParams	Controls for the separation process

Output Resources

Name	Description
RunList	List of separated pages or separated raster bytemaps.

6.3.10 Trapping

Trapping is a prepress process that modifies PDF files to compensate for a type of error that occurs on presses. Specifically, when more than one colorant is applied to a piece of media using more than one inking station, the media may not stay in perfect alignment when moving between inking stations. Any misalignment will result in an error called misregistration. The visual effect of this error is either that inks are erroneously layered on top of one another, or, more seriously, that gaps occur between inks that should abut. In this second case, the color of the media is revealed in the gap and is frequently quite noticeable.

So **Trapping**, in short, is the process of modifying PDL files so that abutting colorant edges intentionally overlap slightly, in order to reduce the risk of gaps.

The **Trapping** process specifies that a set of document pages should be modified to reduce or (ideally) eliminate visible misregistration errors in the final printed output. The process may be combined with **RIPping** or specified as a stand-alone process.

Input Resources

Name	Description
ColorantControl	Identifies color model used by the job.
ColorPool	Identifies the specifics of individual colorants used by the job.
RunList	Structured list of incoming page contents that are to be trapped.
TrappingDetails	Describes the general setting needed to perform trapping.
TrappingParams	A set of TrappingParams resources that are referenced from the TrapRegion resources.
TrapRegion	A set of TrapRegion resources that identify the pages to be trapped, the geometry of the areas to trap on each page, and the trapping settings to use for each area.

Output Resources

Name	Description
RunList	Structured list of the modified page contents to which traps have been added.

6.3.11 Imposition

The **Imposition** process is responsible for combining several pages of input graphical content on to a single surface whose dimensions are reflective of the physical output media. Printer's marks can be added to the surface in order to facilitate various aspects of the production process. Among other things, these marks are used for press alignment, color calibration, job identification, and as guides for cutting and folding.

There are two mechanisms provided for controlling the flow of page images onto **Media**. The default mechanism, which provides the functionality of **Layout** in PJTF, explicitly identifies all page content for each **Sheet** imaged and references these pages by means of the **Documents** and/or **MarkDocuments** array. Setting the **Automated** attribute of the **Layout** resource to *true* activates a template approach to printing and relies upon the full **Documents** hierarchy to specify the page content to image. Automated impositioning is equivalent to the **PrintLayout** functionality in PJTF.

In JDF, there is a single **Layout** resource definition. Its structure is broad enough to encompass the needs of both fully specified and template-driven imposition. When described fully, the **Layout** resources include a **SignaturesPool** which specifies an array of **Signatures**. Each **Signature** in turn specifies an array of **Sheets**, and each **Sheet** can have up to two **Surfaces** (*Front* and *Back*), on which the page images and any marks are to be placed using **PlacedObjects**. A **Sheet** that specifies no **Surface** content will be blank. Pages that are to be printed must be placed onto **Surfaces** using **ContentObject** sub-elements which explicitly identify the page (via the **Ord** attribute which specifies an index into the document **RunList**). Thus, the **Layout** hierarchy specifies explicitly which pages will be imaged.

When describing automated imposition, **Layout** resources specify a single **Signature** of **Sheet(s)** where page contents are imaged. The (virtual) sequence of pages which is to be imaged via automated layout is defined by the Document **RunList**. Pages are drawn in order from this sequence to satisfy the **ContentObjects** in the **Surfaces** for the **Signature** in the **Layout**, and the **Signature** is repeated until all pages of the sequence are consumed. Each time the **Signature** is repeated, pages are consumed in 'chunks' whose size is determined by the value of $MaxOrd + 1$ (if present in the **Layout**), or by the largest **Ord** value or calculated **OrdExpression** value for any **ContentObject** in the **Signature** (if **MaxOrd** is absent).

Attributes of the **Media** are given for each **Sheet** used in printing. Because the same **Signature** is repeated until all pages are consumed, the **Layout** hierarchy can provide hints or preferences about special needs for sets of page content via **InsertSheet** elements. Inserting media is a way to separate sections of the document content. Thus alternate content is printed only as necessary to fill areas which would normally have page content because new media has been added, or to designate where a document section will begin as specified by the odd or even position of the **Signature**.

In a JDF model, impositioning is defined separately from other processes, which may precede or follow it. A **Combined** node may combine **Imposition** with other processes (such as **Separation** or **Interpreting**) to describe a device that happens to perform both in a single execution module.

Input Resources

Name	Description
Layout	A Layout resource that indicates how the content pages from the Document RunList and marks from the Marks RunList (see

	below) shall be combined onto imposed surfaces.
RunList (Document)	Structured list of incoming page contents which is transformed to produce the imposed surface images.
RunList ? (Marks)	Structured list of incoming marks. These are typically printer's marks such as fold marks, cut marks, punch marks, or color bars.

Output Resources

Name	Description
RunList	Structured list of imposed surfaces. The value of the <i>Type</i> attribute of the LayoutElement resources must all be <i>surface</i> .

6.3.12 PDFToPSConversion

The **PDFToPSConversion** process controls the generation of PostScript from a single PDF document. This process may be used at any time in a host-based PDF workflow to exit to PostScript for use of tools that consume such data. Additionally, it may be used to actively control the physical printing of data to a device that consumes PostScript data. The JDF model of this may include a **PDFToPSConversion** process in a *Combined* node with a **PSToPDFConversion** process.

Input Resources

Name	Description
PDFToPSConversionParams	Set of parameters required to control the generation of PostScript.
RunList	List of documents and pages to be converted to PostScript.

Output Resources

Name	Description
RunList	Stream or streams of resulting PostScript code. This PostScript code may end up physically stored in a file or be piped to another process. The <i>GeneratePageStreams</i> attribute of the PDFToPSConversionParams resource determines whether there is a single stream generated for all pages in the RunList or whether each page is generated in to a separate consecutive stream.

6.3.13 PSToPDFConversion

This section defines the controls required to invoke a device that accepts a PostScript stream and produces a set of PDF pages as output.

Input Resources

Name	Description
FontParams ?	These parameters determine how the conversion process will handle font errors encountered in the PostScript stream.
ImageCompressionParams ?	This resource provides a set of controls that determines how images will be compressed in the resulting PDF pages.
PSToPDFConversionParams ?	These parameters control the operation of the process that interprets the PostScript stream and produces the resulting PDF

interprets the PostScript stream and produces the resulting PDF pages.

RunList

This resource specifies where the PostScript stream is to be found.

Output Resources

Name	Description
RunList	This resource identifies the location of the resulting PDF pages.

6.3.14 RIPping

RIPping is, in the context of a workflow, a *Combined* process that is an amalgamation of at least two processes. Most often it includes *Interpreting* and *Rendering*, but it may also include *Trapping*, *Separation*, *Imposition*, and *Screening*. Thus a typical RIP node is of *Type Combined*, as shown in the following example:

```
<JDF Type="Combined" Types="Interpreting Rendering Screening" ... />
```

The **RIPping** process consumes page descriptions and instructions for producing the graphical output. It parses the graphical contents in the page descriptions, renders the contents, and produces a rasterized image of the page. This raster may contain contone data and be represented upon output as a **ByteMap**. Alternatively, the **RIPping** process may also perform halftone screening, in which case the output is in the form of a bitmap. It is also responsible for resolving all system resource references that include font handling and resource aliasing.

Instructions read by the RIP include information about the media, halftoning, color transformations, colorant controls and other items that affect that rasterized output. They do not, however, represent any specific controls for the physical output device, nor do they deal with any instructions intended for the finishing device.

When a **RIPping** process is comprised of only the *Interpreting* and *Rendering* processes, various intermediary steps are required before the output can be run through a **ConventionalPrinting** process. In theory, however, a workflow could include no intermediary steps between a **RIPping** process and a **DigitalPrinting** process. The following workflow scenarios represent possible process chains in each circumstance:

- RIP→Screening→ImageSetting→FilmToPlateCopying→ConventionalPrinting
- RIP→(Screening)→DigitalPrinting

Since RIPping never stands alone as a process, see the processes that contribute to the RIP for input and output resources.

6.3.15 Interpreting

The interpreting device consumes page descriptions and instructions for controlling the printing device. The parsing of graphical content in the page descriptions produces a canonical display list of the elements to be drawn on each page.

The interpreter may encounter, and must act upon, device control instructions that affect the physical functioning of the printing device, such as media selection and page delivery. **Media** selection determines

which type of medium is used for printing and where that medium can be obtained. Page delivery controls the location, orientation and quantity of physical output.

The interpreter is also responsible for resolving all system resource references. This includes handling font substitutions and dealing with resource aliases. However, the interpreter specifically does not get involved with any functions of the device that could be considered finishing features, such as stapling, duplexing and collating.

Input Resources

Name	Description
ColorantControl	Identifies the color model used by the job.
ColorPool	Identifies the specifics of individual colorants used by the job.
FontPolicy ?	Describes the behavior of the font machinery in absence of requested fonts.
InterpretingParams	Provides the parameters needed to interpret the PDL pages specified in the RunList resource.
PDLResourceAlias *	These resources allow a JDF to reference resources which are defined in a Page Description Language (PDL). For example, a PDLResourceAlias resource could refer to a font embedded in a PostScript file.
RunList	This resource identifies a set of PDL pages which will be interpreted.

Output Resources

Name	Description
InterpretedPDLData	Pipe of streamed data which represents the results of Interpreting the pages in the RunList . The format and detail of these data is implementation specific. In particular, it is assumed that the Interpreting and Rendering processes are tightly coupled and that there is no value in attempting to develop a general specification for the format of this data.

6.3.16 Rendering

The **Rendering** process consumes the display list of graphical elements generated by an interpreter. It color manages and scan converts the graphical elements according to the geometric and graphic state information contained within the display list.

The controls governing the external rendering processes provide overrides and additional parameters for controlling the behavior of the process.

Input Resources

Name	Description
Media	This resource provides a description of the physical media which will be marked. The physical characteristics of the media may affect decisions made during Rendering .
InterpretedPDLData	Pipe of streamed data that represents the results of Interpreting the pages in the RunList . The format and detail of these data is implementation specific. In particular, it is assumed that the

Interpreting and **Rendering** processes are tightly coupled and that there is no value in attempting to develop a general specification for the format of this data.

RenderingParams ?

This resource describes the format of the **ByteMap** resources to be created.

Output Resources

Name	Description
RunList	Ordered list of rasterized ByteMap resources representing pages

6.3.17 ContoneCalibration

This process specifies the process of contone calibration. It consumes contone raster data, such as that output from an interpreting and rendering process. It produces contone raster data, which has been calibrated to a press using a well-defined screening process.

Input Resources

Name	Description
RunList	Ordered list of rasterized ByteMap resource representing pages or surfaces.
ScreeningParams	Parameters specifying which halftoning mechanism is to be applied and with what specific controls.
TransferCurvePool	Specifies which calibration to apply.

Output Resources

Name	Description
RunList	Ordered list of rasterized ByteMap resources representing pages or surfaces.

6.3.18 Screening

This process specifies the process of halftone screening. It consumes contone raster data, such as that output from an interpreting and rendering process. It produces monochrome which has been filtered through a halftone screen to identify which pixels are required to approximate the original shades of color in the document.

This process definition includes capabilities for post-RIP halftoning according to the PostScript definitions. Alternatively it allows for the selection of FM screening/error diffusion techniques. However, in these circumstances no specific parameter sets are defined.

In general, an actual screening process will be a **Combined** process of **Calibration** and **Screening**.

Input Resources

Name	Description
RunList	Ordered list of rasterized ByteMap resources representing pages or surfaces.

ScreeningParams Parameters specifying which halftoning mechanism is to be applied and with what specific controls.

Output Resources

Name	Description
RunList	Ordered list of rasterized and screened output pages. Assumes that the resolution remains the same and that resulting data is one bit per component. Furthermore, the organization of planes within the data does not change.

6.3.19 SoftProofing

SoftProofing is the process of reviewing final-form output on a monitor rather than in paper form.

The inputs are a **RunList**, which identifies the pages to proof; the **ProofingParams** resource, which describes the type of proof to be created.

Within the **ProofingParams** resource, the proof device parameter specifies the characterization the monitor on which the proof will be viewed. This processor must create and perform a transformation from the final target device to the proof device colors before displaying the document contents.

The soft proofing parameters allow sufficient control to determine whether any images are displayed in the proof. If so, the ability to select low-resolution proxies or full resolution images is provided. The mechanism for approving proofs requires the generation of a PDF file containing the proofing parameters and a digital signature noting the acceptance of them. The approval PDF file need not contain any graphical data.

Like all other color manipulation supported in JDF, the color conversion controls are based on the use of ICC profiles. While the assumed characterization of input data can take many forms, each can internally be represented as an ICC Profile. In order to perform the transformations input profiles must be paired with the identified final target device profile to create the transformation.

Input Resources

Name	Description
ColorantControl	Identifies the color model used by the job.
ColorPool	Identifies the specifics of individual colorants used by the job.
ColorSpaceConversionParams ?	This resource provides information needed to convert colorspaces in the pages for proofing. Generally present if a color proof is desired, unless the pages in the RunList have already been operated on by a previous colorspace conversion process.
Layout ?	Required if an imposition proof is desired.
ProofingParams	Provides the parameters needed to produce the desired proof.
RunList (Document)	Identifies the pages to be proofed. When the Layout resource is present in the ProofingParams resource, <i>Ord</i> values from ContentObject sub-elements refer to pages in this RunList .

RunList ? (Marks)

Structured list of incoming marks. These are typically printer's marks such as fold marks, cut marks, punch marks, or color bars.

When the **Layout** resource is present in the **ProofingParams** resource, *Ord* values from **MarkObject** sub-elements refer to pages in this **RunList**.

Output Resources

None. The **SoftProofing** process is always combined with an **Approval** process.

6.3.20 Proofing

The **Proofing** process results in the creation of a physical proof, represented by an **ExposedMedia** resource. Proofs can be used to check an imposition, or the expected colors for a job.

The inputs of this process are a **RunList**, which identifies the pages to proof; the **ProofingParams** resource, which describes the type of proof to be created; and a **Media** resource to describe the physical media that will be used.

Input Resources

Name	Description
ColorantControl	Identifies the color model used by the job.
ColorPool	Identifies the specifics of individual colorants used by the job.
ColorSpaceConversionParams ?	This resource provides information needed to convert colorspaces in the pages for proofing. Generally present if a color proof is desired, unless the pages in the RunList have already been operated on by a previous colorspace conversion process.
Layout ?	Required if an imposition proof is desired.
Media	This resource characterizes the output media for the proof.
ProofingParams	This resource provides the parameters needed to produce the desired proof.
RunList (Document)	Identifies the pages to be proofed. When the Layout resource is present in the ProofingParams resource, <i>Ord</i> values from ContentObject sub-elements refer to pages in this RunList .
RunList ? (Marks)	Structured list of incoming marks. These are typically printers marks, such as fold, cut or punch marks, or color bars. When the Layout resource is present in the ProofingParams resource, <i>Ord</i> values from MarkObject sub-elements refer to pages in this RunList .

Output Resources

Name	Description
ExposedMedia	The resulting physical proof.

6.3.21 PreviewGeneration

The **PreviewGeneration** process produces a low-resolution **Preview** of each separation that will be printed. The **Preview** can be used in later processes such as **InkZoneCalculation**. The **PreviewGeneration** process typically takes place after **Imposition** or **RIPping**.

The **PreviewGeneration** can be performed in one of the following two ways: either the imaged printing plate is scanned by a conventional plate scanner or high resolution digital data are used to generate the **Preview** for the separation(s).

The extent of the PDL coordinate system as specified by the **MediaBox** attribute, the resolution of the preview image, and width and height of the image must fulfill the following requirements:

$$\begin{aligned} \text{MediaBox length} / 72 * \text{x-resolution} &= \text{width} \pm 1 \\ \text{MediaBox height} / 72 * \text{y-resolution} &= \text{height} \pm 1 \end{aligned}$$

A gray value of 0 represents full ink, while a value of 255 represents no ink (see the DeviceGray color model in chapter 4.8.2. of the *PostScript Language Reference Manual*).

Rules for the Generation of the Preview Image

To be useful for the ink consumption calculation, the preview data must be generated with an appropriate resolution. This does not only mean spatial resolution, but also color or tonal resolution. Spatial resolution is important for thin lines, while tonal resolution becomes important with large areas filled with a certain tonal value.

The maximum error caused by limited spatial and tonal resolution should be less than 1 %.

Spatial Resolution

Since some pixel of the preview image might fall on the border between two zones, their tonal values must be split up. In a worst-case scenario, the pixels fall just in the middle between a totally white and a totally black zone. In this case, the tonal value is 50%, but only 25% contributes to the black zone. With the resolution of the preview image and the zone width as variables, the maximum error can be calculated using the following equation:

$$\text{error}[\%] = \frac{100}{4 * \text{resolution}[L / mm] * \text{zone_width}[mm]}$$

For zone width broader than 25 mm, a resolution of 2 lines per mm will always result in an error less than 0.5 %. Therefore a resolution of 2 lines per mm (equal to 50.8 dpi) is suggested.

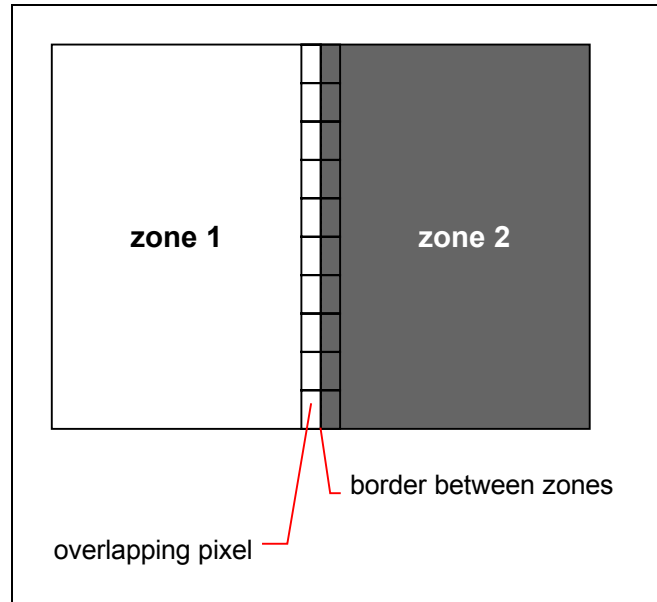


Figure 6.1 Worst-case scenario for area coverage calculation

Tonal Resolution

The kind of error caused by color quantization depends on the number of shades available. If the real tonal value is rounded to the closest (lower or higher) available shade, the error can be calculated using the following equation:

$$error[\%] = \frac{100}{2 * number_of_shades}$$

Therefore, at least 64 shades should be used.

Line-Art Resolution

When rasterizing line-art elements, the minimal line width is 1 pixel, which means $1/resolution$. Therefore, the relationship between the printing resolution and the (spatial) resolution of the preview image is important for these kind of elements. In addition, a specific characteristic of PostScript RIPs adds another error—within PostScript, each pixel that is touched by a line is set.

Tests with different PostScript jobs have shown that a line-art resolution of more than 300 dpi is normally sufficient for ink-consumption calculation.

Conclusion

There are quite a few different ways to meet the requirements listed above. The following list includes several examples:

- The job can be RIPped with 406.4 dpi monochrome.
- With anti-aliasing, the image data can be filtered down by a factor of 8 in both directions. This results in an image of 50.8 dpi with 65 color shades.
- High-resolution data can also be filtered using anti-aliasing. First, the RIPped data, at 2540 dpi monochrome, is taken and filtered down by a factor of 50 in both directions. This produces an

image of 50.8 dpi with 2501 color shades. Finally those shades are mapped to 256 shades, without affecting the spatial resolution.

Rasterizing a job with 50.8 dpi and 256 shades of gray is not sufficient. The problem in this case is the rendering of thin lines (see: Line Art Resolution).

Recommendations for Implementation

The following three guidelines are strongly recommended:

- The resolution of RIPPed line art must be at least 300 dpi.
- The spatial resolution of the preview image must be approximately 20 pixel/cm (= 50.8 dpi).
- The tonal resolution of the preview image must be at least 64 shades.

Input Resources

Name	Description
ExposedMedia ?	The PreviewGeneration process can use an exposed printing plate to produce a Preview resource. This task is performed using an analog plate-scanner.
PreviewGenerationParams	Parameters specifying the size and the type of the preview.
RunList ?	High-resolution bitmap data is consumed by the PreviewGeneration process. These data represent the content of a separation that is recorded on a printing plate or other such item.

Output Resources

Name	Description
Preview	The Preview data are comprised of low-resolution bitmap files representing, for example, the content of a separation that is recorded on a printing plate or other such item.

6.3.22 InkZoneCalculation

The **InkZoneCalculation** process takes place in order to preset the ink zones before printing. The **Preview** data are used to calculate a coverage profile that represents the ink distribution along and perpendicular to the ink zones within the printable area of the preview. The **InkZoneProfile** can be combined with additional, vendor-specific data in order to preset the ink zones and the oscillating rollers of an offset printing press.

Input Resources

Name	Description
InkZoneCalculationParams	Specific information about the printing press geometry (such as the number of zones) to calculate the InkZoneProfile .
Preview	A low-resolution bitmap file representing the content to be printed.
Sheet ?	Specific information about the Media (including type and color) and about the Sheet (placement coordinates on the printing cylinder).
TransferCurvePool ?	Function to apply FilmToPlateCopying , DigitalPrinting , and ConventionalPrinting process characteristics such as press, climate, and substrate under certain standardized circumstances.

This function can be used to generate an accurate **InkZoneProfile**.

Output Resources

Name	Description
InkZoneProfile	Contains information about ink coverage along and perpendicular to the ink zones for a specific press geometry.

6.3.23 Tiling

The **Tiling** process allows the contents of **Surfaces** to be imaged onto separate pieces of media. Note that many different workflows are possible. **Tiling** must always follow **Imposition**, but it can operate on imposed PDL page contents or on contone or halftone data.

Tiling will generally be combined with other prepress processes. For example, **Tiling** might be combined with **ImageSetting**. In that case, the input would be a **RunList** that contains **ByteMap** resources for each **Surface**.

Input Resources

Name	Description
RunList (Surface)	Structured list of imposed page contents or ByteMap resources that are to be decomposed to produce the images for each tile. The <i>Type</i> value of LayoutElement resources must all be <i>surface</i> .
RunList ? (Marks)	Structured list of incoming marks. These are typically printer's marks that provide the information needed to combine the tiles.
Tile	A partitioned Tile resource that describes how the Surface contents are to be decomposed.

Output Resources

Name	Description
RunList	Structured list of portions of the decomposed surfaces. The value of the <i>Type</i> attribute of the LayoutElement resources must be <i>tile</i> .

6.3.24 ImageSetting

The image recording process is executed by an image setter or plate setter that images a bitmap onto the film or plate media. Its inputs are **Media**, a **RunList** of bytemaps or images that represents the image, and some additional **ImageSetterParams**.

Input Resources

Name	Description
ImageSetterParams	Controls the device specific features of the image setter.
Media	The unexposed media.
RunList	Identifies the set of bitmaps to image. May contain bytemaps or images.

Output Resources

Name	Description
ExposedMedia	The exposed media resource.

6.3.25 FilmToPlateCopying

FilmToPlateCopying is the process of making an analog copy of a film onto a printing plate.

Input Resources

Name	Description
ExposedMedia	The film or films to be copied onto the plate.
Media	The unexposed plate.
PlateCopyParams	The settings of the exposure task.

Output Resources

Name	Description
ExposedMedia	The resulting exposed plate.

6.4 Press Processes

Press processes are various technological procedures involving the transfer of ink to a substrate. From a technical standpoint they are often classified in impact and non-impact printing technologies. The impact printing class can be further subdivided into relief, intaglio, planograph or screen technologies, which in turn can be divided in further subparts. Because of the way a workflow is constructed in JDF, however, a different approach to classification was used. All of the various printing technologies are gathered into two categories: **ConventionalPrinting**, which involves printing from a physical master, or **DigitalPrinting**, which involves printing from a digital master.

The most prominent physical, planographic printing technologies are offset-lithography and electrophotography and they are also the printing processes with the highest adoption in today's Graphic Arts industry. Consequently, the **ConventionalPrinting** process in JDF takes them as models. That does not mean, however, that other printing techniques can not make use of the **ConventionalPrinting** process and its resources. The extensibility features of JDF may be used to fill other requirements related to printing technology.

6.4.1 ConventionalPrinting

This process covers several conventional printing tasks, including sheet-fed printing, web printing, web/ribbon coating, converting, and varnishing. Typically, each takes place after prepress and before postpress processes.

Press machinery often includes postpress processes such as **Folding**, **Numbering**, and **Cutting** as inline finishing operations. The **ConventionalPrinting** process itself does not cover this postpress tasks.

Using a conventional printing press for producing a pressproof can be performed in the following two ways:

- A proof of type **Component** is produced with a **ConventionalPrinting** process. The result of this process is then sent to the **Approval** process, which in turn produces an **ApprovalSuccess**

resource. That resource is then passed on to a second **ConventionalPrinting** process, which requires that the press be set up a second time.

- The *DirectProof* attribute of the **ConventionalPrintingParams** can be used to specify the proof if it is produced during the **ConventionalPrinting** process. In this case, the press need only be set up once.

Note, the definition and ordering of separations is specified by the *DeviceColorantOrder* attribute of the appropriate **ColorantControl** resource, which is located in the **Layout** tree.

Input Resources

Name	Description
ColorantControl ?	The ColorantControl resources that define the ordering and usage of inks in print modules.
ColorPool ?	Identifies the specifics of individual colorants used by the job.
Component ? (input)	Various components in the form of preprints can be used in ConventionalPrinting in lieu of Media . Examples include waste or a set of pre-printed sheets.
Component ? (Proof)	A Proof component is used if a proof was produced during an earlier ConventionalPrinting process.
ConventionalPrintingParams	Specific parameters to set up the press.
ExposedMedia ? (Proof)	A Proof is used to compare color and content during ConventionalPrinting . This Proof is produced by a prepress proofing device.
ExposedMedia (Plate)	The printing plate and information about it (such as <i>Thickness</i> and <i>RegisterPunch</i>) is used to set up the press.
Ink	Information (brand, type, clone) about the ink is useful to set up the press.
InkZoneProfile *	The InkZoneProfile contains information about how much ink is needed along the printing cylinder of a specific printing press. It is only useful for Offset-Lithography presses with ink key adjustment functions.
Layout ?	Sheet elements such as the CIELABMeasuringField , DensityMeasuringField , or ColorControlStrip can be used for quality control at the press. The quality control field value and position can be of interest for automatic quality control systems. RegisterMark can be used to line up the printing plates for the press run, and its position can in turn be used to position items such as a camera.
Media ?	The physical substrate—for example, paper or foil—and information about the Media —such as thickness, type, and size—are useful in setting up paper travel in the press. This resource must be present if no pre-printed Component (input) resource is used.

Output Resources

Name	Description
Component (Good)	Describes the printed sheets or ribbons which may be used by

	another printing process or postpress processes.
Component (Waste)	Produced waste of printed sheets or ribbons.

6.4.2 DigitalPrinting

DigitalPrinting is a direct printing process that, like **ConventionalPrinting**, occurs after prepress processes but before postpress processes. In **DigitalPrinting**, the data to be printed are not stored on an extra medium (such as a printing plate or a printing foil), but instead are stored digitally. The printed image is generated for every print out using the digital data, and electrophotography, inkjet and other technologies are used for transferring ink (both liquid ink and dry toner) onto the substrate. Furthermore, both sheet and web presses can be used as machinery for **DigitalPrinting**.

DigitalPrinting is often used to image a small area on preprinted **Components** to perform actions such as addressing or numbering another **Component**. This kind of process can be executed by imaging with an inkjet printer during press, postpress, or packaging operations. Therefore, **DigitalPrinting** is not only a press or prepress operation but sometimes also a postpress process.

Note: Putting a label on a product or package is *not* **DigitalPrinting** but **Inserting**.

Input Resources

Name	Description
ColorantControl ?	The ColorantControl resources that define the ordering and usage of inks in print modules.
ColorPool	Identifies the specifics of individual colorants used by the job.
Component ? (input)	Various components can be used in DigitalPrinting instead of Media . Examples include waste, pre-cut Media , or a set of pre-printed sheets or webs.
Component ? (Proof)	A Proof component is used if a proof was produced during an earlier ConventionalPrinting process (see description a. above).
DigitalPrintingParams	Specific parameters to set up the machinery.
ExposedMedia ?	A Proof is useful for comparisons (completeness, color accuracy) with the print out of the DigitalPrinting process.
Ink	Toner and information about it is needed for DigitalPrinting .
Layout ?	Sheet elements such as the CIELABMeasuringField , DensityMeasuringField , or ColorControlStrip can be used for quality control at the press. The value and position of the quality can be of interest for automatic quality control systems. RegisterMarks can be used to line up the printing registration during press run, and its position can in turn be used to position an item such as a camera.
Media ?	The physical Media and information about the Media , such as thickness, type, and size, is used to set up paper travel in the press. This has to be present if no pre-printed Component (input) resource is present. Note: Printing a job on more than one web or sheet at the same time is parallel processing.
RunList	RIPped data that will be printed on the digital press is needed for DigitalPrinting . The RunList contains only ByteMap resources.

Output Resources

Name	Description
Component (Good)	Components are produced for other printing processes or postpress processes.
Component (Waste)	Produced waste, may be used by other processes.

6.4.3 IDPrinting

IDPrinting, which stands for Integrated Digital Printing, is a specific form of digital printing. It combines functionality that might be represented by the **Interpreting**, **Rendering**, **Screening** and **DigitalPrinting** processes in a single process.

Controls for **IDPrinting** are provided in the **IDPrintingParams** resource. These controls are intended to be somewhat limited in their scope. If greater control over various aspects of the printing process is required, **IDPrinting** should not be used. Ultimately, the controls specified for **IDPrinting** can be used to generate an Internet Printing Protocol job.

IDPrinting may be combined with other processes, such as **Trapping** or **ColorSpaceConversion**.

Input Resources

Name	Description
ColorantControl ?	The ColorantControl resources that define the ordering and usage of inks in print modules.
ColorPool	Identifies the specifics of individual colorants used by the job.
Component ? (cover)	A finished cover may be combined with the pages that will be output by this process.
Component ? (input)	Various components can be used in IDPrinting instead of Media . Examples include waste, precut Media , or a set of pre-printed sheets or webs.
Component ? (Proof)	A Proof component is used if a proof was produced during an earlier ConventionalPrinting process.
ExposedMedia ?	A Proof is useful for comparisons (completeness, color accuracy) with the print out of the IDPrinting process.
FontPolicy ?	Describes the behavior of the font machinery in absence of requested fonts.
InterpretingParams *	A set of resources that specify how the device should interpret the pdl files which are referenced by the RunList for the process. Note that InterpretingParams is an abstract resource – instances are pdl-specific.
IDPrintingParams	Specific parameters to set up the machinery.
Media ?	The physical Media and information about the Media , such as thickness, type, and size, is used to set up paper travel in the press. This has to be present if no pre-printed Component (input) resource is present. Note: Printing a job on more than one web or sheet at the same time is parallel processing.
RenderingParams ?	This resource describes the format of the ByteMaps to be created.

RunList	The set of pages to be printed.
ScreeningParams ?	Parameters specifying which halftoning mechanism is to be applied and with what specific controls.

Output Resources

Name	Description
Component (Good)	Components are produced for other printing processes or postpress processes. Note that the <i>Amount</i> attribute of the ResourceLink to this resource indicates the number of copies which shall be produced.
Component (Waste)	Produced waste, may be used by other processes.

6.5 Postpress Processes

In this specification, the postpress processes are divided into sub-chapters for structuring purposes. This structuring is useful to find specific processes. Please note that processes, in some cases can be used to describe operations that go beyond the scope of the a specific chapter. Therefore, it is a good idea not only to look at certain processes within a subchapter but also to find out what functionality other processes offer if a specific task needs to be addressed.

6.5.1 Web Processes

This sub-chapter of the postpress processes is dedicated to web and ribbon operations—that is, operations that require a web or a ribbon to execute. In essence, a ribbon is a web that has been slit or cross-cut. More specifically, a web is a continuous strip of **Media** to be used for printing, such as paper or foil. This substrate is called “web” while it is treaded through the printing machinery, but once it has run through the Dividing process and been slit, the web no longer exists. In its place are ribbons or sheets.

A ribbon, then, is the part of the web that enters the folder. If the web is never slit, however, the web and the ribbon are identical. Slitting and salvage-trim operations on a web can result in one or more ribbons; a ribbon can be further subdivided after it has been slit.

After the **Dividing** process, sheets are treated further. The **Gathering** process and **Folding** process also handle web and ribbon applications.

6.5.1.1 Dividing

Inline finishing of web presses often include equipment for cutting the ribbon(s) in cross direction. This operations can be described with the **Dividing** process. **Dividing** in cross direction is likely to happen after former folding, which is a **LongitudinalRibbonOperations** process. It may affect one or more ribbons at the same time that are all part of one **Component**.

Input Resources

Name	Description
Component	The Dividing process consumes one Component : the web(s) or ribbon(s) entering the cross-cutting machinery. The substrate might have been treated with LongitudinalRibbonOperations and folded with a former fold.

DividingParams Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	One Component is produced: either the divided web or ribbon.

6.5.1.2 LongitudinalRibbonOperations

Inline finishing within web printing presses can include folding, perforating, or applying a line of glue on the ribbon while it is traveling in longitudinal direction.

Input Resources

Name	Description
Component	The Component can consist of more than one web or ribbon that have been combined with the Gathering process.
LongitudinalRibbonOperationsParams	Specific parameters to set up the machinery tools for the LongitudinalRibbonOperations process.

Output Resources

Name	Description
Component +	A ribbon is produced that is used in other postpress processes. If the LongitudinalRibbonOperations process was slitting, more than one Component is produced.

6.5.2 HoleMaking

A variety of machines, such as those responsible for stamping and drilling, can perform the **HoleMaking** process. This postpress process is needed for different binding techniques, such as spiral binding. One or several holes with different shapes can be made that are later on used for binding the book block together.

Input Resources

Name	Description
Component	One Component , such as a printed sheet or a pile of sheets, are modified in the HoleMaking process.
HoleMakingParams	Specific parameters, including hole diameter, and positions, used to set up the machinery.

Output Resources

Name	Description
Component	A Component with holes, such as a book block or a single sheet, is produced for further postpress processes.

6.5.3 Tip-on/in

The following processes (*EndSheetGluing*, *Inserting*) are part of the postpress operations. They can be grouped together as the tip-on/in-processes. Both processes can be performed by hand, tip-on/in machine, or by a press.

6.5.3.1 EndSheetGluing

EndSheetGluing finalizes the folded **Sheet** or book block in preparation for case binding. It requires three **Components**—the back-end sheet, the book block, and the front-end sheet—and information about how they are merged together.

Back-end sheets and front-end sheets are in most cases sheets folded once before *EndSheetGluing* takes place. The end sheets serve as connections between the book block and the cover boards.

Input Resources

Name	Description
Component (back-end sheet)	A back-end sheet to be mounted on the book block.
Component (book block)	A back-end sheet and a front-end sheet are glued onto the book block.
Component (front-end sheet)	A front-end sheet to be mounted on the book block.
EndSheetGluingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	A book block is produced that includes the end sheets.

6.5.3.2 Inserting

This process can be performed at several stages in postpress. The process can be used to describe the labeling of products, of packages, or the gluing-in of a **Component** (such as Card, Sheet, or CD-ROM). Two **Components** are required for the *Inserting* process: the “mother” **Component** and the “child” **Component**. Inserting can be a selective process by means of inserting different “*child*”-**Components**. Information about the placement is needed to perform the process.

Input Resources

Name	Description
Component (mother)	Designates where to insert the child Component .
Component (child)	The Component to be inserted in the mother Component .
InsertingParams	Specific parameters, such as placement, to set up the machinery.
DBRules ?	Database input that describes whether the child should be inserted for a particular instance component. In this version the schema is only human readable text.
DBSelection ?	Database input that describes whether the child should be inserted for a particular instance component.
IdentificationField ?	Information about identification marks on the component.

Output Resources

Name	Description
Component	A mother Component is produced containing the inserted child Component .

6.5.4 Block Production

This subcategory of the postpress processes merges together all the processes for making a book block. First the block is compiled using the Collecting and Gathering processes. After that, it is combined using one or several of the block joining processes, including AdhesiveBinding, SaddleStitching, SideSewing, Stitching, and ThreadSewing. The workflow using these processes eventually produces a **Component** that can be trimmed.

6.5.4.1 Block Compiling

The **Gathering** and **Collecting** processes are used to position unfolded sheets and/or folded sheets in a planned order. These operations set a fixed page sequence in preparation for three-side trimming and binding.

6.5.4.1.1 Collecting

This process collects folded sheets or partial products, some of which may have been cut. The first **Component** to enter the workflow lies at the bottom of the pile collected on a saddle, and the sequence of the input components that follows depends upon the produced component.

The operation coordinate system is defined as follows:

The y-axis is aligned with the binding edge. It increases from the registered edge to the edge opposite to the registered edge. The x-axis is aligned with the registered edge. It increases from the binding edge to the edge opposite to the binding edge (i.e. the product front edge).

Input Resources

Name	Description
CollectingParams ?	Specific parameters to set up the machinery.
Component +	Variable amount of sheets to be collected.
DBRules *	Database input that describes which sheets should be collected for a particular instance component. In this version the schema is only human readable text. One rule is applied for each individual component.
DBSelection ?	Database input that describes which sheets should be collected for a particular instance component.
IdentificationField ?	Information about identification marks on the component.

Output Resources

Name	Description
Component	A block of collected sheets is produced. This Component can be joined in further postpress processes.

6.5.4.1.2 Gathering

In the **Gathering** process, ribbons, sheets, or other **Components** are accumulated on a pile that will, eventually, be stitched or glued in some way. The input **Components** may be output resources of a web printing machine used in **Collecting** or of any machine that executes a **ConventionalPrinting** or **DigitalPrinting** process. In sheet applications, a moving gathering channel is used to transport the pile. But no matter what the inception of the **Gathering** process, the sequence of the input components dictates the produced component.

Input Resources

Name	Description
Component +	Variable amount of components including single sheets or folded sheets are used in the Gathering process.
GatheringParams	Specific parameters to set up the machinery.
DBRules *	Database input that describes which sheets should be gathered for a particular instance component. The schema are only in the form of human-readable text. One rule is applied for each individual component.
DBSelection ?	Database input that describes which sheets should be gathered for a particular instance component.
IdentificationField ?	Information about identification marks on the component.

Output

Name	Description
Component	Components gathered together, such as a pile of folded sheets.

6.5.4.2 Block Joining

The block joining processes can be grouped into two major subcategories: conventional binding methods, which includes the processes of **Stitching**, **SaddleStitching**, **AdhesiveBinding**, **ThreadSewing**, and **SideSewing**; and single-leaf binding methods, which are listed in section 6.5.4.2.6 Single Leaf Binding Methods. Together they form a sub-category of block-production processes. All of these processes, which are known as block-joining processes, unite sheets and/or folded sheets lying loose on top of each other.

There are numerous possible binding methods. The most prominent ones are modeled by the processes described in the following sections. Many of them can be part of a combined production chain being performed as inline tasks.

6.5.4.2.1 AdhesiveBinding

AdhesiveBinding is a process that addresses the following binding operations:

- perfect binding
- back preparation including milling and notching
- glue application
- spine taping
- cover application

Input Resources

Name	Description
AdhesiveBindingParams	Specific parameters to set up the machinery.
Component (bookblock)	The book block on which the cover is applied.
Component ? (cover)	An additional component for many AdhesiveBinding processes is the cover.

Output Resources

Name	Description
Component	The bound components forming an item such as a raw book.

6.5.4.2.2 SaddleStitching

In **SaddleStitching**, signatures are gathered so that all sections have a common spine, then stitched with staples through the spine.

Input Resources

Name	Description
Component	The only required Component is the collected pile.
SaddleStitchingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	The stitched-together components.

6.5.4.2.3 SideSewing

This is a binding technique resulting in robust products that have a significant loss of inner margin space and poor handling characteristics. For these reasons, other binding techniques are used more often.

In **SideSewing**, the first step is to create the holes in the book block and inject the glue (see section 6.5.2 **HoleMaking**). Then the entire book is sewn at once with a **ThreadMaterial** such as *Cotton or Polyester*. If the book block is rather thick, a **Stitching** process using wire might be performed before **SideSewing**.

Input Resources

Name	Description
Component	The only required Component is the gathered sheets.
SideSewingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	The book is produced.

6.5.4.2.4 Stitching

Gathered or collected sheets or signatures are stitched together with the cover.

Input Resources

Name	Description
Component	The only required Component is the pile of gathered or collected sheets, including the cover.
StitchingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	One Component is produced: the gathered or collected sheets including the cover stitched together.

6.5.4.2.5 ThreadSewing

This process may include a gluing application, which would be used principally between the first and the second or the last and the last sheet but one. Gluing may also be necessary if different types of paper are used.

Input Resources

Name	Description
Component	The operation requires one component: the gathered sheets.
<i>Template</i>	Specific parameters to set up the machinery. string Template to define a sequence of variables consumed by <i>Format</i> . A list of pre-defined values is found in the description of the FileSpec resource. In addition, DynamicInput elements of a RunList define further variables.
ThreadSewingParams	

Output Resources

Name	Description
Component	One Component is produced: the thread sewn components forming an item such as a raw book.

6.5.4.2.6 Single Leaf Binding Methods

Besides the conventional binding methods, there is a multifaceted group of binding methods for single leaf bindings. This group can again be subdivided into two subtypes: loose leaf binding and mechanical binding, each of which is described in the sections that follow.

6.5.4.2.6.1 Loose Leaf Binding Method

This binding techniques allow contents to be changed, inserted, or removed at will. There are two essential groups of loose-leaf binding systems: those that require the paper to be punched or drilled and those that do

not. The **RingBinding** method, described in the next section, is the most prominent binding in the loose leaf binding category.

6.5.4.2.6.1.1 RingBinding

In this process, pre-punched sheets are placed in a ring-binder. Ring-binders have different numbers of rings that are fixed to a metal backbone. In most cases, two, three, or four metal rings hold the sheets together as long as the binding is closed. Depending on the amount of sheets to be bound together different thicknesses of ring binders must be used.

Input Resources

Name	Description
Component (bookblock)	The operation requires one component: the pile of pre-punched sheets to be inserted into the ring-binder.
Component ? (ringbinder)	The empty ring-binder that might have been printed, for example, before it is used during the RingBinding process.
RingBindingParams	Specific parameters to set up the process/machinery.

Output Resources

Name	Description
Component	One Component is produced: the thread sewn components forming an item such as a raw book.

6.5.4.2.6.2 Mechanical Binding Methods

Single leafs are fastened into what is essentially a permanent system that is not meant to be reopened. However, special machinery can be used to re-open some of the mechanical binding systems described below.

In mechanical binding, printing and folding can be done in a conventional manner. The gathered sheets, however, often require the back to be trimmed, as well as the other three sides. Mechanical bindings are often used for short-run jobs such as ones that have been printed digitally. The most prominent mechanical binding processes are described in the sections that follow.

6.5.4.2.6.2.1 ChannelBinding

Various sizes of metal clamps can be used in **ChannelBinding**. The process can be executed in two ways. In the first, a pile of single sheets—sometimes together with a front and back cover—is inserted into a U-shaped clamp and crimped in a special machinery. In the second, a pre-assembled cover that includes the open U-shaped clamp is used instead of the U-shaped clamp alone. The thickness of the pile of sheets determines in both cases the width of the U-shaped clamp to be used for forming the fixed document, which is not meant to be re-opened later.

Input Resources

Name	Description
Component (bookblock)	The operation requires one component: the block of sheets to be bound.
Component ? (cover)	The empty cover with the U-shaped clamp that might, for example,

have been printed before it is used during the **ChannelBinding** process.

ChannelBindingParams Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	One Component is produced: the channel-bound component forming an item such as a brochure.

6.5.4.2.6.2.2 CoilBinding

CoilBinding is a technique that creates bindings not meant to be re-opened later. Another name for **CoilBinding** is *spiral binding*. Metal wire, wire with plastic, or pure plastic is used to fasten pre-punched sheets of paper, cardboard, or other such materials. First, automated machinery forms a spiral of proper diameter and length. The ends of the spiral are then “tucked-in”. Finally, the content is permanently fixed. Note that every time a coil-bound book is opened, a vertical shift occurs as a result of the coil action. This is a characteristic of the process.

Input Resources

Name	Description
Component	The operation requires one component: the pile of pre-punched sheets often including a top and button cover.
CoilBindingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	One Component is produced: the coil-bound component forming an item such as a calendar.

6.5.4.2.6.2.3 PlasticCombBinding

In the **PlasticCombBinding** process, a plastic insert wraps through pre-punched holes in the substrate. Most often, these holes are rectangular and elongated. After the plastic comb is opened with a special tool, the pre-punched block of sheets—often together with a top and button cover—is inserted onto the “teeth” of the plastic comb. When released from the machine, the “teeth” return to their original cylindrical positions with the points tucked into the backside of the spine area. Special machinery can be used to re-open the plastic comp binding.

Input Resources

Name	Description
Component	The operation requires one component: the pile of sheets often including a top and button cover.
PlasticCombBindingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
------	-------------

Component	One Component is produced: the plastic-comb-bound component forming an item such as a calendar.
------------------	--

6.5.4.2.6.2.4 VeloBinding

Hard plastic strips are held together by plastic pins, which in turn are bound to the strips with heat. The sheets to be bound must be pre-punched so that the top strip with multiple pins fits through the assembled material. It is then connected to the bottom strip with matching holes for the pins. The binding edge is often compressed in a special machine before the excess pin length is cut off. The backstrip is permanently fixed with plastic clamping bars and cannot be removed without a special tool.

Input Resources

Name	Description
Component	The operation requires one component: the block of sheets to be bound.
VeloBindingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	One Component is produced: the velo bound component forming an item such as a book.

6.5.4.2.6.2.5 WireCombBinding

The **WireCombBinding** is a technique that creates bindings not meant to be re-opened later. **WireCombBinding** is often named *Wire-O[®]-binding*. Metal wire, wire with plastic, or pure plastic is used to fasten pre-punched sheets of paper, cardboard, or other such materials. The wire—often formed as a double wire—is inserted into the holes, then curled to create a circular enclosure.

Input Resources

Name	Description
Component	The operation requires one component: the pile of pre-printed sheets often including a top and button cover.
WireCombBindingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	One Component is produced: the wire-comb bound component forming an item such as a calendar.

6.5.5 Numbering

Numbering is the process of stamping or applying variable marks in order to produce unique components, for items such as lottery notes or currency. No database access is required, and the counters automatically increase incrementally. **Numbering** is also used for alphanumeric, automatic, and unique marking.

Input Resources

Name	Description
Component	One Component , such as a printed sheet or a pile of sheets, are modified in the Numbering process.
NumberingParams	Specific parameters, including start counter and positions, to set up the machinery.

Output Resources

Name	Description
Component	One Component is produced: the numbered sheet.

6.5.6 Sheet Processes

Many printing processes produce sheets that must be processed further in finishing operations. The web processes presented in the preceding sections result in sheets that are treated in much the same way as sheets produced by sheet-fed printing presses. The following processes describe these sheet finishing operations.

6.5.6.1 Cutting

Sheets are cut using a guillotine **Cutting** machine. Before **Cutting**, the sheets might be jogged and buffered. **CutBlocks** and or **CutMarks** can be used for positioning the knife. After the **Cutting** process is performed, the blocks are often again buffered on a pallet.

Since **Cutting** is described here in a way that is, as much as possible, machine independent, the **CutBlock** elements specified do not directly imply a certain cutting sequence. Therefore, a sequence must be determined by a specialized agent.

Input Resources

Name	Description
Component ?	This process consumes one Component : the printed sheets.
CutBlock *	One or several CutBlocks can be used to find the Cutting sequence.
CutMark *	CutMark s can be used to adapt the theoretical cut positions to the real positions of the corresponding blocks on the Component to be cut.
Media ?	Cutting can be performed to unprinted Media in order to adjust size or shape.

Output Resources

Name	Description
Component +	One or several blocks of cut components are produced. When Media are cut, the output Components can be input resources for processes such as ConventionalPrinting .

6.5.6.2 Folding

Buckle folders or knife folders are used for **Folding** sheets. One or more sheets can be folded at the same time.

Web presses often provide inline **Folding** equipment. Longitudinal **Folding** is often performed using a former, a plow folder, or a belt, while jaw folding, chopper folding, or drum folding equipment is used for folding the sheets that have been divided.

The JDF **Folding** process covers both operations done in stand alone **Folding** machinery—typically found for processing sheet fed printed materials—and inline equipment of web printing presses.

Creasing and/or slot perforating are sometimes necessary parts of the **Folding** operation that guarantee exact process execution. They depend on the folder used, the **Media**, and the folding layout. The decision to perform this operations is left to the agent.

Besides **Folding**, other processes that “add value” to the product, such as cutting creasing, gluing, perforating, and thread-sealing might be performed in the **Folding** machine or in an extra machine. The **FoldingParams** resource can be used to address these variations.

Input Resources

Name	Description
Component	Components including a printed Sheet or a pile of Sheets are used in the Folding process.
FoldingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component +	The process produces components, which in most cases are folded Sheets that might be cut, creased, glued, perforated, or thread-sealed for further postpress processes. If the input Component is cut, several Component resources are produced.

6.5.7 Trimming

The **Trimming** process is performed to adjust the book block to its final size. In most cases, it follows a block joining process, and the process is often executed as an inline operation of a production chain. For example, the binding station may deliver the book blocks to the trimmer. A **Combined** operation in the trimming machinery would then execute a cut at the front, head, and tail in a cycle of two operations. Closed edges of folded signatures would then be opened while the book block is trimmed to its predetermined dimensions.

Some trimming machines, such as magazine production systems, can produce multiple-ups. In every case, however, the additional trimming cuts that divide the multiple-ups result in separated book blocks. Sometimes a stripe is trimmed out between the book blocks. To describe these operations, multiple **Trimming** processes must be defined in JDF.

Input Resources

Name	Description
Component	A bound book block is required for Trimming .

TrimmingParams Specific parameters, such as trim size, to set up the machinery.

Output Resources

Name	Description
Component	One Component is produced: the trimmed component.

Chapter 7 Resources

Resources represent inputs and outputs, the ‘things’ that are produced, modified, consumed or in any way used by nodes. A more thorough description was provided in section 3.6 Resources.

The resources in this chapter are divided into two sections. The first section documents all of the resources of class *Intent*. The second section documents the rest of the resources that have been defined for JDF.

7.1 Intent Resources

As was described in section 4.1.1 Product Intent Constructs, intent resources are designed to narrow down the available options when defining a JDF job.

All intent resources share a set of sub-elements that allow a Request for Quote to describe a range of acceptable values for various aspects of the product. These elements, taken together, allow an administrator to provide a specific value for the quote. Section 7.1.1, below, describes these elements.

Each of the following sections begins with a brief narrative description of the resource. Following that is a list containing details about the properties of the resource, as shown below. The first item in the list provides the class of the resource, which, in this section is always *Intent*. For more information on resource class, see Table 3.9. A template of this list is shown below.

After the list describing the resource properties, each section contains tables that outline the structure of each resource and, when applicable, the abstract or sub-element information that pertains to the resource structure. The first column contains the name of the attribute or element. In some cases, a resource will contain an element with more than one value associated with it. If this is the case, the element name is listed as often as it appears, and a term in parentheses that identifies the kind of element is included in the column. A template of these tables is also provided below.

Resource Properties Template

Resource class:	Defines the resource class or resource element.
Resource referenced by:	List of parent resources that contain elements of this type. Only valid for elements.
Partition:	List of valid partitioning boundaries: <i>PartVersion</i> , <i>Separation</i> , <i>Side</i> , <i>SheetName</i> , <i>SignatureName</i> , <i>TileID</i> . If a partition is specified, the resource may contain a nested elements of its own type.
Input of processes:	List of node types that use the resource as an input resource.
Output of processes:	List of node types that create the resource as an output resource

Resource Structure Template

Name	Data Type	Description
Name of attribute or element	data type of attribute or element	Usage of the structure.

7.1.1 an Resource Sub-elements^[DH3]

Intent resources contain sub-elements that allow spans of values to be specified. These sub-elements also provide mechanisms to select a set of values from the range and map them to a set of quotes. These sub-

elements are called **span elements**. Depending on the data type of the values to be recorded, different abstract span elements exist. These elements are listed in the following table in the column entitled “Span Element Types.” Furthermore, each span element contains further attributes or sub-elements. The contents shared by all span elements are listed in the section 7.1.1.1 Structure of Abstract Span Elements, below, and the contents particular to each span element type are described in the sections that follow.

Span Element Types	Data Type	Description
IntegerSpan	element	Describes a numerical range of integer values.
NameSpan	element	Describes a set of NMTOKEN values.
NumberSpan	element	Describes a numerical range of values.
OptionSpan	element	Describes an intent in which the principal information is that a specific option is requested.
StringSpan	element	Describes a set of string values.
TimeSpan	element	Describes a set of timeInstant values.

7.1.1.1 Structure of Abstract Span Elements

Abstract span elements of intent resources have a common set of attributes and elements that define the priority, data type, and requested identity of the element. These attributes are described in the following table.

Name	Data Type	Description
<i>Data Type ?</i>	enumeration	Describes the data type of the span element within an intent resource. Possible values are: <i>IntegerSpan</i> <i>NumberSpan</i> <i>NameSpan</i> <i>StringSpan</i> <i>TimeSpan</i> <i>OptionSpan</i> <i>ColorSpan</i>
<i>Priority ?</i>	enumeration	Indicates the importance of the specific intent. The following values have prescribed meanings: <i>none</i> – Default value. <i>suggested</i> – The customer will accept a value of <i>Actual</i> that is different than the value of <i>Preferred</i> or outside of <i>Range</i> . <i>required</i> – <i>Actual</i> must be equal to <i>Preferred</i> or within <i>Range</i> . Note that the attribute <i>Preferred</i> is available in the data types which inherit from this abstract type. These are the span element types <i>IntegerSpan</i> , <i>NameSpan</i> , <i>NumberSpan</i> , <i>StringSpan</i> , and <i>TimeSpan</i> , all of which are described in the following sections. In the case of the <i>OptionSpan</i> type, the <i>Priority</i> attribute refers to the <i>Detail</i> attribute.

<i>QuoteAll</i> ?	boolean	If <i>true</i> , the customer wants a quote for all specified options in <i>Range</i> . Default = <i>false</i> , which means that the customer allows the supplier to select a set of values from <i>Range</i> .
QuoteSelection *	element	Abstract sub-element that serves as a placeholder for any selection element. Possible selection elements are: IntegerSelection NameSelection NumberSelection OptionSelection StringSelection TimeSelection.

7.1.1.2 Structure of the Span-Element Type IntegerSpan

This sub-element is used to describe ranges of integer values. The span-element type *IntegerSpan* inherits from the abstract span-element described in section 7.1.1.1 *Structure of Abstract Span Elements*.

Name	Data Type	Description
<i>Preferred</i> ?	integer	Provides a value specified by the person submitting the request, indicating what that person prefers. The value of <i>Preferred</i> must fall within the range of values specified in <i>Range</i> .
<i>Range</i> ?	IntegerRange-List	Provides either a set of discreet values, a range of values, or a combination of the two that comprise all allowed values for.
IntegerSelection *	element	Indicates a set of proposed values that a responder has chosen to use to build the product. Depending on the value of <i>Priority</i> , the <i>Actual</i> attribute of an <i>IntegerSelection</i> may not be within <i>Range</i> .

Structure of the IntegerSelection Sub-element

The *IntegerSelection* element inherits from the *QuoteSelection* element.

Name	Data Type	Description
<i>Actual</i>	integer	The value associated with the quote index.
<i>Index</i> ?	IntegerRange-List	Defines the list of options that the <i>Actual</i> attribute of this Selection is valid for. Default is “0~1” which specifies that this Quote Selection is valid for all quotes.

7.1.1.3 Structure of the Span-Element Type NameSpan

This sub-element is used to describe name ranges. The span-element type *NameSpan* inherits from the abstract span-element described in section 7.1.1.1 *Structure of Abstract Span Elements*.

Name	Data Type	Description
<i>Range</i> ?	NMTOKENS	Provides a set of discreet values.
<i>Preferred</i> ?	NMTOKEN	Provides a value specified by the person submitting the request, indicating what that person prefers. <i>Preferred</i> must fall within the range of values specified in <i>Range</i> .
NameSelection *	element	Indicates a set of proposed values that a responder has chosen to use to build the product. Depending on the value of <i>Priority</i> , the <i>Actual</i> attribute of an NameSelection may not be within <i>Range</i> .

Structure of the NameSelection Sub-element

The NameSelection element inherits from the QuoteSelection element.

Name	Data Type	Description
<i>Actual</i>	string	The value associated with the quote index.
<i>Index</i> ?	IntegerRange-List	Defines the list of options that the <i>Actual</i> attribute of this Selection is valid for. Default is "0~1" which specifies that this Quote Selection is valid for all quotes.

Specifying New Values in a NameSpan Sub-element

NameSpan attributes will generally define an open list of pre-defined values. If a value that is not included in the list shall be specified, a comment that defines that value can be included in the NameSpan using the new name as a Name attribute of the comment, as demonstrated in the following example:

```
<HoleType DataType="NameSpan" Range="36Hole 42Hole">
<Comment Name="36Hole">6 equidistant holes on each side of a hexagonal
piece of paper </Comment>
<Comment Name="42Hole">7 equidistant holes on each side of a hexagonal
piece of paper </Comment>
</HoleType>
```

7.1.1.4 Structure of the Span-Element Type NumberSpan

This sub-element is used to describe a numerical range of values. The span-element type NumberSpan inherits from the abstract span-element described in section 7.1.1.1 Structure of Abstract Span Elements.

Name	Data Type	Description
<i>Preferred</i> ?	number	Provides a value specified by the person submitting the request, indicating what that person prefers. <i>Preferred</i> must fall within the range of values specified in <i>Range</i> .
<i>Range</i> ?	NumberRange-List	Provides either a set of discreet values, a range of values, or a combination of the two.
NumberSelection *	element	Indicates a set of proposed values that a responder has chosen to use to build the product. Depending on the value of <i>Priority</i> , the <i>Actual</i> attribute of an NumberSelection may not be within <i>Range</i> .

Structure of the NumberSelection Sub-element

The NumberSelection element inherits from the QuoteSelection element.

Name	Data Type	Description
<i>Index</i> ?	IntegerRange-List	Defines the list of options that the <i>Actual</i> attribute of this Selection is valid for. Default is “0~-1” which specifies that this Quote Selection is valid for all quotes.
<i>Actual</i>	double	The value associated with the quote index.

7.1.1.5 Structure of the Span-Element Type OptionSpan

The span-element type OptionSpan inherits from the abstract span-element described in section 7.1.1.1 Structure of Abstract Span Elements.

Name	Data Type	Description
<i>Detail</i> ?	string	<i>Detail</i> provides information about the option.
<i>OptionSelection</i> *	element	Indicates a set of proposed values that a responder has chosen to use to build the product. Depending on the value of <i>Priority</i> , the <i>Actual</i> attribute of an OptionSelection may not be within <i>Range</i> .

Structure of the OptionSelection Sub-element

The OptionSelection element inherits from the QuoteSelection element.

Name	Data Type	Description
<i>Actual</i>	string	The value associated with the quote index.
<i>Index</i> ?	IntegerRange-List	Defines the list of options for which the <i>Actual</i> attribute of this Selection is valid. Default is “0~-1” which specifies that this Quote Selection is valid for all quotes.

7.1.1.6 Structure of the Span-Element Type StringSpan

This sub-element is used to describe string ranges. The span-element type StringSpan inherits from the abstract span-element described in section 7.1.1.1 Structure of Abstract Span Elements.

Name	Data Type	Description
<i>Preferred</i> ?	telem	Provides a value specified by the person submitting the request, indicating what that person prefers. <i>Preferred</i> must fall within the range of values specified in <i>Range</i> .
<i>Range</i> *	telem	Range provides a set of discreet string values.
<i>StringSelection</i> *	element	Indicates a set of proposed values that a responder has chosen to use to build the product. Depending on the value of <i>Priority</i> , the value of the <i>Actual</i> attribute of an StringSelection element may not be within the range of values specified in the <i>Range</i> attribute.

Structure of the StringSelection Sub-element

The StringSelection element inherits from the QuoteSelection element.

Name	Data Type	Description
<i>Actual</i>	string	The value associated with the quote index.
<i>Index ?</i>	IntegerRange-List	Defines the list of options that the <i>Actual</i> attribute of this Selection is valid for. Default is “0~1” which specifies that this Quote Selection is valid for all quotes.

7.1.1.7 Structure of the TimeSpan Sub-element

Name	Data Type	Description
<i>Range ?</i>	TimeRange	Range provides a valid time period.
<i>Preferred ?</i>	timeInstant	Provides a value specified by the person submitting the request, indicating what that person prefers. <i>Preferred</i> must fall within the range of values specified in <i>Range</i> .
TimeSelection *	element	Indicates a set of proposed values that a responder has chosen to use to build the product. Depending on the value of <i>Priority</i> , the value of the <i>Actual</i> attribute of an TimeSelection element may not be within the range of values specified in the <i>Range</i> attribute.

Structure of the TimeSelection Sub-element

The StringSelection element inherits from the QuoteSelection element.

Name	Data Type	Description
<i>Actual</i>	timeInstant	The value associated with the quote index.
<i>Index ?</i>	IntegerRange-List	Defines the list of options that the <i>Actual</i> attribute of this Selection is valid for. Default is “0~1” which specifies that this Quote Selection is valid for all quotes.

7.1.2 Named Span resources

The datatypes defined in this section are all instances of NameSpan with a restricted list of allowed NMTOKEN values. More values may be created.

NamedColorSpan

This data type provides a definition of named colors. It is not sufficient for process color definition, but rather serves to define the colors of preprocessed products such as wire-o binders and cover leaflets.

wed values are any entry defined in Table A.1 Mapping of named colors to sRGB colors:
NamedColor.[DH4]


7.1.3 ArtDeliveryIntent

This resource specifies the prepress art delivery intent for a JDF job and maps the items to the appropriate reader pages and separations.

Resource Properties



Resource class:	Intent
Resource referenced by:	-
Partition:	-
Input of processes:	-
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>Method</i> ?	NameSpan	 tifies a required delivery method, such as <i>eMail</i> , <i>PressMail</i> or <i>InterofficeMail</i> . [RP5]
ArtDelivery *	element	Individual delivery
Company ?	element	Address and further information of the addressee.

Structure of ArtDeliveryIntent Elements

Each ArtDelivery element defines a set of existing products that are required to create the specified product.

Name	Data Type	Description
<i>ArtDeliveryType</i>	NameSpan	Type of artwork supplied. Possible values include:  talMedia <i>DigitalNetwork</i> [RP6] <i>ImposedFilm</i> <i>LooseFilm</i> <i>OriginalArt</i> <i>ProofsOfScans</i> <i>None</i> [RP7]
<i>Method</i> ?	NameSpan	 tifies a required delivery method, such as <i>eMail</i> , <i>PressMail</i> or <i>InterofficeMail</i> . [RP8]
<i>PageList</i> ?	IntegerRangeList	Set of Pages that are filled by this ArtDelivery.
<i>rRef</i> ?	IDREF	Reference to the resource to which this ArtDelivery refers. This resource will typically be an ExposedMedia (film, plate or hardcopy proof), Component (complete prefabricated product) or RunList (digital delivery) resource. If <i>rRef</i> is not specified, no details except the <i>ArtDeliveryType</i> are known.
Company ?	element	Address and further information about the addressee.
Part *	element	ArtDelivery may contain any partitioning or amount attributes valid for a ResourceLink in a ResourceLinkPool. This is only valid if rRef is specified.

7.1.4 BindingIntent

This resource specifies the binding intent for a JDF job using information that identifies the type of binding required and which side is to be bound.


Resource Properties

Resource class:	Intent
Resource referenced by:	-
Partition:	-
Input of processes:	-
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>BindingType</i>	NameSpan	<p>This resource describes the desired binding for the job. Possible values are:</p> <p><i>Adhesive</i> – This type of binding can be handled with the AdhesiveBinding process. It includes perfect binding.</p> <p><i>AdhesiveTaped</i> – This type of binding can be handled with the AdhesiveBinding process.</p> <p><i>ChannelBinding</i> – This type of binding can be handled with the ChannelBinding process.</p> <p><i>CoilBinding</i> – This type of binding can be handled with the CoilBinding process.</p> <p><i>PlasticComb</i> – This type of binding can be handled with the PlasticCombBinding process.</p> <p><i>Ring</i> – This type of binding can be handled with the RingBinding process.</p> <p><i>SaddleStitch</i> – This type of binding can be handled with the SaddleStitching process.</p> <p><i>Sewn</i> – This type of binding can be handled with the ThreadSewing process.</p> <p><i>SideSewn</i> – This type of binding can be handled with the SideSewing process.</p> <p><i>SideStitch</i> – This type of binding can be handled with the Stitching process.</p> <p><i>TheadSealing</i> – This type of binding can be handled with the Folding process.</p> <p><i>VeloBind</i> – This type of binding can be handled with the VeloBinding process.</p> <p><i>WireComb</i> – This type of binding can be handled with the WireCombBinding process.</p>
<i>BindingColor?</i>	ColorSpan	Defines the color of the binding spine's material.
<i>BindingSide</i>	NameSpan	<p>Indicates which side should be bound. Possible values are:</p> <p><i>Top</i></p>

		<i>Button</i>
		<i>Right</i>
		<i>Left</i>
		Each of these values is intended to identify an edge of the job. These edges are defined relative to the orientation of the first page in the job with content on it.
<i>CastingMaterial ?</i>	NameSpan	Casting material of the thread in ThreadSewing . Possible values are: <i>Cotton</i> <i>Nylon</i> <i>Polyester</i>
<i>ChannelCover ?</i>	OptionSpan	If true the clamp used in ChannelBinding includes a pre-assembled cover. Default = <i>false</i>
<i>CoilMaterial ?</i>	NameSpan	The following coil materials are available for CoilBinding : <i>LaqueredSteel</i> <i>NylonCoatedSteel</i> <i>PVC</i> <i>TinnedSteel</i> <i>ZincsSteel</i>
<i>CoreMaterial ?</i>	NameSpan	Core material of ThreadSewing . This attribute must be used to define the thread material if there is no casting. Possible values are: <i>Cotton</i> <i>Nylon</i> <i>Polyester</i>
<i>EndSheetGlue ?</i>	NameSpan	Glue type used to define EndSheetGluing procedures. Possible values are: <i>ColdGlue</i> <i>Hotmelt</i> <i>PUR</i> – Polyurethane <i>None</i>
<i>Milling ?</i>	OptionSpan	Milling features for AdhesiveBinding .
<i>Notching ?</i>	OptionSpan	Notching features for AdhesiveBinding .
<i>PlasticCombType ?</i>	NameSpan	The distance between the “teeth” in PlasticCombBinding and the distance between the holes of the pre-punched sheets must be the same. The following standards exist: <i>Euro</i> (Distance = 12 mm; Holes = 7 mm x 3 mm) <i>USA1</i> (Distance = 14.28 mm; Holes = 8 mm x 3 mm)
<i>RingMechanic ?</i>	OptionIntent	The ring binder used includes a lever for opening and closing

		closing. Default = <i>false</i>
<i>RingSystem ?</i>	NameSpan	The following RingBinding systems are used: <i>2Hole</i> – in Europe <i>3Hole</i> – in North America <i>4Hole</i> – in Europe
<i>Scoring ?</i>	NameSpan	Scoring option for AdhesiveBinding : <i>TwiceScored</i> <i>QuadScored</i> <i>None</i>
<i>Sealing ?</i>	OptionSpan	If <i>true</i> , thermo-sealing is required in ThreadSewing .
<i>SpineGlue ?</i>	NameSpan	Glue type used to define AdhesiveBinding procedures. Possible values are: <i>ColdGlue</i> <i>Hotmelt</i> <i>PUR</i> – Polyurethane <i>None</i>
<i>StapleShape ?</i>	NameSpan	Shape of staples for SaddleStitching and Stitching processes. Possible values are:  <i>Down</i> <i>Overlap</i> <i>Butted</i> <i>ClinchOut</i> <i>Eyelet</i>
<i>StapleOpening ?</i>	NameSpan	These values are displayed in Figure 7.14. Defines the side where the staple is open. One of <i>inside</i> (the default) or <i>outside</i> .
<i>ThreadSewingGlue ?</i>	NameSpan	Glue type used to define ThreadSewing procedures. Possible values are: <i>ColdGlue</i> <i>Hotmelt</i> <i>PUR</i> – Polyurethane <i>None</i>
<i>WireCombMaterial ?</i>	NameSpan	The material used for forming the WireCombBinding : <i>LaqueredSteel</i> <i>TinnedSteel</i> <i>ZincsSteel</i>
<i>WireCombShape ?</i>	NameSpan	The shape of the WireCombBinding :

single: each “tooth” is made with one wire

twin: the shape of each “tooth” is made with a double wire



7.1.5 ColorIntent



This resource specifies the type of ink to be used. Typically, the parameters consist of a manufacturer name and additional identifying information. The resource also specifies any coatings and colors to be used, including the process color model and any spot colors.

Resource Properties

Resource class:	Intent
Partition:	<i>Side, Sheet, Signature</i>
Resource referenced by:	-
Input of processes:	Any product node
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>Coatings</i>	NameSpan	Material usually applied on press as a protective or gloss enhancing layer over ink. Possible values include: <i>Varnish</i> <i>Laminant</i> [RP9]
<i>ColorsUsed</i>	NameSpan	 Possible values are:[DH10] <i>CMYK</i> <i>6ColorProcess</i> <i>GreyScale</i> <i>DuoTone</i> <i>ProcessPlusSpot</i> <i>SpotColors</i>
 <i>ColorModel ?</i>	NameSpan	The color space definition. Possible values include: SNAP GRACOL SWAP CieLAB[RP11]
<i>Coverage ?</i>	NameSpan	Describes the relative amount of the surface area that is covered. Possible values are: <i>Light</i> <i>Medium</i> <i>Heavy</i> <i>Unknown</i>
<i>InkManufacturer ?</i>	NameSpan	Name of the manufacturer of the ink requested.

 <i>Family</i> ?[RP12]	NameSpan	A name that the manufacturer uses to describe the family of inks to be used.
 <i>arationSpec</i> [RP13]*	element	Array of color names that are requested.

7.1.6 DeliveryIntent

Summarizes the options that describe pickup or delivery time and location options of a job. It also defines the number of copies that are requested for a specific job or delivery.

Resource Properties

Resource class:	Intent
Resource referenced by:	-
Partition:	-
Input of processes:	Any product node
Output of processes:	-


Resource Structure

Name	Data Type	Description
<i>Earliest</i> ?	TimeSpan	Specifies the earliest time after which the delivery may be made.
<i>Method</i> ?	NameSpan	Identifies a required delivery method, such as <i>eMail</i> , <i>ExpressMail</i> or <i>InterofficeMail</i> .
<i>Pickup</i> ?	boolean	Specifies whether the delivery brings or picks up the merchandise. Default = false, which means that the drop is delivered.
<i>Required</i> ?	TimeSpan	Specifies the time by which the delivery must be made.
Company	element	Address and further information of the addressee.
DroIntent +	element	Includes all locations where the product will be delivered.

Structure of DeliveryIntent Elements

DroIntent

This element contains information about the intended individual drop of a delivery. Attributes that are specified in a DroIntent element overwrite those that are specified in their parent DeliveryIntent element.

Name	Data Type	Description
<i>Earliest</i> ?	TimeSpan	Specifies the earliest time after which the delivery may be made.
<i>Method</i> ?	NameSpan	 tifies a required delivery method, such as <i>ExpressMail</i> or <i>rofficeMail</i> . [RP14]
<i>Pickup</i> ?	boolean	If <i>true</i> , the merchandise is picked up. If <i>false</i> , the merchandise is delivered. Default = <i>false</i>
<i>Required</i> ?	TimeSpan	Specifies the time by which the delivery must be made.
Company	element	Address and further information of the addressee.

PackageIntent *	element	A DroplIntent may consist of multiple products, which are represented by their respective Component resources. Each PackageIntent describes a number of individual resources that is part of this DroplIntent .
-----------------	---------	---

Structure of the PackageIntent Sub-element

Name	Data Type	Description
<i>Amount</i> ?	IntegerSpan	Specifies the number of resources ordered. If not specified, defaults to the total amount of the resource that is referenced by <i>rRef</i> .
<i>rRef</i>	IDREF	Reference to the resource that this PackageIntent contains.
<i>Unit</i> ?	string	Unit of measurement for the <i>Amount</i> specified in the <i>ComponentLink</i> attribute. Defaults to the value of <i>Unit</i> defined in the resource linked by the <i>rRef</i> attribute.
Part *	element	PackageIntent may contain any partitioning or amount attributes valid for a ResourceLink in a ResourceLinkPool element.

7.1.7 FoldingIntent

This resource specifies the fold intent for a JDF job using information that identifies the number of folds, the height and width of the folds, and the folding catalog number. Note that the folding catalog is in JDF Spec 3.0 section 7.35 and that the number of folds and the folding catalog are related.

Resource Properties

Resource class:	Intent
Resource referenced by:	-
Partition:	-
Input of processes:	-
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>FoldingCatalog</i>	NameSpan	Description of the folding scheme as specified in the FoldingParams folding catalog attribute. (See JDF Folding Catalog descriptions in Figure 7.10 FoldCatalog part 1 and Figure 7.11 FoldCatalog part 2). Note that the folding scheme in this context refers to the folding of the finished product as seen after cutting, not the folding of the flat as seen in production.

7.1.8 HoleMakingIntent

This resource specifies the holemaking intent for a JDF job, using information that identifies the type of HoleMaking operation or alternatively, an explicit list of holes.

Resource Properties

Resource class:	Intent
Resource referenced by:	-
Partition:	-
Input of processes:	-
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>HoleType</i>	NameSpan	Type of holemaking operation. Possible values include: <i>2Hole</i> – Holes for the RingBinding process with the 2Hole RingSystem used in Europe. <i>3Hole</i> – Holes for the RingBinding process with the 3Hole RingSystem used in North America. <i>4Hole</i> – Holes for the RingBinding process with the 4Hole RingSystem used in Europe. <i>PlasticCombEuro</i> – Holes for PlasticCombBinding with the (Distance = 12 mm; Holes = 7 mm x 3 mm) <i>PlasticCombUSA1</i> – Holes for PlasticCombBinding with the (Distance = 14.28 mm; Holes = 8 mm x 3 mm) <i>Explicit</i> – Holes are defined in an array of Hole elements.
HoleIntent *	element	Array of all Hole elements. Used when <i>HoleType</i> = <i>Explicit</i> . This is common in CoilBinding , VeloBinding , WireCombBinding .

Structure of HoleIntent Sub-element

Name	Data Type	Description
<i>Center</i>	XYPair	Position of the center of the hole relative to the Component coordinate system. For more information, see section 6.5.2.
<i>Extent</i>	XYPair	Size of the hole in pt. If <i>Shape</i> is <i>round</i> , only the first entry of <i>Extent</i> is evaluated and defines the hole diameter.
<i>Shape</i>	NameSpan	Shape of the hole. Possible values are: <i>round</i> <i>rectangular</i>






7.1.9 InsertingIntent

This resource specifies the inserting for a JDF job, using information that identifies ...

Resource Properties

Resource class: Intent
 Resource referenced by: -
 Partition: -
 Input of processes: Any product node
 Output of processes: -

Resource Structure

Name	Data Type	Description
 <i>TopNumbers</i> [RP15]	IntegerRange-List	(14~15 23~24)
<i>GlueType</i>	NameSpan	 Possible values are:[DH16] <i>ColdGlue</i> <i>Hotmelt</i> <i>PUR</i> <i>None</i>
 <i>Part</i>	list	[RP17]
 <i>Position</i>	enumeration	Possible values are: <i>front</i> <i>overFoldLeft</i> [RP18]
<i>Method</i>	NameSpan	Possible values are: <i>bind-in</i> <i>blow-in</i>
 <i>Printivity</i> [RP19]		
<i>SheetOffset</i>	XYPair	
<i>StartPosition</i>	XYPair	

7.1.10 LaminatingIntent


This resource specifies the finish laminating intent for a JDF job using information that identifies whether or not the product is laminated and, if desired, the temperature and thickness of the laminant.

Resource Properties

Resource class: Intent
 Resource referenced by: -
 Partition: -
 Input of processes: -
 Output of processes: -

Resource Structure

Name	Data Type	Description
<i>Laminated</i>	OptionSpan	If <i>true</i> , the product is laminated.

		Default = <i>false</i>
<i>Temperature</i>	NameSpan	Temperature used in the lamination process. Possible values are: <i>Hot</i> <i>Cold</i>
 <i>Thickness</i> [MS20]	NumberSpan	Thickness of the laminant.

7.1.11 MediaIntent

This resource describes the media to be used for the product component. In some cases, the exact identity of the medium is known, while in other cases, the characteristics are described and a particular stock is matched to those characteristics.

Resource Properties

Resource class:	Parameter
Partition:	-
Resource referenced by:	IDPrintingParams
Input of processes:	Any product node, <i>DigitalPrinting</i>
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>BuyerSupplied ?</i>	OptionSpan	Indicates whether the customer will supply the media.
<i>Coatings</i>	NMTOKENs	What pre-process coating has been applied to the front and back of the media. Either 1 or 2 values may be provided. If 1 is provided, it is used for the front of the media. If 2 are provided, the first is used for the front, and the second is used for the back. Possible values are: <i>none</i> <i>any</i> <i>glossy</i> <i>high-gloss</i> <i>semi-gloss</i> <i>satın</i> <i>matte</i>
<i>Finish ?</i>	NameSpan	The intended finish of the media.
<i>Grade ?</i>	NameSpan	The intended grade of the media.
<i>Opacity ?</i>	enumeration	The opacity of the media. Possible values are: <i>opaque</i> – the media is opaque <i>transparent</i> – the media is transparent
<i>Recycled ?</i>	boolean	If <i>true</i> , recycled media is requested.
<i>Size ?</i>	XYPair	Specifies the size of the media in inches.

<i>StockType</i> ?	NameSpan	Strings describing the available stock.
<i>Weight</i> ?	NumberSpan	The intended weight of the media, measured in (g/m ²).

7.1.12 Numbering Intent


*** Tentative Intent resource definition (9/19) ***

NOTE: this intent not ready for prime time ...

Resource Properties

Resource class:	Intent
Resource referenced by:	-
Partition:	-
Input of processes:	-
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>Generic</i>	enumeration	 Possible values are:[DH21] <i>inkJet</i> <i>Crash</i>
<i>Numbers</i>	list	
<i>Orientation</i>		
<i>Position</i>	XYPair	
<i>StartValue</i>		
<i>Step</i>		

7.1.13 PackagingIntent_[RP22]

This resource specifies the packaging intent for a JDF job, using information that identifies the type of package, the wrapping used, and the shape of the package.

Resource Properties

Resource class:	Intent
Resource referenced by:	-
Partition:	-
Input of processes:	-
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>Shape</i> ?	shape	Describes the length, width and height of the package in pts. For example, 288 544 1012
<i>Type</i> ?	NameSpan	Type of package. Possible values include:



STE[RP23]

None – Default value.

Wrapping ?

NameSpan

Type of wrapping technology used. Possible values are:

Paperband

Rubberband

ShrinkWrap

None – Default value.

7.1.14 PocketingIntent

NOTE: this intent not ready for prime time ...

Resource Properties

Resource class: Intent
 Resource referenced by: -
 Partition: -
 Input of processes: -
 Output of processes: -

Resource Structure

Name	Data Type	Description[DH24]
<i>Capacity</i>		
<i>Generic</i>	enumeration	Possible values are: <i>Folded</i> <i>Glued</i>
<i>Location</i>	enumeration	Possible values are: <i>left</i> <i>right</i> <i>both</i>
<i>Pockets</i>	list	
<i>Size</i>	XYPair	

7.1.15 ProofingIntent



This resource specifies the prepress proofing intent for a JDF job, using information that identifies the type, quality, brand name and overlay of the proof.

Resource Properties

Resource class: Intent
 Resource referenced by: -

Partition: -
 Input of processes: **Proofing**
 Output of processes: -

Resource Structure

Name	Data Type	Description
<i>BrandName</i> ?	StringSpan	Brand name of the proof, such as “Iris”
<i>NumberOfCopies</i> ?	IntegerSpan	Specifies how many copies of this proofs are required
<i>ProofingTarget</i> ?	uri	Identifies a remote target for the proof output.
 <i>ProofingType</i>	NameSpan	Technology used for making the proof. Possible values are: <i>DyeSub</i> <i>InkJet</i> <i>Laser</i> <i>RemoteDigital</i> <i>SoftProof</i> <i>PressProof</i> <i>Unspecified</i> [RP25]
<i>Quality</i> ?	NameSpan	Specification level required for the proof. Possible values are:  <i>cColor</i> = <i>conceptual</i> <i>ContractColor</i> = <i>half-tone, contone</i> (w / wo screening) <i>Imposition</i> = <i>bw?</i> [RP26]

Notes:

1. Proof dates will be in the business object (i.e. *RFQ*, *Quote*, *Order*).
2. *Film-based* will not be used as a proofing type since it specifies a manufacturing process and is not commonly referred to by buyers.
3. JDF “ProofIntent” (*none*, *half-tone*, *contone*, *conceptual*) is part of JDF *ExposedMedia* Object, and will not be explicitly used in *ProofingIntent*.



7.1.16 *ScanningIntent*[RP27]

This resource specifies the prepress scanning intent for a JDF job using information that identifies the type and resolution of the scan, size of the supplied media, size and type of the output media, and screen frequency needed for the print job to be used for a digital scan.

Resource Properties

Resource class: Intent
 Resource referenced by: -
 Partition: -
 Input of processes: **Scanning**
 Output of processes: -

Resource Structure

Name	Data Type	Description
<i>BitDepth</i>	IntegerSpan	Bit depth of a one-color separation.
<i>InputBox ?</i>	rectangle	Rectangle that describes the image section to be scanned, in points. The origin of the coordinate system is the lower left corner of the physical item to be scanned.
<i>Magnification ?</i>	XYPair	Size of the output/size of the input for each dimension. Defaults to 1.0.
<i>OutputResolution</i>	XYPair	X and Y resolution of the output bitmap (in DPI).
<i>OutputSize ?</i>	XYPair	X-,Y-dimension of the intended output image (in pt).
<i>OutputColorSpace ?</i>	NumberSpan	 Possible values are:[DH28] <i>RGB</i>  <i>YK[RP29]</i> <i>CIELab</i> <i>GreyScale</i>
<i>OutputType ?</i>	enumeration	Type of item resulting from the scan, such as a page for display on a website. Possible values are: <i>Digital</i> <i>Film</i>

7.1.17 ScreeningIntent


This resource specifies the screening intent for a JDF job using information that identifies the family, frequency, and spot function of the job that will be screened.

Resource Properties

Resource class:	Intent
Resource referenced by:	-
Partition:	-
Input of processes:	Screening
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>Frequency ?</i>	NumberSpan	Specifies the line frequency of the screen when AM screening is used.
<i>ScreeningType ?</i>	NameSpan	General type of the screen. Possible values are: <i>AM</i> <i>FM</i> <i>Adaptive</i>
<i>ScreeningFamily ?</i>	StringSpan	Vendor specific screening family name. Possible values include: <i>Rational Tangent</i> <i>Adobe Accurate</i>

		<i>Agfa Balanced</i>
		<i>Soft-IS</i>
		<i>ErrorDiffusion</i>
<i>SpotFunction ?</i>	StringSpan	Specifies the spot function of the screen when AM screening is used.
		 list of spot functions is the same as that defined in <i>ScreeningParams</i> [DH30]

7.1.18 ShapeIntent

This resource specifies form and line cutting for a JDF job. The cutting processes are applied for producing special shapes like an envelope-window or a heart-shaped beer mat. Information that identifies the type and shape of cuts can be described. The cutting process(es) can be performed using tools such as hollow form punching, perforating or die-cutting equipment.



Resource Properties

Resource class:	Intent
Resource referenced by:	-
Partition:	-
Input of processes:	-
Output of processes:	-

Resource Structure

Name	Data Type	Description
ShapeCut *	element	Array of all ShapeCut elements. Used when each shape is exactly specified.

Structure of Shape Sub-element

Name	Data Type	Description
<i>CutBox ?</i>	rectangle	Specification of a rectangular window.
<i>CutOut ?</i>	boolean	If <i>true</i> , the inside of a specified shape shall be removed. If <i>false</i> , the outside of a specified shape shall be removed. An example of an inside shape is a window, while an example of an outside shape is a shaped greeting card.
<i>CutPath ?</i>	path	Specification of a complex path. This may be an open path in the case of a single line.
<i>Material ?</i>	StringSpan	Transparent material that fills a shape, such as a window, that was cut out when <i>CutOut = true</i> .
<i>CutType ?</i>	NameSpan	Type of cut or perforation used. Possible values are: <i>Cut</i> <i>Interruptedandlers</i>  <i>rf</i> [MS31]  <i>roPerforation</i> [MS32]
<i>Pages ?</i>	IntegerRange-	List of pages to which this shape shall be applied.

	List	
<i>ShapeType</i>	NameSpan	Describes any precision cutting other than hole making. Possible values are: <i>Rectangular</i> <i>Round</i> <i>Path</i>
<i>TeethPer-Dimension ?</i>	NumberSpan	Number of teeth in a given perforation extent in teeth/point.

7.1.19 SizeIntent

This resource records the size of the finished pages for the product component. It does not, however, specify the size of any intermediate results, such as press sheets.

Resource Properties

Resource class:	Intent
Partition:	-
Resource referenced by:	-
Input of processes:	Any Product Node
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>Height</i>	NumberSpan	Specifies the height of the product component.
<i>Pages ?</i>	IntegerSpan	Specifies the number of pages of the product component.
<i>Type ?</i>	enumeration	Specifies whether the product component referred to is flat or finished. Possible values are: <i>Finished</i> = Default value <i>Flat</i>
<i>Width</i>	NumberSpan	Specifies the width of the product component.

7.1.20 StampingIntent

NOTE: this intent not ready for prime time ...

Resource Properties

Resource class:	Intent
Resource referenced by:	-
Partition:	-
Input of processes:	-
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>Area</i>	XYPair	
<i>Dies</i>	list	
<i>Generic</i>	enumeration	Possible values are: <i>embossed</i> <i>debossed</i> <i>foil</i>

7.2 Process Resources

The rest of the resources described in this chapter are what are known as process resources. This means that they serve as necessary components in each of the JDF processes.

Section 7.2.1 describes the template for all of the sections that follow. Then every resource already defined for JDF is chronicled, in alphabetical order, below.

7.2.1 Process Resource Template

Each of the following sections begins with a brief narrative description of the resource. Following that is a list containing details about the properties of the resource, as shown below. The first item in the list provides the class of the resource. As was described in section 3.2 *Common Node Types*, all resources are derived from one of the following eight superclasses: *Intent*, *Parameter*, *Implementation*, *Consumable*, *Quantity*, *Handling*, *PlaceHolder* and *Selector*. All resources inherit additional contents (which may be attributes or elements) from their respective superclasses, and those attributes and elements are not repeated in this section. Thus those attributes associated with a resource of class *Parameter*, for example, can be found in Table 3.9.

If the resource described is not an atomic resource, the resource class item in the resource properties list defines an element known as a resource element, rather than a class. Resource elements are listed in separate sections if they may be referenced by more than one resource. For an example, see the resource element **SeparationSpec**. If the resource may not be referenced by multiple resources, it is described inside the resource section of the resource to which it belongs. For an example, see the abstract **FoldOperation** element of the **FoldingParams** resource. The resource class of an atomic resource also defines the superclasses from which the resource inherits additional contents. The **Consumable**, **Quantity**, and **Handling** resource elements inherit from the **PhysicalResource** element, which in turn inherits from the **Resource** element. **Parameter** and **Implementation** resource elements inherit from the **Resource** element directly. Non-atomic resources—that is, resource sub-elements—do not inherit contents from resource superclasses.

After the list describing the resource properties, each section contains tables that outline the structure of each resource and, when applicable, the abstract or sub-element information that pertains to the resource structure. The first column contains the name of the attribute or element. In some cases, a resource will contain an element with more than one value associated with it. If this is the case, the element name is listed as often as it appears, and a term in parentheses that identifies the kind of element is included in the column. For an example, see section 7.2.36 *EndSheetGluingParams*.

An example of the tables in this section is provided below.

Resource Properties Template

Resource class: Defines the resource class or resource element.

Resource referenced by:	List of parent resources that contain elements of this type. Only valid for elements.
Partition:	List of valid partitioning boundaries: <i>PartVersion, Separation, Side, SheetName, SignatureName, TileID</i> . If a partition is specified, the resource may contain a nested elements of its own type.
Input of processes:	List of node types that use the resource as an input resource.
Output of processes:	List of node types that create the resource as an output resource

Resource Structure Template

Name	Data Type	Description
Name of attribute or element	data type of attribute or element	Usage of the structure.

7.2.2 Address

Definition of an address. The structure is derived from the vCard format and therefore is comprised of all address sub-types (ADR:) of the delivery address of the vCard format. The corresponding XML types of the vCard are quoted in the table.

Resource Properties

Resource class:	Element
Resource referenced by:	Contact
Partition:	-
Input of processes:	-
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>City</i> ?	string	City or locality of address (vCard: ADR:locality).
<i>Country</i> ?	string	Country of address (vCard: ADR:country).
<i>CountryCode</i> ?	string	Country of address. This value conforms to the ISO 3166 standard in which countries are represented as 2-character codes.
<i>PostBox</i> ?	string	Post office address (vCard: ADR:pobox. For example: P.O. Box 101).
<i>PostalCode</i> ?	string	Zip code or postal code of address (vCard: ADR:pcode).
<i>Region</i> ?	string	State or province (vCard: ADR:region).
<i>Street</i> ?	string	Street address (vCard: ADR:street).
<i>ExtendedAddress</i> ?	telem	Extended address (vCard: ADR:extadd. For example: Suite 245).

7.2.3 AdhesiveBindingParams

This resource describes the details of the following four sub-processes of the **AdhesiveBinding** process:

- back preparation
- multiple glue applications
- spine taping
- cover application

These subprocesses are identified as instances of the abstract **ABOperation** element. Although a workflow may exist that groups these processes according to its own capabilities, it is likely that they will be performed in the order presented. A description of each follows the table containing the contents of the **AdhesiveBindingParams** resource.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	AdhesiveBinding
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>FlexValue</i> ?	double	Flex quality parameter given in [N/cm].
<i>PullOutValue</i> ?	double	Pull out quality parameter given in [N/cm].
ABOperation +	element	Each ABOperation element describes the parameters of one single operation of the complete AdhesiveBinding process.

Structure of AdhesiveBinding Elements

ABOperation

Resource class:	Abstract element
-----------------	------------------

ABOperation is an abstract element that describes the **AdhesiveBinding** process. The defined instances (sub-classes) of **ABOperation** are **BackPreparation**, **GlueApplication**, **SpineTaping** and **CoverApplication**.

BackPreparation

Resource class:	ABOperation
-----------------	--------------------

Name	Data Type	Description
<i>MillingDepth</i>	double	Milling depth.
<i>NotchingDistance</i> ?	double	Notching distance.
<i>NotchingDepth</i> ?	double	Notching depth.
<i>StartPosition</i>	double	Starting position of milling tool (along the Y-axis of the operation coordinate system).
<i>WorkingLength</i>	double	Working length of milling operation.

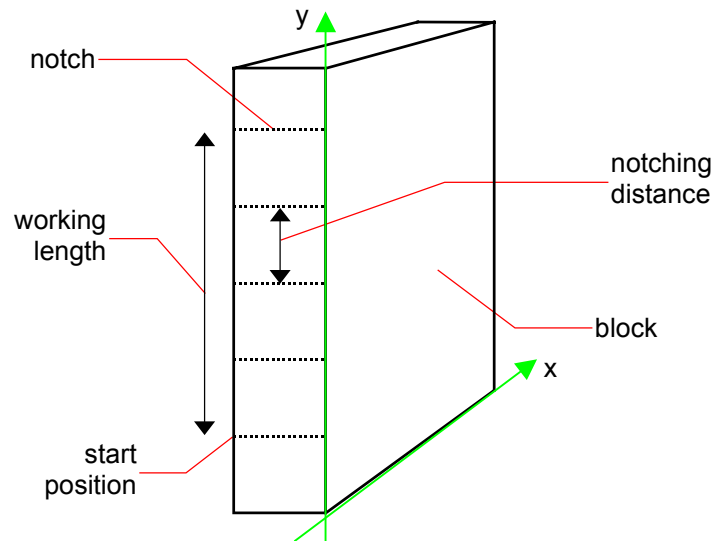


Figure 7.1 Parameters and coordinate systems for back-preparation process

GlueApplication

Resource class: ABOperation

Name	Data Type	Description
<i>GluingTechnique</i>	enumeration	Type or technique of gluing application. Possible values are: <i>SpineGluing</i> <i>SideGluingFront</i> <i>SideGluingBack</i>
GlueLine	element	Structure of the glue line.

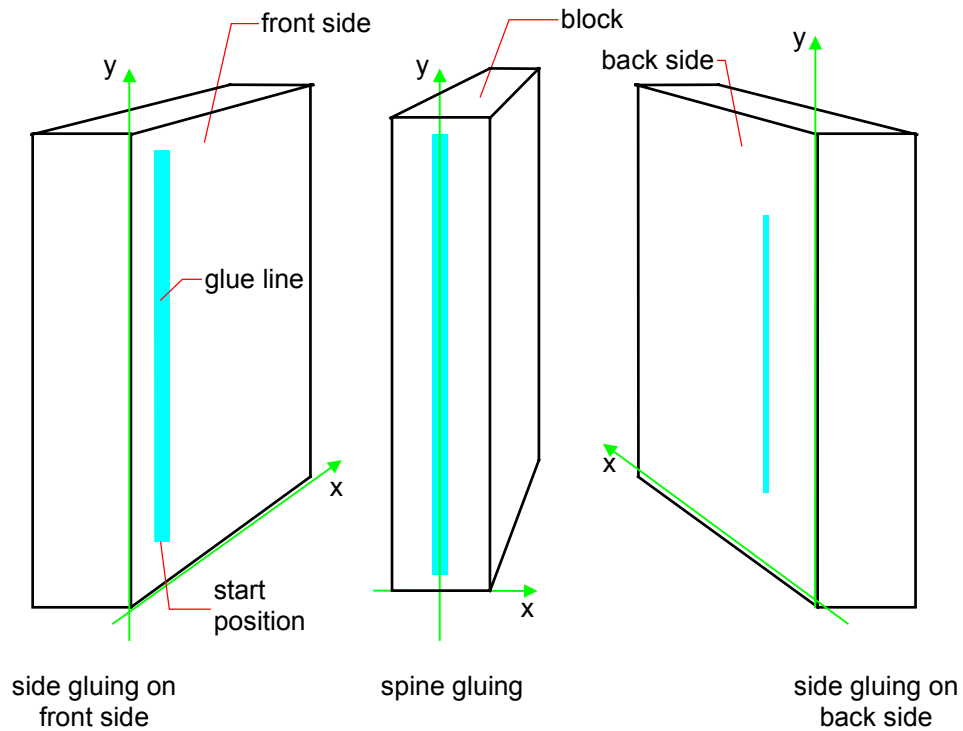


Figure 7.2 Parameters and coordinate system for glue application

SpineTaping

Resource class: ABOperation

Name	Data Type	Description
<i>HorizontalExcess</i>	double	Taping spine excess on each side. The tape is assumed to be centered between left and right.
<i>StripBrand ?</i>	string	Strip brand.
<i>StripColor ?</i>	colorant	Color of the strip.
<i>StripLength</i>	double	Length of strip material along binding edge.
<i>StripMaterial ?</i>	enumeration	Strip material. Possible values are: <i>Gauze</i> <i>Calico</i> <i>PaperlinedMules</i> <i>CrepePaper</i> <i>Tape</i>
<i>TopExcess</i>	double	Top spine taping excess. This value may be negative.

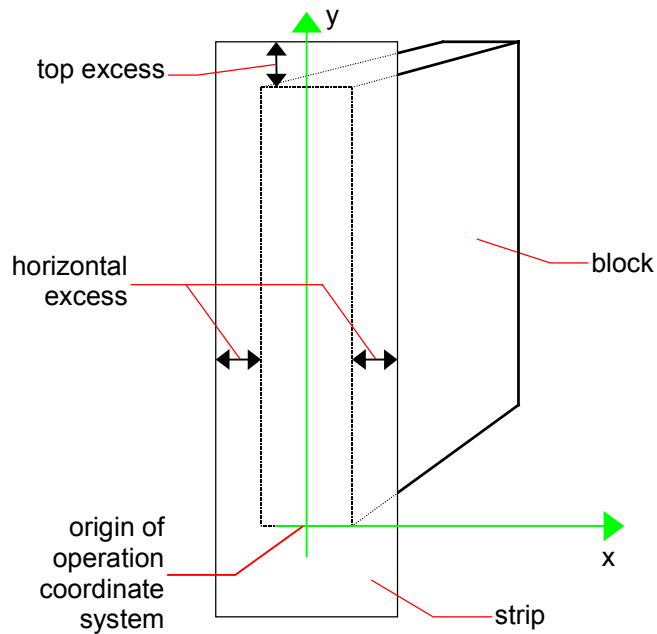


Figure 7.3 Parameters and coordinate system for the spine-taping process

CoverApplication

Resource class: ABOperation

Name	Data Type	Description
<i>CoverOffset</i>	XYPair	Position of the cover in relation to the book block given in the cover-sheet coordinate system.
Score *	element	Describes where and how to score the cover.

Structure of Score Sub-element

Resource class: Element

Name	Data Type	Description
<i>Offset</i>	double	Position of scoring given in the operation coordinate system.
<i>Side</i>	enumeration	Specifies the side from which the scoring tool works. Possible values are: <i>FromInside</i> <i>FromOutside</i>

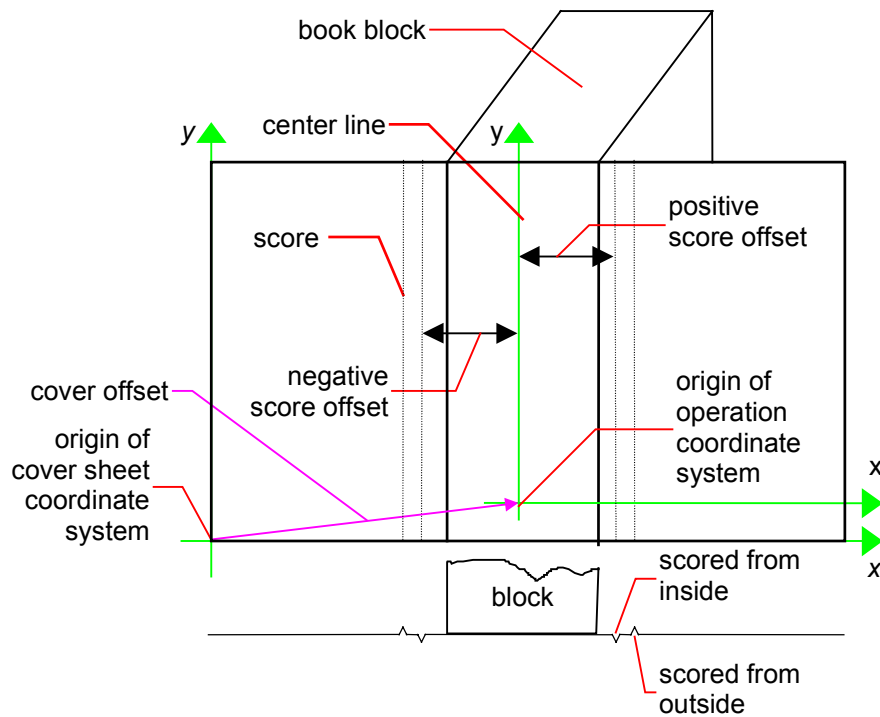


Figure 7.4 Parameters and coordinate system for cover application

7.2.4 ApprovalParams

This resource provides the details of an approval process.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	Approval
Output of processes:	Verification

Resource Structure

Name	Data Type	Description
ApprovalPerson *	element	List of people (such as a customer, printer, or manager) who can sign the approval.

Structure of ApprovalPerson Sub-element

Name	Data Type	Description
<i>Obligated ?</i>	boolean	If <i>true</i> , the person has to sign this approval. Default = <i>true</i>
Contact	element	Contact (such as a customer, printer, or manager) who must sign the approval. The value of the <i>ContactType</i> attribute

of this **Contact** element should be *Administrator*.

7.2.5 ApprovalSuccess

The signed **ApprovalSuccess** resource indicates the success of a soft proof, color proof, printing proof, or any other sort of proof.

Resource Properties

Resource class: Parameter
 Resource referenced by: -
 Partition: -
 Input of processes: -
 Output of processes: *Approval*

Resource Structure

Name	Data Type	Description
FileSpec ?	element	The file that contains the approval signature. If FileSpec does not exist, ApprovalSuccess is a logical placeholder.

7.2.6 ByteMap

This resource specifies the structure of bytemaps produced by various processes within a JDF system. A **ByteMap** represents a raster of image data. This data may have multiple bits per pixel, may represent a varying set of color planes, and may or may not be interleaved.

A Bitmap is a special case of a **ByteMap** in which each pixel is represented by a single bit per color.

Personalized printing requires that certain regions of a given page be dynamically replaced. The optional mask associated with each band of data allows for the omitting certain pixels from the base image represented by the **ByteMap** so that they may be replaced.

Resource Properties

Resource class: General
 Resource references: **RunList**
 Partition: -
 Input of processes: *Screening*
 Output of processes: *RIPping, Scanning, Rendering, Screening*

Resource Structure

Name	Data Type	Description
<i>BandOrdering ?</i>	enumeration	Identifies the precedence given when ordering the produced bands. Possible values are: <i>BandMajor</i> – The position of the bands on the page is prioritized over the color. <i>ColorMajor</i> – All bands of a single color are played in order before progressing to the next plane. This is only

		possible with non-interleaved data.
		This field is required for non-interleaved data and is ignored for interleaved data.
<i>FrameHeight</i>	integer	Height of the overall image that may be broken into multiple bands
<i>FrameWidth</i>	integer	Width of overall image that may be broken into multiple columns
<i>Halftoned</i>	boolean	Indicates whether or not the data has been halftoned.
<i>Interleaved</i>	boolean	If <i>true</i> , the data is interleaved, or chunky. Otherwise the data is non-interleaved, or planar.
<i>PixelSkip ?</i>	integer	Number of bits to skip between pixels of interleaved data.
<i>Resolution</i>	XYPair	Output resolution.
Band +	element	Array of bands containing raster data.
PixelColorant +	element	Ordered list containing information about which colorants are represented and how many bits per pixel are used.

Structure of Band Sub-element

Name	Data Type	Description
<i>Data</i>	URL	Actual bytes of data.
<i>Height</i>	integer	Height in pixels of the band.
<i>Mask ?</i>	URL	1-bit mask of raster data indicating which bits of the band data should actually be used. It is required that the mask dimensions and resolution be equivalent to the contents of the band itself.
<i>WasMarked</i>	boolean	Indicates whether any rendering marks were made in this band. This attribute allows a band to be skipped if no marks were made in the band.
<i>Width</i>	integer	Width in pixels of the band.

Structure of PixelColorant Sub-element

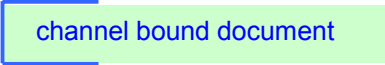
Name	Data Type	Description
<i>ColorantName</i>	string	Name of colorant.
<i>PixelDepth</i>	integer	Number of bits per pixel for each colorant.

7.2.7 ChannelBindingParams

This resource describes the details of the **ChannelBinding** process. The following figure depicts the **ChannelBinding** process.



L


W - ClampD 

The symbols W, L, and ClampD of the illustration above are described by the attributes *ClampD* and *ClampSize* of the table below.

Resource Properties

Resource class: Parameter
 Resource referenced by: -
 Partition: -
 Input of processes: **ChannelBinding**
 Output of processes: -

Resource Structure

Name	Data Type	Description
<i>Brand</i> ?	string	The name of the clamp (or pre-assembled cover with clamp) manufacturer and the name of the specific item.
<i>ClampColor</i> ?	colorant	Determines the color of the clamp/cover. If the clamp is inside of a pre-assembled cover then the color of the cover is meant.
<i>ClampD</i> ?	double	The distance of the clamp that was “pressed away” (see illustration above).
<i>ClampSize</i> ?	shape	 shape size of the clamp: the first number of the shape data corresponds to the clamp width W (see illustration above) which is determined by the final height of the block of sheets to be bound. The second number corresponds to the length L (see illustration above) and the third to the spine length (not visible in the illustration above).[MM33]
<i>ClampSystem</i> ?	boolean	If true the clamp is inside of a pre assembled cover. Default = <i>false</i>

7.2.8 CIELABMeasuringField

Information about a color measuring field. The color is specified as CIE-L*a*b* value.

Resource Properties

Resource class: Element
 Resource referenced by: **Surface**
 Partition: -
 Input of processes: Any printing process
 Output of processes: **Imposition**

Resource Structure

Name	Data Type	Description
------	-----------	-------------

<i>Center</i>	XYPair	Position of the center of the color measuring field in the coordinates of the <i>SurfaceContentsBox</i> .
<i>CIE-Lab</i>	Lab color	L*a*b* color specification.
<i>Diameter</i>	double	Diameter of measuring field.
<i>DensityStandard ?</i>	enumeration	Density filter norm. Possible values are: <i>DIN16536</i> <i>DIN16536NB</i> <i>ANSIA</i> <i>ANSIT</i>
<i>Light</i>	NMTOKEN	Type of light. Possible values include: <i>D50</i> <i>D65</i>
<i>Observer</i>	integer	Observer in degree (2 or 10)
<i>Percentages ?</i>	NumberList	Film percentage values for each separation. The number of array elements must match the number of separations.
<i>ScreenRuling ?</i>	NumberList	Screen ruling values in lines per inch for each separation. The number of array elements must match the number of separations.
<i>ScreenShape ?</i>	string	Shape of screening dots.
<i>Setup ?</i>	string	Description of measurement setup.
<i>Tolerance</i>	double	Tolerance in ΔE .

7.2.9 CoilBindingParams

This resource describes the details of the **CoilBinding** process.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	CoilBinding
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>Brand ?</i>	string	The name of the coil manufacturer and the name of the specific item.
<i>Color ?</i>	colorant	Determines the color of the coil.
<i>Diameter ?</i>	double	The coil diameter to be produced is determined by the height of the block of sheets to be bound.
<i>Material ?</i>	enumeration	The material used for forming the wire-comb binding: <i>LaqueredSteel</i> <i>NylonCoatedSteel</i>

		<i>PVC</i>
		<i>TinnedSteel</i>
		<i>ZincsSteel</i>
<i>Shift ?</i>	double	Amount of vertical shift that occurs as a result of the coil action while opening the document. It is determined by the distance between the holes.
<i>Thickness ?</i>	double	The coil's thickness.
<i>Tucked ?</i>	boolean	If <i>true</i> , the ends of the coil are "tucked in".

7.2.10 CollectingParams

The **Collecting** process needs no special attributes. However, this resource is provided as a container for extensions of the **Collecting** process.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Input of processes:	Collecting
Output of processes:	-

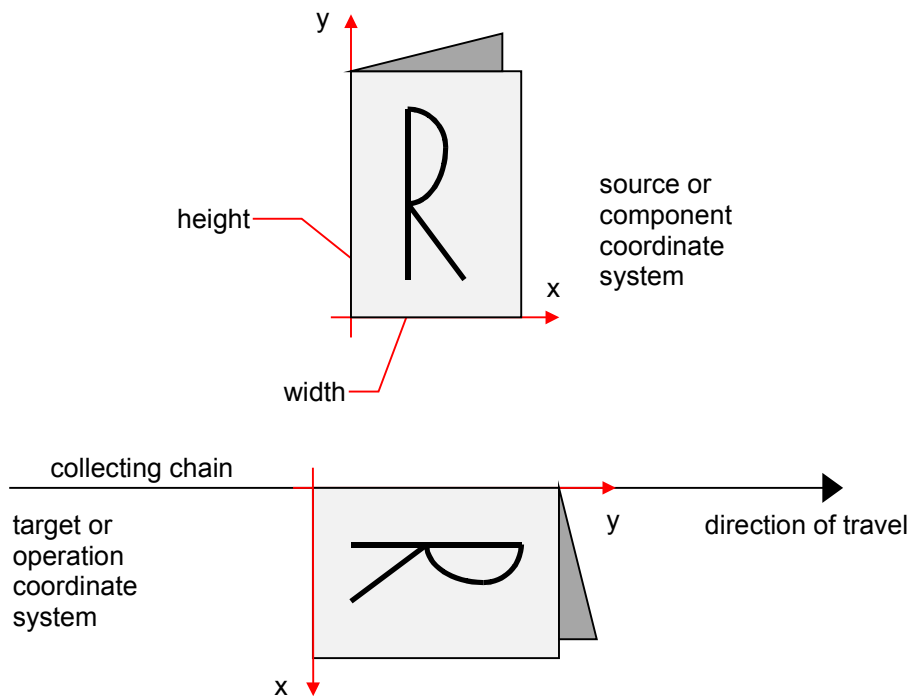


Figure 7.5 Coordinate Systems Used for Collecting

7.2.11 Color

JDF describes spot color inks and, along that line, process color (inks). Spot colors are named colors that may either be separated or converted to process colors.

It is important to know the density of the colorant (for trapping) and, in many cases, the *Lab* values (for representing them on screen). If you know the *Lab* value, you can calculate the density. When representing colors on screen, a conversion to process colors must be defined. This conversion is a simple linear interpolation between the *CMYK* value of the 100% spot color and its tint.

A color is represented by a *Color* element. It has a required *Name* attribute, which represents the name of either a spot color or a process color. The four names that are reserved for representing ProcessCMYK color names are *Cyan*, *Magenta*, *Yellow*, and *Black*. Every colorant can have a *Lab* and/or *CMYK* color value. If both are specified and a system is capable of interpreting both values, the *Lab* value overrides the *CMYK* definition, unless the target device is compatible with the *CMYKTarget*. In this case the *CMYK* value has precedence.



The *Lab* value represents the lab readings of the ink on certain media. This means that spot inks printed on two different kinds of stocks have different *Lab* values. Pantone books, for example, provide *Lab* values for two kinds of paper: *coated* (not necessarily glossy) and *uncoated*. Thus a color of ink should identify the media for which it is specified.

CMYK colors are used to approximate spot colors when they are not separated. This conversion can be done by a color management system or there can be fixed CMYK representation defined by colorbooks such as Pantone.

Resource Properties

Resource class:	Element
Resource referenced by:	ColorPool, Media, TrappingDetails
Partition:	-
Input of processes:	-
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>CMYK</i> ?  [4]	CMYKColor	CMYK value of the 100 % tint value of the colorant. Although optional, it is highly recommended that this value be filled.
<i>ColorType</i> ?	enumeration 	A name that characterizes the colorant. If no value is specified, the device shall provide a default value. Possible values are: <i>DieLine</i> – Marks made with colorants of this type are ignored for trapping. Trapping processes need not generate a color plane for this colorant. Transparent can be used for auxillary process separations. <i>Normal</i> – Marks made with colorants of this type, marks covered by colorants of this type, and marks on top of colorants of this type are trapped. <i>Transparent</i> – Marks made with colorants of this type are ignored for trapping. Trapping processes need not generate a color plane for this colorant. Transparent can be used for varnish. <i>Opaque</i> – Marks covered by colorants of this type are ignored for trapping. Opaque can be used for metallic inks.

		<i>OpaqueIgnore</i> – Marks made with colorants of this type and marks covered by colorants of this type are ignored for trapping. <i>OpaqueIgnore</i> can be used for metallic inks.
<i>Lab</i> ?	LabColor	Lab value of the 100 % tint value of the colorant.
<i>MediaType</i> ?	string	Specifies the mediatype. Possible values are: <i>coated</i> <i>uncoated</i>
<i>Name</i>	string	Name of the colorant.
<i>NeutralDensity</i> ?	number	A number in the range of 0.001 to 10 that represents the neutral density of the colorant. If no value is specified, the device shall provide a default.
<i>sRGB</i> ?	sRGBColor	sRGB value of the 100 % tint value of the colorant.
<i>TargetProfile</i> ?	URL	ICC profile that defines the target output device in case the object that uses the Color has been colorspace converted to a device color space.
<i>UsePDALternateCS</i> ?	boolean	If <i>true</i> , the alternate colorspace definition defined in the PDL shall be used for color space transformations when available. If <i>false</i> , the alternate color space definitions defined in <i>sRGB</i> , <i>CMYK</i> or <i>DeviceNColor</i> of this Color shall be used depending on the value of <i>ProcessColorModel</i> in <i>ColorantControl</i> . Default = <i>true</i>
<i>DeviceNColor</i> *	element	Elements that defines the colorant in a non-standard device-dependent process color space.
<i>TransferCurve</i> *	element	A list of color transfer functions that is used to convert a tint value to one of the alternative colorspace. The transfer functions that are not specified here default to a linear transfer: “0 0 1 1”

Structure of DeviceNColor [RP35]Sub-element

Name	Data Type	Description
<i>ColorList</i>	NumberList	Value of the 100 % tint value of the colorant in the ordered DeviceN space. The list must have <i>N</i> elements. A value of 0 specifies no ink and a value of 1 specifies full ink. The mapping of indices to colors is specified in the <i>DeviceNSpace</i> element of the ColorantControl resource.
<i>N</i>	integer	Number of colors that define the color space.
<i>Name</i> ?	string	Color space name, such as HexaChrome or HiFi. <i>Name</i> must match the <i>Name</i> attribute of a <i>DeviceNSpace</i> element defined in a <i>ColorantControl</i> resource.

Structure of TransferCurve Sub-element¹

Name	Data Type	Description
<i>Curve</i>	TransferFunction	The transfer function.
<i>Separation</i>	string	The name of the separation. If <i>Separation</i> = <i>All</i> , this curve should be applied to all separations.

Color Example

This is an example of the structure for colorant. The transfer curves in this example are defined for process CMYK and sRGB, independently.

```
<Color Name="Pantone Deep Blue" Density="3.14" MediaType="Coated"
Lab="0.2 0.3 0.4" CMYK="0.2 0.3 0.4 0.5" sRGB="0.6 0.7 0.9">
<TransferCurve Separation ="Cyan" Curve="0 0 .5 .4 1 1"/>
<TransferCurve Separation ="Magenta" Curve="0 0 .5 .6 1 1"/>
<TransferCurve Separation ="Yellow" Curve="0 0 1 1"/>
<TransferCurve Separation ="Black" Curve="0 0 1 1"/>
<TransferCurve Separation ="sRed" Curve="0 0 1 1"/>
<TransferCurve Separation ="sGreen" Curve="0 0 1 1"/>
<TransferCurve Separation ="sBlue" Curve="0 0 1 1"/>
</Color/>
```

7.2.12 ColorantControl

ColorantControl is a resource used to control the use of color when processing PDL pages. The attributes and elements of the **ColorantControl** resource describe how color information embedded in PDL pages shall be translated into device colorant information.

Colorants are referenced in **ColorantControl** by name only. Additional details about individual colorants can be found in the **Color** element of the **ColorPool** resource.

ColorantControl resources control which device colorants will be used as well as how document colors will be converted into device color spaces and how conflicting color information should be resolved.

Resource Properties

Resource class:	Parameter
Resource referenced by:	Any process that uses RunList resources
Partition:	-
Input of processes:	Separation , ColorCorrection , ConventionalPrinting , DigitalPrinting , IDPrinting
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>ForceSeparations</i> ?	boolean	If <i>true</i> , forces all colorants to be output as individual separations, regardless of the values defined in

¹ Note that this is identical to the TransferCurve element in a **TransferCurvePool** resource.

		ColorSpaceSubstitute and ColorantAlias. Default = <i>false</i> , which means respect the above elements.
<i>ProcessColorModel</i>	NMTOKEN	Specifies the model to be used for rendering the colorants defined in color spaces into process colorants. Possible values include: <i>DeviceCMY</i> <i>DeviceCMYK</i> <i>DeviceGray</i> <i>DeviceN</i> <i>DeviceRGB</i>
<i>ReplacementColorant-Name</i>	string	The name of the colorant to be substituted for the colorants named in the <i>Alias</i> element list.
ColorantAlias *	element	Identify one or more named colorants that should be replaced with a specified named colorant.
ColorantOrder ?	element	The ordering of named colorants to be processed, for example in the RIP. All of the colorants named must either occur in the <i>ColorantParams</i> list, or be implied by the <i>ProcessColorModel</i>
ColorantParams ?	element	A set of named colorants. This list defines all the colorants that are expected to be available on the device where the process will be executed. The colorants implied by the value of <i>ProcessColorModel</i> are assumed.
ColorPool	element	Pool of <i>Color</i> elements that define the specifics of the colors named in <i>ColorantControl</i> . ²
ColorSpaceSubstitute *	element	These sub-elements identify a colorant that should be replaced by another colorant.
DeviceColorantOrder ?	element	The ordering of named colorants to be output on the device ³ , such as press modules. All of the colorants named must occur in the <i>ColorantParams</i> list, or be implied by the <i>ProcessColorModel</i> . If the <i>DeviceColorantOrder</i> element is not specified, the element defaults to <i>ColorantOrder</i> .
DeviceNSpace *	element	Defines the colorants that make up a DeviceN color space.

Structure of ColorantAlias Sub-element

Name	Data Type	Description
SeparationSpec *	telem	The names of the colorants to be replaced in PDL files.

Structure of ColorantOrder, ColorantParams, and DeviceColorantOrder Elements

Name	Data Type	Description
SeparationSpec *	element	The names of the colorants that define the respective lists.

² Note that this will generally be an inter-resource link.

³ Note that this must be synchronized with the device output ICC profile.

Structure of ColorSpaceSubstitute Sub-element

Name	Data Type	Description
PDLResourceAlias	element	A reference to a color space description that replaces the color space defined by TargetColorantName.
TargetColorantName +	telem	A list of names that defines the colorants to be replaced. This could be a single name in the case of a <i>Separation</i> color space, or more than one name in the case of a DeviceN color space.

Structure of DeviceNSpace Sub-element

Name	Data Type	Description
<i>Name</i> ?	string	Color space name, such as <i>HexaChrome</i> or HiFi.
<i>N</i>	integer	The number of colors that define the color space.
<i>SeparationSpec</i> *	element	Ordered list of colorant names that define the DeviceN colorspace. The ordering maps to the ordering of elements in the corresponding Color::DeviceNColor::ColorList attribute. Note that these colorants must have a ColorantUsage attribute of <i>process</i> or <i>spot</i> . In other words, they must be real physical colorants.

7.2.13 ColorControlStrip

This resource describes a color control strip. The type of the color control strip is given in the *StripType* attribute. If it is known at the system reading the JDF file, there is no need to define the elements of the strip, and the attribute *DensityMeasuringFields* is not needed. Otherwise, this attribute must contain a definition of the contained measuring fields. The lower left corner of the control strip box is used as the origin of the coordinate system used for the definition of the measuring fields. It can be calculated using the following formula:

$$x_0 = x - \frac{w}{2} \cos(\varphi) + \frac{h}{2} \sin(\varphi)$$

$$y_0 = y - \frac{w}{2} \sin(\varphi) - \frac{h}{2} \cos(\varphi)$$

where

x = X element of the *Center* attribute
 y = Y element of the *Center* attribute
 w = X element of the *Size* attribute
 h = Y element of the *Size* attribute
 φ = Value of the *Rotation* attribute

Resource Properties

Resource class: Element
 Resource referenced by: **Surface**
 Partition: -
 Input of processes: *Any printing process*
 Output of processes: **Imposition**

Resource Structure

Name	Data Type	Description
<i>Center</i>	XYPair	Position of the center of the color control strip in the coordinates of the <i>SurfaceContentsBox</i> .
<i>Rotation ?</i>	double	Rotation in degrees. Positive graduation figures indicate counter-clockwise rotation; negative figures indicate clockwise rotation.
<i>Size</i>	XYPair	Size of the color control strip.
<i>StripType ?</i>	NMTOKEN	Type of color control strip. This attribute can be used for specifying a predefined, company-specific color control strip.

7.2.14 ColorCorrectionParams

This resource provides the information needed for an operator to correct colors on some PDL pages or content elements such as image, graphics, or formatted text.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	<i>ColorCorrection</i>
Output of processes:	-

Name	Data Type	Description
<i>ColorManagement-System ?</i>	string	Identifies the preferred ICC color-management system to use when performing corrections. Overrides the default selection of the application or the selection contained in any of the profiles when specified.
<i>FinalTargetDevice ?</i>	URI	Describes the characterization of the final output target device using an ICC profile.
<i>WorkingColorSpace ?</i>	URI	Describes the assumed characterization of <i>CMYK</i> , <i>RGB</i> and <i>Gray</i> colorspace using ICC Profiles.
ColorCorrectionOp *	element	List of ColorCorrectionOp sub-elements.

It is assumed that color correction will be performed by a human operator; no attempt is made to encode specific types of operations.

Sub-elements of the **ColorCorrectionParams** resource should contain a **Comment** to describe the desired correction operation, and, optionally, to provide a region to be corrected via the **Comment::Path** or **Comment::Box** elements.

Structure of ColorCorrectionOp Sub-element

Name	Data Type	Description
<i>SourceObjects ?</i>	enumerations	Identifies which class of incoming graphical objects will be operated on. Possible values are: <i>All</i> – Default value.

ImagePhotographic – Contone images.

ImageScreenShot – Images largely comprised of rasterized vector art.

Text

LineArt

SmoothShades – Gradients and blends.

7.2.15 ColorPool

The **ColorPool** resource contains a pool of all **Color** elements referred to in the job.

Resource Properties

Resource class: Parameter
 Resource referenced by: -
 Partition: -
 Input of processes: *ColorSpaceConversion, Proofing, SoftProofing, RIPping, Trapping*
 Output of processes: -

Resource Structure

Name	Data Type	Description
Color *	element	Individual named color.

7.2.16 ColorSpaceConversionParams

This set of parameters defines the rules for a **ColorSpaceConversion** process, the elements of which define the set of operations to be performed.

Information inside the **ColorSpaceConversionOp** elements, described below, defines the operation and identifies the colorspaces and types of objects to operate on.

Other attributes define the color management system to use, as well as the working color space and the final target device.

Resource Properties

Resource class: Parameter
 Resource referenced by: -
 Partition: -
 Input of processes: *ColorSpaceConversion, Proofing, SoftProofing*
 Output of processes: -

Resource Structure

Name	Data Type	Description
<i>ColorManagement-System ?</i>	string	Identifies the preferred ICC color management system to use when performing transformations. Overrides the default selection of the application or that contained in any of the profiles when specified.

<i>ConvertDevIndepColors</i> ?	boolean	When <i>true</i> , incoming device-independent colors are processed to the selected device space. If the chosen operation is <i>untag</i> and the characterization data are in the form of an ICC profile, then the profile is removed. Otherwise, these colors are left untouched.
<i>FinalTargetDevice</i> ?	URI	Describes the characterization of the final output target device using an ICC profile. This item is required when converting, but optional for tagging or untagging.
<i>WorkingColorSpace</i> ?	URI	Describes the assumed characterization of <i>CMYK</i> , <i>RGB</i> and <i>Gray</i> colorspaces using ICC Profiles. This item is required for converting or tagging, but optional for untagging.
ColorSpaceConversion-Op *	element	List of <i>ColorSpaceConversionOp</i> sub-elements.

Structure of ColorSpaceConversionOp Sub-element

Name	Data Type	Description
<i>IgnoreEmbeddedICC?</i>	boolean	If <i>true</i> , specifies that embedded source ICC profiles shall be ignored and that the ICC profile defined by <i>SourceProfile</i> shall be used instead. Default= <i>false</i> .
<i>Operation</i>	enumeration	Controls which of three functions the color space conversion utility performs. Possible values are: <i>Convert</i> – Transforms graphical elements to final target color space. <i>Tag</i> – Associates appropriate working space profile with uncharacterized graphical element. <i>Untag</i> – Removes all profiles and color characterizations from graphical elements <i>Retag</i> – Removes all profiles and color characterizations from graphical elements and replaces them with the appropriate values. Equivalent to a sequence of <i>UnTag</i> → <i>Tag</i> <i>ConvertIgnore</i> – Removes all profiles and color characterizations from graphical elements and converts to the appropriate values. Equivalent to a sequence of <i>UnTag</i> → <i>Convert</i>
<i>RenderingIntent</i> ?	enumeration	Identifies the rendering intents associated with <i>SourceObjects</i> elements. Possible ICC-defined rendering intent values are: <i>Saturation</i> <i>Perceptual</i> <i>RelativeColorimetric</i> <i>AbsoluteColorimetric</i>
<i>RGBGray2Black</i> ?	boolean	This feature controls what happens to gray values (R = G = B) when converting from RGB to CMYK. In the case of MSOffice applications and screendumps, there are a number of gray values in the images and lineart. Printers

		do not want to have CMY under the K (Registration). Therefore, they prefer to have K only. This is not true for photographic images. In that case, everything is moved through a link.
		Default = <i>false</i>
SourceCS	enumeration	Identifies which of the incoming color spaces will be operated on. Possible values are: <i>CMYK</i> – Operates on <i>deviceCMYK</i> or 4-component ICCBased colorspaces. <i>RGB</i> – Operates on <i>deviceRGB</i> , <i>calRGB</i> or 3-component ICCBased colorspaces <i>Gray</i> – Operates on <i>deviceGray</i> , <i>calGray</i> or 1-component ICCBased colorspaces.
SourceObjects ?	enumerations	List of objects that identifies which class of incoming graphical objects will be operated on. Possible values are: <i>All</i> – Default value. <i>ImagePhotographic</i> – Contone images. <i>ImageScreenShot</i> – Images largely comprised of rasterized vector art. <i>Text</i> <i>LineArt</i> <i>SmoothShades</i> – Gradients and blends.
SourceProfile ?	url	Link to an ICC profile that describes the assumed color space. The default is to use embedded profiles.

7.2.17 ComChannel

A communication channel to a person or company such as an email address, phone number, or fax number.

Resource Properties

Resource class:	Element
Resource referenced by:	Contact, Person
Partition:	-
Input of processes:	-
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>ChannelType</i>	enumeration	Type of the communication channel. Possible values are: <i>Phone</i> – Telephone number. <i>Email</i> – E-mail address. <i>Fax</i> – Fax machine. <i>WWW</i> – WWW home page or form.

		<i>JMF</i> – JMF messaging channel.
<i>Locator</i>	string	Locator of this type of channel in a form such as a phone number or an email address.

7.2.18 Company

Specifies contacts to a company including detailed information about contact persons and addresses. This structure can be used in many situations where addresses or contact persons are needed. Examples of contacts are customer, supplier, company, and addressees. The structure is derived from the vCard format. It comprises the organization name and organizational units (ORG) of the organizational properties defined in the vCard format. The corresponding XML types of the vCard are quoted in the table.

Resource Properties

Resource class:	Element
Resource referenced by:	ArtDeliveryIntent, DeliveryIntent, DeliveryParams, DroplIntent, OrderingParams
Partition:	-
Input of processes:	-
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>OrganizationName</i>	string	Name of the organization or company (vCard: ORG:orgnam. For example: ABC, Inc.).
Contact *	element	A contact of the company.
OrganizationalUnit *	telem	Describes the organizational unit (vCard: ORG:orgunit. For example, if two elements are present: 1. “North American Division” and 2. “Marketing”).

7.2.19 Component

Component is used to describe the various versions of semi-finished goods in the press and postpress area, such as a pile of folded sheets that have been collected and must then be joined and trimmed.

Nearly every postpress process has a **Component** resource as an input as well as an output. Typically the first components in the process chain are some printed sheets, while the last component is a book or a brochure.

Component resources are grouped by kind, in much the same way that nodes are classified as Combined, Process, or Product Intent. The four categories of **Component** resources are: *Sheet*, *Block*, *PartialProduct*, and *FinalProduct*. These categories are defined in greater detail below:

<i>Sheet</i>	This source type is appropriate if a flat sheet—such as a postcard to be glued in—is used as an input component. "Flat" in this case means that the sheet has not been folded or cut before the operation.
<i>Block</i>	This source type is appropriate if a folded sheet, a cut portion of the sheet, or a cut and folded portion of a sheet is used as an input component.

PartialProduct This source type is appropriate if a partial product should be used as an input component.

FinalProduct This source type is appropriate if this **Component** is the final product.



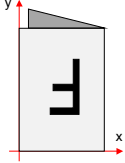

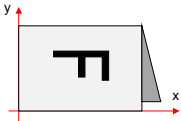

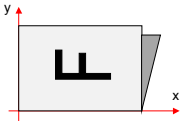

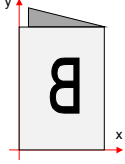

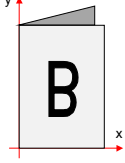

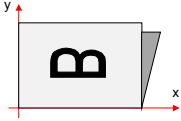

Coordinate Systems of Components and Processes

Each input **Component** of a process has, by default, its own coordinate system, which is called source or component coordinate system. The coordinate system also implies a specific orientation of that component.

On the other hand there is a coordinate system that is used to define various process-specific parameters. This coordinate system is called target or process coordinate system.

It is often necessary to change the orientation of an input **Component** before executing the operation. This can be done by specifying a transformation matrix. It is stored in the *Transformation* attribute of the **Component**. This provides the ability to specify different matrices for the individual input components of a process.

The following table shows some matrices that can be used to change the orientation of an input **Component**. Most of the transformations require the X- (**w**) and the Y-dimension (**h**) of the **Component** as specified in the *Dimension* element.

Source Coordinate System	Transformation Matrix According Action	Target Coordinate System
	$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$ No Action	
	$\begin{bmatrix} -1 & 0 & 0 & -1 & w & h \end{bmatrix}$ 180° Rotation	
	$\begin{bmatrix} 0 & 1 & -1 & 0 & h & 0 \end{bmatrix}$ 90° Counterclockwise Rotation	
	$\begin{bmatrix} 0 & -1 & 1 & 0 & 0 & w \end{bmatrix}$ 90° Clockwise Rotation	
	$\begin{bmatrix} -1 & 0 & 0 & 1 & w & 0 \end{bmatrix}$ Horizontal Flip	
	$\begin{bmatrix} 1 & 0 & 0 & -1 & 0 & h \end{bmatrix}$ Vertical Flip	
	$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$ 90° Counterclockwise Rotation + Horizontal Flip	

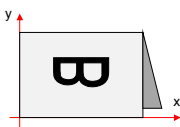
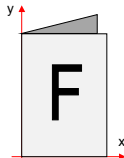
	$[0 \ -1 \ -1 \ 0 \ h \ w]$ 90° Clockwise Rotation + Horizontal Flip	
---	--	---

Table 7.1 Matrices used to change the orientation of a Component

Terms and Definitions for Components

The descriptions of **Component**-specific attributes use some terms whose meaning depends on the culture in which they are used. For example, different cultures mean different things when they refer to the “front” side of a magazine. Other terms, such as binding, are defined by the production process and therefore do not depend on the culture.

Whenever possible, this specification endeavors to use culture-independent terms. In cases where this is not possible, Western style (left-to-right writing) is assumed. Please note that these terms may have a different meaning in other cultures (such as those writing from right to left).

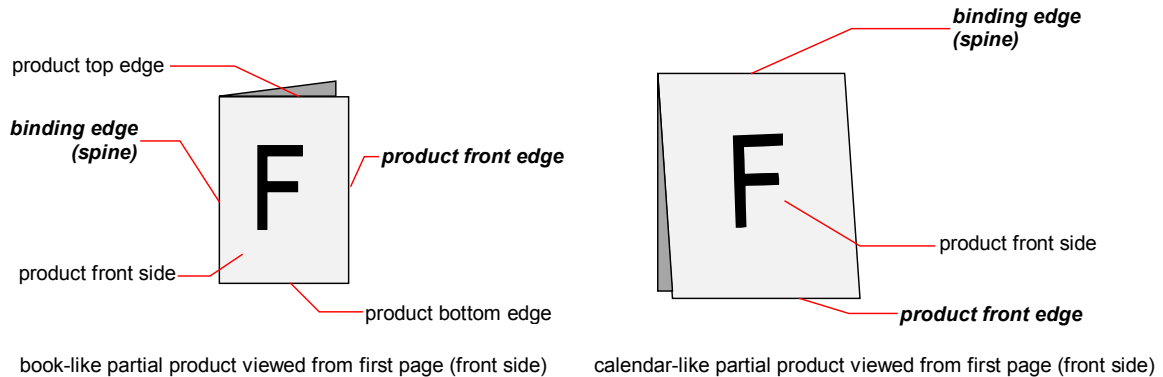


Figure 7.6 Terms and Definitions for Components

The table below describes the terms used to define the components.

Table 7.2 Terms and Definitions for Components

Edge	Description
Binding edge	The edge on which the (partial) product is glued or stitched. This edge is also often called <i>working edge</i> or <i>spine</i> .
Product front edge	The side, where you open the (partial) product. This edge is opposite to the binding edge.
Registered edge	A side on which a collection of sheets or partial products is aligned during a production step. All production steps require two registered edges, which must not be opposite to each other. The two registered edges define the coordinate system used within the production step. When there is a binding edge, this is one of the registered edges.

Resource Properties

Resource class: *Quantity*

Resource referenced by: -
 Partition: *Side, SheetName, Signature*
 Input of processes: Many
 Output of processes: Many

Resource Structure

Name	Data Type	Description
<i>ComponentType</i>	enumeration	Specifies the category of the component. Possible values are: <i>Sheet</i> <i>Block</i> <i>PartialProduct</i> <i>FinalProduct</i>
<i>IsWaste ?</i>	boolean	If true, the component waste may be used to set up a machine. Default = <i>false</i>
<i>MaxHeat ?</i>	double	Maximum temperature the Component can resist (in degree centigrade).
<i>SheetPart ?</i>	rectangle	Only useful when <i>ComponentType</i> = <i>Block</i> and when <i>SourceSheet</i> is present. Part of the <i>Sheet</i> in <i>SurfaceContentsBox</i> coordinates used in this Component .
<i>SourceSheet ?</i>	string	Only required when <i>ComponentType</i> = <i>Sheet</i> or <i>Block</i> . <i>SheetName</i> of the sheet used in this Component .
<i>Transformation ?</i>	matrix	Matrix describing the transformation of the orientation of a component for the next process. This is needed to convert the coordinate system of the component to the coordinate system of the process. When this attribute is not present, the identity matrix (1 0 0 1 0 0) is assumed.
<i>Dimensions ?</i>	element	The dimensions of the component. At minimum, the three dimensions X, Y and Z are required. These dimensions differ from the original size of the original product. For example, the dimensions of a folded sheet may not be equal to the dimensions of the sheet before it was folded.
<i>Disjointing ?</i>	element	A stack of components can be processed using physical separators. This is useful in operations such as feeding.

Dimensions

Name	Data Type	Description
X	double	Expansion in dimension X.
Y	double	Expansion in dimension Y.
Z	double	Expansion in dimension Z.

7.2.20 Contact

Element describing a contact to a person or address.

Resource Properties

Resource class:	Element
Resource referenced by:	ApprovalParams, Company
Partition:	-
Input of processes:	-
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>ContactType</i>	NMTOKENS	Classification of the contact. Possible values include: <i>Administrator</i> – Person to contact for queries concerning the execution of the job. <i>Accounting</i> – Address of where to send to the bill. <i>Delivery</i> – Delivery address for all products of this job. <i>Supplier</i> – Address of a supplier of needed goods. <i>Customer</i> – The end customer. <i>Owner</i> – The owner of a resource.
Address ?	element	Element describing the address.
ComChannel *	element	Communication channels to the company, not to a specific person.
Person ?	element	Name of the contact person.

7.2.21 ConventionalPrintingParams

This resource defines the attributes and elements of the **ConventionalPrinting** process. The specific parameters of individual printer modules are modelled by using the standard partitioning methods. These methods are described in section 3.8.2.


Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	<i>Separation, Side, SheetName, SignatureName</i>
Input of processes:	ConventionalPrinting
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>DirectProof?</i>	boolean	If <i>true</i> , the proof is directly produced and subsequently an approval may be given by a person such as the customer, foreman, or floor manager shortly after the first final-quality printed sheet is printed. The approval is not required for setup, but it is required for the actual print

		run. If the ConventionalPrinting process is waiting for a DirectProof , its Status is switched to <i>stopped</i> with the StatusDetails = <i>WaitForApproval</i> .
Drying ?	enumeration	<p>The way in which ink is dried after a print run. Possible values are:</p> <p><i>UV</i> – Ultraviolet dryer</p> <p><i>Heatset</i> – Heatset dryer</p> <p><i>IR</i> – Infrared dryer</p> <p><i>On</i> – Use the device default drying unit.</p> <p><i>Off</i> – Default value.</p>
FirstSurface ?	enumeration	<p>Printing order of the surfaces. Possible values are:</p> <p><i>Either</i> – Default value. The printer may choose.</p> <p><i>Front</i></p> <p><i>Back</i></p>
FountainSolution ?	enumeration	<p>State of the fountain solution module in the printing units. Possible values are:</p> <p><i>On</i></p> <p><i>Off</i></p> <p>If not specified use the device default setting, which may be either <i>On</i> or <i>Off</i>.</p>
MediaLocation ?	string	Identifies the location of the Media . The value identifies a physical location on the press, such as unwinder 1, unwinder 2, and unwinder 3.
ModuleDrying ?	enumeration	<p>The way in which ink is dried in individual modules. Possible values are:</p> <p><i>UV</i> – Ultraviolet dryer</p> <p><i>Heatset</i> – Heatset dryer</p> <p><i>IR</i> – Infrared dryer</p> <p><i>On</i> – Use the device default drying unit.</p> <p><i>Off</i> – Default.</p>
ModuleIndex ?	IntegerRange List	Zero-based list of print modules that are used. Defaults to device default.
Powder ?	double	Quantity of powder (in g/m ²).
PrintingType	enumeration	<p>Type of printing machine. Possible values are:</p> <p><i>SheetFed</i></p> <p><i>WebFed</i></p> <p>The principal difference between <i>SheetFed</i> and <i>WebFed</i> is the shape of the paper each is equipped to accept. Presses that execute <i>WebFed</i> processes use substrates that are continuous and cut after printing is accomplished. Most newspapers are printed on web-fed presses. <i>SheetFed</i> printing, on the other hand, accepts pre-cut substrates.[MS37]</p>

<i>SheetLay</i> ?	enumeration	Lay of input media. Possible values are: <i>Left</i> <i>Right</i> <i>Center</i>
 <i>Speed</i> ?	number	Default is the device default. Maximum print speed in sheets/hour (sheet fed) or meters/hour (web fed). Defaults to device specific full speed.
<i>WorkStyle</i> ?	enumeration	The direction in which to turn. Possible values are: <i>Simplex</i> – No turning <i>WorkAndBack</i> – This <i>WorkStyle</i> describes the printing on both sides of the substrate with a different plate (set) in the second run. After the first run the side lays are altered but the front lays stay as they were. Lays can be turned by hand or using a pile reverser. Two-plate sets are necessary for <i>WorkAndBack</i> . <i>Perfecting</i> – Many sheet-fed printing presses have perfecting cylinder(s) build in. The leading edge of the print sheet changes as the sheet is turned by the perfecting cylinder, but the side lays remain unaltered. In this regard, this <i>WorkStyle</i> is similar to <i>WorkAndTumble</i> , but <i>Perfecting</i> is an inline operation during the press run. Therefore, an additional plate (set) is required during this press run. <i>WorkAndTurn</i> – Refers to the turning of the first-run sheet for subsequent perfecting. The front lays remain unchanged but the side lays must be altered. The alteration can be made by hand or using a pile turner. The plate (set) stay(s) in the machine and, during each run, half of the surface is imaged. <i>WorkAndTumble</i> – The <i>WorkAndTumble</i> method is also used for perfecting. The leading edge of the print sheet changes as the sheet is turned, but the side lays remain unaltered. Tumbling happens after the first press run and the plate (set) is used again in the second press run, imaging the other sheet surface. <i>WorkAndTwist</i> – Done between two press runs. The palette is twisted 180 degree before the second run is performed so that the front lay and the side lay both change. The surface to be imaged is the same at both runs. Each run prints only part of the surface. The plate (set) stay in the machine. This <i>WorkStyle</i> is used for saving plate or film material. It is no longer a common <i>WorkStyle</i> .
<i>Ink</i> ?	element	Kind of varnishing. Defines the varnish to be used for coatings on printed sides. Coatings are applied after printing all the colors. Other coating sequences must use the partition mechanism of this parameter resource. Selective varnishing has to use a separate separation for the respective varnish.

Note, the color inks are direct input resources of the *ConventionalPrinting* process.

7.2.22 CostCenter

Defines a cost center.[RP38]

Resource Properties

Resource class: Element
 Resource referenced by: **Device, Employee**
 Partition: -
 Input of processes: -
 Output of processes: -

Resource Structure

Name	Data Type	Description
<i>CostCenterID</i>	string	Identification of the cost center
<i>Name ?</i>	string	Name of the cost center.
<i>IsActive?</i>	boolean	If multiple cost centers are specified, this defines the active cost center. Default=true.

7.2.23 CutBlock

Defines a cut block on a sheet.

It is possible to define a block that contains a matrix of elements of equal size. In this scenario, the intermediate cut dimension is calculated from the information about element size, block size and the number of elements in both directions.

Each cut block has its own coordinate system, which is defined by the *BlockTrf* attribute.

Resource Properties

Resource class: Parameter
 Resource referenced by: -
 Partition: -
 Input of processes: **Cutting**
 Output of processes: **Imposition**

Resource Structure

Name	Data Type	Description
<i>BlockElementSize ?</i>	XYPair	Element dimension in X and Y direction.
<i>BlockElementType ?</i>	enumeration	Element type. Possible values are: <i>CutElement</i> – Cutting element. <i>PunchElement</i> – Punching element.
<i>BlockName</i>	NMTOKEN	Name of the block. Used for reference by the CutMark resource.

		resource.
<i>BlockSize</i>	XYPair	Size of the block.
<i>BlockSubdivision ?</i>	XYPair	Number of elements in X and Y direction.
<i>BlockTrf</i>	matrix	Block transformation matrix. Defines the position and orientation of the block relative to the Component coordinate system.
<i>BlockType</i>	enumeration	Block type. Possible values are: <i>CutBlock</i> – Block to be cut. <i>SaveBlock</i> – Protected block, cut only via outer contour. <i>TempBlock</i> –Auxiliary block that is not taken into account during cutting. <i>MarkBlock</i> –Contains no elements, only marks.

7.2.24 CutMark

This resource, along with **CutBlock**, provides the means to position cut marks on the sheet. After printing, these marks can be used to adapt the theoretical block positions (as specified in **CutBlock**) to the real position of the corresponding blocks on the printed sheet.

Resource Properties

Resource class:	Element
Resource referenced by:	Surface
Partition:	-
Input of processes:	Cutting
Output of processes:	Imposition

Resource Structure

Name	Data Type	Description
<i>Blocks</i>	NMTOKENS	Values of the <i>BlockName</i> attributes of the blocks defined by the CutMark resource.
<i>MarkType</i>	enumeration	Mark type. Possible values are: <i>CrossCutMark</i> <i>TopVerticalCutMark</i> <i>BottomVerticalCutMark</i> <i>LeftHorizontalCutMark</i> <i>RightHorizontalCutMark</i> <i>LowerLeftCutMark</i> <i>UpperLeftCutMark</i> <i>LowerRightCutMark</i> <i>UpperRightCutMark</i>
<i>Position</i>	XYPair	Position of the logical center of the cut mark in the coordinates of the <i>SurfaceContentsBox</i> . Please note that the logical center of the cut mark does

not always directly specify the center of the visible cut mark symbol.










Symbol	Name	Position of symbol
	<i>CrossCutMark</i>	Centered at logical position
	<i>TopVerticalCutMark</i>	Slightly above logical position
	<i>BottomVerticalCutMark</i>	Slightly below logical position
	<i>LeftHorizontalCutMark</i>	Slightly to the left of logical position
	<i>RightHorizontalCutMark</i>	Slightly to the right of logical position
	<i>LowerLeftCutMark</i>	Corner at logical position
	<i>UpperLeftCutMark</i>	Corner at logical position
	<i>LowerRightCutMark</i>	Corner at logical position
	<i>UpperRightCutMark</i>	Corner at logical position

Figure 7.7 Cut Mark Types

7.2.25 DBMergeParams

This resource specifies the parameters of the ***DBTemplateMerging*** process.

Resource Properties

Resource class:	Parameter
Resource references:	-
Resource inheritance:	-
Partition	-
Input of processes:	<i>DBTemplateMerging</i>
Output of processes:	-

Resource Structure

Name	Data Type	Description
------	-----------	-------------

<i>SplitDocuments</i> ?	integer	Indicates how often to split documents to create a new file.
FileSpec ?	element	URL of the generated destination file. This is most often a printable file type, such as PDF or PPML. If FileSpec is not specified, DBMergeParams must be a Pipe resource.

7.2.26 DBRules

This resource specifies the rules that should be applied to convert a database record into a graphic element. It is described by a text element with a human-readable description of the selection rules.

For example:

```
insert the "Age" field behind the birthday;
if income>100,000 use Porsche.gif, else use bicycle.jpeg for image #2.
```

The internal representation of the mapping of database fields to graphic content within the document template is implementation-dependent. It can vary from fully variable, multi-page, automated document layout to simply inserting some line-feed characters between database records in an address field. Therefore, **DBRules** is defined as a simple human-readable text element.

Resource Properties

Resource class:	Parameter or Element
Resource references:	-
Resource inheritance:	-
Partition:	-
Input of processes:	<i>DBDocTemplateLayout, Inserting, Collecting, Gathering</i>
Output of processes:	-

Resource Structure

Name	Data Type	Description
Comment +	telem	Human-readable description of the database rules that map database fields to image or text content.

7.2.27 DBSchema

This resource specifies the formal structure of a database record, regardless of type. It is encoded as a text element with a human-readable description of the database schema.

Resource Properties

Resource class:	Parameter or Element
Resource references:	-
Resource inheritance:	-
Partition:	-
Input of processes:	<i>DBDocTemplateLayout, Verification</i>
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>DBSchemaType</i>	enumeration	Database type. Possible values are:

		<i>CommaDelimited</i>
		<i>SQL</i>
		<i>XML</i>
Comment +	telem	Human-readable description of the database schema.

7.2.28 DBSelection

This resource specifies a selection of records from a database.

Resource Properties

Resource class:	Parameter or Element
Resource references:	-
Resource inheritance:	-
Partition	-
Input of processes:	<i>DBTemplateMerging, Inserting, Gathering, Collecting, Verification</i>
Output of processes:	<i>Verification</i>

Resource Structure

Name	Data Type	Description
<i>DataBase</i>	URL	URL of the database
<i>Records ?</i>	IntegerRangeList	The indices of the database records.
<i>Select ?</i>	string	Database selection criteria in the native language of the database, such as SQL.

7.2.29 DeliveryParams

Provides information needed by a ***Delivery*** process. A ***Delivery*** process consists of sending a quantity of a product to a specific location at, in some cases, a required date and time.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	<i>Delivery</i>
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>Earliest ?</i>	timeInstant	Specifies the earliest time after which the delivery may be made.
<i>Method ?</i>	string	Identifies a required delivery method, such as <i>ExpressMail</i> or <i>InterofficeMail</i> .
<i>Pickup ?</i>	boolean	If <i>true</i> , the merchandise is picked up. If <i>false</i> , the merchandise is delivered. Default = <i>false</i>

<i>Required ?</i>	timeInstant	Specifies the time by which the delivery must be made.
Company	element	Address and further information of the addressee.
Drop +	element	All locations where the product will be delivered.

Structure of DeliveryParams Sub-elements

Drop

Name	Data Type	Description
<i>Earliest ?</i>	timeInstant	Specified the earliest time after which the delivery may be made.
<i>Method ?</i>	string	Identifies a required delivery method, such as <i>ExpressMail</i> or <i>InterofficeMail</i> .
<i>Pickup ?</i>	boolean	If <i>true</i> , the merchandise is picked up. If <i>false</i> , the merchandise is delivered. Default = <i>false</i>
<i>Required ?</i>	timeInstant	Specifies the time by which the delivery must be made.
Company	element	Address and further information of the addressee.
Package *	element	A Drop may consist of multiple products, which are represented by their respective Component resources. Each Package describes an individual resource that is part of this Drop .

Package

Name	Data Type	Description
<i>Amount ?</i>	integer	Specifies the number of Components ordered. If <i>Amount</i> is not specified, defaults to the total amount of the Component that is referenced by <i>rRef</i> .
<i>rRef</i>	IDREF	Reference to the Component that this Package contains.
<i>Unit ?</i>	string	Unit of measurement for the <i>Amount</i> specified in <i>ComponentLink</i> . Defaults to the value of <i>Unit</i> defined in the Component resource linked by <i>rRef</i> .
Part *	element	Package may contain any partitioning or amount attributes valid for a ResourceLink in a ResourceLinkPool .

7.2.30 DensityMeasuringField

This resource contains information about a density measuring field.

Resource Properties

Resource class:	Element
Resource referenced by:	ColorControlStrip , Surface
Partition:	-
Input of processes:	<i>Any printing process</i>
Output of processes:	Imposition

Resource Structure

Name	Data Type	Description
<i>Center</i>	XYPair	Position of the center of the density measuring field in the coordinates of the <i>SurfaceContentsBox</i> . If this measuring field is inside a ColorControlStrip , the position is relative to the coordinates of that strip.
<i>Density</i>	CMYK color	Density value measured with filter.
<i>Diameter</i>	double	Diameter of measuring field.
<i>DotGain</i>	double	Percentage of dot gain.
<i>Percentage</i>	double	Film percentage or equivalent.
<i>Screen</i>	string	Description of the screen.
<i>Separation</i>	string	Reference to separation.
<i>Setup ?</i>	string	Description of measurement setup.
<i>ToleranceCyan</i>	XYPair	Upper and lower cyan tolerance (in density units).
<i>ToleranceMagenta</i>	XYPair	Upper and lower magenta tolerance (in density units).
<i>ToleranceYellow</i>	XYPair	Upper and lower yellow tolerance (in density units).
<i>ToleranceBlack</i>	XYPair	Upper and lower black tolerance (in density units).
<i>ToleranceDotGain</i>	XYPair	Upper and lower tolerance (in percentage).

7.2.31 Device

Information about a specific device. For more information, see section 3.6.1.3 Implementation Resources.

Resource Properties

Resource class:	<i>Implementation</i>
Resource referenced by:	-
Partition:	-
Input of processes:	-
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>DeviceFamily?</i>	string	Manufacturer family type ID.
<i>DeviceID</i>	string	Name of the device.
<i>DeviceType?</i>	string	Manufacturer type ID, including a revision stamp.
CostCenter *	element	MIS cost center ID.

7.2.32 DigitalPrintingParams

This resource contains attributes and elements used in executing the **DigitalPrinting** process.

The *PrintingType* attribute in this resource defines two types of printing: *SheetFed* and *WebFed*. The principal difference between them is the shape of the paper each is equipped to accept. Presses that execute *WebFed* processes use substrates that are continuous and cut after printing is accomplished. Most

newspapers are printed on web-fed presses. *SheetFed* printing, on the other hand, accepts pre-cut substrates.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	<i>Separation, Side, SheetName, SignatureName</i>
Input of processes:	DigitalPrinting
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>PrintingType</i>	enumeration	Type of printing machine. Possible values are: <i>SheetFed</i> <i>WebFed</i>
<i>SheetLay</i> ?	enumeration	Lay of input media. Possible values are: <i>Left</i> <i>Right</i> <i>Center</i> <i>Default</i> = The device-specific machine default
<i>MediaSource</i> ?	element	Describes the source and physical orientation of the media to be used.

MediaSource

Name	Data Type	Description
<i>Class</i> ?	string	Product-specific classification of media, which may influence rendering. For example, transparent or glossy media may affect the selection of a color rendering method or a post-rendering technique specific to the device.
<i>LeadingEdge</i> ?	number	Specifies the size, in points, of the edge of the media that represents the scanline direction. If this attribute is absent, the scanline direction is assumed to be along the x-axis of the <i>Dimension</i> parameter for the Media .
<i>ManualFeed</i> ?	boolean	Indicates whether the media will be fed manually.
<i>Position</i> ?	integer	In a device that has numbered input sources, identifies which source is to be used.

7.2.33 Disjointing

The **Disjointing** resource describes how individual components are separated from one another on a stack.

Resource Properties

Resource class:	Element
Resource referenced by:	Component, GatheringParams

Partition: -
 Input of processes: -
 Output of processes: -

Resource Structure

Name	Data Type	Description
<i>InsertSheet</i> ?	element	Some kind of physical marker (such as a paper strip or a yellow paper sheet) that separates the components.
<i>Number</i> ?	integer	Number of sheets that make up one component.
<i>Offset</i> ?	XYPair	Offset dimension in X- and Y-dimensions that separates the components.
<i>OffsetAmount</i> ?	integer	The number of components that are shifted in <i>OffsetDirection</i> simultaneously.
<i>OffsetDirection</i> ?	enumeration	Offset-shift action for the first component. Possible values are: <i>left</i> <i>right</i> <i>straight</i> <i>alternate</i>
<i>Overfold</i> ?	double	Expansion of the overfold of a sheet. This attribute may be needed for the Inserting process.
IdentificationField +	element	Marks that identify the range of sheets to be used in a process. A scanner will scan the sheets and detect a component boundary by scanning a mark, such as a barcode, that matches the description in the IdentificationField element.

7.2.34 DividingParams

This resource contains attributes and elements used in executing the **Dividing** process.

Resource Properties

Resource class: Parameter
 Resource referenced by: -
 Partition: -
 Input of processes: **Dividing**
 Output of processes: -

Resource Structure

Name	Data Type	Description
<i>DivideDistance</i>	number	Distance between the cross cuts.

7.2.35 Employee

Information about a specific device or machine operator (see section 3.6.1.3 Implementation Resources). **Employee** is also used to describe the contact person who is responsible for executing a node, as defined in the `NodeInfo` field of a JDF node.

Resource Properties

Resource class: Implementation, Element
 Resource referenced by: -
 Partition: -
 Input of processes: -
 Output of processes: -

Resource Structure

Name	Data Type	Description
<i>PersonID</i> ?	string	ID of the relevant MIS employee.
<i>Shift</i> ?	string	Defines the shift to which the employee belongs.
<i>CostCenter</i> ?	element	MIS cost center ID.
<i>Person</i> ?	element	Describes the employee. If no Person element is specified, the Employee resource represents any employee who fulfills the selection criteria.

7.2.36 EndSheetGluingParams

This resource describes the attributes and elements used in executing the **EndSheetGluing** process.

Resource Properties

Resource class: Parameter
 Resource referenced by: -
 Partition: -
 Input of processes: **EndSheetGluing**
 Output of processes: -

Resource Structure

Name	Data Type	Description
EndSheet (Front)	element	Information about the front-end sheet. The <i>Side</i> attribute of this element must be <i>Front</i> .
EndSheet (Back)	element	Information about the back-end sheet. The <i>Side</i> attribute of this element must be <i>Back</i> .

Structure of EndSheetGluingParams Elements

EndSheet

Name	Data Type	Description
<i>Offset</i>	XYPair	Offset of end sheet in X and Y direction.

<i>Side</i>	enumeration	Location of the end sheet. Possible values are: <i>Front</i> <i>Back</i>
GlueLine	element	Description of the glue line.

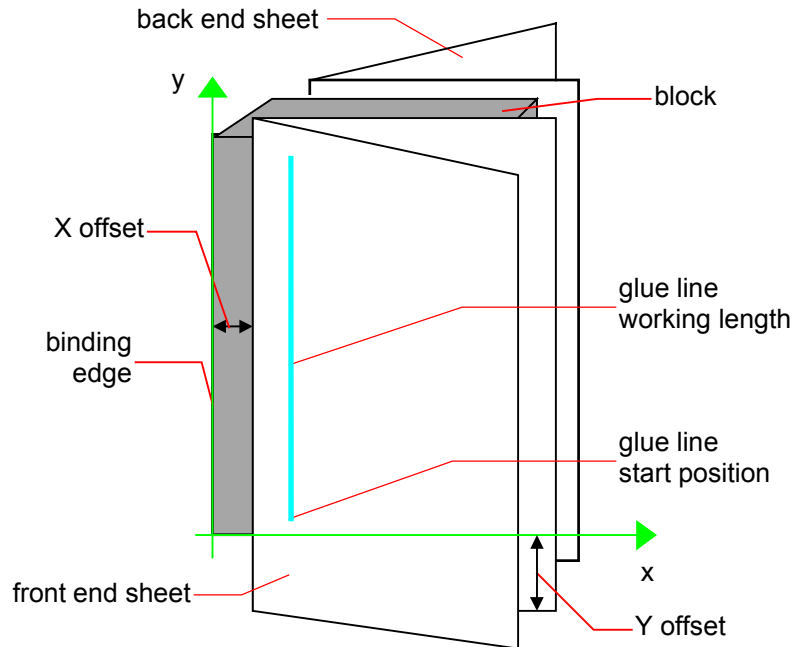


Figure 7.8 Parameters and coordinate system used for end-sheet gluing

The process coordinate system is defined as follows: The y-axis is aligned with the binding edge of the book block. It increases from the registered edge to the edge opposite to the registered edge. The x-axis is aligned with the registered edge. It increases from the binding edge to the edge opposite the binding edge (i.e. the product front edge).

7.2.37 ExposedMedia

This resource represents a processed **Media**-based handling resource such as film, plate or paper proof. It is also used as an input resource for the **Scanning** process.

Resource Properties

Resource class:	Handling
Resource referenced by:	-
Partition:	<i>Separation, Side, SheetName, SignatureName, TileID</i>
Input of processes:	<i>FilmToPlateCopying, ConventionalPrinting, PreviewGeneration, DigitalPrinting, IDPrinting, Scanning, Proofing</i>
Output of processes:	<i>ImageSetting, FilmToPlateCopying, Proofing</i>

Resource Structure

Name	Data Type	Description
------	-----------	-------------

<i>ColorType ?</i>	enumeration	Possible values are: <i>Color</i> <i>GreyScale</i> <i>BW</i> – Black and white.
<i>OutputProfile ?</i>	URL	ICC Profile of the output process for which this media was exposed.
<i>Polarity ?</i>	boolean	<i>False</i> if the media contains a negative image.
<i>ProofIntent ?</i>	enumeration	This attribute is present if the ExposedMedia resource describes a proof. Possible values are: <i>none</i> – Not a proof or don't know. the default. <i>halftone</i> – the halftones are emulated. <i>contone</i> – No halftones but exact color. <i>conceptual</i> – Color does not match precisely.
<i>PunchType ?</i>	string	Name of the registration punch scheme. Possible values include, but are not limited to: <i>Bacher</i> <i>Stoesser</i> Defaults to no punch holes.
<i>Resolution ?</i>	XYPair	Resolution of the output.
Media	element	Describes media specifics such as size and type.
ScreeningParams ?	element	Used to describe the screening in case of rasterized media

7.2.38 FileSpec

Specification of a file or a set of files.

Resource Properties

Resource class:	Element
Resource referenced by:	DBMergeParams, LayoutElement, PDLResourceAlias, ScanParams
Partition:	<i>Separation</i>
Input of processes:	-
Output of processes:	-

Name	Data Type	Description
<i>Application ?</i>	string	Creator application, such as Photoshop.
<i>AppOS ?</i>	enumeration	Operating system of the application that created the file. Possible values are: <i>Unknown</i> – Default value <i>Mac</i> <i>Windows</i> <i>Linux</i> <i>Solaris</i>

		<i>IRIX</i>
		<i>DG_UX</i>
		<i>HP_UX</i>
<i>AppVersion</i> ?	string	Version of the value of the <i>Application</i> attribute.
<i>Compression</i> ?	enumeration	Indicates how the file is compressed. Possible values are: <i>none</i> – Default value. The file is not compressed. <i>Deflate</i> – The file is compressed using ZIP public domain compression (RFC 1951) <i>gzip</i> – GNU zip compression technology (RFC 1952) <i>compress</i> –UNIX compression (RFC 1977)
<i>Disposition</i> ?	enumeration	Indicates what the device should do with the file when the process that uses this resource as an input resource completes. Possible values are: <i>Unlink</i> – The device should release the file. <i>Delete</i> – The device should attempt to delete the file. <i>Retain</i> – Default value. The device should do nothing with the file.
<i>FileFormat</i> ?	string	A formatting string used with the <i>Template</i> attribute to define a sequence of filenames in a batch process. If <i>FileName</i> is not present, both <i>FileFormat</i> and <i>FileTemplate</i> must be present. For more information, see the text following this table.
<i>FileName</i> ?	URL	Location of the file. This value is ignored if the referencing resource is a pipe.
<i>FileTemplate</i> ?	string	A template, used with <i>FileFormat</i> , to define a sequence of filenames in a batch process. If <i>FileName</i> is not present, both <i>FileFormat</i> and <i>FileTemplate</i> must be present.
<i>MimeType</i> ?	string	Mime type of the file.
<i>OSVersion</i> ?	string	Version of the operating system.
<i>FileAlias</i> *	element	Defines a set of mappings between file names that may occur in the document and URIs (which may refer to external files or parts of a MIME message).

Structure of FileAlias Sub-element

Name	Data Type	Description
<i>Alias</i>	string	The filename which is expected to occur in the file.
<i>Disposition</i>	enumeration	Indicates what the device should do with the file referenced by this alias when the process that uses this resource as an input resource completes. Possible values are: <i>Unlink</i> – The device should release the file. <i>Delete</i> – The device should attempt to delete the file.

		<i>Retain</i> – The device should do nothing with the file.
<i>MimeType ?</i>	string	Mime type of the file.
<i>URI</i>	uri	The uri which identifies the file the alias refers to.

Usage of Format and Template

The function defined when using the attributes *FileFormat* and *FileTemplate* is drawn from the same root as the standard C print function, and therefore overtly resembles the model of that function. *FileFormat* is the first argument and *FileTemplate* is a comma-separated list of the additional arguments. *FileTemplate* may contain the following operators : +, -, *, /, %, (,) which are evaluated using standard C-operator precedence and the variables defined in the following table:

Table 7.3 Predefined variables used in *FileTemplate*

Name	Description
<i>i</i>	Integer iterator over all files produced by this process. 0-based numbering.
<i>r</i>	Integer iterator over all Runs in an input RunList .
<i>ri</i>	Integer iterator over all files in an input Run of a RunList .
<i>sep</i>	Separation as defined in the separation element of a partitioned resource.
<i>surf</i>	Surface string, “Front” or “Back”
<i>SheetName</i>	SheetName string of a partitioned resource.
<i>SignatureName</i>	SignatureName string of a partitioned resource.
<i>TileX</i>	X coordinate of a Tile
<i>TileY</i>	Y coordinate of a Tile
<i>PartVersion</i>	PartVersion string of a partitioned resource.
<i>jobPartID</i>	JobPartID string
<i>jobID</i>	Job ID string
<i>Time</i>	current <i>Time</i> in ISO format.
<i>Date</i>	current <i>Date</i> in ISO format.
<i>CustomerID</i>	CustomerID

Example:

```
<FileSpec FileFormat = "file://here/next/%s/%4.i/m%4.i.pdf"
FileTemplate = "JobID,i/100,i%100"/>
```

with JobID = “j001” and a **RunList** defining 2023 created files will iterate all created files and place them into:

```
"file://here/next/j001/0000/m0000.pdf"
...
"file://here/next/j001/0002/m0023.pdf"
```

7.2.39 FoldingParams

This resource describes the folding parameters, including the sequence of folding steps. It is also possible to execute the predefined steps of the folding catalog. The following six applications of a folding device are possible:

- folding
- cutting
- creasing
- gluing
- perforating
- thread sealing

The cutting information contained in this section is intended only for the cutting procedures within the folding equipment, not for cut blocks. Although additional cutting information can be defined using cut blocks, this resource does not apply to cut-block specification.

At the beginning of a folding procedure definition it is necessary to specify the size of the input sheet in the *FoldSheetIn* attribute. If the specified size does not match the size of the *X* and *Y* dimensions of the input component, all coordinates of the folding procedure are scaled accordingly. This allows for the specification of a folding procedure that can be used with different sizes.

After each folding or cutting step of a folding procedure, the origin of the coordinate system is moved to the lower left corner of the intermediate folding product.

The specification of reference edges (*Front*, *Rear*, *Left* and *Right*) for the description of an operation (such as the positioning of a tool) is done by means of determined names. These names are case-sensitive; they must be written exactly as shown in figure 7.9, below.

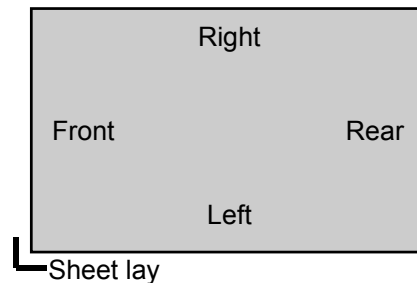


Figure 7.9 Names of the reference edges of a sheet in the *FoldingParams* resource

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	Folding
Output of processes:	Imposition

Resource Structure

Name	Data Type	Description
<i>DescriptionType</i>	enumeration	How the folding operations are described. Possible

		values are: <i>FoldProc</i> <i>FoldCatalog</i>
<i>FoldCatalog ?</i>	string	Description of the type of fold according to the folding catalog. Required when <i>DescriptionType = FoldCatalog</i> .
<i>FoldSheetIn ?</i>	XYPair	Input sheet format. Required when <i>DescriptionType = FoldProc</i> .
<i>SheetLay</i>	enumeration	Lay of input media. Possible values are: <i>Left</i> <i>Right</i>
FoldOperation*	element	Steps of folding operation in the sequence in which the steps should be carried out. Required when <i>DescriptionType = FoldProc</i> .

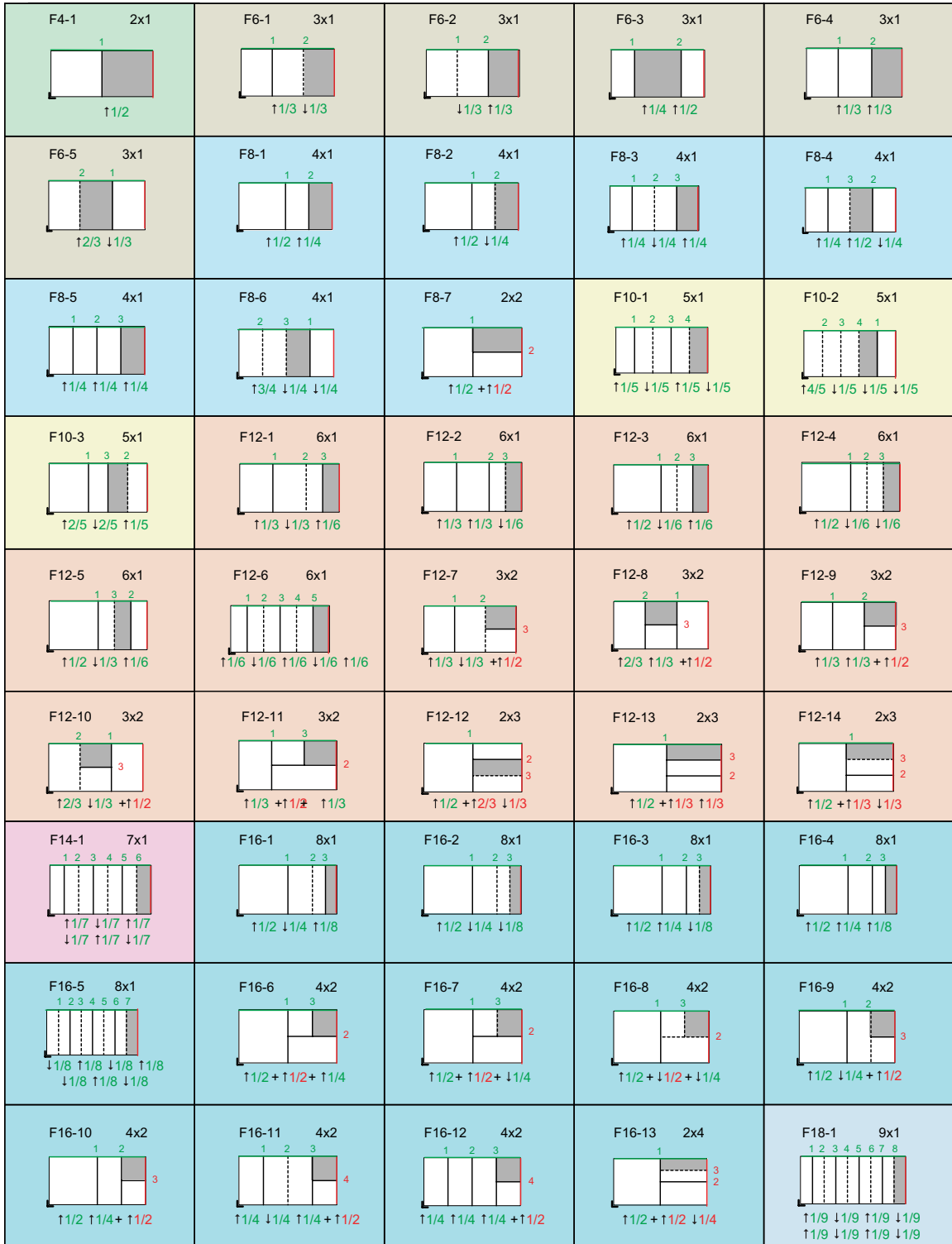


Figure 7.10 FoldCatalog part 1

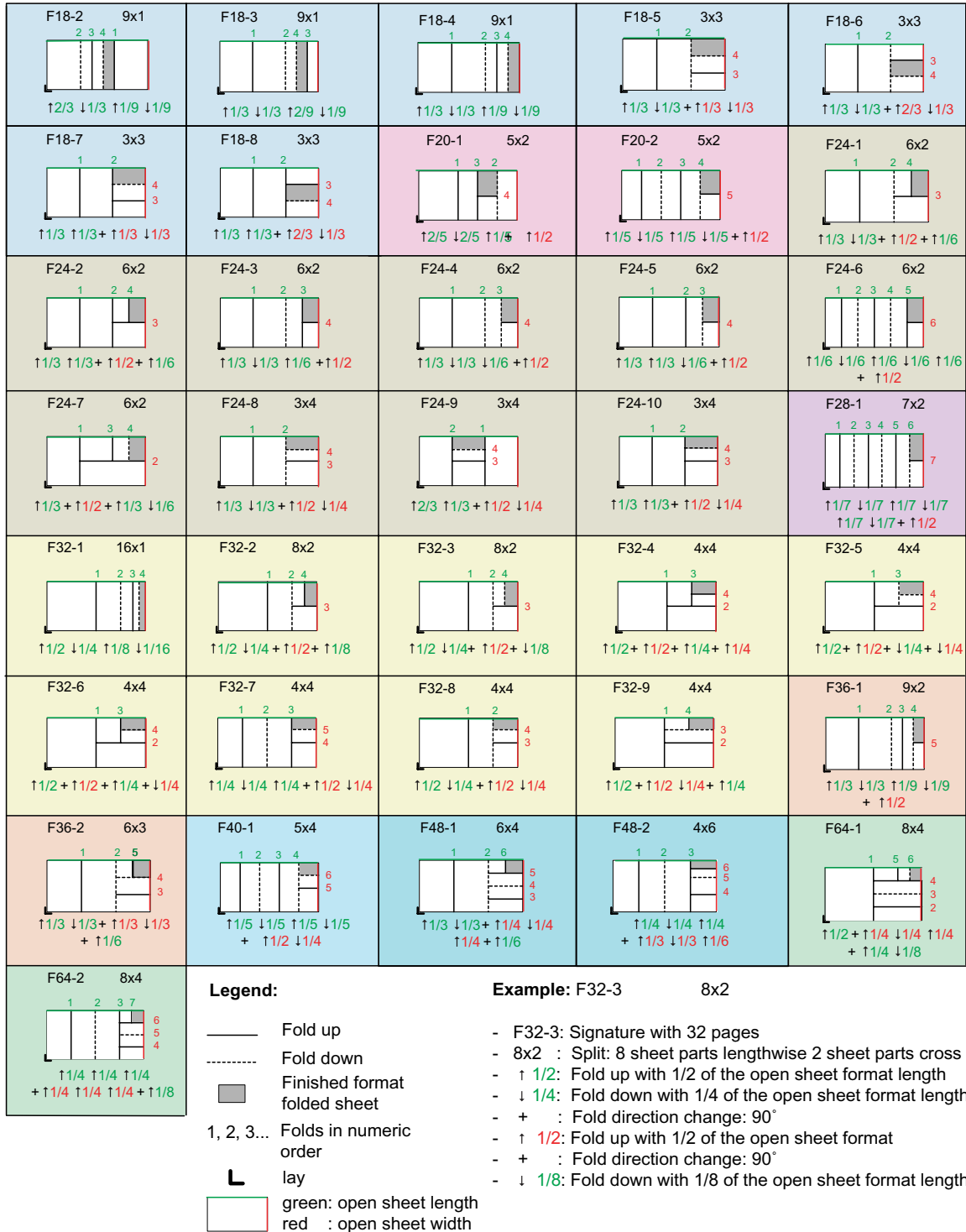


Figure 7.11 FoldCatalog part 2

Structure of FoldingParams Elements

FoldOperation

Resource class: Abstract element

FoldOperation describes the operations that can be executed by the various modules of a folding machine. The sequence is relevant.

FoldOperation	Description
Fold	Make one straight fold of a component.
Cut	Make one cut
Crease	Make one crease
Glue	Add a glue line
Perforate	Perforating the Component
ThreadSeal	Perform a thread sealing operation

Fold

Resource class: FoldOperation

Name	Data Type	Description
<i>From</i>	enumeration	Edge from which the page is folded. Possible values are: <i>Front</i> <i>Left</i>
<i>To</i>	enumeration	Direction in which it is folded. Possible values are: <i>Up</i> – upwards <i>Down</i> – downwards
<i>Travel</i>	double	Distance of the reference edge relative to <i>From</i>

Cut

Resource class: FoldOperation

Name	Data Type	Description
<i>StartPosition</i>	XYPair	Starting position of the tool.
<i>WorkingPath</i>	XYPair	Relative working path of the tool. Since the tools can only work parallel to the edges, one coordinate must be zero.
<i>WorkingDirection</i>	enumeration	Direction from which the tool is working. Possible values are: <i>Top</i> – from above <i>Bottom</i> – from below

Crease

Resource class: FoldOperation

Name	Data Type	Description
<i>StartPosition</i>	XYPair	Starting position of the tool.

<i>WorkingPath</i>	XYPair	Relative working path of the tool. Since the tools can only work parallel to the edges, one coordinate must be zero.
<i>WorkingDirection</i>	enumeration	Direction from which the tool is working. Possible values are: <i>Top</i> –from above <i>Bottom</i> –from below

Glue

Resource class: FoldOperation

Name	Data Type	Description
<i>WorkingDirection</i>	enumeration	Direction from which the tool is working. Possible values are: <i>Top</i> –from above <i>Bottom</i> –from below
GlueLine	element	Description of the glue line.

Perforate

Resource class: FoldOperation

Name	Data Type	Description
<i>StartPosition</i>	XYPair	Starting position of the tool.
<i>WorkingPath</i>	XYPair	Relative working path of the tool. Since the tools can only work parallel to the edges, one coordinate must be zero.
<i>WorkingDirection</i>	enumeration	Direction from which the tool is working. Possible values are: <i>Top</i> –from above <i>Bottom</i> –from below

ThreadSeal

Resource class: FoldOperation

Name	Data Type	Description
<i>ThreadMaterial ?</i>	enumeration	Thread material. Possible values are: <i>Cotton</i> <i>Nylon</i> <i>Polyester</i>
<i>ThreadPositions</i>	NumberList	Array containing the Y-coordinate of the center positions of the thread.
<i>ThreadLength</i>	double	Length of one thread.
<i>ThreadStitchWidth</i>	double	Width of one stitch.

SealingTemperature ? integer Temperature needed for sealing thread and sheets together (in degree centigrade).

7.2.40 FontParams

This resource describes how fonts shall be handled when converting PostScript files to PDF.

Resource Properties

Resource class: Parameter
 Resource referenced by: -
 Partition: -
 Input of processes: *PSToPDFConversion*
 Output of processes: -

Resource Structure

Name	Data Type	Description
<i>AlwaysEmbed ?</i>	NMTOKENS	One or more names of fonts that are always to be embedded in the PDF file. Each name must be the PostScript language name of the font. An entry that occurs in both the <i>AlwaysEmbed</i> and <i>NeverEmbed</i> lists constitutes an error.
<i>CannotEmbedFontPolicy ?</i>	enumeration	Determines what occurs when a font cannot be embedded. Possible values are: <i>Error</i> – Log an error and abort the process if any font can not be found or embedded. <i>Warning</i> – Warn and continue if any font cannot be found or embedded. <i>OK</i> – Continue without warning or error if any font can not be found or embedded.
<i>EmbedAllFonts ?</i>	boolean	If <i>true</i> , specifies that all fonts, except those in the <i>NeverEmbed</i> list, are to be embedded in the PDF file.
<i>EmbedFontPolicy ?</i>	enumeration	Determines what occurs when a font cannot be embedded. Possible values are: <i>Error</i> – Abort the process and record an error (in an <i>Audit</i> record or via a message) if any font can not be found or embedded. <i>Warning</i> – Continue the process but record a warning record an error (in an <i>Audit</i> record or via a message) if any font cannot be found or embedded. <i>OK</i> – Continue without warning or error if any font can not be found or embedded.
<i>MaxSubsetPct ?</i>	integer	The maximum percentage of glyphs in a font that can be used before the entire font is embedded instead of a subset. This value is only used if <i>SubsetFonts = true</i> .

<i>NeverEmbed</i> ?	NMTOKENS	One or more names of fonts that are never to be embedded in the PDF file. Each name must be the PostScript language name of the font. An entry that occurs in both the <i>AlwaysEmbed</i> and <i>NeverEmbed</i> lists constitutes an error.
<i>SubsetFonts</i> ?	boolean	If <i>true</i> , font subsetting is enabled. If <i>false</i> , it is not. Font subsetting embeds only those glyphs that are used, instead of the entire font. This reduces the size of a PDF file that contains embedded fonts. If font subsetting is enabled, the decision whether to embed the entire font or a subset is determined by number of glyphs in the font that are used, and the value of <i>MaxSubsetPct</i> . Note: Embedded instances of multiple master fonts are always subsetted, regardless of the setting of <i>SubsetFonts</i> . The <i>AlwaysEmbed</i> and <i>NeverEmbed</i> fonts lists are restored to their default values between each job.

7.2.41 FontPolicy

This resource defines the policies that devices must follow when font errors occur while PDL files are being processed. When fonts are referenced by PDL files but are not provided, devices may provide one of the following two fallback behaviors:

1. The device may provide a standard default font which is substituted whenever a font cannot be found.
2. The device may provide an emulation of the missing font.

If neither fallback behavior is requested (i.e., both *UseDefaultFont* and *UseFontEmulation* are false), then the job shall fail if a referenced font is not provided.

FontPolicy allows jobs to specify whether or not either of these fallback behaviors should be employed when missing fonts occur.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	<i>Interpreting, PDFToPSConversion</i>
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>PreferredFont</i>	NMTOKEN	The name of a font to be used as the default font for this job. It is not an error if the device cannot use the specified font as its default font.
<i>UseDefaultFont</i>	boolean	If <i>true</i> , the device shall resort to a default font if a font cannot be found. This is the normal behavior of the PostScript interpreter, which defaults to courier when a

font cannot be found.

UseFontEmulation boolean If *true*, the device shall emulate a required font if a font cannot be found.

7.2.42 GatheringParams

This resource contains the attributes of the **Gathering** process.

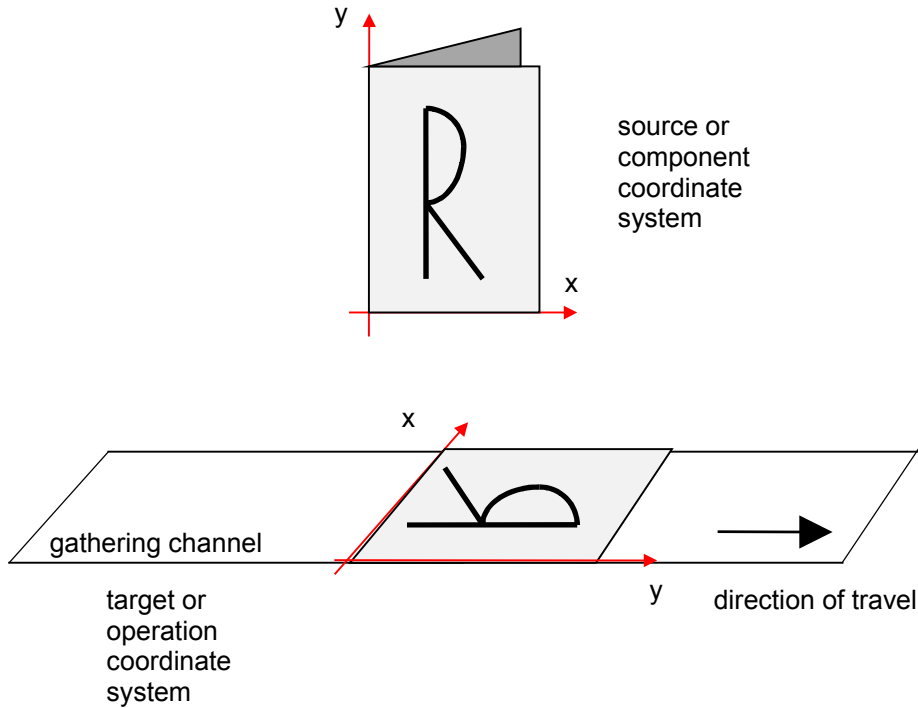


Figure 7.12 Coordinate system used for gathering

Resource Properties

Resource class: Parameter
 Resource referenced by: -
 Input of processes: **Gathering**
 Output of processes: -

Resource Structure

Name	Data Type	Description
<i>Disjointing?</i>	element	Description of the separation properties between individual components on a gathered pile.

7.2.43 GlueLine

This resource provides the information to determine where and how to apply glue.

Resource Properties

Resource class:	Element
Resource referenced by:	AdhesiveBindingParams, EndSheetGluingParams, FoldingParams, InsertingParams, ThreadSewingParams
Partition:	-
Input of processes:	-
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>StartPosition</i>	XYPair	Start position of glue line. The start position is given in the coordinate system of the mother sheet.
<i>WorkingPath</i>	XYPair	Relative working path of the gluing tool.
<i>GlueType</i> ?	enumeration	Glue type. Possible values are: <i>ColdGlue</i> <i>Hotmelt</i> <i>PUR</i> – Polyurethane
<i>GlueBrand</i> ?	string	Glue brand.
<i>GluingPattern</i> ?	XYPair	Glue line pattern defined by the length of a glue line segment (X element) and glue line gap (Y element). A solid line is expressed by the pattern (1 0).
<i>GlueLineWidth</i> ?	double	Width of the glue line.
<i>MeltingTemperature</i> ?	integer	Required temperature for melting the glue (in degrees centigrade). Use only when <i>GlueType</i> = <i>Hotmelt</i> .

7.2.44 HoleMakingParams

This resource specifies where to make a hole of what shape in components. This information is used by the **HoleMaking** process.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	HoleMaking
Output of processes:	-

Resource Structure

Name	Data Type	Description
Hole *	element	Array of all Hole elements.

Structure of HoleMakingParams Elements

Hole

Name	Data Type	Description
<i>Shape</i>	enumeration	Shape of the hole. Possible values are: <i>round</i> <i>rectangular</i>
<i>Center</i>	XYPair	Position of the center of the hole relative to the Component coordinate system. For more information, see section 6.5.2.
<i>Extent</i>	XYPair	Size of the hole in pt. If <i>Shape</i> is <i>round</i> , only the first entry of <i>Extent</i> is evaluated and defines the hole diameter.

7.2.45 IdentificationField

This resource contains information about a mark on a document, such as a bar code, used for OCR-based verification purposes or document separation.

Resource Properties

Resource class:	Element
Resource referenced by:	Disjointing, Sheet, Surface
Partition:	-
Input of processes:	Verification, Inserting, Collecting, Gathering
Output of processes:	Imposition

Resource Structure

Name	Data Type	Description
<i>BoundingBox ?</i>	rectangle	Box that provides the boundaries in the coordinate system of the mark that indicates where the component is to be placed. If no <i>BoundingBox</i> is defined, the complete visible surface must be scanned for an appropriate bar code.
<i>Encoding</i>	enumeration	Encoding of the information. Possible values are: <i>ASCII</i> – Plain-text font. <i>BarCode1D</i> – One-dimensional barcode. <i>BarCode2D</i> – Two-dimensional barcode.
<i>EncodingDetails</i>	NMTOKEN	Details about the encoding type. An example is the barcode scheme. Possible values are: <i>Code39</i> <i>Interleave25</i> <i>Plessey</i> <i>EAN</i>
<i>Format ?</i>	string	Regular expression ⁴ that defines the format of the expression, such as the number of digits, alphanumeric,

⁴ This is a regular expression as in UNIX `grep`.

		or numeric. Note that this field may also be used define constant fields, such as the end of document markers. Default is that any expression is valid (<i>Format</i> = “*”).
<i>Orientation ?</i>	matrix	Orientation of the contents within the field. The coordinate system is defined in the system of the sheet or component where the IdentificationField resides. Default = unit matrix.
<i>Position</i>	enumeration	Position with respect to the instance document to which the IdentificationField resource refers. Possible values are: <i>header</i> – Sheet before the document. <i>trailer</i> – Sheet after the document. <i>page</i> – A page of the document.
<i>Page ?</i>	integer	If <i>Position</i> = <i>page</i> , this refers to the page where the IdentificationField can be found. Negative values denote an offset relative to the last page in a stack of pages.
<i>Purpose</i>	enumeration	Purpose defines the usage of the field. Possible values are: <i>verification</i> – used for verification of documents. <i>separation</i> – used to separate documents.

7.2.46 IDPrintingParams

This resource contains the parameters needed to control the **IDPrinting** process.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	IDPrinting
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>AttributesNaturalLang</i>	language	Language selected for communicating attributes.
<i>CoverUsage ?</i>	enumeration	Specifies whether covers are to be used for each copy of the whole job or for each copy of an instance document. <i>Job</i> – The covers are for the whole job. <i>Document</i> – The covers are for each instance document within the job. Default = <i>Job</i>
<i>IDPAttributeFidelity</i>	boolean	Indicates whether or not the device must reject the job if there are attribute values or elements that it does not support.

<i>IDPVersion</i>	number	A number indicating the version of the IDP protocol to use when communicating to IDP devices.
<i>OutputBin</i>	enumeration	<p>Specifies the bin to which the finished document should be output. Possible values are:</p> <p><i>Top</i> – The bin that, when facing the device, can best be identified as ‘top’.</p> <p><i>Middle</i> – The bin that, when facing the device, can best be identified as ‘middle’.</p> <p><i>Bottom</i> – The bin that, when facing the device, can best be identified as ‘bottom’.</p> <p><i>Side</i> – The bin that, when facing the device, can best be identified as ‘side’.</p> <p><i>Left</i> – The bin that, when facing the device, can best be identified as ‘left’.</p> <p><i>Right</i> – The bin that, when facing the device, can best be identified as ‘right’.</p> <p><i>Center</i> – The bin that, when facing the device, can best be identified as ‘center’.</p> <p><i>Rear</i> – The bin that, when facing the device, can best be identified as ‘rear’.</p> <p><i>FaceUp</i> – The bin that can best be identified as ‘face up’ with respect to the device.</p> <p><i>FaceDown</i> – The bin that can best be identified as ‘face down’ with respect to the device.</p> <p><i>LargeCapacity</i> – The bin that can best be identified as the ‘large capacity’ bin (in terms of the number of sheets) with respect to the device.</p> <p><i>FitMedia</i> – Requests the device to select a bin based on the size of the media.</p>
<i>PageDelivery</i>	enumeration	<p>Indicates how pages are to be delivered to the output bin or finisher. Possible values are:</p> <p><i>same-order-face-up</i> – Order as defined by the RunList, with the ‘front’ sides of the media up.</p> <p><i>same-order-face-down</i> – Order as defined by the RunList, with the ‘front’ sides of the media up.</p> <p><i>reverse-order-face-up</i> – Order reversed, as defined by the RunList, with the ‘front’ sides of the media up.</p> <p><i>reverse-order-face-down</i> – Order reversed, as defined by the RunList, with the ‘front’ sides of the media down.</p> <p><i>system-specified</i> – Order and face-up/face-down as defined by the system.</p> <p>Default is system-specified.</p>
<i>PageDelivery</i>	enumeration	<p>Indicates how pages are to be delivered to the output bin or finisher. Possible values are:</p> <p><i>same-order-face-up</i></p>

		<i>same-order-face-down</i>
		<i>reverse-order-face-up</i>
		<i>reverse-order-face-down</i>
		<i>system-specified</i>
<i>SheetCollate</i>	boolean	Determines whether the sequencing of the pages in the output of the job. If <i>true</i> , pages for each copy of the document are sequenced together, followed by the pages for the next copy. If <i>false</i> , all copies of the first page are sequenced, followed by the second and subsequent pages. <i>SheetCollate</i> describes the order of the final pages, but does not prescribe the order in which they are produced.
Cover *	element	0, 1 or 2 Cover elements.
IDPFinishing *	element	These elements provide the details of how media should be finished.
IDPLayout ?	element	This element provides the details of how page contents shall be imaged onto media.
IDPMediaIntent	element	An IDPMediaIntent element. This element describes the media to be used for the job. This element is ignored if a Media resource is present and can be honored for the IDPrinting process.

Structure of the Cover Sub-element

This element describes the cover requested for the job. Covers may be applied to the whole job, or to each instance document in the job. Note that front and back covers may be specified.

Name	Data Type	Description
<i>CoverType</i> ?	enumeration	Specifies whether this Cover element specifies the front or back cover. <i>Front</i> – The front cover. <i>Back</i> – The back cover. Default = <i>Front</i> .
<i>MediaName</i> ?	NMTOKEN	A name that identifies the media to the device. Ignored if Media is present.
IDPMediaIntent	element	An IDPMediaIntent element. This element describes the media to be used for the job. This element is ignored if a Media resource is present and can be honored for the IDPrinting process.
Media ?	element	A Media resource which identifies the media to be used for the cover. If Media is specified, <i>MediaName</i> is ignored.

Structure of abstract IDPFinishing Element

IDPFinishing elements describe finishing operations that should be applied to sets of pages that are output by the **IDPrinting** process. Operation-specific elements are derived from the abstract definition.

Additional derived elements are expected to be defined over time.

Also, more detail will be added to the currently defined elements.

Name	Data Type	Description
<i>Pages ?</i>	IntegerRange-List	Identifies a set of pages in the RunList for the process that this finishing operation should be applied to. If this attribute is absent, the operation shall be applied to all pages

Structure of the IDPLayout Sub-element

This element provides details describing how page contents are to be imaged onto media.

Name	Data Type	Description
<i>Border</i>	number	A real number that indicates the width of a border which will be drawn around the page images on the media. Default = 0, meaning that no border will be drawn.
<i>Duplex</i>	enumeration	Indicates how pages should be imposed onto sides of the medium. Possible values are: <i>OneSided</i> – Page contents will only be imaged on one side of the media. <i>LongEdge</i> – Impose consecutive pages upon the front and back sides of media sheets so that the orientation of the pages on each side is appropriate for binding along the long edge. Equivalent to ‘ <i>work-and-turn</i> ’. <i>ShortEdge</i> – Impose consecutive pages upon the front and back sides of media sheets so that the orientation of the pages on each side is appropriate for binding along the short edge. Equivalent to ‘ <i>work-and-tumble</i> ’.
<i>ForceFrontSide</i>	NumberRange-List	A set of numbers which identify a set of pages in the RunList that should always be imaged on the front side of a piece of media.
<i>LeftToRight</i>	boolean	Indicates whether page images should be imaged along the major axis from left to right.
<i>MajorAxis</i>	enumeration	Indicates which axis shall be treated as the major axis for Number-Up printing. Possible values are: <i>xFirst</i> – Treat the edge of the media that matches the first entry in the MediaSize pair as the major axis. <i>yFirst</i> – Treat the edge of the media that matches the second entry in the MediaSize pair as the major axis.
<i>NumberUp</i>	XYPair	The number of pages to impose onto a single side of media. The first number specifies the number of page images for each row along the major axis as defined by the <i>MajorAxis</i> attribute. The second number specifies the number of page images for each column across the major

<i>Orientation</i>	enumeration	<p>axis as defined by the <i>MajorAxis</i> attribute.</p> <p>Describes the desired orientation of printed pages of the job. Possible values are:</p> <p><i>Portrait</i> – The content will be imaged across the short edge of the medium</p> <p><i>Landscape</i> – The content will be imaged across the long edge of the medium. Landscape is defined to be a rotation of the page contents to be imaged by +90 degrees (i.e., anti-clockwise) with respect to the medium.</p> <p><i>ReverseLandscape</i> – The content will be imaged across the long edge of the medium. Landscape is defined to be a rotation of the page contents to be imaged by -90 degrees (i.e., clockwise) with respect to the medium.</p> <p><i>ReversePortrait</i> – The content will be imaged across the short edge of the medium. Landscape is defined to be a rotation of the page contents to be imaged by +180 degrees anti-clockwise with respect to the medium.</p>
<i>Rotate</i>	number	A number of degrees which the page contents are to be rotated prior to being imaged onto page contents. A positive value is taken to mean an counter-clockwise rotation.
<i>TemplateName</i>	NMTOKEN	The name of an imposition template that the device is expected to recognize. If present, all other attributes in this element are ignored.
<i>TopToBottom</i>	boolean	Indicates whether page images should be imaged across the major axis from top to bottom.

Structure of the IDPMediaIntent Sub-element


This element provides describing the intended media for the job. This element is used as an alternative to the **Media** resource which may be provided for the *IDPrinting* process or within IDPOverride sub-elements. It is ignored if the **Media** resource is present and can be fulfilled.

Name	Data Type	Description
<i>MediaHoles</i>	integer	The number of pre-drilled holes the media should have.
<i>MediaLabel</i>	NMTOKEN	The label characteristics of the media. Possible values include: <i>none</i> – Not labeled stock. <i>standard</i> – The site-defined standard labeled stock.
<i>MediaTabs</i>	enumeration	Indicates whether the media should have tabs. Possible values are: <i>none</i> – No tabs. <i>pre-cut</i> – Extend only partially along the edge. <i>full-cut</i> – Extend the entire length of the edge.
MediaIntent	element	A MediaIntent resource which specifies the general characteristics of the media.

Structure of the IDPOverride Sub-element

IDPOverride elements describe overrides to the controls specified for the whole process in the IDPrintingParams resource.

Overrides may apply to a set of input pages or documents, or to a set of output surfaces, sheets of media, or to instance documents in a personalized printing job. Note that if more than one override refers to the same content, the lowest level override takes precedence. That is, page overrides supercede document overrides, and **Surface** overrides supercede **Sheet** overrides. If both input and output overrides are specified for the same content, the output overrides take precedence.

Name	Data Type	Description
<i>Copies ?</i>	integer	This value indicates the number of documents (when OverrideScope is either <i>InputDocs</i> or <i>OutputDocs</i>) or pages (when OverrideScope is <i>InputPages</i>) to include in each copy of the job. <i>Copies</i> is ignored when OverrideScope is <i>OutputSheets</i> or <i>OutputSurfaces</i> .
<i>Docs ?</i>	IntegerRange-List	Only meaningful if OverrideScope is <i>InputDocs</i> or <i>OutputDocs</i> . Identifies a set of documents to which the overrides apply.
<i>IDPAttributeFidelity</i>	boolean	Indicates whether or not the device must reject the job if there are attribute values or elements that it does not support for the scope of this override.
<i>OverrideScope</i>	enumeration	Indicates how the overrides shall be applied. Possible values are: <i>InputDocs</i> – The overrides are to be applied to a set of input pages in the RunList. <i>InputPages</i> – The overrides are to be applied to a set of input pages in the RunList.  <i>OutputSheets</i> – The overrides are to be applied to a set of sheets of physical media to be produced. <i>OutputSurfaces</i> – The overrides are to be applied to a set of surfaces of physical media to be produced.[BW39] <i>OutputDocs</i> – The overrides are to be applied to the set of pages which comprise the output documents to be produced. Note that these are instance documents in a personalized printing job.
<i>PageDelivery</i>	enumeration	Indicates how paages for the scope of this override are to be delivered to the output bin or finisher. Possible values are: <i>same-order-face-up</i> – Order as defined by the RunList, with the ‘front’ sides of the media up. <i>same-order-face-down</i> – Order as defined by the RunList, with the ‘front’ sides of the media up. <i>reverse-order-face-up</i> – Order reversed, as defined by the RunList, with the ‘front’ sides of the media up. <i>reverse-order-face-down</i> – Order reversed, as defined by the RunList, with the ‘front’ sides of the media down. <i>system-specified</i> – Order and face-up/face-down as

		defined by the system. Default is <i>system-specified</i> .
<i>Pages ?</i>	IntegerRange-List	Only meaningful if OverrideScope is <i>InputPages</i> , <i>OutputSheets</i> or <i>OutputSurfaces</i> . Identifies a set of pages in the input RunList or a set of sheets or surfaces of finished media to which the overrides apply.
<i>SheetCollate</i>	boolean	Determines whether the sequencing of the pages in the output of the job. If <i>true</i> , pages for each copy of the document are sequenced together, followed by the pages for the next copy. If <i>false</i> , all copies of the first page are sequenced, followed by the second and subsequent pages. <i>SheetCollate</i> describes the order of the final pages, but does not prescribe the order in which they are produced.
<i>Cover *</i>	element	0, 1 or 2 Cover elements. Note that these cover elements, by definition, are to be used for instance documents. It is an error to provide Cover elements in the DocumentOverrides element when <i>CoverUsage</i> in the IDPrintingParams is <i>Job</i> .
<i>IDPFinishing *</i>	element	These elements provide the details of how media within the scope of this override should be finished.
<i>IDPLayout ?</i>	element	This element provides the details of how page contents within the scope of this override shall be imaged onto media.
<i>IDPMediaIntent</i>	element	An IDPMediaIntent element. This element describes the media to be used for the for the scope of this override. This element is ignored if a Media resource is present and can be honored for this IDPOverride element.
<i>Media</i>	element	A Media resource which specifies the media to be used for the scope of this override.

Structure of Stitching Sub-element

This element describes the stitching requested for a set of pages in the document.

Name	Data Type	Description
<i>StitchingLocations ?</i>	IntegerRange-List	A list of absolute offsets along the Stitching Axis at which a stitch MUST occur. MUST be in increasing values. Units are millimeters.
<i>StitchingOffset ?</i>	integer	The perpendicular distance of the stitching axis from the stitching reference edge, in units of one hundredth of a millimeter (1/2540th of an inch).
<i>StitchingPosition ?</i>	enumeration	Specifies the location for stitching. All locations are interpreted as if the document were a portrait document. Ignored if <i>StitchingOffset</i> and <i>StitchingPostions</i> are present. Possible values are: <i>None</i> – The document is not to be stitched.

TopLeft – Bind the document with one or more staples in the top left corner.

BottomLeft – Bind the document with one or more staples in the Bottom left corner.

TopRight – Bind the document with one or more staples in the top right corner.

BottomRight – Bind the document with one or more staples in the bottom right corner.

LeftEdge – Bind the document with one or more staples across the left edge.

TopEdge – Bind the document with one or more staples across the top edge.

RightEdge – Bind the document with one or more staples across the right edge.

BottomEdge – Bind the document with one or more staples across the bottom edge.

DualLeftEdge – Bind the document with two staples across the left edge.

DualTopEdge – Bind the document with two staples across the top edge.

DualRightEdge – Bind the document with two staples across the right edge.

DualBottomEdge – Bind the document with two staples across the bottom edge.

StitchingReference-Edge ?

enumeration

The edge of the output media relative to which the stapling or stitching MUST be applied. Possible values are:

bottom – The bottom edge coincides with the x-axis of the coordinate system.

top – The top edge is opposite and parallel to the bottom edge.

left – The left edge coincides with the y-axis of the coordinate system.

right – The right edge is opposite and parallel to the left edge.

7.2.47 ImageCompressionParams

This resource provides information describing how images are to be compressed in PDF files.

Resource Properties

Resource class:	Parameter
Resource referenced by:	Sheet
Partition:	-
Input of processes:	<i>PSToPDFConversion</i>
Output of processes:	-

Resource Structure

Name	Data Type	Description
ImageCompression *	element	Specifies how images are to be compressed.

Structure of ImageCompression Sub-element

Name	Data Type	Description
<i>AntiAliasImages</i> ?	boolean	<p>If <i>true</i>, anti-aliasing is permitted on images. If <i>false</i>, anti-aliasing is not permitted.</p> <p>Anti-aliasing increases the number of bits per component in downsampled images to preserve some of the information that is otherwise lost by downsampling. Anti-aliasing is only performed if the image is actually downsampled and if <i>ImageDepth</i> has a value greater than the number of bits per color component in the input image.</p> <p>Default = <i>false</i></p>
<i>AutoFilterImages</i> ?	boolean	<p>Used only if <i>EncodeColorImages</i> is <i>true</i>. This attribute is not used if <i>ImageType</i> = <i>monochrome</i>.</p> <p>If <i>true</i>, the <i>DCTEncode</i> filter is applied to photos and the <i>FlateEncode</i> filter is applied to screen shots. If <i>false</i>, the <i>ImageFilter</i> compression method is applied to all images. This parameter is ignored for monochrome images.</p> <p>Default = <i>true</i></p>
<i>ConvertImagesToIndexed</i> ?	boolean	<p>If <i>true</i>, the application converts images that use fewer than 257 colors to an indexed colorspace for compactness. This attribute is used only when <i>ImageType</i> = <i>color</i>.</p>
<i>DCTQuality</i> ?	number	<p>A value between 0 and 1 that indicates ‘how much’ the process should compress images when using a <i>DCTEncode</i> filter. 0.0 means ‘do as loss-less compression as possible;’. 1.0 means do the maximum compression possible.’ Default=0;</p>
<i>DownsampleImages</i> ?	boolean	<p>If <i>true</i>, sampled color images are downsampled using the resolution specified by <i>ColorImageResolution</i>. If <i>false</i>, downsampling is not carried out, and the image resolution in the PDF file is the same as that in the source file.</p>
<i>EncodeColorImages</i> ?	boolean	<p>If <i>true</i>, color images are encoded using the compression filter specified by the value of the <i>ColorImageFilter</i> key. If <i>false</i>, no compression filters are applied to color sampled images.</p>
<i>ImageDepth</i> ?	integer	<p>Specifies the number of bits per component in the downsampled image when <i>DownsampleImages</i> = <i>true</i>. Allowed values are 1, 2, 4, and 8 (for 1, 2, 4, and 8 bits per color component) and -1 (which forces the downsampled image to have the same number of bits per sample as the original image.)</p>
<i>ImageDownsample-</i>	number	<p>Sets the image downsample threshold for images. This</p>

<i>Threshold ?</i>		<p>is the ratio of image resolution to output resolution above which downsampling may be performed. Allowable values must be between 1.0 through 10.0, inclusive. If the threshold is set out of range, the value reverts to a default of 2.0. The following short examples provide a hypothetical configuration:</p> <p>To use <i>ImageDownsampleThreshold</i>, set the following attributes to the values indicated:</p> <p><i>ImageResolution</i> = 72</p> <p><i>ImageDownsampleThreshold</i> = 1.5</p> <p>The input image would not be downsampled unless it has a resolution greater than</p> <p>$\text{trunc}((72 * 1.5) + .5) = 108\text{dpi}$</p>
<i>ImageDownsampleType ?</i>	enumeration	<p>Downsampling algorithm for images. Possible values are:</p> <p><i>Average</i> – The program averages groups of samples to get the new downsampled value.</p> <p><i>Bicubic</i> – The program uses bicubic interpolation on a group of samples to get a new downsampled value.</p> <p><i>Subsample</i> – The program picks the middle sample from a group of samples to get the new downsampled value.</p>
<i>ImageFilter ?</i>	enumeration	<p>Specifies the compression filter to be used for images. Ignored if <i>AutoFilterImages</i> = <i>true</i> or if <i>EncodeImages</i> = <i>false</i>. Possible values are:</p> <p><i>CCITTFaxEncode</i> – Used to select CCITT Group 3 or 4 facsimile encoding. Used only if <i>ImageType</i> = <i>monochrome</i>.</p> <p><i>DCTEncode</i> – Used to select JPEG compression.</p> <p><i>FlateEncode</i> – Used to select ZIP compression.</p> <p>If <i>DCTEncode</i> is specified, it is only used if the output image has 8 bits per color component (that is, if <i>ImageDepth</i> is 8, or if it is -1 and the original image has 8 bits per color component). Otherwise, <i>FlateEncode</i> is used regardless of the value of <i>ImageFilter</i>.</p>
<i>ImageResolution ?</i>	number	<p>Specifies the minimum resolution for downsampled color images in dots per inch. This value is used only when <i>DownsampleImages</i> is <i>true</i>. The application downsamples to that actual resolution. The legal values are from 9.0 to 2400.0, inclusive.</p>
<i>ImageType</i>	enumeration	<p>Specifies the kind of images that are to be manipulated. Possible values are:</p> <p><i>color</i></p> <p><i>grayscale</i></p> <p><i>monochrome</i></p>

7.2.48 ImageReplacementParams

This resource specifies parameters required to control image replacement within production workflows.

Resource Properties

Resource class:	<i>Parameter</i>
Resource referenced by:	-
Partition:	-
Input of processes:	<i>ImageReplacement, Rendering</i>
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>ImageReplacement-Strategy</i>	enumeration	Identifies how externally referenced images will be handled within the associated process. Possible values are: <i>Omit</i> – Complete process maintaining only references to external data. <i>Proxy</i> – Complete process using available proxy images. <i>Replace</i> – Replace external references with image data during processing. <i>AttemptReplacement</i> – Attempt to replace external references with image data during processing. If replacement fails, complete the process using available proxy images.
<i>MaxResolution?</i>	double	Reduces the resolution of images with a resolution higher than <i>MaxResolution</i> Default = 0, which means “do not downsample.”
<i>MinResolution ?</i>	double	Specifies the minimum resolution that an image must have in order to be embedded. Default = 0, which means “don’t care”
<i>ResolutionReduction-Strategy ?</i>	enumeration	Identifies the mechanism used for reducing the image resolution. Possible values are: <i>Downsample</i> – Default value. <i>Subsample</i> <i>Bicubic</i>
<i>IgnoreExtensions</i>	NMTOKENS	Identifies a set of filename extensions that will be trimmed during searches for high-resolution images. These extensions are what will be stripped from the end of an image name to find a base name. Examples include: <i>.lay</i> <i>.e</i> <i>.samp</i>
<i>MaxSearchRecursion ?</i>	integer	Identifies how many levels of recursion in the search path will be traversed while trying to locate images. A value

of 0 indicates that no recursion is desired.

SearchPath + telem String that identifies the paths to search when trying to locate the referenced data.

7.2.49 ImageSetterParams

This resource specifies the settings for the imagesetter. A number of settings are OEM-specific, while others are so widely used they may be supported between vendors.

Both filmsetter settings and platesetter settings are described with this resource.

Resource Properties

Resource class: Parameter
 Resource referenced by: -
 Partition: -
 Input of processes: *ImageSetting*
 Output of processes: -

Resource Structure

Name	Data Type	Description
<i>AdvanceDistance ?</i>	double	Additional media advancement beyond the media dimensions on a roll-fed device.
<i>CenterAcross ?</i>	enumeration	This attribute specifies the axis around which a device may center an image, if the device is capable of doing so. Possible values are: <i>None</i> – Default value. <i>FeedDirection</i> – Image is centered around the feed-direction axis. <i>MediaWidth</i> – Image is centered around the media-width axis. <i>Both</i> – Image is centered around both axes.
<i>CutMedia ?</i>	boolean	Indicates whether or not to cut the media (roll-fed). Default = device default.
<i>MirrorAround ?</i>	enumeration	This attribute specifies the axis around which a device may mirror an image, if the device is capable of doing so. Possible values are: <i>None</i> – Default value. Used if the device is incapable of mirroring an image. <i>FeedDirection</i> – Image is mirrored around the feed-direction axis. <i>MediaWidth</i> – Image is mirrored around the media-width axis. <i>Both</i> – Image is mirrored around both possible axes.
<i>Polarity ?</i>	enumeration	Some devices can invert the image (in hardware). Possible values are: <i>Positive</i> – Default value.

		<i>Negative</i>
<i>Punch?</i>	boolean	If <i>true</i> , indicates that the device may create registration punch holes.
<i>PunchType ?</i>	string	Name of the registration punch scheme, such as <i>Bacher</i> .
<i>Resolution ?</i>	XYPair	Resolution of the output
<i>RollCut ?</i>	double	Length of media to be cut off of a roll, in points.
<i>TransferCurve ?</i>	Transfer-Function	Area coverage correction of the device.

7.2.50 Ink

Resource describing what kind of ink or other colorant (such as toner or varnish) is to be used during printing or varnishing.

Resource Properties

Resource class:	Consumable
Resource referenced by:	ConventionalPrintingParams
Partition:	<i>Separation, Side, SheetName, SignatureName</i>
Input of processes:	ConventionalPrinting, DigitalPrinting, IDPrinting
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>ColorName ?</i>	string	Link to a definition of the color specifics. The value of <i>ColorName</i> color should match the <i>Name</i> attribute of a Color defined in a ColorPool resource that is linked to the process using the Ink resource.
<i>Family ?</i>	NMTOKEN	Ink family. Possible values include: <i>HKS</i> <i>Pantone</i> <i>TOYO</i> <i>ISO</i> <i>EURO</i> <i>InkJet</i> It is also possible to specify liquids that are similar to ink. Possible values of this type include: <i>Varnish</i> <i>Silicon</i> <i>Toner</i>
<i>InkName</i>	string	The name of ink is dependent on its <i>Family</i> . For example, the <i>InkName 138 CVC</i> is a member of the <i>Pantone Family</i> .
<i>SpecialInk ?</i>	NMTOKEN	Specific ink attributes. Possible values include:

		<i>metallic</i>
<i>SpecificYield ?</i>	double	Weight per area at total coverage in g/m ² .

7.2.51 InkZoneCalculationParams

This resource specifies the parameters for the *InkZoneCalculation* process.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	<i>InkZoneCalculation</i>
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>PrintableArea ?</i>	rectangle	Position and size of the printable area of the print cylinder in the coordinates of the Preview resource. Default = the complete image.
<i>ProfileOffset ?</i>	XYPair	Offset of the lower left corner of the printable area of the preview with respect to the press coordinate system. Default = 0 0.
<i>ZoneHeight ?</i>	double	The width of one zone in the feed direction of the printing machine being used. Typically, the height of a zone is the height of an ink slide.
<i>ZoneWidth</i>	double	The width of one zone of the printing machine being used. Typically, the width of a zone is the width of an ink slide.
<i>Zones</i>	integer	The number of ink zones of the press.
<i>ZonesY ?</i>	integer	Number of ink zones in feed direction of the press. Default = 0, which means not required.

7.2.52 InkZoneProfile

This resource specifies ink zone settings that are specific to the geometry of the printing device being used. **InkZoneProfiles** are independent of the device details.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	<i>Separation, Side, SheetName, SignatureName</i>
Input of processes:	<i>ConventionalPrinting</i>
Output of processes:	<i>InkZoneCalculation</i>

Resource Structure

Name	Data Type	Description
------	-----------	-------------

<i>PlatePosition</i> ?	XYPair	Position of the plate. The ink-zone settings depend on the plate mounting position. After remounting, the <i>PlatePosition</i> may be used for a fine adjustment of the ink-zone settings.
<i>ZoneHeight</i> ?	double	The width of one zone in the Y-Direction of the printing machine being used.
<i>ZoneSettingsX</i>	NumberList	Each entry of the <i>ZoneSettings</i> attribute is the value of one ink zone. The first entry is the first zone, and the number of entries equals the number of zones of the printing device being used.
<i>ZoneSettingsY</i> ?	NumberList	Each entry of the <i>ZoneSettingsY</i> attribute is the value of one ink zone in Y-Direction. The first entry is the first zone and the number of entries equals the number of zones of the printing device being used.
<i>ZoneWidth</i>	double	The width of one zone of the printing machine being used.

7.2.53 InsertingParams

This resource specifies the parameters for the *Inserting* process. Figure 7.13 shows the various components involved in an inserting process, and how they interact.

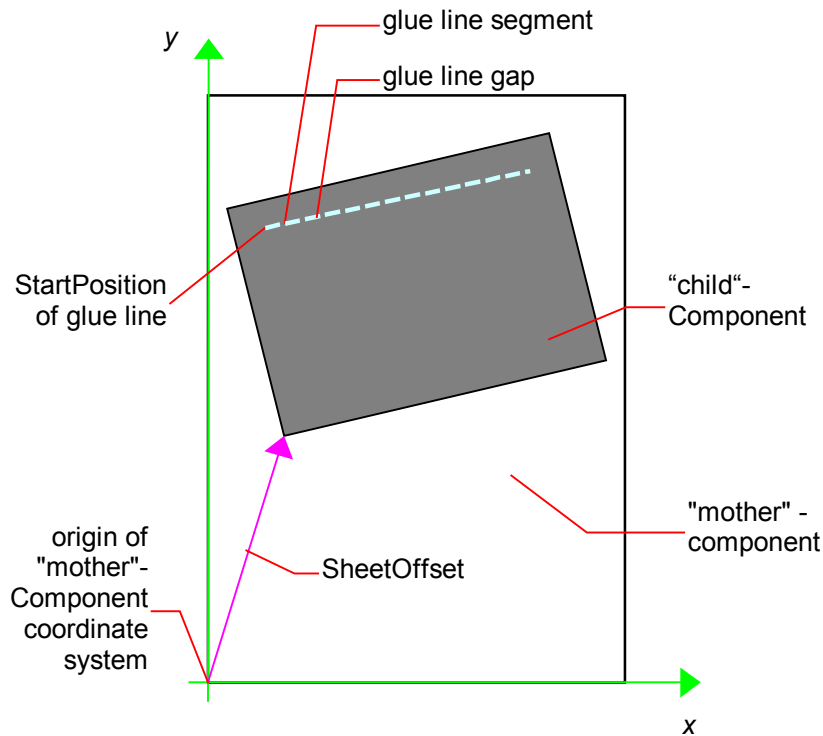


Figure 7.13 Parameters and coordinate system used for inserting

The process coordinate system is defined as follows:

The Y-axis is aligned with the binding edge, and increases from the registered edge to the edge opposite the registered edge. The X-axis, meanwhile, is aligned with the registered edge. It increases from the binding edge to the edge opposite the binding edge, which is the product front edge.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	<i>Inserting</i>
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>SheetOffset</i>	XYPair	Offset between the sheet to be inserted and the “mother” sheet.
<i>Location</i>	enumeration	Where to place the “child” sheet. Possible values are: <i>Front</i> <i>Back</i> <i>OverfoldLeft</i> <i>OverfoldRight</i>
GlueLine *	element	Array of all GlueLine elements.

7.2.54 InsertSheet

In some cases, an ***Imposition*** process may encounter RunList elements that do not provide enough pages to complete a **Sheet** resource. **InsertSheet** resources are used to provide a standard way of completing such **Sheet** resources. **InsertSheet** resources may also be used to start new **Sheet** resources (for example, to ensure that a new chapter starts on a right-hand page.) In addition, **InsertSheet** may specify whether new media should be inserted, once the current **Sheet** is completed.

InsertSheets may be used at the beginning or end of RunLists (as *NewSheet* elements or as *Trailer* elements).

Resource Properties

Resource class:	Parameter, Element
Resource referenced by:	Layout, Sheet
Partition:	-
Input of processes:	<i>Imposition</i>
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>BlankSheet</i>	boolean	If <i>true</i> , the next Sheet will be blank.
<i>Usage</i>	enumeration	Indicates the way in which this InsertSheet will be used. Possible values are: <i>header</i> – the sheet contents are prepended in front of the current signature or layout <i>trailer</i> – the sheet contents are appended behind the current signature or layout <i>filler</i> – the sheet contents are filled into the current

		signature
RunList ?	element	A RunList that provides the content for the InsertSheet .
Sheet ?	element	A Sheet that will be inserted after the current Sheet is completed or before the current Sheet is begun. The various Sheets are identified by the value of <i>Usage</i> .

7.2.55 InterpretedPDLData

Represents the results of the PDL Interpretation process. The details of this resource are not specified, as it is assumed to be implementation dependent.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	<i>Rendering</i>
Output of processes:	<i>Interpreting</i>

7.2.56 InterpretingParams

The **InterpretingParams** resource contains the parameters needed to interpret PDL pages. The resource itself is an abstract resource that contains attributes that are relevant to all PDLs. PDL-specific instances of **InterpretingParams** resources are then derived from this abstract resource.

This specification defines one PDL-specific resource instance: **PDFInterpretingParams**.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	<i>Interpreting, IDPrinting</i>
Output of processes:	-

Structure of the abstract InterpretingParams Resource

Name	Data Type	Description
<i>Center</i>	boolean	Indicates whether or not the page image should be centered within the imagable area of the media.
<i>FitToPage ?</i>	boolean	Specifies whether the page contents should be scaled to fit the media. Default = <i>false</i> .
<i>Poster ?</i>	XYPair	Specifies whether the page contents should be expanded such that each page covers X by Y pieces of media. Default = 1,1.
<i>PosterOverlap</i>	XYPair	This pair of real numbers identifies the amounts of overlap, in points, that shall be calculated for the poster tiles across the horizontal and vertical axes, respectively.

		Default = 0,0
<i>Resolution ?</i>	XYPair	Indicates the resolution at which the PDL contents will be interpreted in DPI. This value must be a positive integer, and may be different from the <i>Resolution</i> attribute provided for the RenderingParams resource.
<i>Scaling ?</i>	XYPair	A pair of positive real values that indicates the scaling factor for the page contents. Values between 0 and 1 specify that the contents are to be reduced, while values greater than 1 specify that the contents are to be expanded. This attribute is ignored if <i>FitToPage</i> = <i>true</i> or if <i>Poster</i> is present and has a value other than 1,1. Default = 1.
<i>ScalingOrigin</i>	XYPair	A pair of real values that identify the point in the unscaled page that is to become the origin of the new, scaled page image. This point is defined in the coordinate system of the unscaled page. Default = 0,0

Structure of PDFInterpretingParams resource

Name	Data Type	Description
<i>EmitPDFBG</i>	boolean	Indicates whether BlackGeneration functions should be emitted.
<i>EmitPDFHalftones</i>	boolean	Indicates whether Halftones should be emitted.
<i>EmitPDFTransfers</i>	boolean	Indicates whether Transfer functions should be emitted.
<i>EmitPDFUCR</i>	boolean	Indicates whether UnderColorRemoval functions should be emitted.
<i>HonorPDFOverprint</i>	boolean	Indicates whether or not overprint settings in the file will be honored. If <i>true</i> , the setting for overprint will be honored. If <i>false</i> , it is expected that the device does not directly support overprint, and that the PDF is pre-processed to simulate the effect of the overprint settings.
<i>ICCColorAsDeviceColor</i>	boolean	Indicates whether colors specified by ICC colorspaces should be treated as device colorants.
<i>PrintPDFAnnotations</i>	boolean	Indicates whether the contents of annotations on PDF pages should be included in the output. This only refers to annotations that are set to print in the PDF file.
<i>TransparencyRenderingQuality</i>	number	Possible values are 0 to 1. 0 represents the lowest allowable quality. 1 represents the highest desired quality.

7.2.57 Layout

Represents the root of the layout structure. **Layout** is used both for fixed-layout and for automated printing.

Resource Properties

Resource class: Parameter

Resource referenced by: -
 Partition: -
 Input of processes: ***Imposition, Proofing, ConventionalPrinting, DigitalPrinting, SoftProofing***
 Output of processes: -

Resource Structure

Name	Data Type	Description
<i>Automated</i> ?	boolean	If <i>true</i> , the <i>Imposition</i> process is expected to perform automated page consumption. Automated page consumption is equivalent to the PrintLayout functionality provided in PJTF. Default = <i>false</i>
<i>MaxOrd</i> ?	integer	Maximum number of placed objects that are consumed from a RunList each time the Layout is executed, assuming the <i>Imposition</i> process is automated.
InsertSheet *	element	Additional sheets that should be inserted before and/or after a document.
Signature *	element	The signatures that are defined by the layout.

Structure of Signature Sub-element

This element groups individual **Sheet** resources into one **Signature** sub-element.

Name	Data Type	Description
<i>Name</i> ?	NMTOKEN	Unique name of the signature. <i>Name</i> is used for external reference to a signature, as in a Part element.
InsertSheet *	element	Specifies how to complete a signature in an automated printing environment.
Sheet *	element	Sheet resources that comprise the signature.

7.2.58 LayoutElement


This resource is needed for ***LayoutElementProduction***. It describes some text, an image, one or more pages, or anything else that is used in the production of the layout of a product.

Resource Properties

Resource class: Parameter or Element
 Resource referenced by: **RunList, Surface**
 Partition: -
 Input of processes: ***DBDocTemplateLayout, DBTemplateMerging, LayoutElementProduction***
 Output of processes: ***DBDocTemplateLayout, DBTemplateMerging, LayoutElementProduction***

Resource Structure

Name	Data Type	Description
------	-----------	-------------

<i>ClipPath ?</i>	path	Path that describes the outline of the LayoutElement in the coordinate space of the LayoutElement of <i>ElementType</i> page that results from the LayoutElementProduction process.
<i>ElementType</i>	enumeration	Describes the content type for this LayoutElement . Possible values are: <i>text</i> – Formatted or unformatted text. <i>image</i> – Bitmap image. <i>graphic</i> – Line art. <i>reservation</i> – Empty element. Content for this area of the page may be provided by a subsequent process. <i>composed</i> – Combination of elements that define an element that is not bound to a document page. <i>page</i> – Representation of one document page. <i>document</i> – An ordered set of pages. <i>surface</i> – Representation of an imposed surface. <i>tile</i> – Representation of the contents of one tile. <i>unknown</i> – Unknown element type,  of the above[DH40].
<i>IsPrintable ?</i>	boolean	If <i>true</i> , the file can be printed. Possible file types include PDF or PostScript files.
<i>IsTrapped ?</i>	boolean	If <i>true</i> , the file has been trapped.
<i>SourceClipBox ?</i>	rectangle	A rectangle that defines the region of the element to be included. This rectangle is expressed in the default user space of the source document page.
<i>Template ?</i>	boolean	Template is <i>false</i> when this layout element is self-contained. This attribute is <i>true</i> if the LayoutElement represents a template that must be completed with information from a database. Default = <i>false</i>
FileSpec	element	URL + meta-data about the physical characteristics of a file representing the LayoutElement .
SeparationSpec *	element	List of used separation names.

7.2.59 LongitudinalRibbonOperationParams

This resource provides the parameters of the **LongitudinalRibbonOperation** process. It is defined as a list of abstract *LROperation* elements.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	LongitudinalRibbonOperation
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>LROperation</i> +	element	Instance of an abstract LROperation element.

Structure of LongitudinalRibbonOperationParams Elements

LROperation

Resource class: Abstract element

LROperation is an abstract element that describes the **LongitudinalRibbonOperation** process. The defined instances (sub-classes) of LROperation are Slit, FormerFold, Glue and Perforate. All instances of LROperation have the following common contents.

Name	Data Type	Description
<i>WorkingDistance</i> ?	double	Length of the <i>Operation</i> to be performed in point.
<i>XOffset</i>	number	Position of the tool for longitudinal action along the cylinder axis.

LongFold

Resource class: ABOperation

LongSlit describes a longitudinal cut operation and has no further contents in addition to those of LROperation.

LongFold

Resource class: ABOperation

LongFold describes a longitudinal fold operation and has no further contents in addition to those of LROperation.

LongGlue

Resource class: ABOperation

LongGlue describes a longitudinal gluing operation and has the following contents in addition to those of LROperation.

Name	Data Type	Description
<i>GlueBrand</i> ?	string	Glue brand. Use only when <i>Operation</i> = Glue
<i>GlueType</i> ?	enumeration	If <i>Operation</i> = Glue the following values can be used: <i>ColdGlue</i> <i>Hotmelt</i> <i>PUR</i> – Polyurethane
<i>LineWidth</i> ?	double	Width of the <i>Operation</i> line.
<i>MeltingTemperature</i> ?	integer	Required temperature for melting the glue (in degrees centigrade). Use only when <i>GlueType</i> = <i>Hotmelt</i> and <i>Operation</i> =

Glue

LongPerforate

Resource class: ABOperation

LongPerforate describes a longitudinal gluing operation and has the following contents in addition to those of LROperation.

Name	Data Type	Description
<i>TeethPer-Dimension ?</i>	integer	If <i>Operation</i> = Perforate, the number of teeth in a given perforation extent is defined in teeth/point.

7.2.60 Media

This resource describes a physical element that represents a raw, unexposed printable surface such as sheet, film, or plate.

Resource Properties

Resource class: Consumable or Element

Resource referenced by: **Color, ExposedMedia, IDPrintingParams, Sheet**

Partition: *SheetName, TileID*

Input of processes: **ConventionalPrinting, Cutting, DigitalPrinting, FilmToPlateCopying, IDPrinting, ImageSetting, Proofing, Rendering**

Output of processes: -

Resource Structure

Name	Data Type	Description
<i>Dimension ?</i>	XYPair	The X and Y dimensions of the chosen medium. Measured in points [pt].
<i>Grade ?</i>	string	The quality of the chosen medium. Examples include newsprint and glossy.
<i>ImagableSide ?</i>	enumeration	Side of the chosen medium that may be marked. Possible values are: <i>Front</i> <i>Back</i> <i>Both</i> – Default value. <i>Neither</i>
<i>Material ?</i>	NMTOKEN	Material of the chosen medium. Possible values include: <i>Aluminum</i> <i>DryFilm</i> <i>Paper</i> <i>Polyester</i> <i>WetFilm</i>
<i>MediaType</i>	enumeration	Describes the medium being employed. Possible values are: <i>Film</i>

		<i>Foil</i>
		<i>Paper</i>
		<i>Plate</i>
<i>MediaUnit ?</i>	enumeration	Describes the format of the media as it is delivered to the device. Possible values are: <i>Roll</i> <i>Sheet</i> – Default value.
<i>Polarity ?</i>	enumeration	Polarity of the chosen medium. Possible values are: <i>Positive</i> – Default value. <i>Negative</i>
<i>Transparent ?</i>	boolean	If <i>true</i> , the medium is transparent. Default = <i>false</i>
<i>Thickness ?</i>	double	The thickness of the chosen medium. Measured in micro meters [μm].
<i>Weight ?</i>	double	Weith of the chosen medium. Measured in grams per square meter [g/m^2].
<i>Color ?</i>	element	A Color resource that provides the color of the chosen medium.

7.2.61 NumberingParams

This resource describes the parameters of the **Numbering** process. One **NumberingParams** element must be defined per numbering machine.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	Numbering
Output of processes:	-

Resource Structure

Name	Data Type	Description
NumberingParams *	element	Set of parameters for one numbering machine

Structure of NumberingParams Sub-element

Name	Data Type	Description
<i>StartValue ?</i>	string	First value of the numbering machine.
<i>XPosition</i>	number	Position of the numbering machine along the printer axis.
<i>YPosition</i>	NumberList	List of stamp positions, in points, starting from the leading edge.
<i>Orientation</i>	integer	Rotation of the numbering machine. If <i>Orientation</i> = 0, the top of the numbers is along the leading edge.

<i>Step ?</i>	integer	Number that specifies the difference between two subsequent numbers of the numbering machine. Default = 1
---------------	---------	--

7.2.62 OrderingParams

Attributes of the **Ordering** process, which results in an acquisition.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	Ordering
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>Amount</i>	double	Amount of the ordered resource.
<i>Unit</i>	string	Unit of measurement for <i>Amount</i> .
Comment	telem	OrderingParams require a Comment element that contains a human-readable description of what to order.
Company ?	element	The supplier company.

7.2.63 PDFToPSConversionParams

This resource specifies a set of configurable options that may be used by processes that generate PostScript files from PDF files.

Font controls are applied in the following order:

1. IncludeBaseFonts
2. IncludeEmbeddedFonts
3. IncludeType1Fonts
4. IncludeType3Fonts
5. IncludeTrueTypeFonts
6. IncludeCIDFonts

For example, an embedded Type-1 font follows the rule for embedded fonts, not the rule for Type-1 fonts. In other words, if *IncludeEmbeddedFonts* is true, and *IncludeType1Fonts* is false, embedded Type-1 fonts would be included in the PostScript stream.

Resource Properties

Resource class:	<i>Parameter</i>
Resources referenced:	-
Partition:	-

Input of processes: *PDFToPSConversion*
 Output of processes: -

Resource Structure

Name	Data Type	Description
<i>BinaryOK ?</i>	boolean	If true, binary data are to be included in the PostScript stream. Default = <i>true</i>
<i>BoundingBox ?</i>	rectangle	If all zeroes, this attribute is ignored. Otherwise, it is used for <i>BoundingBox</i> DSC comment, in <i>CenterCropBox</i> calculations and for <i>SetPageDevice</i> . Default = [0 0 0 0]
<i>CenterCropBox ?</i>	boolean	If <i>true</i> , CropBox output is centered on the page when the CropBox < MediaBox. Default = <i>true</i>
<i>IgnoreAnnotForms ?</i>	boolean	If <i>true</i> , ignores annotations that contain an XObject form. Default = <i>false</i>
<i>IgnoreColorSeps ?</i>	boolean	If <i>true</i> , ignores images for Level-1 separations. Default = <i>false</i> .
<i>IgnoreDeviceExtGState ?</i>	boolean	If <i>true</i> , ignores all extended graphic state parameters. This overrides <i>IgnoreHalftones</i> . Default = <i>true</i>
<i>IgnoreDSC ?</i>	boolean	If <i>true</i> , ignores DSC (Document Structuring Conventions). Default = <i>true</i>
<i>IgnoreExternStreamRef ?</i>	boolean	If an image resource uses an external stream and <i>IgnoreExternStreamRef = true</i> , ignores code that points to the external file. Default = <i>false</i>
<i>IgnoreHalftones ?</i>	boolean	If <i>true</i> , ignores any halftone screening in the PDF file. Default = <i>false</i>
<i>IgnorePageRotation ?</i>	boolean	If <i>true</i> , ignores a concat provided at the beginning of each page that orients the page so that it is properly rotated. Used when emitting EPS. Default = <i>false</i>
<i>IgnoreRawData ?</i>	boolean	If <i>true</i> , no unnecessary filters should be added when emitting image data. Default = <i>false</i>
<i>IgnoreSeparableImages-Only ?</i>	boolean	If <i>true</i> , and if emitting EPS, ignores only CMYK and gray images. Default = <i>false</i>
<i>IgnoreShowPage ?</i>	boolean	If <i>true</i> , ignores save-and-restore showpage in

		PostScript files. Default = <i>false</i>
<i>IgnoreTTFontsFirst ?</i>	boolean	If <i>true</i> , ignores TrueType fonts before any other fonts. Default = <i>false</i>
<i>GeneratePageStreams ?</i>	boolean	If <i>true</i> , the process emits individual streams of data for each page in the RunList . Default = <i>false</i>
<i>IncludeBaseFonts ?</i>	enumeration	Determines when to embed the base fonts. Possible values are: <i>IncludeNever</i> – Default value <i>IncludeOncePerDoc</i> <i>IncludeOncePerPage</i>
<i>IncludeCIDFonts ?</i>	enumeration	Determines when to embed CID fonts. Possible values are: <i>IncludeNever</i> <i>IncludeOncePerDoc</i> – Default value. <i>IncludeOncePerPage</i>
<i>IncludeEmbeddedFonts ?</i>	enumeration	Determines when to embed fonts in the document that are embedded in the PDF file. This attribute overrides the <i>IncludeType1Fonts</i> , <i>IncludeTrueTypeFonts</i> , and <i>IncludeCIDFonts</i> attributes. Possible values are: <i>IncludeNever</i> <i>IncludeOncePerDoc</i> – Default value. <i>IncludeOncePerPage</i>
<i>IncludeOtherResources ?</i>	enumeration	Determines when to include all other types of resources in the file. Possible values are: <i>IncludeNever</i> <i>IncludeOncePerDoc</i> – Default value. <i>IncludeOncePerPage</i>
<i>IncludeProcSets ?</i>	enumeration	Determines when to include ProcSets in the file. Possible values are: <i>IncludeNever</i> <i>IncludeOncePerDoc</i> – Default value. <i>IncludeOncePerPage</i>
<i>IncludeTrueTypeFonts ?</i>	enumeration	Determines when to embed TrueType fonts. Possible values are: <i>IncludeNever</i> <i>IncludeOncePerDoc</i> – Default value. <i>IncludeOncePerPage</i>
<i>IncludeType1Fonts ?</i>	enumeration	Determines when to embed Type-1 fonts. Possible values are: <i>IncludeNever</i>

		<i>IncludeOncePerDoc</i> – Default value.
		<i>IncludeOncePerPage</i>
<i>IncludeType3Fonts ?</i>	enumeration	Determines when to embed Type-3 fonts. Must always be set to <i>IncludeOncePerPage</i> . It is included here to complete the precedence hierarchy.
<i>OutputType</i>	enumeration	Describes the kind of output to be generated. Possible values are: <i>PostScript</i> – Default value <i>EPS</i>
<i>PSLevel ?</i>	integer	Number that indicates the PostScript level. Default = 2
<i>Scale ?</i>	Number	Number that indicates the wide-scale factor of documents. Full-size = 100. Default = 100
<i>SetPageSize ?</i>	boolean	(PostScript level 2 only) If <i>true</i> , sets page size on each page automatically. Use media box for outputting PostScript files and crop box for EPS. Default = <i>false</i> .
<i>SetupProcsets ?</i>	boolean	If <i>true</i> , indicates that if procsets are included, the init/term code is also included. Default = <i>true</i>
<i>ShrinkToFit ?</i>	boolean	If <i>true</i> , the page is scaled to fit the printer page size. This field overrides scale. Default = <i>false</i>
<i>SuppressCenter ?</i>	boolean	If <i>true</i> , suppresses automatic centering of page contents whose crop box is smaller than the page size.
<i>SuppressRotate ?</i>	boolean	If <i>true</i> , suppresses automatic rotation of pages when their dimensions are better suited to landscape orientation. More specifically, the application that generates the PostScript compares the dimensions of the page. If the width is greater than the height, then pages are not rotated if <i>SuppressRotate</i> is <i>true</i> . On the other hand, if <i>SuppressRotate</i> is <i>false</i> , the value of the PDF Rotate key for each page is honored, regardless of the dimensions of the pages (as defined by the <i>MediaBox</i> attribute). Default = <i>false</i>
<i>TTasT42 ?</i>	boolean	If including TrueType fonts, converts to Type-42 instead of Type-1 fonts when <i>TTasT42</i> = <i>true</i> . Default = <i>false</i>
<i>UseFontAliasNames ?</i>	boolean	If <i>true</i> , font alias names are used when printing with system fonts. Default = <i>false</i>

7.2.64 PDLResourceAlias

This resource provides a mechanism for referencing resources that occur in files, or that are expected to be provided by devices.

Prepress and printing processes have traditionally used the word ‘resource’ to refer to reusable data structures that are needed to perform processes. Examples of such resources include fonts, halftones and functions. The formats of these resources are defined within PDLs, and instances of these resources may occur within PDL files, or may be provided by devices.

JDF does not provide a syntax for defining such resources directly within a job. Instead, resources continue to occur within PDL files, and continue to be provided by devices. However, since it is necessary to be able to refer to these resources from JDF jobs, the **PDLResourceAlias** resource is provided to fulfill this need.

Resource Properties

Resource class: Parameter
 Resource referenced by: **ColorantControl**
 Partition: -
 Input of processes: *Interpreting*
 Output of processes: -

Resource Structure

Name	Data Type	Description
<i>ResourceType</i>	string	The type of PDL resource that is referenced. The semantic of this attribute is defined by the PDL.
<i>SourceName</i> ?	string	The name of the resource in the file referenced by the <i>FileSpec</i> attribute or by the device.
<i>FileSpec</i> ?	element	Location of the file containing the PDL resource. If <i>FileSpec</i> is absent, the device is expected to provide the resource defined by this PDLResourceAlias resource.

7.2.65 Person

This resource provides detailed information about a person. It also has the ability to specify different communication channels to this person. The structure of the resource is derived from the vCard format—it contains all of the same name sub-types (N:) of the identification and the title of the organizational properties. The corresponding XML types of the vCard are quoted in the description field of the table below.

Resource Properties

Resource class: Element
 Resource referenced by: **Contact, Employee**
 Partition: -
 Input of processes: -
 Output of processes: -

Resource Structure

Name	Data Type	Description
------	-----------	-------------

<i>AdditionalNames</i> ?	string	Additional names of the contact person (vCard: N:other).
<i>FamilyName</i> ?	string	The family name of the contact person (vCard: N:family).
<i>FirstName</i> ?	string	The first name of the contact person (vCard: N:given).
<i>JobTitle</i> ?	string	Job function of the person in the company or organization (vCard: title).
<i>NamePrefix</i> ?	string	Prefix of the name, may include title (vCard: N:prefix).
<i>NameSuffix</i> ?	string	Suffix of the name (vCard: N:suffix).
<i>ComChannel</i> *	element	Communication channels to the person.

7.2.66 PlaceholderResource

This resource is used to link *ProcessGroup* nodes when the exact nature of interchange resources is still unknown. In this way, a skeleton of process networks can be constructed, with the **PlaceholderResource** resources serving as place holders in lieu of the appropriate resources.

This resource needs no structure besides that provided in an abstract **Resource** element, as it has no inherent value except as a stand-in for other resources.

Resource Properties

Resource class:	Placeholder
Resource referenced by:	-
Partition:	-
Input of processes:	any <i>ProcessGroup</i> nodes
Output of processes:	any <i>ProcessGroup</i> nodes

Resource Structure

The resource has no additional structure.

7.2.67 PlasticCombBindingParams

This resource describes the details of the **PlasticCombBinding** process.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	PlasticCombBinding
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>Brand</i> ?	string	The name of the comb manufacturer and the name of the specific item.
<i>Color</i> ?	colorant	Determines the color of the plastic comb.
<i>Diameter</i> ?	double	The comb diameter is determined by the height of the block of sheets to be bound.

<i>Thickness ?</i>	double	The material thickness of the comb.
<i>Type ?</i>	enumeration	The distance between the “teeth” and the distance between the holes of the pre-punched sheets must be the same. The following standards exist: <i>Euro</i> (Distance = 12 mm; Holes = 7 mm x 3 mm) <i>USAI</i> (Distance = 14.28 mm; Holes = 8 mm x 3 mm)

7.2.68 PlateCopyParams

This resource specifies the parameters of the **FilmToPlateCopying** process.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	FilmToPlateCopying
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>Cycle ?</i>	integer	Number of exposure light units to be used. The amount depends on the subject to be exposed.
<i>Diffusion ?</i>	enumeration	The diffusion foil setting. Possible values are: <i>on</i> <i>off</i> .
<i>Vacuum ?</i>	double	Amount of vacuum pressure to be used. Measured in bar.

7.2.69 PreflightAnalysis

PreflightAnalysis resources record the results of a **Preflight** process. The semantics for results are specific to the **FileType** of the file. The elements in this resource, detailed in the table below, place the results in specific categories. The value for each of these elements is an array of **PreflightResultsDetail** and **PreflightInstance** sub-elements. Within the **PreflightInstance** sub-elements, results are further broken down into **PreflightInstanceDetails**.

Each **PreflightResultsDetail** and **PreflightInstance** sub-element in the **PreflightAnalysis** hierarchy describes the results of a comparison of the properties of the file against one **PreflightConstraint** in the **PreflightProfile**.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	-
Output of processes:	Preflight

Name	Data Type	Description
------	-----------	-------------

ColorsResultsPool ?	element	A pool of PreflightDetail and PreflightInstance sub-elements that provides analysis about color.
DocumentResultsPool ?	element	A pool of PreflightDetail and PreflightInstance sub-elements that provides analysis about documents.
FontsResultsPool ?	element	A pool of PreflightDetail and PreflightInstance sub-elements that provides analysis about fonts.
FileTypeResultsPool ?	element	A pool of PreflightDetail and PreflightInstance sub-elements that provides analysis about file types.
ImagesResultsPool ?	element	A pool of PreflightDetail and PreflightInstance sub-elements that provides analysis about images.
PagesResultsPool ?	element	A pool of PreflightDetail and PreflightInstance sub-elements that provides analysis about pages.

Structure of PreflightDetail Sub-element

PreflightDetail sub-elements are used to describe one property within the **PreflightAnalysis** category in which they occur.

This sub-element is also used by **PreflightInventory** resource.

Name	Data Type	Description
<i>PageRefs</i>	rangelist	Identifies the set of pages in a RunList resource that exhibit the characteristic identified by the combination of the <i>Property</i> attribute and the Value element.
<i>Property</i> ?	string	Identifies the property described by this element.
<i>Status</i> ?	enumeration	Possible values are: <i>Error</i> – Value violates the ConstraintValue specified in the associated PreflightConstraint element. The constraint was flagged as an Error in the profile. <i>Warning</i> – Value violates the ConstraintValue specified in the associated PreflightConstraint element. The constraint was flagged as a Warning in the profile. <i>Ignore</i> – The constraint is ignored, and no PreflightDetail or PreflightInstanceDetail elements are created for this constraint. <i>IgnoreValue</i> – No comparison was made against a ConstraintValue . In other words, either the <i>Status</i> for the PreflightConstraint was <i>Ignore</i> or <i>IgnoreValue</i> , or this PreflightDetail is part of a PreflightInventory hierarchy.
<i>Value</i> ?	element	Identifies the value of the property. The semantics are PDL-specific.

Structure of PreflightInstance Sub-element

PreflightInstance sub-elements are used to collect **PreflightInstanceDetail** elements for one instance of some object which occurs in the PDL files referenced by a run list. For example, there might be one **PreflightInstance** element for each font that occurs in the pages of a run list.

This sub-element is also used by **PreflightInventory** resources.

Name	Data Type	Description
<i>Identifier ?</i>	string	Identifies the instance this element collects PreflightInstanceDetail elements.
<i>PageRefs</i>	rangelist	Identifies the set of pages in a RunList on which the instance occurs.
Properties *	element	A pool of PreflightInstanceDetail elements that describe the properties for this instance

Structure of PreflightInstanceDetail Sub-element

PreflightInstanceDetail sub-elements describe one property of one instance of some object type that occurs in a PDL file. For example, several **PreflightInstanceDetail** elements might describe the properties of a single font.

This sub-element is also used by **PreflightInventory** resources.

Name	Data Type	Description
<i>Property ?</i>	string	Identifies the property described by this element.
<i>Status ?</i>	enumeration	Specifies the results of the comparison between the value of the property for this instance with the ConstraintValue for the associated PreflightConstraint element. Possible values are: <i>Error</i> – Value violates the ConstraintValue specified. The constraint was flagged as an Error in the profile. <i>Warning</i> – Value violates the ConstraintValue specified. The constraint was flagged as a Warning in the profile. <i>IgnoreValue</i> – No comparison was made against a ConstraintValue . In other words, either the Status for the Constraint was <i>Ignore</i> or <i>IgnoreValue</i> , or this PreflightInstanceDetail is part of a PreflightInventory hierarchy.
Value ?	element	Identifies the value of the property. The semantics are PDL-specific.

7.2.70 PreflightInventory

PreflightInventory resources, like **PreflightAnalysis** resources, record the results of a **Preflight** process. The semantics for results are specific to the **FileType** of the file. The elements in this resource, detailed in the table below, place the results in specific categories. The value of each of these elements is an array of **PreflightResultsDetail** and **PreflightInstance** sub-elements. Within the **PreflightInstance** sub-elements, results are further broken down into **PreflightInstanceDetails**.

Each **PreflightResultsDetail** or **PreflightInstance** sub-element in the **PreflightInventory** hierarchy describes the results of a comparison of the properties of the file against one **PreflightConstraint** in the **PreflightProfile**.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	Preflight
Output of processes:	Preflight

Name	Data Type	Description
ColorsResultsPool ?	element	A pool of PreflightDetail and PreflightInstance sub-elements that provides a color inventory.
DocumentResultsPool ?	element	A pool of PreflightDetail and PreflightInstance sub-elements that provides a document inventory.
FontsResultsPool ?	element	A pool of PreflightDetail and PreflightInstance sub-elements that provides a font inventory.
FileTypeResultsPool ?	element	A PreflightDetail and PreflightInstance sub-element that provides a file-type inventory.
ImagesResultsPool ?	element	A pool of PreflightDetail and PreflightInstance sub-elements that provides an image inventory.
PagesResultsPool ?	element	A pool of PreflightDetail and PreflightInstance sub-elements that provides a page inventory.

7.2.71 PreflightProfile

PreflightProfile resources specify a set of constraints against which a file may be tested. The semantics for constraints are specific to the **FileType** of the for the file. The elements in this resource, detailed in the table below, place the results in specific categories. The value for each of these elements is an array of **PreflightConstraint** sub-elements. Within the **PreflightConstraint** resources, the **ConstraintValue** element indicates allowable values and the **Status** attribute indicates the error level (if any) to be flagged when exceptions to the constraints are identified.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	Preflight
Output of processes:	-

Name	Data Type	Description
ColorsConstraintsPool ?	element	A pool of PreflightConstraint sub-elements. Each element in this pool identifies a specific constraint concerning colors against which to test the file
DocumentConstraintsPool ?	element	A pool of PreflightConstraint sub-elements. Each element in this pool identifies a specific constraint concerning documents against which to test the file
FontsConstraintsPool ?	element	A pool of PreflightConstraint sub-elements. Each element in this pool identifies a specific constraint concerning fonts against which to test the file
FileTypeConstraintsPool ?	element	A Preflight constraint. The Type attribute must have a value of <i>array</i> and must contain string objects that

		identify the allowable types of data in the file. The strings in the <i>Value</i> array must be MIME-file types as recorded by the Internet Assigned Numbers Authority (IANA). IANA has procedures for registering new file types if needed.
ImagesConstraintsPool ?	element	A pool of PreflightConstraint sub-elements. Each element in this pool identifies a specific constraint concerning images against which to test the file
PagesConstraintsPool ?	element	A pool of PreflightConstraint sub-elements. Each element in this pool identifies a specific constraint concerning pages against which to test the file

Structure of PreflightConstraint Sub-element

Name	Data Type	Description
<i>AttemptFixupErrors</i> ?	boolean	If <i>true</i> , the device performing preflight should attempt to fix errors that are identified during preflight. Errors that are corrected are not given a <i>Status</i> attribute.
<i>AttemptFixupWarnings</i> ?	boolean	If <i>true</i> , the device performing preflight should attempt to fix warnings that are identified during preflight. Warnings that are corrected are not given a <i>Status</i> attribute.
<i>Constraint</i> ?	string	Describes the specific file characteristic to be checked.
<i>Status</i>	enumeration	Possible values are: <i>Error</i> – Values that violate the ConstraintValue specified are flagged as Errors in PreflightDetail and PreflightInstanceDetail elements. <i>Warning</i> – Values that violate the ConstraintValue specified are flagged as Warnings in PreflightDetail and PreflightInstanceDetail elements. <i>Ignore</i> – The constraint is ignored, and no PreflightDetail or PreflightInstanceDetail elements are created for this constraint. <i>IgnoreValue</i> – No comparison is made against the ConstraintValue .
<i>ConstraintValue</i> ?	element	Provides a value against which to test occurrences of the characteristic in the file. Note that the semantics of the ConstraintValue element depend on the PDL characteristic in question.

7.2.72 Preview

The preview of the content of a surface. It can be used for the calculation of the ink coverage (*PreviewType* = *Separation*) or as a preview of what is currently processed in a device (*PreviewType* = *Viewable*). When the preview is of *Type* = *Separation*, a gray value of 0 represents full ink, while a value

of 255 represents no ink (for more information, see DeviceGray color model chapter 4.8.2. of the *PostScript Language Reference Manual*).

Resource Properties

Resource class: Parameter
 Resource referenced by: **Surface**
 Partition: *Separation, Side, SheetName*
 Input of processes: ***InkZoneCalculation***
 Output of processes: ***PreviewGeneration***

Resource Structure

Name	Data Type	Description
<i>PreviewType</i>	enumeration	Type of the preview. Possible values are: <i>Separation</i> <i>Viewable</i>
<i>URL</i>	URL	URL identifying the image file. This is a normally a URL to a MIME sub-part (see section A.4.1). Note that a preview will generally be partitioned by separation.
<i>Compensation ?</i>	enumeration	Compensation of the image to reflect the application of transfer curves to the image. Possible values are: <i>unknown</i> – Default value. <i>none</i> – No compensation. <i>film</i> – Compensated until film exposure. <i>plate</i> – Compensated until plate exposure. <i>press</i> – Compensated until press.

7.2.73 PreviewGenerationParams

Parameters specifying the size and the type of the preview.

Resource Properties

Resource class: Parameter
 Resource referenced by: -
 Partition: -
 Input of processes: ***PreviewGeneration***
 Output of processes: -

Resource Structure

Name	Data Type	Description
<i>PreviewType</i>	enumeration	The kind of preview to be generated. Possible values are: <i>Separation</i> <i>Viewable</i>
<i>Resolution ?</i>	XYPair	Resolution of the preview, in dpi.

<i>Size</i> ?	XYPair	<p>Default = “50.8 50.8” dpi.</p> <p>Size of the preview, in pixels. If this attribute is present, the <i>Resolution</i> attribute is ignored.</p> <p>If <i>Size</i> is not specified, it may be calculated using the <i>Resolution</i> attribute and the input image size.</p>
---------------	--------	---

7.2.74 ProofingParams

This resource specifies the settings needed for all proofing operations, including both ‘hard’ or ‘soft’ proofing, of color and imposition proofs.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	<i>Proofing</i> , <i>SoftProofing</i>
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>DisplayTraps</i> ?	boolean	If <i>true</i> , the trap networks are shown in the proof.
<i>ImageViewingStrategy</i> ?	string	Identifies which images will be displayed during the SoftProofing process. Possible values are: <i>NoImages</i> – Default value. <i>OmitReference</i> – Displays only images actually embedded in the file. <i>UseProxies</i> – Displays images embedded in the file and proxy versions of referenced data. <i>UseReplacements</i> – Displays embedded images plus the full resolution version of referenced images.
<i>ProoferProfile</i> ?	URL	ICC profile of the proofer device.
<i>ProofType</i> ?	enumeration	This string identifies the type of proof requested. When absent, the type of proof desired may be inferred from the other attributes and elements within this resource. Possible values are: <i>colorconceptual</i> <i>contone</i> <i>halftone</i> <i>imposition</i>
<i>Resolution</i> ?	XYPair	Resolution of the output.

7.2.75 PSToPDFConversionParams

This resource contains the parameters that control the conversion of PostScript streams to PDF pages.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	<i>PSToPDFConversion</i>
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>ASCII85EncodePages ?</i>	boolean	If <i>true</i> , binary streams such as page contents streams, sampled images, and embedded fonts are ASCII85-encoded, resulting in a PDF file that is almost pure ASCII. If <i>false</i> , they are not, resulting in a PDF file that may contain substantial amounts of binary data.
<i>AutoRotatePages ?</i>	name	Allows the device to try to orient pages based on the predominant text orientation. Only used if the file does not contain <code>%%ViewingOrientation</code> , <code>%%PageOrientation</code> , or <code>%%Orientation</code> DSC comments. If the file does contain such DSC comments, it honors them. <code>%%ViewingOrientation</code> takes precedence over others, then <code>%%PageOrientation</code> , then <code>%%Orientation</code> . Possible values are: <i>None</i> – Turns <i>AutoRotatePages</i> off. <i>All</i> – Takes the predominant text orientation across all pages and rotates all pages the same way. <i>PageByPage</i> – Does the rotation on a page-by-page basis, rotating each page individually. Useful for documents that use both portrait and landscape orientations.
<i>AutoRotatePages ?</i>	name	Allows the device to try to orient pages based on the predominant text orientation. Only used if the file does not contain <code>ViewingOrientation</code> , <code>PageOrientation</code> , or <code>Orientation</code> DSC comments. If the file does contain such DSC comments, it honors them. <code>ViewingOrientation</code> takes precedence over others, then <code>PageOrientation</code> , then <code>Orientation</code> . Possible values are: <i>None</i> – Turns <i>AutoRotatePages</i> off. <i>All</i> – Takes the predominant text orientation across all pages and rotates all pages the same way. <i>PageByPage</i> – Does the rotation on a page-by-page basis, rotating each page individually. Useful for documents that use both portrait and landscape orientations.
<i>Binding ?</i>	name	Determines how the printed pages would be bound. Specify <i>Left</i> for left binding or <i>Right</i> for right binding.
<i>CompressPages ?</i>	boolean	Enables compression of pages and other content streams like forms, patterns and Type 3 fonts. If <i>true</i> , use Flate compression.
<i>DefaultRenderingIntent ?</i>	name	Selects the rendering intent for the current job. Possible values are:

		<i>Default</i>
		<i>Perceptual</i>
		<i>Saturation</i>
		<i>RelativeColorimetric</i>
		<i>AbsoluteColorimetric</i>
		See the <i>Portable Document Format Reference Manual</i> for more information on rendering intent.
<i>DetectBlend ?</i>	boolean	Enables or disables blend detection. If <i>true</i> , and if <i>PDFVersion</i> is 1.3 or higher, then blends will be converted to smooth shadings.
<i>DoThumbnails ?</i>	boolean	If <i>true</i> , thumbnails are created.
<i>EndPage ?</i>	integer	Number that indicates the last page that is displayed when the PDF file is viewed. <i>EndPage</i> must equal be to anything less than <i>StartPage</i> or be greater than or equal to 1. If not, then it must be greater than or equal to <i>StartPage</i> . When combined with <i>StartPage</i> , <i>EndPage</i> selects a range of pages to be displayed. The entire file may or may not be distilled, but only <i>StartPage</i> to <i>EndPage</i> pages, inclusive, are opened and viewed in Acrobat.
<i>ImageMemory ?</i>	integer	Number of bytes in the buffer used in sample processing for color, grayscale, and monochrome images. Its contents are written to disk when the buffer fills up. This is a read-only attribute.
<i>OverPrintMode ?</i>	integer	Controls the overprint mode strategy of the job. Set to 0 for full overprint or 1 for non-zero overprint. For more information, see http://partners.adobe.com/asn/developer/PDFS/TN/5044.ColorSep_Conv.pdf
<i>Optimize ?</i>	boolean	If <i>true</i> , the PS-to-PDF converter optimizes the PDF file. See the <i>Portable Document Format Reference Manual</i> for more information on optimization.
<i>PDFVersion ?</i>	real	Specifies the version number of the PDF file produced. Possible values include all legal version designators (e.g., 1.2, 1.3, 1.4).
<i>StartPage ?</i>	integer	Sets the first page that is displayed when the PDF file is opened with Acrobat. <i>StartPage</i> must be greater than or equal to 1. If <i>EndPage</i> is not -1, then it must be greater than or equal to <i>StartPage</i> .
<i>AdvancedParams ?</i>	element	Advanced parameters which control how certain features of PostScript are handled.
<i>ThinPDFParams ?</i>	element	Parameters that control the optional content or form of PDF files that will be created.

Structure of AdvancedParams Sub-element

Name	Data Type	Description
<i>AutoPostitionEPSInfo ?</i>	boolean	If <i>true</i> , the process automatically resizes and centers EPS

		information on the page.
<i>EmitDSCWarnings ?</i>	boolean	If <i>true</i> , warning messages about questionable or incorrect DSC comments appear during the distilling of the PS file.
<i>LockDistillerParams ?</i>	boolean	If <i>true</i> , the incoming PS content that specifies any of the PSToPDFConversionParams settings is used. If <i>false</i> , any PSToPDFConversionParams settings configured by the PS content are ignored.
<i>ParseDSCComments ?</i>	boolean	If <i>true</i> , the process parses the DSC comments for any information that might be helpful for converting the file or for information that must be stored in the PDF file. If <i>false</i> , the process treats the DSC comments as pure PS comments and ignores them.
<i>ParseDSCCommentsFor- DocInfo ?</i>	boolean	If <i>true</i> , the process parses the DSC comments in the PS file and extracts the document information. This information is recorded in the Info dictionary of the PDF file.
<i>PreserveCopyPage ?</i>	boolean	If <i>true</i> , the copypage operator of PostScript Level 2 is maintained. If <i>false</i> , the PostScript Level 3 definition of copypage operator is used. In PostScript Levels 1 and 2, the copypage operator transmits the page contents to the current output device (similar to showpage). However, copypage does not perform many of the reinitializations that showpage does. Many PostScript Level-1 and -2 programs used the copypage operator to perform such operations as printing multiple copies and implementing forms. These programs produce incorrect results when interpreted using the Level-3 copypage semantics. This attribute provides a mechanism to retain Level-2 compatibility for this operator.
<i>PreserveEPSInfo ?</i>	boolean	If <i>true</i> , preserves the EPS information in the PS file and stores it in the resulting PDF file.
<i>PreserveOPIComments ?</i>	boolean	If <i>true</i> , encapsulates OPI low-resolution images as a form and preserves information for locating the high-resolution images. OPI stands for Open Prepress Interface.
<i>UsePrologue ?</i>	boolean	If <i>true</i> , the process shall prepend a PostScript prologue file to the job and append a PostScript epilog file to the job. Such files are used to control the PostScript environment for the conversion process. The expected location and allowable contents for these files is defined by the process implementation.

Structure of ThinPDFParams Sub-element

Name	Data Type	Description
<i>FilePerPage ?</i>	boolean	If <i>true</i> , the process generates 1 PDF file per page. Default = <i>false</i>
<i>SidelineFonts ?</i>	boolean	If <i>true</i> , font data are stored in external files during PDF

		generation.
		Default = <i>false</i>
<i>SidelinelImages ?</i>	boolean	If <i>true</i> , image data are stored in an external stream during the PDF Generation phase. This prevents large amounts of image data from having to be passed through all phases of the code generation process.
		Default = <i>false</i>

7.2.76 RegisterMark

Defines a register mark, which can be used for setting up and monitoring a printing process. The position and rotation of each register mark can be specified with the help of the following attributes. It is important that the register marks are defined in such a way that their centers are on the point of origin of the coordinate system, as otherwise they are not positioned properly.

Resource Properties

Resource class:	Element
Resource referenced by:	Surface
Partition:	-
Input of processes:	Any printing process
Output of processes:	Imposition

Resource Structure

Name	Data Type	Description
<i>Center</i>	XYPair	Position of the center of the register mark in the coordinates of the SurfaceContentsBox .
<i>MarkType ?</i>	NMTOKEN	Type of register mark. Possible value include: <i>arc</i> <i>circle</i> <i>cross</i>
<i>Rotation ?</i>	double	Rotation in degrees. Positive graduation figures indicate counter-clockwise rotation; negative figures indicate clockwise rotation.
SeparationSpec *	element	Set of separations to which the register mark is bound.

7.2.77 RenderingParams

This set of parameters identifies how the **Rendering** process should operate. Specifically, these parameters define the expected output of the **Bytemap** resource that the **Rendering** process creates.

Resource Properties

Resource class:	Parameters
Resource referenced by:	-
Partition:	-
Input of processes:	IDPrinting, Rendering
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>BandHeight</i>	integer	Height of output bands expressed in lines. For a frame device, the band height is simply the full height of the frame.
<i>BandOrdering ?</i>	enumeration	Indicates whether output buffers are generated in band-major or color-major order. Possible values are: <i>band-major</i> – Default value. <i>color-major</i> – Only an option when dealing with non-interleaved data.
<i>BandWidth</i>	integer	Width of output bands expressed in pixels.
<i>ColorantDepth</i>	integer	Number of bits per colorant. Determines whether the output is bitmaps or bytemaps. A value of 1 implies that a bitmap is used and that halftone screening is performed by the interpretation process.
<i>Interleaved</i>	boolean	If <i>true</i> , the resulting colorant values are interleaved and <i>BandOrdering</i> is ignored.
RenderingResolution +	element	Elements which define the resolutions to render the contents at. More than one element may be used to specify different resolutions for different SourceObject types.

RenderingResolution Structure

Name	Data Type	Description
<i>Resolution</i>	XYPair	Horizontal and vertical output resolution in DPI.
<i>SourceObjects ?</i>	enumeration	Identifies the class(es) of incoming graphical objects to render at the specified resolution. Possible values are: <i>All</i> – Default value. <i>ImagePhotographic</i> – Contone images. <i>ImageScreenShot</i> – Images largely comprised of rasterized vector art. <i>LineArt</i> <i>SmoothShades</i> – Gradients and blends. <i>Text</i>

7.2.78 RingBindingParams

This resource describes the details of the *RingBinding* process.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	<i>RingBinding</i>
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>BinderColor</i> ?	colorant	Color of the ring binder.
<i>BinderName</i> ?	string	The name of the binder manufacturer and the name of the specific item.
<i>RingDiameter</i> ?	double	The diameter of the ring determines the diameter of the sheet's holes.
<i>RingMechanic</i> ?	boolean	If <i>true</i> , a hand lever is available for opening. Default = <i>false</i>
<i>RingSystem</i> ?	enumeration	The following ring binding systems are used: <i>2Hole</i> – in Europe <i>3Hole</i> – in North America <i>4Hole</i> – in Europe
<i>SpineColor</i> ?	colorant	Color of the binders spine.
<i>SpineWidth</i> ?	double	The spine width is determined by the final height of the block of sheets to be bound.

7.2.79 RunList

RunList resources describe an ordered set of **LayoutElement** or **ByteMap** elements. **RunList** resources are an ordered list of **Run** elements.

RunList resources are used whenever an ordered set of page descriptions elements are required. Depending on the process usage of a **RunList**, only certain *Types* of **LayoutElement** may be valid. For example, a pre-rip imposition process requires **LayoutElement** elements of *Type page* or *document*, whereas a post-rip imposition process requires **ByteMap** elements. The usage is detailed in the descriptions of the processes that use the **RunList** resource.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	RunLists are used as input resources by most processes that act on content data
Output of processes:	RunLists are used as output resources by most processes that act on content data

Resource Structure

Name	Data Type	Description
<i>NPage</i> ?	integer	Total number of pages (placed object slots) that are defined by the RunList . If <i>NPage</i> is not specified, it defaults to all pages in the Run elements that make up the RunList .
Run *	element	Describes a range of LayoutElement or ByteMap elements.

Structure of Run Sub-element

Run elements describe the separations a range of pages.

Name	Data Type	Description
<i>EndOfDocument</i> ?	boolean	If <i>true</i> , the last page in the Run is the last page of an instance document. The precise handling of instance-document changes is defined in the InsertSheet resource. Default = <i>false</i>
<i>NPage</i> ?	integer	Total number of pages (placed object slots) that are defined by the Run element. If <i>NPage</i> is not specified, the number of pages defaults to all pages in the files that make up the run.
DynamicInput *	element	Replacement text for a DynamicField element. This information defines the contents of a dynamic mark on the Layout for automated page layout. The mark must be filled using information from the document runlist, such as the barcode of the recipient. This information varies with the document content. DynamicInput elements have one optional <i>Name</i> attribute that, when linked to the <i>ReplaceField</i> attribute of the DynamicField element, defines the string that should be replaced.
RunSeparation ?	element	Describes a range of LayoutElement or ByteMap elements for one or more separations.

Structure of RunSeparation Sub-element

RunSeparation elements describe either a separation or an unseparatedrange of pages, and must contain either a **LayoutElement** or **ByteMap** element that describes the data stream.

Name	Data Type	Description
<i>DocNames</i> ?	NameRangeList	A list of named documents in a multi-document file that supports named access to individual documents. <i>DocNames</i> defaults to all documents. If <i>DocNames</i> occurs in the RunList , <i>Docs</i> is ignored if it is also present.
<i>Docs</i> ?	IntegerRange-List	0-based list of document indices in a multi-document file specified by the LayoutElement element.
<i>EndOfDocument</i> ?	boolean	If <i>true</i> , the last page in the Run is the last page of an instance document. The precise handling of instance-document changes is defined in the InsertSheet resource. Default = <i>false</i>
<i>FirstPage</i> ?	integer	First page in the document that is described by this run. This attribute is generally used to describe pre-separated files. Default = 0

<i>NPage</i> ?	integer	Total number of pages (placed object slots) that are defined by the Run element. If <i>NPage</i> is not specified, the number of pages defaults to all pages in the files that make up the run.
<i>Pages</i> ?	IntegerRange-List	0-based list of indices in the documents specified by the LayoutElement element and the <i>Docs</i> attribute. If <i>Pages</i> is present, <i>FirstPage</i> , and <i>SkipPage</i> are ignored.
<i>PageNames</i> ?	NameRangeList	A list of named pages in a multi-page file that supports named access to individual pages. <i>PageNames</i> defaults to all pages. If <i>PageNames</i> occurs in the RunList , <i>FirstPage</i> , <i>Npage</i> , <i>SkipPage</i> and <i>Pages</i> are ignored if any of them is also present.
<i>Separation</i> ?	string	The name of the separation. The default separation name “All” has a special meaning. In that case, this RunSeparation element should be applied to all separations.
<i>SkipPage</i> ?	integer	Used when the run comprises every Nth page of the file. <i>SkipPage</i> indicates the number of pages to be skipped between each of the pages that comprise the run. This is generally used to describe pre-separated files, or to select only even or odd pages. Default = 0
<i>ByteMap</i> ?	element	Describes the page or stream of pages. Only one of ByteMap or LayoutElement may be specified in one run.
<i>DynamicInput</i> *	element	Replacement text for a DynamicField element. This information defines the contents of a dynamic mark on the Layout for automated page layout. The mark must be filled using information from the document runlist, such as the barcode of the recipient. This information varies with the document content. DynamicInput elements have one optional <i>Name</i> attribute that, when linked to the <i>ReplaceField</i> attribute of the DynamicField element, defines the string that should be replaced.
<i>LayoutElement</i> ?	element	Describes the document, page or image. Only one of ByteMap or LayoutElement may be specified in one run.

Structure of a DynamicInput Sub-element

DynamicInput defines the contents of a dynamic mark on a **Surface** resource for automated page layout. The mark must be filled using information from the document runlist, such as the barcode of the recipient. This information varies with the document content. For details on dynamic marks, see the **DynamicField** element description in section 7.2.88 *Surface*.

Name	Data Type	Description
<i>Name?</i>	string	Label that must match the <i>ReplaceField</i> attribute of the appropriate <i>DynamicField</i> element
-	text	Defines the text string that should be inserted as a replacement for the text defined in <i>ReplaceField</i> of a <i>DynamicField</i> element.

7.2.80 SaddleStitchingParams

This resource provides the parameters of the **SaddleStitching** process

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	SaddleStitching
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>NumberOfStitches</i>	integer	The number of stitches that will be made.
<i>StitchPositions ?</i>	NumberList	Array containing the stitch positions along the saddle. The center of the stitch must be specified, and the number of entries must match the number given in the <i>NumberOfStitches</i> attribute.
<i>StapleShape ?</i>	enumeration	Shape of staples. Possible values are: <i>Crown</i> <i>Overlap</i> <i>Butted</i> <i>ClinchOut</i> <i>Eyelet</i> These values are displayed in Figure 7.14, below.
<i>StitchWidth ?</i>	double	Width of each stitch.
<i>WireGauge ?</i>	double	Width of the wire being used.
<i>WireBrand ?</i>	string	Brand of wire being used.

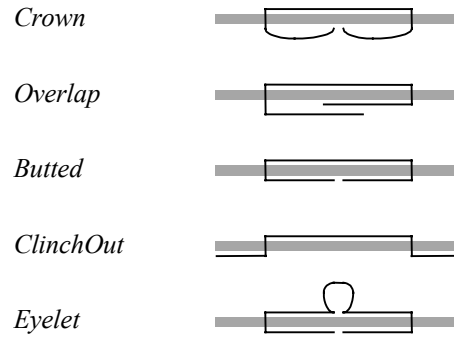


Figure 7.14 Staple shapes

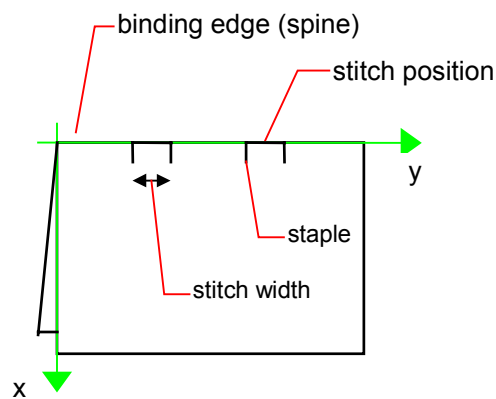


Figure 7.15 Parameters and coordinate system used for saddle stitching

The process coordinate system is defined as follows:

The Y-axis is aligned with the binding edge, and increases from the registered edge to the edge opposite the registered edge. The X-axis, meanwhile, is aligned with the registered edge. It increases from the binding edge to the edge opposite the binding edge, which is the product front edge.

7.2.81 ScanParams

This resource provides the parameters for the **Scanning** process.

Resource Properties

Resource class: Parameter
 Resource referenced by: -
 Partition: -
 Input of processes: **Scanning**
 Output of processes: -

Resource Structure

Name	Data Type	Description
<i>BitDepth</i>	integer	Bit depth of a one-color separation.
<i>CompressionFilter ?</i>	enumeration	Specifies the compression filter to be used. Possible

		values include: <i>CCITTFaxEncode</i> – Used to select CCITT Group 3 or 4 facsimile encoding. <i>DCTEncode</i> – Used to select JPEG compression. <i>FlateEncode</i> – Used to select ZIP compression. <i>DCTEncode</i> – Used to select JPEG compression.
<i>DCTQuality</i> ?	number	A value between 0 and 1 that indicates ‘how much’ the process should compress images. 0.0 means ‘do as loss-less compression as possible;’. 1.0 means do the maximum compression possible.’
<i>CorrectionProfile</i> ?	URL	ICC Profile with color corrections.
<i>InputBox</i> ?	rectangle	Rectangle that describes the image section to be scanned, in points. The origin of the coordinate system is the lower left corner of the physical item to be scanned.
<i>TargetProfile</i> ?	URL	ICC Profile that defines the target output device for a device specific scan, such as the profile of a CMYK press.
<i>Magnification</i> ?	XYPair	Size of the output/size of the input for each dimension. Default = 1.0.
<i>MountID</i> ?	string	ID of the drum or other mounting device upon which the media should be mounted.
<i>Mounting</i> ?	enumeration	Specifies how to mount originals. Possible values are: <i>unfixed</i> – Original lies unfixed on the scanner tray/drum. <i>fixed</i> – Original is fixed on the scanner tray/drum with transparent tape. <i>wet</i> – Original is put in gel or oil and fixed on the scanner tray/drum. <i>registered</i> – Original is fixed with registration holes. This value is used for copix.
<i>OutputColorSpace</i>	enumeration	Color space of the output images. Possible values are: <i>LAB</i> <i>RGB</i> <i>CMYK</i> <i>GreyScale</i> .
<i>OutputResolution</i>	XYPair	X and Y resolution of the output bitmap (in DPI).
<i>OutputSize</i> ?	XYPair	X-,Y-dimension of the intended output image (in pt).
<i>ScanProfile</i> ?	URL	ICC Profile of the scanner.
<i>SplitDocuments</i> ?	integer	A number representing how many images are scanned before a new file is created.
<i>DecompressionDictionary</i> ?	element	Details of the image data compression algorithm.
<i>FileSpec</i> ?	element	Name of output image file or files.

7.2.82 ScreeningParams

This resource specifies the parameter of the screening process. Since screening is, in most cases, very OEM specific, the following parameters are generic enough that they can be mapped onto a number of OEM controls.

Resource Properties

Resource class: Parameter
 Resource referenced by: **ExposedMedia**
 Partition: *Side, SheetName, SignatureName*
 Input of processes: **Screening, IDPrinting**
 Output of processes: -

Resource Structure

Name	Data Type	Description
<i>IgnoreSourceFile ?</i>	boolean	Specifies whether to ignore the screen settings (such as <i>setscreen</i> , <i>setcolorscreen</i> , and <i>sethalftone</i>) specified in the source files. Default = <i>true</i> Note: In some cases, Halftones are used to create patterns. In these cases, the halftone in the source PDL file will not be overridden.
<i>AbortJobWhenScreen-MatchingFails ?</i>	boolean	Specifies what happens when the device can not fulfill the screening requests. If <i>true</i> , it flushes the job. If <i>false</i> , it ignores matching errors using the default screening. Default = <i>false</i>
ScreenSelector +	element	List of screen selectors. A screen selector is included for each separation, including a default specification.

Structure of ScreenSelector Sub-elements

Description of screening for a selection of source object types and separations.

Name	Data Type	Description
<i>Angle ?</i>	real	Specifies the angle of the screen when AM screening is used.
<i>Frequency ?</i>	real	Specifies the line frequency of the screen when AM screening is used.
<i>ScreeningFamily ?</i>	string	Vendor specific screening family name. Possible values include: <i>Rational Tangent</i> <i>Adobe Accurate</i> <i>Agfa Balanced</i> <i>Soft-IS</i> <i>ErrorDiffusion</i>

<i>Separation ?</i>	string	 name of the separation. The default separation is "All" has a special meaning. In that case, this <i>ScreenSelector</i> should be applied to all separations.[DH41]
<i>SpotFunction ?</i>	NMTOKEN	<p>Specifies the spot function of the screen when AM screening is used. These names are the same as the spot function names defined in PDF:</p> <p><i>Round</i></p> <p><i>Diamond</i></p> <p><i>Ellipse</i></p> <p><i>EllipseA</i></p> <p><i>InvertedEllipseA</i></p> <p><i>EllipseB</i></p> <p><i>EllipseC</i></p> <p><i>InvertedEllipseC</i></p> <p><i>Line</i></p> <p><i>LineX</i></p> <p><i>LineY</i></p> <p><i>Square</i></p> <p><i>Cross</i></p> <p><i>Rhomboid</i></p> <p><i>DoubleDot</i></p> <p><i>InvertedDoubleDot</i></p> <p><i>SimpleDot</i></p> <p><i>InvertedSimpleDot</i></p> <p><i>CosineDot</i></p> <p><i>Double</i></p> <p><i>InvertedDouble</i></p>
<i>SourceObjects ?</i>	enumerations	<p>Identifies the class(es) of incoming graphical objects on which to use the selected screen. Possible values are:</p> <p><i>All</i> – Default value.</p> <p><i>ImagePhotographic</i> – Contone images.</p> <p><i>ImageScreenShot</i> – Images largely comprised of rasterized vector art.</p> <p><i>Text</i></p> <p><i>LineArt</i></p> <p><i>SmoothShades</i> – Gradients and blends.</p>

7.2.83 SeparationControlParams

This resource provides the controls needed to separate composite color files.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	Separation
Output of processes:	-

Resource Structure

Name	Data Type	Description
AutomatedOverprint-Params ?	element	Optional controls for overprint substitutions. Defaults to no automated overprint generation.
TransferFunctionControl ?	element	Controls whether the device performs transfer functions and what values are used when doing so.

Structure of SeparationControlParams Sub-elements

AutomatedOverprintParams

Name	Data Type	Description
OverPrintBlackText ?	boolean	Indicates whether OverPrint should be set to true for black text. If false, <i>TextSizeThreshold</i> and <i>TextBlackLevel</i> are ignored.
TextSizeThreshold ?	integer	Indicates the point size for text below which black text will be set to overprint.
TextBlackLeve ?!	number	A value between 0.0 and 1.0 which indicates the minimum black level for the text stroke or fill colors that cause the text to be set to overprint.
OverPrintBlackLineArt ?	boolean	Indicates whether overprint should be set to true for black line art. If false, <i>LineArtBlackLevel</i> is ignored.
LineArtBlackLevel ?	number	A value between 0.0 and 1.0 which indicates the minimum black level for the stroke or fill colors that cause the line art to be set to overprint.

TransferFunctionControl

Name	Data Type	Description
TransferFunctionSource	enumeration	Identifies the source of transfer curves which should be applied during separation. <i>Document</i> – Use the transfer curves provided in the document. <i>Device</i> – Use transfer functions provided by the output device. (When Separation is being performed pre-RIP, this may mean that no transfer curves will be applied.) <i>Custom</i> – Use the transfer curves provided in the <i>TransferCurvePool</i> element of this element.
TransferCurvePool ?	element	Provides a set of transfer curves to be used by the

Separation process.

7.2.84 SeparationSpec

This resource specifies a specific separation, and is usually used to define a list or sequence of separations.

Resource Properties

Resource class: Element
 Resource referenced by: **ColorantControl, LayoutElement, RegisterMark, TrappingDetails**
 Partition: -
 Input of processes: -
 Output of processes: -

Resource Structure

MarkSeparation

Name	Data Type	Description
<i>SeparationName</i>	string	Name of one specific separation.

7.2.85 Sheet

This resource provides a description of a sheet, as well as the marks on that sheet.

Resource Properties

Resource class: Parameter
 Resource referenced by: **InsertSheet, Layout**
 Partition: -
 Input of processes: ***InkZoneCalculation***
 Output of processes: -

Resource Structure

Name	Data Type	Description
<i>LockOrigins ?</i>	boolean	Determines the relationship of the coordinate systems for front and back surfaces. When <i>false</i> , all contents for all surfaces are transformed into the first quadrant, in which the origin is at the lower left corner of the surface. When <i>true</i> , contents for the front surface are imaged into the first quadrant (as above), but contents for the back surface are imaged into the second quadrant, in which the origin is at the lower right.
<i>Name</i>	NMTOKEN	Unique name of the sheet. <i>Name</i> is used for external reference to a sheet in, for example, a Part element.
<i>SurfaceContentsBox ?</i>	rectangle	This box, specified in surface-coordinate space, defines the area into which contents and marks will occur for all

Surfaces in the **Sheet**.

CTMs for **MarkObjects** or **ContentObjects** transform page contents or marks into this rectangle.

InsertSheet *	element	Specifies how to complete a sheet in an automated printing environment.
Surface *	element	Describes the surface to be used. Two surfaces may be attached: one front surface and one back surface.

7.2.86 SideSewingParams

This resource provides the parameters for the **SideSewing** process.

The process coordinate system is defined in the following way: the Y-axis is aligned with the binding edge. It then increases from the registered edge to the edge opposite to the registered edge. The X-axis is aligned with the registered edge, which then increases from the binding edge to the edge opposite to the binding edge, i.e. the product front edge.

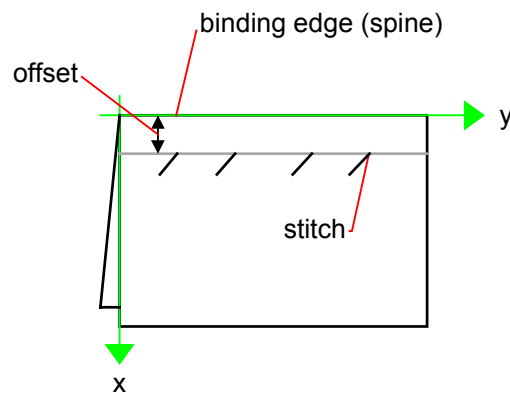


Figure 7.16 Parameters and coordinate system used for side sewing

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	SideSewing
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>NumberOfNeedles</i>	integer	Specifies the number of needles to be used.
<i>NeedlePositions ?</i>	NumberList	Array containing the Y-coordinates of the needle positions. The number of entries must match the number given in <i>NumberOfNeedles</i> .

<i>Offset</i>	double	Specifies the distance between the stitch and the binding edge.
<i>SewingPattern ?</i>	enumeration	Specifies the sewing pattern to be used. Possible values are: <i>Normal</i> <i>Staggered</i> <i>CombinedStaggered</i>
<i>ThreadMaterial ?</i>	enumeration	Specifies the thread material to be used. Possible values are: <i>Cotton</i> <i>Nylon</i> <i>Polyester</i>
<i>ThreadThickness ?</i>	double	The thickness of the thread to be used.
<i>ThreadBrand ?</i>	string	The brand of thread to be used.

7.2.87 StitchingParams

This resource provides the parameters for the **Stitching** process.

The process coordinate system is defined as follows:

If there is a binding edge, the y-axis is aligned with this edge. Otherwise the y-axis is aligned with one of the registered edges. The y-axis increases from the (first) registered edge to the edge opposite to the registered edge. The x-axis is aligned with the (second) registered edge. It increases from the binding edge (or first registered edge) to the edge opposite to the binding edge (or first registered edge).

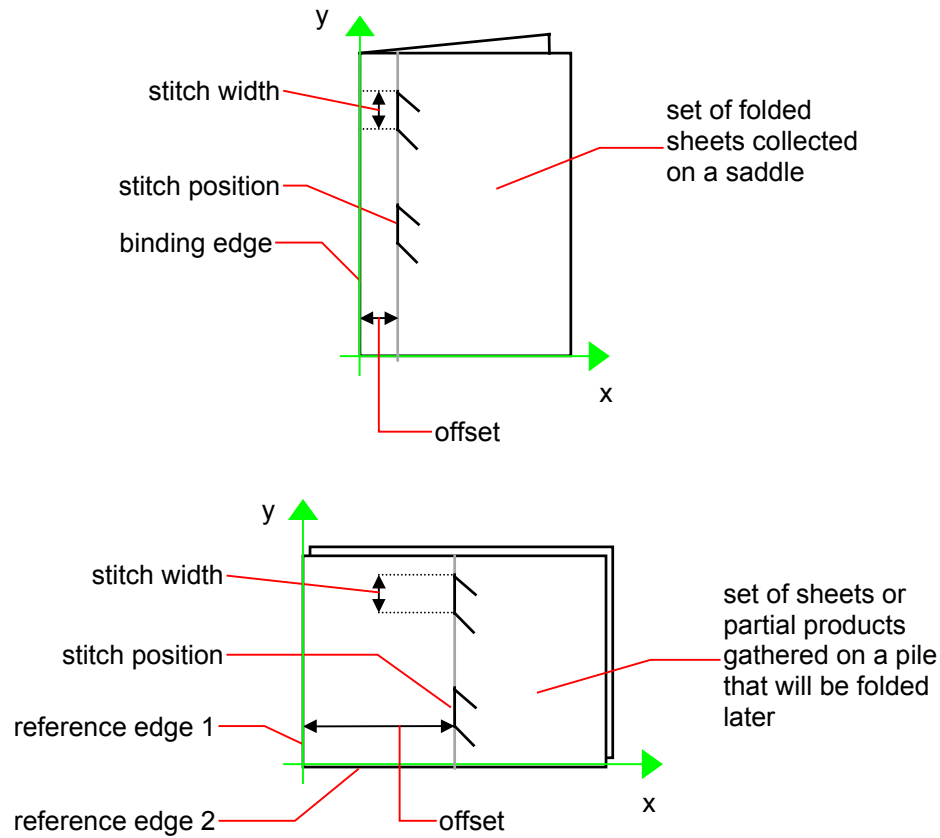


Figure 7.17 Parameters and coordinate system used for stitching

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	Stitching
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>Angle ?</i>	double	Angle of stitch in degree. The angle increases in a counterclockwise direction. 0 = horizontal, which means that it is parallel to the X-axis of the operation coordinate system.
<i>NumberOfStitches</i>	integer	Number of stitches.
<i>Offset</i>	double	Distance between stitch and binding edge.
<i>StapleShape ?</i>	enumeration	Specifies the shape of the staples to be used. Possible values are: <i>Crown</i>

*Overlap**Butted**ClinchOut**Eyelet*

Representations of these values are displayed in Figure 7.14.

<i>StitchFromFront</i>	boolean	If <i>true</i> , Stitching is done from front to back. Otherwise it is done from back to front.
<i>StitchPositions ?</i>	NumberList	Array containing the stitch positions. The center of the stitch must be specified, and the number of entries must match the number given in <i>NumberOfStitches</i> .
<i>StitchWidth ?</i>	double	Width of the stitch to be used.
<i>WireGauge?</i>	double	Width of the wire to be used.
<i>WireBrand?</i>	string	Brand of the wire to be used.

7.2.88 Surface

This resource describes the marks on a sheet surface. Up to two **Surface** resources may exist on a **Sheet**.

Resource Properties

Resource class:	Parameter
Resource referenced by:	Sheet
Partition:	-
Input of processes:	-
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>Side</i>	enumeration	The side of the Sheet that the Surface describes. Possible values are: <i>Front</i> <i>Back</i>
<i>SurfaceContentsBox ?</i>	rectangle?	This rectangle provides the region of the surface into which the contents of ContentObjects and MarkObjects are to be imaged.
PlacedObject *	element	Provides a list of the ContentObject and MarkObject elements to be placed on to the surface. Contains the marks on the surface in rendering order. See the description that follows. Note that PlacedObject is not a container but an abstract type.

Structure of the abstract PlacedObject Sub-element

The marks that may be placed on the designated **Surface** come in two varieties: ContentObject or MarkObject elements. Both inherit characteristics from the abstract PlacedObject element type, and both are described below.

Name	Data Type	Description
<i>ClipBox</i> ?	rectangle	Clip path in the coordinates of the <i>SurfaceContentsBox</i> .
<i>CTM</i>	matrix	Transformation matrix of the object in the <i>SurfaceContentsBox</i> .
<i>SourceClipPath</i> ?	path	Clip path for the <i>PlacedObject</i> in the coordinates of the source page.
<i>Type</i>	enumeration	Describes the kind of <i>PlacedObject</i> . Possible values are: <i>Content</i> <i>Mark</i>

Structure of ContentObject Sub-elements

ContentObject elements describe containers for page content on a surface. They are filled from the *Content RunList* of the *Imposition* process.

Using Expressions in the OrdExpression Attribute

Expressions can use the operators +, -, *, /,%and parentheses, operating on integers and two variables: *s* for signature number (starting at 0) and *n* for number of pages to be imposed in one document. The operators have the same meaning as in the C programming language. Expressions are evaluated with normal “C” operator precedence. Multiplication must be expressed by explicitly including the * operator – that is, use “2*s”, not “2 s”. Remainders are discarded.

For print applications where page count varies from Instance Document to Instance Document, imposition templates can automatically assign pages to the correct **Surface** and *PlacedObject* position.

Name	Data Type	Description
<i>HalfTonePhaseOrigin</i>	XYPair	Location of the origin for screening of this <i>ContentObject</i> . Specified in the coordinate systems of <i>SurfaceContentBox</i> .
<i>Ord</i> ?	integer	Reference to an index in the content RunList .
<i>OrdExpression</i> ?	string	Function to calculate an <i>Ord</i> value dynamically, using a value of <i>s</i> for signature number and <i>n</i> for total number of pages in the instance document. <i>Ord</i> and <i>OrdExpression</i> are mutually exclusive in one <i>PlacedObject</i> .

Structure of MarkObject Elements

MarkObject elements describe containers for page marks on a surface. They are filled from the *Marks RunList* of the *Imposition* process.

Of the last six element described in the following table (*RegisterMark*, *CIELABMeasuringField*, *DensityMeasuringField*, *ColorControlStrip*, *CutMark*, and *IdentificationField*), only one can be valid at any given time.

Name	Data Type	Description
<i>Ord</i> ?	integer	Reference to an index in the marks RunList

CIELABMeasuringField ?	element	Specific information about this kind of mark object.
ColorControlStrip ?	element	Specific information about this kind of mark object.
CutMark ?	element	Specific information about this kind of mark object.
DensityMeasuringField ?	element	Specific information about this kind of mark object.
DynamicField *	element	Definition of text replacement for a MarkObject .
IdentificationField ?	element	Specific information about this kind of mark object.
LayoutElement ?	element	PDL description of the mark. LayoutElement and Ord are mutually exclusive within one MarkObject .
RegisterMark ?	element	Specific information about this kind of mark object.

Only one of these last six elements may be valid at any given time.

DynamicField Sub-element Properties

DynamicField provides a description of dynamic text replacements for **MarkObjects**. This element should be used for production purposes, such as defining bar codes for variable data printing.

DynamicField elements are not intended as a placeholders for actual content such as addresses. Rather, they are marks with dynamic data such as time stamps and database information.

Dynamic objects are **MarkObjects** with optional additional **DynamicField** elements that define text replacement.

Example usage of a DynamicField Element:

```
<!--The RunList entry: -->
<Run ... >
  <LayoutElement Type="graphics">
    <LayoutFile URL="Variable.pdf"/>
    <DynamicInput Name="i1">Joe</DynamicInput>
    <DynamicInput Name="i2">John</DynamicInput>
  </LayoutElement>
</Run>

...

<!--The MarkObject in the Layout hierarchy: -->
<MarkObject Ord="0" CTM=... (...) >
  <LayoutElement Type="graphics">
    <LayoutFile URL="MyReplace.pdf"/>
  </LayoutElement >
  <DynamicField ReplaceField="__xxx__"
    Format="Replacement Text for %s and %s go in here at %s on %s"
    Template="i1,i2,Time,Date" Ord="0"/>
</MarkObject>
```

In the example above, the text “__xxx__” in the file **MyReplace.pdf** would be replaced by the sentence “Replacement Text for Joe and John go in here at 14:00 on Mar-31-2000”.

MyReplace.pdf is placed at the position defined by the CTM of the **MarkedObject** and **Variable.pdf** is placed at the position defined by the CTM of the **PlacedObject**.

Structure of DynamicField Sub-element

Name	Data Type	Description
<i>Format</i>	string	Format string in C printf format that defines the replacement.
<i>InputField</i> ?	string	String that must be replaced by the <i>DynamicInput</i> attribute in the Contents RunList referenced by <i>Ord</i> or <i>OrdExpression</i> .
<i>Ord</i> ?	integer	Reference to an index in the Contents RunList that contains <i>DynamicInput</i> elements.
<i>OrdExpression</i> ?	string	Reference to an index in the Contents RunList that contains <i>DynamicInput</i> fields. For details, see the definition of <i>OrdExpression</i> in the description of the <i>PlacedObject</i> element.
<i>ReplaceField</i>	string	String that must be replaced by the instantiated text expression as defined by the <i>Format</i> and <i>Template</i> attributes in the file referenced by <i>Run</i> .
<i>Template</i>	string	Template to define a sequence of variables consumed by <i>Format</i> . A list of pre-defined values is found in the description of the FileSpec resource. In addition, <i>DynamicInput</i> elements of a RunList define further variables.

7.2.89 ThreadSewingParams

This resource provides the parameters for the **ThreadSewing** process. It may also specify a gluing application, which would be used principally between the first and the second or the last and the last sheet but one. A gluing application might also be necessary if different types of paper are used.

The process coordinate system is defined as follows:

The Y-axis is aligned with the binding edge. It increases from the registered edge to the edge opposite to the registered edge. The X-axis is aligned with the registered edge. It increases from the binding edge to the edge opposite to the binding edge, i.e. the product front edge.

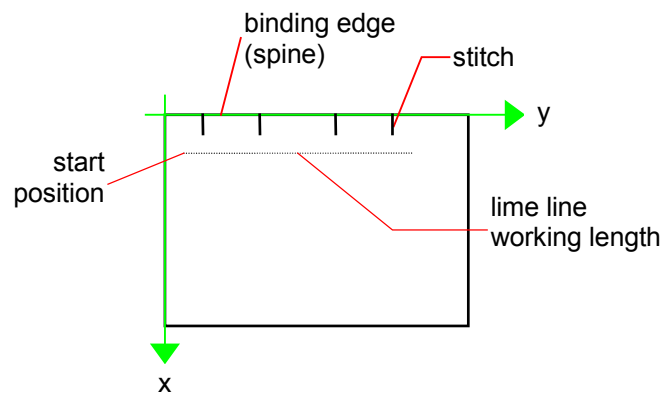


Figure 7.18 Parameters and coordinate system used for thread sewing

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	<i>ThreadSewing</i>
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>BlindStitch</i>	boolean	If <i>true</i> , a blind stitch after last stitch is required.
<i>CastingMaterial</i> ?	enumeration	Casting material of the thread being used. Possible values are: <i>Cotton</i> <i>Nylon</i> <i>Polyester</i>
<i>CoreMaterial</i> ?	enumeration	Core material of the thread being used. This attribute must be used to define the thread material if there is no casting. Possible values are: <i>Cotton</i> <i>Nylon</i> <i>Polyester</i>
<i>GlueLineRefSheets</i>	IntegerList	This dictionary entry is only required if <i>GlueLine</i> is defined. It contains the indices of the loose parts of the input component after which liming should be applied. The index starts with 0.
<i>NumberOfNeedles</i>	integer	Specifies the number of needles to be used.
<i>NeedlePositions</i> ?	NumberList	Array containing the Y-coordinate of the needle positions. The number of entries must match the number given in <i>NumberOfNeedles</i> .
<i>Sealing</i>	boolean	If <i>true</i> , thermo-sealing is required.
<i>SewingPattern</i> ?	enumeration	Sewing pattern. Possible values are: <i>Normal</i> <i>Staggered</i> <i>CombinedStaggered</i>
<i>ThreadThickness</i> ?	double	Thread thickness.
<i>ThreadBrand</i> ?	string	Thread brand.
<i>GlueLine</i> *	element	Gluing parameters.

7.2.90 Tile

Each **Tile** resource defines how content from a **Surface** resource will be imaged onto a piece of media that is smaller than the designated surface.

Tiling occurs in some production environments when pages are imaged on to an intermediate medium and the resulting image of the surface is larger than the media. In this case, instructions are needed to determine how the intermediate media (tiles) will be assembled to achieve the desired output (such as a single plate for the surface). For example, a device might require that four pieces of film be assembled to create the image for the plate.

In general, a **Tile** resource will be partitioned (see section 3.8.2 Description of Partitionable Resources) by *TileID*. Individual tiles are selected and matched by specifying the appropriate *TileID* attribute, which is described in Table 3.18 Contents of the Part element.

Resource Properties

Resource class: Parameter
 Resource referenced by: -
 Partition: *TileID*
 Input of processes: *Tiling*
 Output of processes: -

Resource Structure

Name	Data Type	Description
<i>ClipBox</i>	rectangle	A rectangle that defines the bounding box of the Surface contents which will be imaged on this Tile . The <i>ClipBox</i> is defined in the coordinate system of the Surface .
<i>CTM</i>	matrix	A coordinate transformation matrix mapping the <i>ClipBox</i> for this Tile to the rectangle [0 0 X Y], where X and Y are the extents of the media that the Tile will be imaged onto.

7.2.91 TransferCurvePool

A transfer curve pool is a collection of **TransferCurveSet** elements that each contains information about a **TransferCurve**. Multiple **TransferCurvesSets** may exist at one time. For example, one may exist for the laser calibration of the imagesetter, one for the **FilmToPlateCopy** process and one for the printing process. Each **TransferCurveSet** consists of one or more **TransferCurve** elements. A **TransferCurve** element should be applied to the appropriate correlative *Separation*, or to all *Separations* when *Separation = All*. All **TransferCurveSets** should be concatenated in the order they appear in the **TransferCurvePool**.

Resource Properties

Resource class: Parameter
 Resource referenced by: - SeparationControlParams
 Partition: -
 Input of processes: *InkZoneCalculation*
 Output of processes: -

Resource Structure

Name	Data Type	Description
------	-----------	-------------

TransferCurveSet * element The set of transfer curves.

Structure of TransferCurvePool Sub-element

TransferCurveSet

Name	Data Type	Description
<i>Name</i>	NMTOKENS	The name of the TransferCurveSet. Possible values include: <i>Laser</i> <i>Film2Plate</i> <i>Press</i>
TransferCurve *	element	List of TransferCurve entries.

Structure of TransferCurveSet Sub-element⁵

TransferCurve

Name	Data Type	Description
<i>Curve</i>	TransferFunction	The transfer function.
<i>Separation</i> ?	string	The name of the separation. The default separation name “All” has a special meaning. In that case, this curve should be applied to all separations.


7.2.92 TrappingDetails

This resource identifies the root of the hierarchy of resources. This hierarchy controls the *Trapping* process.



Resource Properties

Resource class: Parameter
 Resource referenced by: Any process that uses **RunList** resources
 Partition: -
 Input of processes: *Trapping*
 Output of processes: -

Resource Structure

Name	Data Type	Description
 <i>ColorantSetName</i> ?	string	A string used to identify the named colorant parameter set.[RP42]
<i>DefaultTrapping</i> ?	boolean	If <i>true</i> , pages that have no defined TrapRegions are trapped using the DefaultParams set of TrappingParameters. The BleedBox is used for the

⁵ Note that this is identical to the TransferCurve element in a **Color** resource

		TrapZone.
		If <i>false</i> , only pages that have TrapRegions are trapped.
		Default = <i>false</i>
<i>IgnoreFileParams</i> ?	boolean	If <i>true</i> , any trapping controls provided within any source files used by this process are ignored.
		If <i>false</i> , trapping controls embedded in the source files are honored.
		Default = <i>true</i>
 <i>Trapping</i> ?	boolean	If <i>true</i> , trapping is enabled. If omitted, the default setting for the device is used.[RP43]
<i>TrappingOrder</i> ?	element	Trapping processes will trap colorants as if they are laid down on the media in the order specified in <i>TrappingOrder</i> . The colorant order may affect which colors to spread, especially when opaque inks are used.
 <i>TrappingType</i> ?	integer	Identifies the trapping method to be used by the trapping process. The number identifies the minor (last three digits) and major (any digits prior to the last three) version of the trapping type requested.[RP44]
<i>DefaultParams</i> ?	element	A TrappingParams resource that is used when <i>DefaultTrapping</i> = <i>true</i> .

Structure of theTrappingOrder Sub-element

Name	Data Type	Description
<i>SeparationSpec</i> *	element	An array of colorant names.

7.2.93 TrappingParams

This resource provides a set of controls that are used to generate traps. The values of the parameters are chosen based on the customer's trapping strategy, and depend largely on the content of the pages to be trapped and the characteristics of the output device (press).

The attributes of this resource that are optional are optional in the sense that each implementation decides a default value for them.

Resource Properties

Resource class:	Parameter
Resource referenced by:	TrapRegion
Partition:	-
Input of processes:	<i>Trapping</i>
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>BlackColorLimit</i>	number	A number between 0 and 1 that specifies the lowest color value required for trapping a colorant according to

		the black trapping rule. This entry uses the subtractive notion of color, where 0 is white, or no colorant, and 1 is full colorant.
<i>BlackDensityLimit</i>	number	A positive number that specifies the lowest neutral density of a colorant for trapping according to the black trapping rule.
<i>BlackWidth ?</i>	number	A positive number that specifies the trap width for trapping according to the black trapping rule. <i>BlackWidth</i> is specified in <i>TrapWidth</i> units; a value of 1 means that the black trap width is one <i>TrapWidth</i> wide. The resulting black trap width is subject to the same device limits as <i>TrapWidth</i> .
<i>Enabled</i>	boolean	If <i>true</i> , trapping is enabled for zones that are defined with this parameter set.
<i>HalftoneName</i>	string	A name that identifies a halftone object to be used when marking traps. The name is the value of the <i>ResourceName</i> attribute of some PDLResourceAlias resource. If absent, the halftone in effect just before traps are marked will be used, which may cause unexpected results.
<i>ImageInternalTrapping ?</i>	boolean	If <i>true</i> , the planes of color images are trapped against each other. If <i>false</i> , the planes of color images are not trapped against each other.
<i>ImageResolution ?</i>	integer	A positive integer indicating the minimum resolution, in dots per inch, for downsampled images. Images can be downsampled by a power of 2 before traps are calculated. The downsampled image is used only for calculating traps, while the original image is used when printing the image.
<i>ImageMaskTrapping ?</i>	boolean	Controls trapping when the <i>TrapZone</i> contains a stencil mask. A stencil mask is a monochrome image in which each sample is represented by a single bit. The stencil mask is used to paint in the current color: image samples with a value of 1 are marked; samples with a value of 0 are not marked. When <i>false</i> , none of the objects covered by the clipped bounding box of the stencil mask are trapped. No traps are generated between the stencil mask and objects that the stencil mask overlays. No traps are generated between objects that overlay the stencil mask and the stencil mask. For all other objects, normal trapping rules are followed. Two objects on top of the stencil mask that overlap each other, may generate a trap, regardless of the value of this parameter. When <i>true</i> , objects are trapped to the stencil mask, and to each other.

<i>ImageToImageTrapping ?</i>	boolean	If <i>true</i> , traps are generated along a boundary between images. If <i>false</i> , this kind of trapping is not implemented.
<i>ImageToObjectTrapping ?</i>	boolean	If <i>true</i> , images are trapped to other objects. If <i>false</i> , this kind of trapping is not implemented.
<i>ImageTrapPlacement ?</i>	string	Controls the placement of traps for images. Possible values are: <i>Center</i> – Trap is centered on the edge between the image and the adjacent object. <i>Choke</i> – Trap is placed in the image. <i>Normal</i> – Trap is placed in the adjacent object. <i>Spread</i> – Trap is based on the colors of the areas.
<i>MinimumBlackWidth ?</i>	number	Specifies the minimum width, in points, of a trap that uses black ink. Allowable values are those greater than or equal to zero. Default = 0
<i>SlidingTrapLimit ?</i>	number	A number between 0 and 1. Specifies when to slide traps towards a center position. If the neutral density of the lighter area is greater than the neutral density of the darker area multiplied by the <i>SlidingTrapLimit</i> , then the trap slides. This applies to vignettes and non-vignettes. No slide occurs at 1.
<i>StepLimit ?</i>	number	A number between 0 and 1. Specifies the smallest step required in the color value of a colorant to trigger trapping at a given boundary. If the higher color value at the boundary exceeds the lower value by an amount that is equal or greater than the larger of 0.05 or <i>StepLimit</i> times the lower value ($\text{low} + \max(\text{StepLimit} * \text{low}, 0.05)$), then the edge is a candidate for trapping. The value 0.05 is set to avoid trapping light areas in vignettes. This entry is used when not specified explicitly by a <i>ColorantZoneDetails</i> sub-element for a colorant.
<i>TrapColorScaling ?</i>	number	A number between 0 and 1. Specifies a scaling of the amount of color applied in traps towards the neutral density of the dark area. 1 means the trap has the combined color values of the darker and the lighter area. 0 means the trap colors are reduced so that the trap has the neutral density of the darker area. This entry is used when not specified explicitly by a <i>ColorantZoneDetails</i> sub-element for a colorant.
<i>TrapEndStyle ?</i>	enumeration	Instructs the trap engine how to form the end of a trap that touches another object. Possible values include: <i>Miter</i> <i>Overlap</i> Other values may be added later as a result of customer requests.

		Default = <i>Miter</i>
<i>TrapJoinStyle ?</i>	enumeration	Specifies the style of the connection between the ends of two traps created by consecutive segments along a path. Possible values are: <i>Bevel</i> <i>Miter</i> <i>Round</i> Default = <i>Miter</i>
<i>TrapWidth ?</i>	number	A positive number. Specifies the trap width in points. Also defines the unit used in trap width specifications for certain types or objects, such as <i>BlackWidth</i> . The valid range is usually at least 1 - 40 pixels.
<i>ColorantZoneDetails</i>	element	A <i>ColorantZoneDetails</i> sub-element. As with the entries in the <i>TrappingDetails::ColorantDetails</i> dictionary, entries in this dictionary reflect the results of any named colorant aliasing specified. Each entry defines parameters specific for one named colorant. If omitted for a specific colorant, the relevant parameters in the <i>TrappingDetails::ColorantDetails</i> dictionary are used. If the colorant named is neither listed in the <i>ColorantParams</i> array, nor implied by the <i>ProcessColorModel</i> , for the <i>ColorantControl</i> object in effect when these <i>TrappingParameters</i> are applied, the entry is not used for trapping.

Structure of ColorantZoneDetails Sub-element

Name	Data Type	Description
<i>Colorant</i>	string	The colorant name that occurs in the <i>ColorantParams</i> array of the <i>ColorantControl</i> object used by the process.
<i>StepLimit</i>	number	A number between 0 and 1. Specifies the smallest step required in the color value of a colorant to trigger trapping at a given boundary. If the higher color value at the boundary exceeds the lower value by an amount that is equal or greater than the larger of 0.05 or <i>StepLimit</i> times the lower value ($low + \max(StepLimit * low, 0.05)$), then the edge is a candidate for trapping. The value 0.05 is set to avoid trapping light areas in vignettes. If omitted, the <i>StepLimit</i> attribute in the TrappingParams resource is used.

<i>TrapColorScaling</i>	number	A number between 0 and 1. Specifies a scaling of the amount of color applied in traps towards the neutral density of the dark area. 1 means the trap has the combined color values of the darker and the lighter area. 0 means the trap colors are reduced so that the trap has the neutral density of the darker area. If omitted, the <i>TrapColorScaling</i> attribute in the TrappingParameters resource is used.
-------------------------	--------	---

7.2.94 TrapRegion

This resource identifies a set of pages to be trapped, an area of the pages to trap, and the parameters to use.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	<i>Trapping</i>
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>TrapZone</i> ?	path	Each element within <i>TrapZone</i> is one sub-path of a complex path. The <i>TrapZone</i> is the area that results when the paths are filled using the non-zero winding rule. When absent, the <i>MediaBox</i> array for the RunList defines the <i>TrapZone</i> .
<i>Pages</i>	IntegerRange-List	Identifies a set of pages from the RunList to trap using the specified geometry and trapping style.
TrappingParams	element	The set of TrappingParams which will be used when trapping in this region.

7.2.95 TrimmingParams

This resource provides the parameters for the *Trimming* process.

The process coordinate system is defined as follows:

The y-axis is aligned with the binding edge. It increases from the registered edge to the edge opposite to the registered edge. The x-axis is aligned with the registered edge. It increases from the binding edge to the edge opposite to the binding edge, i.e. the product front edge.

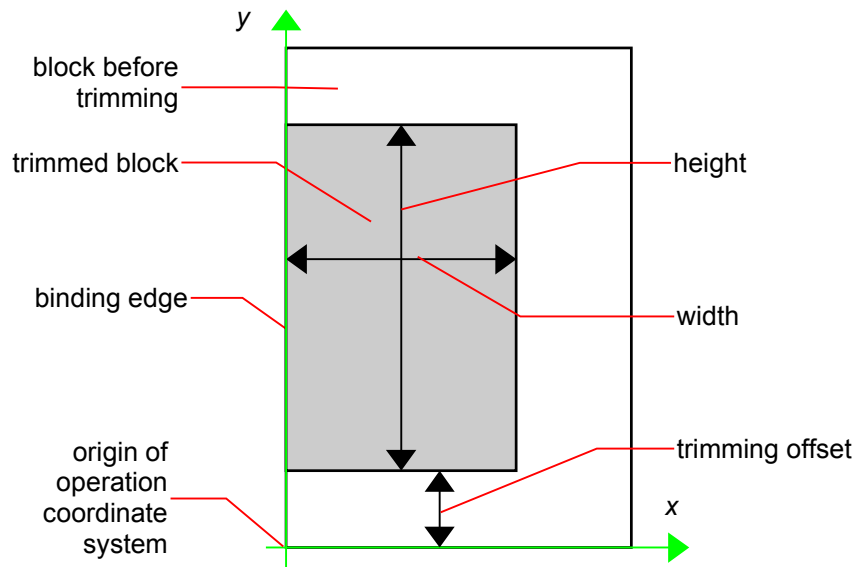


Figure 7.19 Parameters and coordinate system used for trimming

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	Trimming
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>Height ?</i>	double	Height of the trimmed product.
<i>TrimmingOffset ?</i>	double	Amount to be cut at bottom side.
<i>Width ?</i>	double	Width of the trimmed product.

7.2.96 VeloBindingParams

This resource describes the details of the **VeloBinding** process.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	VeloBinding
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>Brand</i> ?	string	The name of the comb manufacturer and the name of the specific item.
<i>Distance</i> ?	double	The distance between the pins and the distance between the holes of the pre-punched sheets must be the same.
<i>Length</i> ?	double	The length of the pin is determined by the height of the pile of sheets to be bound.
<i>StipColor</i> ?	colorant	Determines the color of the strip.

7.2.97 VerificationParams

This resource provides the parameters of a **Verification** process.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Partition:	-
Input of processes:	Verification
Output of processes:	-

Resource Structure

Name	Data Type	Description
<i>FieldRange</i> ?	IntegerRange List	Range list of integers that determines which characters of the data in IdentificationField should be applied to the field formatting strings. Defaults = "0~1", which means first-to-last.
<i>InsertError</i> ?	string	Database insertion statement in C printf format defining how information read from the IdentificationField resource of the Verification process should be stored in case of verification errors. The database is defined by the DBSelection resource of the Verification process. This field must be specified if a database is selected.
<i>InsertOK</i> ?	string	Database insertion statement in C printf format defining how information extracted from the IdentificationField should be stored in case of verification success. The database is defined by the DBSelection resource of the verification node. This field must be specified if a database is selected.
<i>Tolerance</i> ?	double	Ratio of tolerated verification failures to the total number of tests. 0 = none allowed, 1.0 = all.

Usage of FieldRange and Format Strings.

A database field name can be calculated from the characters of the **IdentificationField** using standard C printf notation and the *FieldRange* attribute. Each range that is defined in *FieldRange* is passed to printf as one string that is applied to the format. The order is maintained. Note that SQL was chosen for illustrative purposes only. The mechanism is defined for any database interface.

Example:

IdentificationField string : “1234:John Doe”

FieldRange : “6~1 1~4”

FieldOK : “Insert true into Va where Name = '%s' and ID = %s”

Resulting string: : “Insert true into Va where Name = 'John Doe' and ID = 1234”

7.2.98 WireCombBindingParams

This resource describes the details of the **WireCombBinding** process.

Resource Properties

Resource class: Parameter

Resource referenced by: -

Partition: -

Input of processes: **WireCombBinding**

Output of processes: -

Resource Structure

Name	Data Type	Description
<i>Brand ?</i>	string	The name of the comb manufacturer (e.g. <i>Wire-O</i> [®]) and the name of the specific item.
<i>Color ?</i>	colorant	Determines the color of the comb.
<i>Diameter ?</i>	double	The comb diameter is determined by the height of the block of sheets to be bound.
<i>Distance ?</i>	double	The distance between the “teeth” and the distance between the holes of the pre-punched sheets must be the same.
<i>Material ?</i>	enumeration	The material used for forming the wire comb binding. Possible values are: <i>LaqueredSteel</i> <i>TinnedSteel</i> <i>ZincsSteel</i>
<i>Shape ?</i>	enumeration	The shape of the wire comb binding. Possible values are: <i>single</i> – Each “tooth” is made with one wire <i>twin</i> – The shape of each “tooth” is made with a double wire
<i>Thickness ?</i>	double	The thickness of the comb material.

Chapter 8 Building a System Around JDF

8.1 Implementation Considerations and Guidelines

JDF parsing –JDF devices must implement JDF parsing. At a minimum, a device must be able to search the JDF to find a node whose process type it is able to execute. In addition, a device must be able to consume the inputs and produce the outputs for each process type it is able to execute.

Test run – To reduce failures during processing, it is recommended that either individual devices or their controller support the testrun functionality. This prevents the case where a device begins processing a node that is incomplete or mal-formed.

8.2 JDF and JMF Interchange Protocol

A system of vendor independent elements must define a protocol that allows them to interchange information based on JDF and JMF.

8.2.1 File-Based Protocol (JDF only)

The file-based protocol is only a solution for JDF job tickets, not JMF messages. A JMF-compliant controller must implement the HTTP protocol. A file-based protocol is based on hot-folders. Every processor must define an input hot-folder and an output folder for JDF. In addition the “SubmitQueueEntry” message contains a URL attribute that allows specification of arbitrary JDF locators.

Implementation of JDF file-based protocol is simple, but it is important to note that the protocol does not support acknowledgement receipts for protocol error handling. It requires that the receiver polls the output folder of the processor. Finally, granting read/write access to your hot-folder negates the security functions.

8.2.2 HTTP-Based Protocol (JDF + JMF)

HTTP is a stable, vendor-independent protocol, and it supports a variety of advantageous features. For example, it offers a wide availability of tools, it is already a common technology among vendors who use HTTP, and it has a well-defined query-response mechanism (HTTP POST message). It also offers widespread firewall support and secure connections via SSL when using HTTPS.

8.2.3 Protocol Implementation Details

JDF Messaging will not specify a standard port. We recommend that you use the standard HTTP port 80 in order to avoid firewall problems.

Implementation of Messages

Only HTTP servers may be targeted by queries. This is done with a standard HTTP Post request. The JMF is the body of the HTTP post message. The response is the body of the initiated HTTP post response.

HTTP Push Mechanisms

Since HTTP is a stateless protocol, push mechanisms, such as regular status bar updates, are non-trivial when communicating with a client. Work-arounds can, however, be implemented. For example, a Java applet that polls the server in regular intervals can be used.

8.2.4 Mime Types and File Extensions

JDF and JMF documents have a MIME type of Application/JDF and Application/JMF respectively. It is recommended that the controller use a file extension of .jdf when using file based jdf in an environment that supports file name extensions.

8.3 MIS Requirements

MIS systems may:

- ignore Audit elements when they receive complete information about a process execution via JMF.
- decompose JDF into an internal format such as database tables.

Appendix A Encoding

This appendix lists a number of commonly used JDF data types and structures and their XML encoding. Data types are simple data entities such as strings, numbers and dates. They have a very straightforward string representation and are used as XML attribute values. Data structures, on the other hand, describe more complex structures that are built from the defined data types, such as colors

A.1 XML Schema Data Types

JDF is based on the XML Schema specification. The JDF data types used in this specification are summarized in the table below and comply with the lexical representation of (primitive) data types defined by [[XML Schema Part 2: Datatypes](#)]. For a complete definition of each of these data types, please refer to the final specification of XML Schema Datatypes.

XML Data Type	Description	Example
boolean	Has the value space required to support the mathematical concept of binary-valued logic: {true, false}.	<Example Enable="true"/>
date	Represents a time period that starts at midnight on a specified day and lasts for 24 hours.	<Example StartDate="1999-05-31"/>
double	Corresponds to IEEE double-precision 64-bit floating point type	<Example Pi="3.14"/>
ID	Represents the ID attribute from [XML Specification Version 1.0]. It basically represents a name or string that contains no space characters.	<Example ID="R-16"/>
IDREF	Represents the IDREF attribute from [XML Specification Version 1.0]. For a valid XML-document an element with the ID value specified in IDREF must be present in the scope of the document.	<Example IDREF="R-16"/>
IDREFS	Represents the IDREFS attribute from [XML Specification Version 1.0]. More specifically, this is a whitespace-separated list of IDREFs.	<Example IDREFS="R-12 R-16"/>
integer	Represents numerical integer values.	<Example Copies="36"/>
language	Represents a natural language defined in IETF rfc 1766. http://www.ietf.org/rfc/rfc1766.txt	<Example Language="de"/>
NMTOKEN	Represents the NMTOKEN attribute type from [XML Specification Version 1.0]. It basically represents a name or string that contains no space characters.	<Example Alias="ABC_6"/>
NMTOKENS	Represents the NMTOKENS attribute type from [XML Specification Version 1.0]. More specifically, this is a whitespace-	<Example AliasList="ABC_6 ABCD_3 DEGF"/>

	separated list of NMTOKENs.	
string	Represents character strings in XML.	<Example Name="Test"/>
time	Represents an instant of time that recurs every day.	<Example StartAt="13:20:00-05:00"/>
timeInstant	Represents a combination of date and time values representing a specific instant of time.	<Example Start="1999-05-31T13:20:00-05:00"/>
timeDuration	Represents a duration of time.	<Example Duration="P1Y2M3DT10H30M"/>
URI	Short for URI-reference. Represents a Uniform Resource Identifier (URI) Reference as defined in Section 4 of [RFC 2396] .	<Example URI="file://hubble/test.txt"/>
URL	Short for URL-reference . Represents a Uniform Resource Locator (URL) Reference as defined in Section 4 of [RFC 2396] .	<Example URL="file://hubble/test.txt"/>

A.2 JDF Data Types

The data types listed and described in this section are defined by JDF. They are also found in PJTF and CIP3.

A.2.1 CMYKColor

XML attributes of type CMYKColor are used to specify CMYK colors.

Encoding

CMYKColor attributes are primitive data types and are encoded as a string of four numbers in the range of [0...1.0] separated by spaces. A value of 0 specifies no ink and a value of 1 specifies full ink.

Example

```
<Color cmyk = "0.3 0.6 0.8 0.1"> (brick red)
```

A.2.2 IntegerList

XML attributes of type IntegerList are used to describe a variable length list of integer values.

Encoding

An IntegerList is encoded as a string of integers separated by spaces.

Example

```
<XXX list="0 1 2 3 4 1 3 0"/>
```

A.2.3 IntegerRange

XML attributes of type IntegerRange are used to describe a range of integers. In some cases, ranges are defined for an unknown number of objects. In these cases, a negative value denotes a number counted

from the end. For example, -1 is the last object, -2 the second to last, and so on. IntegerRanges that follow this convention are marked in the respective attribute descriptions.

If the first element of an IntegerRange specifies an element that is behind the second element, the Range specifies a list of integers in reverse order, counting backwards. For example “6~4” = ”6 5 4” and “-1~0” = “last... 2 1 0”.

Encoding

An IntegerRange is represented by two integers, separated by a “~” (tilde) character.

Example

```
<XXX range="-3~-5"/>
```

A.2.4 IntegerRangeList

XML attributes of type IntegerRangeList are used to describe a list of IntegerRanges and/or enumerated integers.

Encoding

A IntegerRangeList is represented by a sequence of IntegerRanges and integers, separated by spaces.

Example

```
<XXX list="-1~-6 3~5 7 9~128 131"/>
```

A.2.5 LabColor

XML attributes of type LabColor are used to specify absolute Lab colors.

The Lab values are normalized to a Light of D50 and an angle of 2 degrees as specified in CIE Publication 15.2 - 1986 "Colorimetry, Second Edition" and ISO 13655:1996 "Graphic technology - Spectral measurement and colorimetric computation for graphic arts images"

This corresponds to a white point of $X = 0.9642$, $Y = 1.0000$, and $Z = 0.8249$ in CIEXYZ color space. L is restricted to a range of [0..100]; a and b are unbounded.

Encoding

LabColors are primitive data types and are encoded as a string of three numbers separated by spaces: "L a b"

Example

```
<Color ... Lab="51.9 12.6 -18.9">
```

A.2.6 Matrix

Coordinate transformation matrices are widely used throughout the whole printing process, especially in layout resources. They represent 2D transformations as defined by the PostScript and PDF Reference manuals. For more information, refer to the respective Reference Manuals, and look for “Coordinate Systems and Transformations.”

Encoding

Coordinate transformation matrices are primitive data types and are encoded as a string attribute of six numbers, separated by spaces:

```
"a b c d Tx Ty"
```

Tx and Ty describe distances and are defined in points.

Example

```
<ContentObject CTM="1 0 0 1 3.14 21631.3" ... />
```

A.2.7 NamedColor

XML attributes of type NamedColor are not sufficient for process color definition, but rather serve to define the colors of preprocessed products such as wire-o binders and cover leaflets.

Table A.1 Mapping of named colors to sRGB colors

sRGBColor value	Color name
1 1 1	white
0.8 0.8 0.8	grey
0.5 0.5 0.5	grey
1 0 0	red
1 1 0	yellow
0 1 0	bright green
0 1 1	turquoise
0 0 1	blue
1 0 1	pink
0.5 0 0.5	violet
0 0 0.5	dark blue
0 0.5 0.5	teal
0 0.5 0	green
0.5 0.5 0	dark yellow
0.5 0 0	dark red

A.2.8 NameRange

XML attributes of type NameRange are used to describe a range of NMTOKEN data that are acquired from a list of named elements, such as named pages in a PDL file. It depends on the ordering of the targeted list, which names are assumed to be included in the NameRange. The following two possibilities exist:

1. There is no explicit ordering. In this case, alphabetical ordering is implied.
2. There is explicit ordering, such as in a list of named pages in a **RunList**. In this case, the ordering of the Runlist defines the order and all pages between the end pages are included in the NameRange.

Encoding

A NameRange attribute is represented by two NMTOKEN- $\{\sim\}$, separated by a “~” (tilde) character.

Example

```
<XXX NameRange="Jack~Jill"/>
```

A.2.9 NameRangeList

XML attributes of type NameRangeList are used to describe a list of NameRanges.

Encoding

A NameRangeList is represented by a sequence of NameRanges and NMTOKEN, separated by spaces.

Example

```
<XXX list="A b~f x z"/>
```

A.2.10 NumberList

XML attributes of type NumberList are used to describe a variable length list of numbers (double or integer).

Encoding

A NumberList is encoded as a string of space-separated numbers.

Example

```
<XXX list="3.14 1 .6"/>
```

A.2.11 NumberRange

XML attributes of type NumberRange are used to describe a range of numbers. Mathematical spoken, the two numbers define a closed interval.

Encoding

A NumberRange is represented by two numbers, separated by a “~” (tilde) character.

Example

```
<XXX range="-3.14~5.13"/>
```

A.2.12 NumberRangeList

XML attributes of type NumberRangeList are used to describe a list of NumberRanges and/or enumerated numbers.

Encoding

A NumberRangeList is a sequence of NumberRanges and numbers separated by spaces.

Example


```
<XXX list="-1~-6 3.14~5.13 7 9~128 131"/>
```

A.2.13 Path

XML attributes of type Path are used in JDF for describing parameters such as Trapzones and Clippaths. In PJTF, paths are encoded as a series of moveto-lineto operations. JDF has a different encoding, which is able to describe more complex paths, such as beziers.

Encoding

Paths are encoded as an XML string attribute formatted with PDF path operators. This allows for easy adoption in PS and PDF workflows.

PDF operators are limited to those described in section 8.6.1 “Path segment operators” in “Portable Document Format Reference Manual”, Version 1.3.

Example

```
<ElementWithPath path="0 0 m 10 10 l 20 20 l"/>
```

A.2.14 Rectangle

XML attributes of type Rectangle are used to describe rectangular locations on the page, sheet, or other printable surface. A rectangle is represented as an array of four numbers—llx lly urx ury—specifying the lower-left x, lower-left y, upper-right x, and upper-right y coordinates of the rectangle, in that order.

All numbers are defined in points.

Encoding

To maintain compatibility with PJTF, rectangles are primitive data types and are encoded as a string of four numbers, separated by spaces:

```
"llx lly urx ury"
```

Example

```
<ContentObject ClipBox="0 0 3.14 21631.3" ... >
```

Implementation Remark

Since all numbers are real numbers, any comparison of boxes should take into account certain rounding errors. For example, different boxes may be considered equal when all numbers are the same within a range of 1 point.

A.2.15 sRGBColor

XML attributes of type sRGBColors are used to specify sRGB colors.

Encoding

sRGBColors are primitive data types and are encoded as a string of three numbers in the range of [0...1.0] separated by spaces. A value of 0 specifies no intensity (black) and a value of 1 specifies full intensity:

```
:  
"r g b"
```

Example

```
<Color sRGB="0.3 0.6 0.8" ... >
```

A.2.16 TimeRange

XML attributes of type TimeRange are used to describe a range of time. More specifically, it describes a time span that has a specific start and end. Mathematically, two timeInstant values define a closed time interval.

Encoding

A TimeRange is represented by two timeInstants, separated by a “~” (tilde) character.

Example

```
<XXX range="1999-05-31T13:20:00-05:00~2000-08-03T23:50:00-05:00"/>
```

A.2.17 TransferFunctions

XML attributes of type TransferFunctions are functions that have a one-dimensional input and output. In JDF, they are encoded as a simple kind of sampled functions and used to describe transfer curves of processes such as Film-to-Plate-copy, LaserCalibration and Press Calibration. They may also be used in Color specifications, for example when converting a spot tint value to a CMYK value.

A transfer curve consists of a series of XY pairs where each pair consist of the stimuli(X) and the resulting value(Y). To calculate the result of a certain stimuli, the following algorithms must be applied:

1. If $x \leq$ first stimuli, then the result is the y value of the first xy pair.
2. If $x \geq$ the last stimuli, then the result is the y value of the last xy pair.
3. Search the interval in which x is located.
4. Return the linear interpolated value of x within that interval.

Encoding

A TransferCurve is encoded as a string of space-separated numbers. The numbers are the XY pairs that build up the transfer curve.

Example

```
<someElementWithTransferCurve someCurve="0 0 .1 .2 .5 .6 .8 .9 1 1"/>
```

A.2.18 XYPair

XML attributes of typ XYPair are used to describe sizes like *MediaSize* and *PageSize*. They can also be used to describe positions on a page.

All numbers that describe lengths are defined in points.

Encoding

XYPair attributes are primitive data types and are encoded as a string of two numbers, separated by spaces: “x y”

Example

```
<CutBlock BlockSize="612 792">
```

Implementation Remark

Since all numbers are real numbers, comparison of XYpairs should take into account certain rounding errors. For example, different XYpairs may be considered equal when all numbers are the same within a range of 1 point.

A.3 JDF Data Structures

The following data structures are unique to JDF, although they may be comprised of existing XML structures.

A.3.1 Links

Links are defined by a combination of XML attributes of type *ID* and XML attributes of type *IDREF*. The referenced element or target of the link contains the actual information and an *ID* attribute, whereas the reference or link itself contains an *IDREF* attribute. The value of an *ID* attribute must be unique within an XML file. In order to keep the implementation burden on JDF compliant processors low, linking between distributed JDF files is not supported. The *ID* attribute of the target is always named *ID*. This is not required by XML, but it makes implementation simpler. The *IDREF* attribute in a link, however, can have varying names depending on the link type. The names of the *IDREF* attributes are defined in this document.

The following example specifies a trivial link and target pair¹:

```
<Target ID="id1" (lots of attributes)><Subelement/></Target>
...
<Link rRef="id1"/>
```

A.4 JDF File Formats

This section describes the specific file formats used by JDF. JDF uses MIME files to package different files in a single file for transmission, and when representing preview images, JDF uses the PNG image file format. The following sections explain in what ways MIME and PNG are used in JDF.

A.4.1 MIME File Packaging

JDF files are XML files but may contain references (URLs) to external data files. The following external data file types are identified, although any valid MIME file type may be referenced:

- Preview images.
They are encoded using the PNG format.
- ICC Profiles.
- Preflight Profiles.
- PDL files (PageDescription files).
- ...

One of the requirements for JDF is to support the ability to make a single, selfcontained job package that contains the JDF with all of its related files, maintaining the external data references. That package will be sent to a remote location where it is used for further processing. This section describes how JDF uses MIME to achieve this requirement.

¹ Note that the element names were chosen for simplicity and do not imply any naming conventions for targets and links.

MIME (Multipurpose Internet Mail Extensions) is an internet standard that defines mechanisms for specifying and describing the format of internet message bodies. One of its applications is the MIME Multipart/related type and is used by JDF.

The MIME Multipart/Related Content-type specification can be found at <http://rfc.roxen.com/rfc/rfc2112.html> “The MIME Multipart/Related Content-type”

A.4.1.1 MIME Basics

MIME is comprised of headers and bodies. In case of Multipart messages, the body consists of multiple messages, each identified by the individual MIME header and separated by a unique boundary string. Normally a MIME-user agent uses the boundary string to separate different message parts, and JDF MIME files are compliant with that mechanism. Furthermore, JDF defines a Content-Length mechanism that enables fast scanning of MIME files for their body parts.

A.4.1.2 MIME Fields

Content Type

This field is always required.

Content Type identifies the MIME type of the message (part). The Multipart header uses this to identify itself as a multipart message and the sub parts also have MIME types to identify their content.

Content ID

This field is required for every part that is referenced by other parts.

Content ID identifies each different part within a multipart MIME message. Its value can be anything as long as it is defined using USASCII. It is good practice to limit yourself to using only alphanumeric characters or only the first 127 characters of the USASCII character set in order to avoid confusing less intelligent MIME agents.

Content Transfer Encoding

This field is optional, and its default = *none*.

MIME prescribes three different encodings: None, Base64 and QuotedPrintable. When no encoding is used, the data are only encapsulated by MIME headers. Base64 and QuotedPrintable encodings are commonly used algorithms for converting 8-bit and binary data into 7-bit data and vice versa. Although these encodings are not imposed, JDF agents that support MIME must be able to handle them.

A.4.1.3 CID URL scheme

One of the benefits of the MIME multipart/related mediatype is the ability to refer from one bodypart to another bodypart. This is done by using the cid: URL addressing scheme, specified in <http://rfc.roxen.com/rfc/rfc2111.html> “Content-ID and Message-ID Uniform Resource Locators”. Please look at the example to see how it is used.

Example

```
MIME-Version: 1.0
Content-Type: multipart/related; boundary=abcdefg0123456789

--abcdefg0123456789
Content-Type: text/xml

<JDF ... >
<PreviewImage separation = "Pantone 128" URL="cid:123456.png" />
</JDF>

--abcdefg0123456789
Content-Type: image/png
Content-Transfer-Encoding: base64
Content-ID: 123456.png
Content-Length: 12345

BASE64DATA
BASE64DATA

--abcdefg0123456789--
```

A.4.1.4 JDF Agent Requirements

All JDF agents must be prepared to receive JDF files that are MIME encoded. They may choose not to support it, but they should be able to handle these JDF files gracefully. Agents that do support MIME must support base64 and QuotedPrintable encodings.

A.4.2 HTTP 1.0 Field

Content Length

Although this field is optional, it is recommended that it be included.

Content Length is used to optimize the performance of scanning multipart messages. Each multipart bodypart may have an optional Content-Length header field. Its syntax is identical to the syntax defined by RFC1945 "HTTP1.0".

When present, the Content Length identifies the number of octets of the encoded bodypart. When no encoding as is the case with 7bit, 8bit, binary, it represents the size of the bodypart. Otherwise it depends on what encoding method is used encoding (base64, QuotedPrintable) and what the relationship is between the encoded size and the bodypart size. If an agent composing a MIME message can not derive a Content Length for its encoded bodyparts, it must omit the Content-Length field.

An agent parsing such a message can use the Content-Length field to seek to the end of the body. This position is calculated by using the position of the first byte of the bodypart and adding the Content Length. At that position (one byte after the bodypart contents), the agent must check if the following characters are one of either "\r\n—boundary" or "—boundary." If not, the agent must ignore the Content-Length field and resume the normal MIME Multipart behaviour and restart scanning for the boundary from the beginning of the bodypart.

A.4.3 PNG Image Format

JDF uses the PNG images for representing preview images. CIP3 defined two formats: composite CMYK and separated. With PNG, only the separated format is supported, the composite CMYK must be

represented as separated CMYK. Thus, preview images are stored as separate PNG images and JDF links them together.

Appendix B Schema

This appendix lists the schemas or URLs of those which are required for the creation and parsing of JDF-documents.

B.1 Schema of the JDF-node

The following shows the schema of a JDF node.
TBD...

Appendix C Converting PJTF to JDF

This appendix is provided as a guide to developers writing applications that will consume PJTF version 1.1 jobs and produce JDF.

C.1 PJTF Object Conversion

Many PJTF objects are directly translatable to JDF processes or resources. Others, especially those containing multiple keys, correspond to multiple processes and resources. For example, the **JobTicketContents** object corresponds to four JDF processes and three JDF resources. And still others, such as **AuditObject**, cannot be translated to JDF at all.

Listed below are the prominent PJTF objects and the JDF components to which they correspond. Each section heading contains the title of the object in question, and each section contains a descriptive table. The first column in the tables, entitled JDFKey or Object, contains a list of the keys or objects contained within the object being described. For example, the **Accounting** object contains an **Address** object, while the **Address** object contains an **Address** key. If no sub-object or key is contained within the object, then the first column is left blank and the process or resource listed is assumed to correspond directly to that object.

C.1.1 Accounting

PJTF Key or Object	JDF Process	JDF Resource	Description
Address	-	Address	-

C.1.2 Address

PJTF Key or Object	JDF Process	JDF Resource	Description
Address	-	Address	Used whenever people or organizations need to be identified.

C.1.3 Analysis

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	Analysis	-

C.1.4 AuditObject

Audit objects may not be translated. PJTF Audit objects describe the results of operations on files, while JDF Audit elements describe the results of processes, so there is a basic incompatibility between the two. In addition, PJTF Audit objects will not be needed to direct further processing of the job after it is converted to JDF.

C.1.5 ColorantAlias

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	-	Maps to a sub-element of the ColorantControl resource.

C.1.6 ColorantControl

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	ColorantControl	-

C.1.7 ColorantDetails

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	-	Keys in the PJTF ColorantDetails dictionary are a set of colorant names. The values are DeviceColorant objects.

C.1.8 ColorantZoneDetails

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	TrappingParams	DeviceColorant map to the ColorantZoneDetails sub-element of the TrappingParams resource.

C.1.9 ColorSpaceSubstitute

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	-	Maps to a sub-element of the ColorantControl resource.

C.1.10 Delivery

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	Delivery	Address	Specifies a quantity of a product to be delivered to an address.

C.1.11 DeviceColorant

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	-	Maps to a Color sub-element of the ColorPool resource. The name is entered in the SeparationSpec of a TrappingDetails resource.

C.1.12 Document

JobTicketContents, **Document** and **PageRange** objects are decomposed into a number of different JDF objects. Most of the key/value pairs translate into various resources.

PJTF Key or Object	JDF Process	JDF Resource	Description
bleed media trim	-	RunList	Maps to attributes of the RunList resource or to processes in which they are

			used.
ColorantControl	-	ColorantControl	-
Files	-	RunList FileSpec	Maps to FileSpec resources contained within Run elements.
Finishing	AdhesiveBinding EndSheetGluing SaddleStitching SideSewing Stitching ThreadSewing	AdhesiveBinding-Params EndSheetGluing-Params SaddleStitching-Params SideSewingParams StitchingParams ThreadSewingParams	-
FontPolicy	-	FontPolicy	The resource is attached to the applicable processes.
IgnoreHalftone	-	-	Maps to the <i>IgnoreHalftone</i> attribute of the PDFToPS-ConversionParams resource.
InsertPage	Imposition	RunList Sheet	Occurs as an attribute either of RunList resources or of Sheet resources referenced by Imposition processes.
InsertPage	Imposition	RunList Sheet	Appears as an attribute of the listed resources and is referenced by the listed process.
NewSheet	Imposition	InsertSheet	See Bill (RunList, Run)
Media	-	Media	Maps to a sub-element of the ExposedMedia resource.
MediaSource	-	-	Maps to a sub-element of the DigitalPrinting-Params resource.
MediaUsage	Dividing	DividingParams	Specifies controls for roll-fed media.
Rendering	Rendering	-	-
Trailer	Imposition	InsertSheet	See Bill (RunList, Run)
Trapping	Trapping	-	-

C.1.13 Finishing

Finishing operations are derived from CIP3 PPF. Conversion of PJTF **Finishing** objects is vendor-dependent, since the PJTF specification does not describe any detail for **Finishing** objects.

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	AdhesiveBinding EndSheetGluing SaddleStitching SideSewing Stitching ThreadSewing	AdhesiveBinding-Params EndSheetGluing-Params SaddleStitching-Params SideSewing-Params StitchingParams ThreadSewing-Params	-

C.1.14FontPolicy

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	Interpreting	Interpreting-Params	See Bill

C.1.15InsertPage

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	-	See Bill

C.1.16InsertSheet

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	InsertSheet	-

C.1.17Inventory

PJTF Key or Object	JDF Process	JDF Resource	Description
--------------------	-------------	--------------	-------------

C.1.18JobTicket

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys except Audit , Scheduling , PreflightResults	Any process	Any resource	Keys may be represented at various levels of the JDF tree. Contents are represented as processes, resources, and versions.

C.1.19JobTicketContents

JobTicketContents, **Document** and **PageRange** objects are decomposed into a number of different JDF objects. Most of the key/value pairs translate into various resources.

PJTF Key or Object	JDF Process	JDF Resource	Description
Accounting	-	-	Maps to the CustomerInfo element.
Administrator	-	-	Maps to the CustomerInfo element.
ColorantControl	-	ColorantControl	-
Delivery	Delivery	DeliveryParams	-
Documents	-	RunList	May require more than one RunList resource.
EndMessage	-	-	Maps to the <i>End</i> attribute of the NodeInfo element.
Finishing	AdhesiveBinding EndSheetGluing SaddleStitching SideSewing Stitching ThreadSewing	AdhesiveBinding-Params EndSheetGluing-Params SaddleStitching-Params SideSewing-Params StitchingParams ThreadSewing-Params	-
FontPolicy	Interpreting PDFToPS-Conversion	FontPolicy	The FontPolicy resource is attached to any process that uses it.
IgnoreHalftone	-	-	Maps to the <i>IgnoreHalftone</i> attribute of the PDFToPS-ConversionParams resource.
InsertPage	Imposition	RunList Sheet	Occurs as an attribute either of RunList resources or of Sheet resources referenced by Imposition processes.
JobName	-	-	CustomerJobName in the CustomerInfo element of the JobInfo node.
Layout	Imposition	Layout	-
MarkDocuments	Imposition	RunList	Requires one of two RunList resources, each of which is a resource of the Imposition process.
MediaSource	-	-	Maps to a sub-element of the DigitalPrintingParams resource.
MediaUsage	Dividing	DividingParams	Specifies controls for roll-fed media.

NewSheet	<i>Imposition</i>	InsertSheet	media. See Bill (RunList, Run)
PrintLayout	<i>Imposition</i>	-	Maps to a sub-element of the Layout resource.
Rendering	<i>Rendering</i>	-	Maps to the attribute of the Rendering process.
Scheduling	-	-	The Scheduling object is not translated.
StartMessage	-	-	Maps to the <i>Start</i> attribute of the NodeInfo element.
Submitter	-	-	Maps to the CustomerInfo element.
Trailer	<i>Imposition</i>	InsertSheet	See Bill (RunList, Run)
Trapping	<i>Trapping</i>	-	-

C.1.20JTFile

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	-	See Bill

C.1.21Layout

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	<i>Imposition</i>	Layout	-

C.1.22Media

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	Media	Maps to a sub-element of the ExposedMedia resource.

C.1.23MediaSource

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	-	Maps to a sub-element of the DigitalPrintingParams resource.

C.1.24MediaUsage

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	<i>Dividing</i>	DividingParams	Specifies controls for roll-fed media.

C.1.25PageRange

JobTicketContents, **Document** and **PageRange** objects are decomposed into a number of different JDF objects. Most of the key/value pairs translate into various resources.

PJTF Key or Object	JDF Process	JDF Resource	Description
bleed media trim	-	RunList	Maps to attributes of the RunList resource or to processes in which they are used.
ColorantControl	-	ColorantControl	-
Delivery	<i>Delivery</i>	DeliveryParams	-
Files	-	RunList FileSpec	Maps to FileSpec resources contained within Run elements.
Finishing	<i>AdhesiveBinding</i> <i>EndSheetGluing</i> <i>SaddleStitching</i> <i>SideSewing</i> <i>Stitching</i> <i>ThreadSewing</i>	AdhesiveBinding-Params EndSheetGluing-Params SaddleStitching-Params SideSewing-Params StitchingParams ThreadSewing-Params	-
FontPolicy	<i>Interpreting</i> <i>PDFToPS-Conversion</i>	FontPolicy	The FontPolicy resource is attached to any process that uses it.
IgnoreHalftone	-	-	Maps to the <i>IgnoreHalftone</i> attribute of the PDFToPS-ConversionParams resource.
InsertPage	<i>Imposition</i>	RunList Sheet	Occurs as an attribute either of RunList resources or of Sheet resources referenced by <i>Imposition</i> processes.
Media	-	Media	Maps to a sub-element of the ExposedMedia resource.
MediaSource	-	--	Maps to a sub-element of the DigitalPrintingParams resource.
MediaUsage	<i>Dividing</i>	DividingParams	Specifies controls for roll-fed media.
NewSheet	<i>Imposition</i>	InsertSheet	See Bill (RunList, Run)
Rendering	<i>Rendering</i>	-	-
Trailer	<i>Imposition</i>	InsertSheet	See Bill (RunList, Run)
Trailer			
Trapping	<i>Trapping</i>	-	-

Which - **RunList** See Bill

C.1.26PlacedObject

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	-	Maps to a sub-element of the Surface resource.

C.1.27PlaneOrder

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	RunList	See section ## (Translating the Contents Hierarchy) for more details.

C.1.28Preflight

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	Preflight	-	-

C.1.29PreflightConstraint

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	-	Maps to a sub-element of the PreflightProfile resource.

C.1.30PreflightDetail

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	-	Maps to a sub-element of the PreflightAnalysis resource.

C.1.31PreflightInstance

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	-	Sub-element of the PreflightAnalysis resource

C.1.32PreflightInstanceDetail

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	-	Sub-element of the PreflightAnalysis resource

C.1.33PreflightResults

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	-	This object is not translated.

C.1.34 PrintLayout

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	Imposition	-	Maps to a sub-element of the Layout resource.


C.1.35 Profile

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys			See Bill


C.1.36 Rendering

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	Rendering	RenderingParams	-

C.1.37 ResourceAlias

PJTF Key or Object	JDF Process	JDF Resource	Description
Location			Location is Device
File			File is supported via the SourceFile fileref.
This			This is supported via the SourceFile fileref.
ResourceName			This key is not used. References to the aliased resource run via the ResourceLink element.
SourceFile		 [5]	Source file maps to an attribute of this resource.

PJTF **ResourceAlias** objects provide a unified namespace that allows each PJTF object to refer to the resources it needs to execute the job of which it is a part. More specifically, PJTF version 1.1 supports the use of **ResourceAlias** objects to allow references to halftones and colorspaces.

 the **ResourceAlias::Location** key, the **File** and **This** keys are supported by a **SourceFile** attribute whose value is a fileref.[DH46] The translator must provide a reference to the original PJTF file (for this) or a copy that contains the referenced resources.

C.1.38 Scheduling

Scheduling objects are not translated. It is presumed that translation of PJTF jobs into JDF is performed to allow the reuse of PJTF jobs that have been archived. Thus, the original scheduling information embedded in the PJTF is irrelevant.

C.1.39Signature

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	-	Maps to a sub-element of the Layout resource.

C.1.40Sheet

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	Sheet	-

C.1.41SlipSheet

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	-	See Bill

C.1.42Surface

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	Surface	-

C.1.43Tile

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	Tiling	Tile	-

C.1.44Trapping

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	Trapping	TrappingParams	-

C.1.45TrappingDetails

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	TrappingDetails	See the PJTF DeviceColorant object entry for details on how it is translated.

C.1.46TrappingParameters

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	-	TrappingParams	-

C.1.47TrapRegion

PJTF Key or Object	JDF Process	JDF Resource	Description

All keys	-	TrapRegion	-
----------	---	-------------------	---

C.2 Translating Values

The PJTF version 1.1 specification lists twelve data types that may occur for the values of keys in PJTF objects. The following table describes how each of these datatypes shall be represented in JDF.

PJTF Data Type	JDF Representation	Comment
Boolean	boolean	-
Number	number	-
Name	name	-
Dictionary	element	All PJTF objects are dictionaries. These dictionaries generally become resources or processes as specified above. In addition, some PJTF objects contain embedded dictionaries whose keys are not specified (examples include TrappingParameters and ColorantDetails). These dictionaries are converted to arrays of elements, with the key name from the PJTF dictionary becoming an attribute of the sub-element.
Stream	URI	PJTF supports PDF streams by reference to an object in a PDF file. The same mechanism is supported in JDF, with the JDF URI data type being used to identify the PDF file.
Rectangle	rectangle	-
Filespec	URI	-
Text	string	-
String	string	-
Date	date	-
Phone number	Phone number	The standard for the representation of phone numbers in PJTF is used here as well.

C.3 Translating the Contents Hierarchy

The contents of a PJTF job are represented in the ‘contents hierarchy’. The hierarchy is headed by the JobTicketContents object, with Document, PageRange and JTFile objects occurring below. The hierarchy implicitly specifies the sequence of source pages for the job.

The contents sequence comprises all the pages specified by the first, then second, then ... last PageRange for the first Document, followed by the pages specified by the first, then ... last PageRange for the second Document, followed by ... the pages for the first, then ... last PageRange for the last Document. This sequence of source pages is consumed when the job is printed via PrintLayout (discussed below).

The contents hierarchy must be translated into a JDF **RunList** resource, which contains multiple Run sub-elements. Each Run can reference a file via the *URL* attribute and a set of pages in the file via the *Pages* element.

There are several additional issues related to this translation which are discussed below.

C.4 Representing Pages

In PJTF, source pages are represented as a hierarchy of Document and PageRange objects. Pages are referenced by page number out of files; files are represented in JTFile objects. PageRange objects can reference a single page, or a set of contiguous pages.

In JDF, source pages are represented as a set of Run sub-elements, of the **RunList**, which reference files via URL, and pages from the files via an IntegerRangeList (such as '1,3~5,7~-1').

As a consequence of this difference, pages from more than one PJTF PageRange object can be represented in a single **RunList** resource, assuming that all the other keys for the multiple PageRanges have the same values.

C.5 Representing Pre-separated Documents

In pre-separated workflows, all plane of each page may occur in the same file, or there may be a separate file for each plane. When all the planes occur in a single file, PJTF JTFile objects use a PlaneOrder object to specify which pages in the file represent each colorant plane for each source page. When each plane occurs in a separate file, the JTFile objects use a FilesDictionary to associate files with each colorant.

In JDF, both of these cases are handled through the **RunList** resource. In the case where the planes occur in separate files, the Runs are partitioned; and each partition contains the name of the colorant and the URL for the file for that colorant. In the case where the colorant planes are intermingled via PlaneOrder objects, the Runs are partitioned, but only a single URL is used for each Run. Each PlaneOrder object will become one Run.

C.6 Representing Inherited Characteristics

In PJTF, many of the characteristics of source pages—including MediaBox, ColorantControl, and InsertPage—may occur at all levels of the contents hierarchy.

This inheritance scheme is not provided in JDF. Therefore, the correct values for each of the attributes must be translated to the appropriate element for each **RunList** element.

C.7 Translating Layout

PJTF provides two mechanisms for image a set of source pages onto a larger surface for printing: Layout and PrintLayout. Layout is a mechanism for explicitly associating specific source pages with specific locations on the surface. PrintLayout is a method for automatically positioning a sequence of source pages onto a series of surfaces.

Layout is represented as a hierarchy of PJTF objects: Signatures, Sheets, Surfaces and PlaceObjects. The Layout hierarchy may have one or more Signature objects. Each Signature must have one or more Sheets. Each Sheet must have 1 or 2 Surfaces. Each Surface may have 0 or more PlacedObjects.

PlacedObjects directly reference source pages by referring to a Document object via its Doc key, and a specific page within the sequence of pages specified by all the PageRanges in Pages arrays for that Document.

JDF defines resources which are direct translations of Signature, Sheet and Surface. PlacedObjects and MarkObjects are sub-elements of the Surface resource.

Note that PlacedObjects identify specific source pages via a combination of Ord and either Doc or MarkDoc. Ord identifies one page out of the sequence of pages specified by all the PageRange objects for the document identified by either Doc or MarkDoc.

In the JDF PlacedObject sub-element, the Ord attribute is an index into the entire sequence of pages specified by all the Runs in the **RunList**. So there is a translation required between the PJTF Ord value and the JDF Ord attribute.

Similarly, in the JDF MarkObject sub-element, the Ord attribute is an index into the entire sequence of pages specified by all the Runs in the **RunList** for marks. So there is a translation required between the PJTF Ord value and the JDF Ord attribute.

C.8 Translating PrintLayout

PrintLayout uses the same hierarchy of objects as Layout, but with the restriction that there can be only a single Signature. The Signature is used as a template that is repeated to consume all the source pages specified by the contents hierarchy for the job.

In addition, the PlacedObjects that occur in a PrintLayout hierarchy are not references to specific source pages. Instead, they represent the intent that a page from the sequence of source pages specified by the contents hierarchy be consumed and placed onto the Surface each time the Signature is executed.

In JDF, PrintLayout is represented via the same set of resources as Layout, except that the top of the hierarchy is an AutomatedLayout resource instead of Layout. This resource is constrained to have only one Signature resource.

Note that when translating PJTF PlacedObjects to PlacedObject sub-elements of a Surface resource in the AutomatedLayout hierarchy, the Ord values from the PJTF PlacedObjects need not be modified.

However, as in the creation of Layout, the Ord attribute for JDF MarkObject sub-elements are indices into the entire sequence of pages specified by all the Runs in the **RunList** for marks. So there is a translation required between the PJTF Ord value and the JDF Ord attribute.

C.9 Translating Trapping

Trapping controls are represented in PJTF as several objects: Trapping, TrappingDetails, ColorantDetails and DeviceColorants; TrappingParameters and ColorantZoneDetails; and TrapRegions. These objects can occur in multiple places in the PJTF job, and they work together to determine, for each page in the job, whether it will be trapped and how. There is also a key in the JobTicketContents object, TrappingSourceSelector, which determines which set of trapping controls will be honored.

The trapping controls in PJTF are the same, whether the trapping will be done pre-rip or in-rip.

In translating PJTF trapping controls to JDF, there are several tasks to perform:

- Create the required Trapping node
- Add the resources to represent the TrappingParameters which will be used
- Create the resources which represent the TrapRegions which will be used
- Determine the pages to be trapped
- Determine which controls to use for each page
- Add references to the pages in the **RunList** in the TrapRegion resource

Note that the contents hierarchy for the PJTF job must be translated into **RunLists** before trapping objects can be translated.

Note as well, that paths in JDF are specified as a set of path operators. PJTF TrapZone paths are a sequence of coordinates with an implied moveto at the beginning, and an implied closepath the end.

Appendix D Converting PPF to JDF

This appendix gives advice on how to convert CIP3 PPF 3.0 files to JDF encoded files. Since JDF was designed with the intention of providing the highest possible level of compatibility with PPF, many of these conversions are relatively straightforward. From the point of view of JDF, CIP3's PPF is mainly resource based. Most of the PPF structures were therefore translated to JDF resources of a corresponding process. Meanwhile, the PPF product definition operations are easily translated to JDF processes of the same name, as quoted in **CIP3ProductOperation**. This kind of conversion is possible because the component structure of PPF is adopted by JDF, with some enhancements. Parameters of PPF product definition operations (**CIP3ProductParams**) are given the abbreviated name "Params," and this name is appended to the **CIP3ProductOperation** name. Thus SideSewing becomes SideSewingParams.

In many cases, PPF key names became JDF attribute or element names with the "CIP3" prefix removed. An example of this kind of translation is provided below, and the CIP3 product structure shown in the example is expressed as a JDF process in Figure D.1, following the example.

Example: A CIP3 PPF product definition operation

```

/CIP3Products [
<<
  /CIP3ProductName (sewed book block)
  /CIP3ProductOperation /ThreadSewing
  /CIP3ProductParams <<
    /NumberOfNeedles 4
    /GlueLineRefSheets [ 0 ]
    /GlueLine <<
      ...
    >>
    /BlindStitch false
    /Sealing false
  >>
  /CIP3ProductComponents
  [
    <<
      /SourceType /PartialProduct
      /SourceProduct (book block)
      ...
    >>
  ]
>>

<<
  /CIP3ProductName (book block)
  % ... the definition of the book block operation would go here
  ...
>>
] def

```



Figure D.1 JDF node of a CIP3 product structure

In Figure D.1, the input **Component** represents the “book block,” the output **Component** represents the “sewed book block,” and **ThreadSewingParams** covers all information of the **CIP3ProductParams** structure.

Whenever possible, the formal conversion and translation conventions described above were followed, but because extensions and operations new to PPF are included in JDF, some exceptions were made. These exceptions are explained in detail for each PPF structure in the sections that follow. Before they are explained, however, a translation of PPF data types is provided.

D.1 Converting PPF Data Types

The following table shows all PPF data types, and how they are transformed. All measuring units of CIP3 must be converted to the JDF native unit point (1/72 inch). Comments are only provided when there is something unusual or noteworthy about the translation; thus, not all translations require comment.

Table D.2 Conversion of PPF Data Types

PPF Data Type	JDF Data Type	Comments
Boolean	boolean	-
Integer	integer	-
Real	double	The exponent symbol must be a capital ‘E’ in XML.
Number	double	The exponent symbol must be a capital ‘E’ in XML.
Name	enumeration <i>or</i> NMTOKEN	When PPF Names are used as a closed set of predefined values, they are converted to an enumeration. Otherwise, they are converted to an NMTOKEN.
String	string	Some PostScript string characters cannot be used in XML.
Array	Sequence of elements <i>or</i> IntegerList <i>or</i> DoubleList	If the array consists of homogeneous integers or doubles, it is converted to an IntegerList or DoubleList, otherwise to a sequence of corresponding elements.
Dictionary	element	In most cases, the structure of a Dictionary is directly converted to a XML element. Exceptions to this rule are described in the following sections.

D.2 PPF Product Definitions

The information stored in **CIP3Products** and **CIP3FinalProducts** is implicitly expressed by the structure of the JDF tree. Each product definition step is converted to a JDF node, and a product node is created for every final product of a PPF file. This is also the case for each partial product that is used in two or more final products. The following table provides information that explains how to accomplish these transformations and make these conversions. The content of the entities **CIP3ProductJobName**, **CIP3ProductJobCode**, **CIP3ProductCopyright** and **CIP3ProductCustomer** shall also be copied to the parent product node.

The sections that follow contain information about the conversion requirements of prominent postpress processes.

Table D.8.3 JDF Representation of a product definition step

PPF Key	JDF Representation	Comments
CIP3ProductName	This is expressed by an output resource link.	-
CIP3ProductOperation	JDF node	See section 3.1 JDF nodes.
CIP3ProductParams	Resource identified by the name of the JDF node + “Params”	For example, during a CIP3ProductOperation of the type “ SaddleStitching ”, the JDF representation of the CIP3ProductParams is SaddleStitchingParams
CIP3ProductComponent	Component	See section D.2.1, below
CIP3ProductJobName	Comment element of the JDF node	-
CIP3ProductJobCode	<i>JobID</i> or <i>JobPartID</i> attribute of the JDF node	If the output of this step is a final product and it is only final product, it should be converted into <i>JobID</i> of the root node. Otherwise, it is converted into a <i>JobPartID</i> of the corresponding process node.
CIP3ProductCopyright	Comment element of the JDF node	-
CIP3ProductCustomer	CustomerInfo element of the JDF node	Note that the CustomerInfo element is structured, while the CIP3ProductCustomer is not.
CIP3ProductVolume	<i>Amount</i> attribute of the output Component resource link	-

D.2.1 Comparison of the PPF Component to the JDF Component

The structure of the PPF **Component** is very similar to the structure of the JDF **Component**, so it is easy to convert one to the other. The following table gives advice on how to do this. Some information stored in the PPF **Component** must be used for linking the correct resources to a process. Other implicit information, such as the bounding box of the component or an overfold, must be calculated and explicitly specified in the sub-elements of the **Component**. Furthermore, the appropriate algorithms can be very complex for some operations, such as folding.

For further information about the **Component** resource, see section 7.2.19 Component.

Table D.8.4 Converting a PPF Component

PPF Key	JDF Representation	Comments
SourceType	<i>ComponentType</i> attribute of Component	-
SourceSheet	<i>SourceSheet</i> attribute of Component	-

-	SheetPart attribute of Component	Calculable out of the cut block structure.
SourceBlock	Expressed by an input resource link to an output Component of a previous Cutting process.	see section D.3.6 Cutting Data
SourceProduct	Expressed by an input resource link to a Component .	-
Params	Transformation attribute of Component	In most CIP3 operations, there is only one component parameter called Orientation . This matrix is renamed to Transformation . The only exception is the EndSheetGluing process. See 6.5.3.1 EndSheetGluing for more information.

D.2.2 Collecting

To convert a **Collection** operation, follow the previous descriptions. This process contains no special considerations to take into account.

D.2.3 Gathering

To convert a **Gathering** operation, follow the previous descriptions. This process contains no special considerations to take into account.

D.2.4 ThreadSewing

Convert the entries of **CIP3ProductParams** structure directly to the **ThreadSewingParams** resource. Add this resource as an input resource link to the originated **ThreadSewing** process. See section 0

Template	string	Template to define a sequence of variables consumed by Format . A list of pre-defined values is found in the description of the FileSpec resource. In addition, DynamicInput elements of a RunList define further variables.
-----------------	--------	--

ThreadSewingParams for more information.

D.2.5 SaddleStitching

Convert the entries of **CIP3ProductParams** structure directly to the **SaddleStitchingParams** resource. Add this resource as an input resource link to the originated **SaddleStitching** process. See section 6.5.4.2.2 SaddleStitching for more information.

D.2.6 Sticking

Convert the entries of **CIP3ProductParams** structure directly to the **StickingParams** resource. Add this resource as an input resource link to the originated **Sticking** process. See section 7.2.87 StickingParams for more information.

D.2.7 SideSewing

Convert the entries of **CIP3ProductParams** structure directly to the **SideSewingParams** resource. Add this resource as an input resource link to the originated **SideSewing** process. See section 6.5.4.2.3 **SideSewing** for more information.

D.2.8 EndSheetGluing

The **EndSheetGluing** CIP3 operation is the only operation that requires more information than **Orientation** in the PPF Component **Params**. This additional information of the front and the back end sheet components is transferred to the **EndSheetGluingParams** resource, as described in the following table. See section 7.2.36 for more information.

Table D.8.5 Converting the PPF EndSheetGluing operation to JDF

PPF Key	JDF Representation	Comments
Offset	<i>Offset</i> attribute of the EndSheet element of EndSheetGluing-Params	-
GlueLine	GlueLine element of the EndSheet element of EndSheetGluing-Params	See section 7.2.36 for information on how to convert the GlueLine structure.

D.2.9 AdhesiveBinding

The PPF main adhesive binding operation dictionary is translated to the **AdhesiveBindingParams** resource. All single sub-operations that were resident in the PPF **Processes** array are converted to special elements inside the **AdhesiveBindingParams** (see section 7.2.3 **AdhesiveBindingParams**). For each type of adhesive binding sub-operation there exists one extra element. The sub-operations **BackPreparation** and **GlueApplication** can simply be translated by removing the **ProcessType** entry and converting all other entries directly to the appropriate element.

The following tables show how to convert the main operation and its other sub-operations. Because new features were added, the CIP3 **Lining** operation was renamed to **SpineTaping**.

Table D.8.6 Converting the PPF AdhesiveBinding operation to JDF

PPF Key	JDF Representation	Comments
Processes	Several elements inside the AdhesiveBinding-Params	See description above.
PullOutValue	<i>PullOutValue</i> attribute of AdhesiveBinding-Params	-
PullOutMake	-	Not needed.
FlexValue	<i>FlexValue</i> attribute of AdhesiveBinding-Params	-
FlexMake	-	Not needed.

Table D.8.7 Converting the PPF AdhesiveBinding sub-operation Lining

PPF Key	JDF Representation	Comments
ProcessType	-	There is an extra element for each

		type of AdhesiveBinding sub-operation
TopLiningExcess	<i>TopExcess</i> attribute of SpineTaping	-
LiningExcess	<i>HorizontalExcess</i> attribute of SpineTaping	-
LiningLength	<i>StripLength</i> attribute of SpineTaping	-
LiningMaterial	<i>StripMaterial</i> attribute of SpineTaping	-
LiningBrand	<i>StripBrand</i> attribute of SpineTaping	-

Table D.8.8 Converting the PPF AdhesiveBinding sub-operation CoverApplication

PPF Key	JDF Representation	Comments
ProcessType	-	There is an extra element for each type of AdhesiveBinding sub-operation.
CoverOffset	<i>CoverOffset</i> attribute of CoverApplication	-
ScoringOffsets and ScoringSide	Several Score elements inside of CoverApplication	The Score element is much more structured than these two single entries.

D.2.10 Trimming

Convert the entries of **CIP3ProductParams** structure directly to the **TrimmingParams** resource. Add this resource as an input resource link to the originated **Trimming** process. See section 6.5.7 Trimming for more information.

D.2.11 GluingIn

Because extended features have been added, the PPF **GluingIn** operation was renamed to the **Inserting** process. Consequently, the parameters of this CIP3 operation are transformed into the **InsertingParams** resource. For more information see section 7.2.53 InsertingParams.

Table D.8.9 Converting the PPF GluingIn operation to JDF

PPF Key	JDF Representation	Comments
SheetOffset	<i>SheetOffset</i> attribute of InsertingParams	-
-	<i>Location</i> attribute of InsertingParams	Shall be <i>Front</i>
GlueLines	Several GlueLine elements in InsertingParams	See section 7.2.53 InsertingParams for information on how to convert the GlueLine structure.
Sample	Comment of the corresponding Component	Converted to an input Component of <i>Type PartialProduct</i>

Most of the entries of the PPF **GlueLine** structure can be directly mapped to the **GlueLine** element. Note that the *GluingPattern* attribute cannot have an empty array to describe a solid glue line. For this purpose, use an array of “1 0”.

D.2.12 Folding

Like all formats, JDF follows a structured approach in the description of the folding process. That is why every sub-operation has its own element type and has no need of the function entry. Normally, the names of the CIP3 fold functions was taken for the name of the respective corresponding elements. The fold sub-operation elements are simply placed inside the **FoldingParams** resource (see section 7.2.39 *FoldingParams*). Because of inherent naming obscurities, the CIP3 functions **Groove** and **Lime** were renamed to **Crease** and **Gluing** in JDF. The following tables give advice on how to convert the PPF structures to JDF elements.

Table D.8.10 Converting the PPF Folding operation to JDF

PPF Key	JDF Representation	Comments
CIP3FoldDescription	-	If required, it can be expressed by the <i>FoldCatalog</i> attribute or by the fold operations.
CIP3FoldSheetIn	<i>FoldSheetIn</i> attribute of FoldingParams	-
CIP3FoldProc	Several elements inside the FoldingParams	See previous description

Table D.8.11 Converting the PPF Folding sub-operation of type Fold

PPF Key	JDF Representation	Comments
travel	<i>Travel</i> attribute of Fold	-
from	<i>From</i> attribute of Fold	-
to	<i>To</i> attribute of Fold	-
function	-	There is an extra element for each type of a Folding sub-operation. In this case it is the Fold element.

Table D.8.12 Converting the PPF Folding sub-operation of type Lime

PPF Key	JDF Representation	Comments
start-position	<i>StartPosition</i> attribute of the GlueLine element of the Gluing element	JDF uses the GlueLine element because of the advantage of more optional attributes of this type of element..
working-path	<i>WorkingPath</i> attribute of the GlueLine element of the Gluing element	JDF uses the GlueLine element because of the advantage of more optional attributes of this type of element..
working-direction	<i>WorkingDirection</i> attribute of the Gluing element	-
function	-	There is an extra element for each type of a Folding sub-operation. In this case it is the Gluing element.

Table D.8.13 Converting the PPF Folding sub-operation of all other types

PPF Key	JDF Representation	Comments
start-position	<i>StartPosition</i> attribute of the respective Fold-Operation element	-
working-path	<i>WorkingPath</i> attribute of the respective Fold-Operation element	-
working-direction	<i>WorkingDirection</i> attribute of the respective FoldOperation element	-
function	-	There is an extra element for each type of a Folding sub-operation. The extra elements are: Cut , Crease and Perforate

D.3 PPF Sheet Structure

The conversion of the PPF sheet structures is much more complex than the conversion of the product operations. A JDF layout structure, which is not directly specified in PPF, must be built up in order to place the mark objects such as register mark or density measuring field. All other sheet information is stored in specialized resources. These resources are often partitionable to specify the sheet, surface and separation to which they belong (see section [Description of Partitionable Resources](#)). The result is an inheritance of attributes comparable to the inheritance process in CIP3.

To build the layout structure, create a **Layout** resource that includes one **Signature** element with a unique *Name*. For each PPF **Sheet**, add one **Sheet** resource to the **Signature**. Set the *Name* of the corresponding **Sheet** to the value of **CIP3AdmSheetName**. For each surface (front or back) initiate a **Surface** resource with one **PlacedObjects** element. In order to define a mark object (that is **CutMark**, **CIELABMeasuringField**, **DensityMeasuringField**, **ColorControlStrip** or **RegisterMark**), build a **MarkObject** element inside **PlacedObjects**. In that element, define *CTM* and an appropriate *LayoutElement*. The CIP3 information is added to the **MarkObject** by including the mark-specific element (such as **RegisterMark** for a register mark). Note that the coordinate system of the JDF **Sheet** is specified by the *SurfaceContentsBox*, which defaults to the page coordinates and the coordinate system of the CIP3 **Sheet** is the PSExtent coordinates.

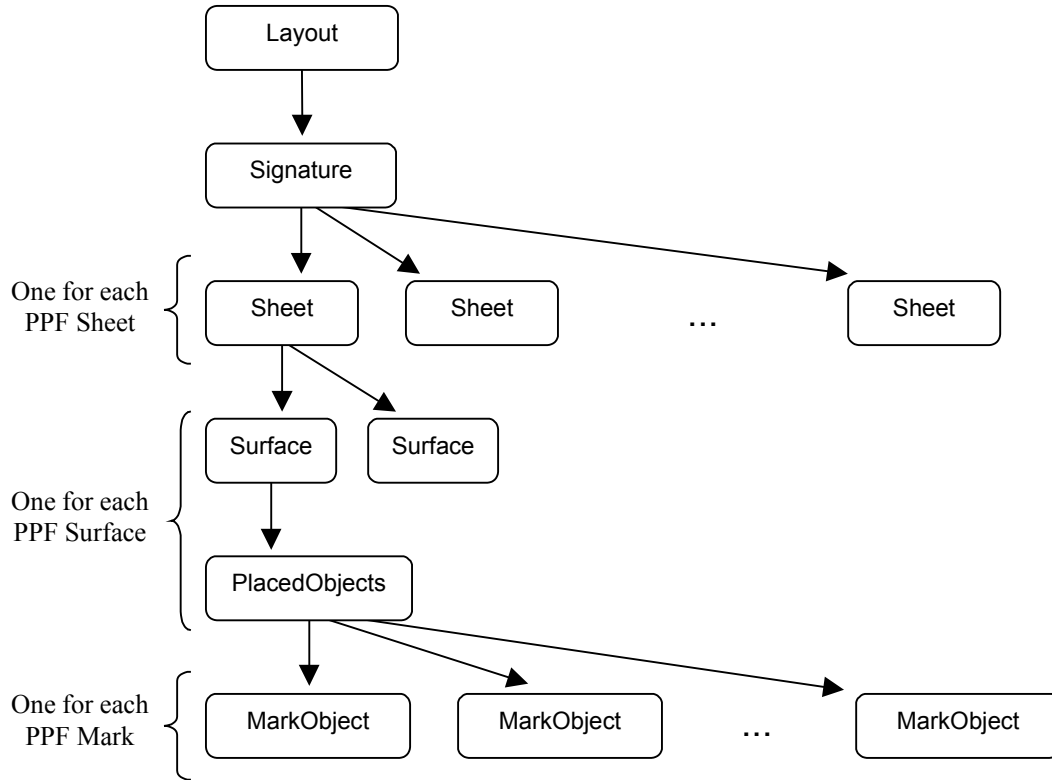


Figure D.8.2 JDF representation of sheets

If there are no product definitions in the PPF file, create JDF product nodes which are the results of all cutting and folding information in the sheet structure.

D.3.1 Administration Data

The following table defines how to convert the administration data of CIP3. In some situations, it may not be clear whether or not conversion is necessary. Processes such as **CIP3AdmFilmType**, for example, contain limited information, making it difficult to tell.

Table D.8.14 Converting administration data

PPF Key	JDF Representation	Comments
CIP3AdmSheetName	<i>Name</i> attribute of the corresponding Sheet	If there is no CIP3AdmSheetName , define a unique new one.
CIP3AdmJobName	Comment of the corresponding product node	-
CIP3AdmJobCode	<i>JobPart</i> of the corresponding product node	May conflict with CIP3ProductJobCode .
CIP3AdmMake	-	Not supported.
CIP3AdmModel	-	Not supported.
CIP3AdmSoftware	-	Not supported.
CIP3AdmCreationTime	-	Not supported.

CIP3AdmArtist	Comment of the corresponding product node	-
CIP3AdmCopyright	Comment of the corresponding product node	-
CIP3AdmCustomer	CustomerInfo element of the corresponding product node	May conflict with CIP3ProductCustomer . Note that the CustomerInfo element is structured while the CIP3AdmCustomer is not.
CIP3AdmPSExtent	indirect	-
CIP3AdmTypeOfScreen	see description	Not possible to convert appropriate
CIP3AdmFilmType	<i>Brand</i> attribute of the corresponding Media resource	<i>MediaType</i> of the Media is <i>Film</i> .
CIP3AdmFilmExtent	<i>Dimension</i> attribute of the corresponding Media resource	-
CIP3AdmFilmTrf	indirect	-
CIP3AdmPlateType	<i>Brand</i> attribute of the corresponding Media resource	<i>MediaType</i> of the Media is <i>Plate</i> .
CIP3AdmPlateExtent	<i>Dimension</i> attribute of the corresponding Media resource	-
CIP3AdmPlateTrf	indirect	-
CIP3AdmPaperGrade	<i>Grade</i> attribute of the corresponding Media resource	<i>MediaType</i> of the Media is <i>Paper</i>
CIP3AdmPaperGrammage	<i>Weight</i> attribute of the corresponding Media resource	See CIP3AdmPaperGrade .
CIP3AdmPaperThickness	<i>Thickness</i> attribute of the corresponding Media resource	See CIP3AdmPaperGrade .
CIP3AdmPaperColor	<i>LabColor</i> attribute of the Color element of the corresponding Media resource	See CIP3AdmPaperGrade .
CIP3AdmPaperExtent	<i>Dimension</i> attribute of the corresponding Media resource	-
CIP3AdmPaperTrf	indirect	-
CIP3AdmSeparationNames	see description	Create a ConventionalPrinting process (see section 6.4.1) and a corresponding ConventionalPrintingParams

		resource of <i>PrintingType</i> = <i>SheetFed</i> . For each separation name create one <i>PrintUnitProcessing</i> operation in the same sequence. Set the <i>Separation</i> and <i>Side</i> attribute. Add the Media resources as input.
CIP3AdmSheetLay	<i>SheetLay</i> attribute of the corresponding ConventionalPrintingParams resource	see CIP3AdmSeparationNames
CIP3AdmPrintVolume	<i>Amount</i> attribute of the output Component resource link of the printing process	-
CIP3AdmPressTrf	indirect	-
CIP3AdmPressExtent	indirect	-
CIP3AdmInkInfo	<i>Name</i> attribute of the <i>Color</i> element of the corresponding Ink resource	Create a partitioned Ink matching the side and separation. Add the Ink to the ConventionalPrinting process of CIP3AdmSeparationNames
CIP3AdmInkColors	<i>LabColor</i> attribute of the <i>Color</i> element of the corresponding Ink resource	see CIP3AdmInkInfo

D.3.2 Preview Images

In PPF, preview images are coded as an inline image. This is not possible in version 1.0 of XML, so JDF uses the *URL* attribute within the **Preview** resource (see section 7.2.72 *Preview*), which points to an external PNG file. The following table shows how to translate the PPF preview structure to the PNG header. Use the partition feature to assign a preview image to a specific separation and surface.

Table D.8.15 PPF preview representation as PNG

PPF Key	JDF Representation	Comments
CIP3PreviewImageWidth	“Width” of the “IHDR” chunk of the PNG file	-
CIP3PreviewImageHeight	“Height” of the “IHDR” chunk of the PNG file	-
CIP3PreviewImageBitsPerComp	“Bit depth” of the “IHDR” chunk of the PNG file	-
CIP3PreviewImageComponents	-	Because of a lack of CMYK composite support by PNG, PPF previews of this type must be separated.
CIP3PreviewImageImageMatrix	-	Not needed. Convert image data to the PNG native sequence.
CIP3PreviewImageResolution	“pHYs” chunk of the PNG file	Use the meter unit and convert dpi to dpm.

CIP3PreviewImageEncoding	-	Not needed.
CIP3PreviewImageCompression	-	Not needed. Use PNG's own compression.
CIP3PreviewImageFilterDict	-	Not needed.
CIP3PreviewImageByteAlign	-	Not needed.
CIP3PreviewImageDataSize	-	Not needed.

To calculate ink zones, JDF uses a process chain of *PreviewGeneration* and *InkZoneCalculation* processes. Add the converted CIP3 previews as an input resource to *InkZoneCalculation*. The *ProfileOffset* attribute of *InkZoneCalculationParams* can be calculated out of the different CIP3 coordinate systems.

D.3.3 Transfer Curves

Simply convert all CIP3 transfer curves to elements of a partitioned **TransferCurvePool** (see section 7.2.90 *Tile*). Add this **TransferCurvePool** as an input resource to a corresponding *InkZoneCalculation* process.

D.3.4 Register Marks

The table provides information about how to create a JDF **RegisterMark** and place this element inside the respective **MarkObject**.

Table D.8.16 Converting the parameter of the CIP3PlaceRegisterMark command

PPF Key	JDF Representation	Comments
translate-x and translate-y	<i>Center</i> attribute of RegisterMark	Apply all transformations of the CIP3 coordinate systems to get from the PS system to the paper system.
rotation	<i>Rotation</i> attribute of RegisterMark	-
type	<i>MarkType</i> attribute of RegisterMark	-
Current CIP3SetRegisterMark-Separations context	Several <i>SeparationSpec</i> elements inside the RegisterMark	-

D.3.5 Color and Ink Control

In CIP3, the two types of measuring fields are specified by an entry of the data dictionary in the **CIP3PlaceMeasuringField** command. In JDF, this approach is replaced by two different types of JDF elements: *CIELABMeasuringField* and *DensityMeasuringField*. All parameters of the **CIP3PlaceMeasuringField** command are merged into these elements. See the following tables as well as section 7.2.8 *CIELABMeasuringField* and section 7.2.30 *DensityMeasuringField* for further information. All PPF entries that are not explicitly listed in the following tables can be directly converted. Place the originated element inside the appropriate **MarkObject**.

Table D.8.17 Converting PPF color-measuring data

PPF Key	JDF Representation	Comments
position-x and position-y of the respective CIP3-PlaceMeasuringField command	<i>Center</i> attribute of CIELABMeasuring-Field	Apply all transformations of the CIP3 coordinate systems to get from the PS system to the paper system.

Type	-	There is an extra resource for each type of CIP3 measuring field.
CIE-L*, CIE-a* and CIE-b*	<i>CIE-Lab</i> attribute of CIELABMeasuringField	-

Table D.8.18 Converting PPF density-measuring data

PPF Key	JDF Representation	Comments
position-x and position-y of the respective CIP3-PlaceMeasuringField command	<i>Center</i> attribute of DensityMeasuringField	Apply all transformations of the CIP3 coordinate systems to get from the PS system to the paper system.
Type	-	There is an extra resource for each type of CIP3 measuring field.
DensityCyan , DensityMagenta , DensityYellow and DensityBlack	<i>Density</i> attribute of DensityMeasuringField	-

Like the measuring fields, the **CIP3PlaceColorControlStrip** command is translated to a structured element. All parameters of this command can be converted to the **ColorControlStrip** element (see section 7.2.13 **ColorControlStrip**) by following the instructions in table D.18, below.

Table D.8.19 Converting the parameter of the CIP3PlaceColorControlStrip command

PPF Key	JDF Representation	Comments
position-x and position-y	<i>Center</i> attribute of ColorControlStrip	Apply all transformations of the CIP3 coordinate systems to get from the PS system to the paper system.
rotation	<i>Rotation</i> attribute of ColorControlStrip	-
width and height	<i>Size</i> attribute of ColorControlStrip	-
data	Sequence of DensityMeasuringField elements within the ColorControlStrip	The entries of the data parameter have to be converted to DensityMeasuringField elements.
name	<i>StripType</i> attribute of ColorControlStrip	-

D.3.6 Cutting Data

CIP3s cut block structure is translated to JDF by defining **Cutting** processes. Since CIP3 has the ability to create nested cut blocks, one separate **Cutting** process is needed for each nested block set. Simply follow the instructions in the following table and add all originated **CutBlock** resources as input the corresponding **Cutting** process. The **CIP3CutModel** entry is not used in JDF.

Table D.8.20 Converting the Cutting Data structure

PPF Key	JDF Representation	Comments
CIP3BlockTrf	<i>BlockTrf</i> attribute of CutBlock	If the CutBlock is at the uppermost level, apply all transformations of the CIP3 coordinate systems to get from

CIP3BlockSize	<i>BlockSize</i> attribute of CutBlock	-	the PS system to the paper system.
CIP3BlockElementSize	<i>BlockElementSize</i> attribute of CutBlock	-	
CIP3BlockSubdivision	<i>BlockSubdivision</i> attribute of CutBlock		Determines how many Components are produced.
CIP3BlockType	<i>BlockType</i> attribute of CutBlock	-	
CIP3BlockElementType	<i>BlockElementType</i> attribute of CutBlock	-	
CIP3BlockName	This is expressed by resource links		Not needed in JDF.
CIP3BlockFoldingProcedure	A Folding process		See section 6.5.6.2 Folding.

For cut marks, follow the instructions in the table below. Place the originated element inside the appropriate MarkObject.

Table D.8.21 Converting the parameter of the CIP3PlaceCutMark command

PPF Key	JDF Representation	Comments
position-x and position-y	<i>Center</i> attribute of CutMark	Apply all transformations of the CIP3 coordinate systems to get from the PS system to the paper system.
mark-type	<i>MarkType</i> attribute of CutMark	-

D.3.7 Folding Data

When a CIP3 cut block has a folding operation defined (**CIP3BlockFoldingProcedure**), append a JDF **Folding** process which uses the respective output **Component** of the respective **Cutting** process as an input **Component**. See section 6.5.6.2 Folding for more information on how to translate the CIP3 folding procedure, which is used to fold the cut block.

D.3.8 Comments and Annotations

PPF comments can either be converted to an XML comment or to a human-readable form by transforming them into a **Comment** telem of the next element. In most cases, PPF comments can simply be ignored. Annotations are not supported by JDF.

D.3.9 Private Data and Content

For your private data, you should first examine if one of the new JDF elements or attributes fit your requirements. If not, please use the extension capabilities of JDF to express your needs. They are described in section 0 JDF Extensibility.

Appendix E Modelling IFRATrack in JDF

Introduction

Job tracking and production control are integral parts of a workflow system. IFRA, describe in this section, has defined a job tracking system called IFRATrack that fulfills a large number of the job-tracking requirements of a production scenario, and is especially effective in newspaper production. The JDF messaging system generalizes the IFRATrack approach, expanding its focus from a newspaper workflow to one that encompasses the entire graphic arts industry. This appendix provides further detail about the way in which JDF expands upon the existing IFRATrack technology.

E.1 IFRA Objects and JDF Nodes

IFRATrack traces the status of objects, and these objects are modified by processes that are only generic. JDF, on the other hand, precisely defines process nodes that create output resources. These JDF output resources are equivalent to IFRATrack objects, so tracking the state of a JDF node conveys a superset of the information communicated by tracking the state of an IFRATrack. The sections that follow define the mapping of IFRA concepts to JDF concepts in greater detail.

E.1.1 Object Identification

IFRATrack defines objects with with an object path. The object path, in turn, may be a unique identifier, or uid. JDF also supports uids for internal linking of objects, although these uid's should not be exported beyond the scope of a JDF document. External references to JDF nodes should be made the JobID/JobPartID pair. These values may be defined by an external system, such as MIS, and can be used to uniquely track JDF nodes.

E.1.2 IFRA Object Hierarchy

IFRATrack defines an explicit hierarchy to define a newspaper,; from Issue through Edition, EditionVersion, and so on. JDF, on the other hand, defines a generic hierarchy of products containing a description attribute that allows the products to be named. An IFRATrack-conforming JDF job consequently includes a product hierarchy with product nodes that contain the appropriate description fields. Furthermore, the abstract IFRA Element type is mapped to the JDF **LayoutElement** type.

E.1.3 Object States

IFRA defines object states that define the status of a resource, although they also define the status of the process that defines a resource. JDF defines explicit states for both processes and resources. In addition, JDF defines a descriptive string to denote the details of each status. The mapping is defined in the following table.

IFRA Object Status	JDF Node Status	JDF Resource Status	Description
<i>Not Started</i>	<i>waiting</i>	<i>unavailable</i>	Status prior to <i>in_progress</i> .
	<i>ready</i>	<i>unavailable</i>	JDF defines a test-run mode that allows generalized pre-flighting. <i>ready</i> is the status after <i>testrun</i> .
<i>In Progress</i>	<i>setup</i>	<i>unavailable</i>	A process is <i>in_progress</i> but not yet producing any output.
	<i>in_progress</i>	<i>unavailable</i>	A process is <i>in_progress</i> .

	<i>cleanup</i>	<i>available</i>	A process is running after all output has been produced.
<i>On Hold</i>	<i>stopped</i>	<i>unavailable</i>	A process is active but not currently producing, as when maintenance is run during a job.
<i>Completed</i>	<i>completed</i>	<i>available</i>	heureka
<i>Aborted</i>	<i>aborted</i>	<i>unavailable</i> ¹	Fatal Error

E.1.4 Deadlines and Scheduling

In IFRATrack, activities may be linked to deadlines. JDF defines deadlines in the `NodeInfo` element of every node. The definition of deadline values is identical.

IFRA defines an integer value for deadline level. JDF defines four explicit enumerations for *DueLevel* in order to assure that devices in a heterogeneous system have the same concept of deadline level.

E.2 JMF Messages that Translate IFRATrack Messages

The messages explained in this section can be used to emulate IFRATrack functionality.

E.2.1 JMF Phase Message

The phase message describes the phase or status of a given process.

Name	Data Type	Description
<i>Status</i>	enumeration	One of the JDF status values defined in Table 3.3 Contents of a JDF node.
<i>StatusDetails</i>	string	String that defines the job state more specifically.
<i>JDF?</i>	element	Complete JDF node that is currently being processed. Optional and supplied only on request.
<i>Part ?</i>	element	Describes which part of a job is currently being processed. Parts of a resource are resources that are created in a very similar way and therefore only defined once, as is the case with separations of plates.
<i>Device *</i>	element	Device resources that are currently in use by the process.
<i>Employee *</i>	element	Employee resources that are currently working on the job.
<i>ModuleStatus *</i>	element	Status of individual modules. For details on using <code>ModuleStatus</code> elements, see section 5.5.2.3 Status.

Table E.8.22 Contents of the Phase message

E.2.2 JMF Progress Message

The progress message describes the numerical progress of a given process.

Name	Data Type	Description
<i>Amount</i>	number	Progress amount in units, as requested.
<i>Unit?</i>	string	Unit of quantity, as defined in the JDF output resource of the node being queried. Defaults to % completed.

¹ Unless aborted during cleanup

<i>DeadLine?</i>	NMTOKEN	Scheduling state of the job. Possible values are: <i>InTime</i> -- Default value. <i>Warning</i> <i>Late</i> For more details on scheduling, see section 3.5 Process and Node Information.
Part?	element	Describes which part of a job is currently being processed.

Appendix F StatusDetails Supported Strings

The *StatusDetails* attribute refines the concept of a job status to be job specific or a device status to be device specific. The following tables define individual *StatusDetail* values and map them to the appropriate job specific state *Status* or device specific state *DeviceStatus*.

Table F.1 *StatusDetails and Status mapping for generic devices*

StatusDetails	Status	DeviceStatus	Description
<i>ControllDeferred</i>	-	<i>stopped</i>	The device is controlled by a master device and cannot be accessed.

Table F.2 *StatusDetails and Status mapping for conventional printing devices*

StatusDetails	Status	DeviceStatus	Description
<i>Good</i>	<i>in_progress</i>	<i>running</i>	Production of sheets in progress, good copy counter is on.
<i>Waste</i>	<i>in_progress</i>	<i>running</i>	Production of sheets in progress, good copy counter is off.
<i>FormChange</i>	<i>setup</i>	<i>setup</i>	In conventional printing. changing of plates or in digital printing changing of images.
<i>SizeChange</i>	<i>setup</i>	<i>setup</i>	Changing setup for media size.
<i>WashUp</i>	<i>cleanup</i>	<i>cleanup</i>	Machine is washed before, during or after production. <i>WashUp</i> is a super-term for <i>BlanketWash</i> , <i>CylinderWash</i> , <i>CleaningInkingUnit</i> , or <i>CleaningInkFountain</i> . <i>WashUp</i> is the default which is assumed if <i>StatusDetails</i> is not specified.
<i>InkingRollerWash</i>	<i>cleanup</i>	<i>cleanup</i>	Washing of the inking roller; sub-term of <i>WashUp</i> .
<i>PlateWash</i>	<i>cleanup</i>	<i>cleanup</i>	Washing of the plate; sub-term of <i>WashUp</i> .
<i>DampeningRoller-Wash</i>	<i>cleanup</i>	<i>cleanup</i>	Washing of the dampening roller; sub-term of <i>WashUp</i> .
<i>BlanketWash</i>	<i>cleanup</i>	<i>cleanup</i>	Washing of the blanket; sub-term of <i>WashUp</i> .
<i>CylinderWash</i>	<i>cleanup</i>	<i>cleanup</i>	Washing of impression cylinders; sub-term of <i>WashUp</i> .
<i>CleaningInkFountain</i>	<i>cleanup</i>	<i>cleanup</i>	Cleaning of the ink fountain; sub-term of <i>WashUp</i> .
<i>Pause</i>	<i>stopped</i>	<i>stopped</i>	Machine paused, restart is possible.
<i>MissResources</i>	<i>stopped</i>	<i>stopped</i>	Production has been stopped because resources are missed. For example, if the machine has consumed paper, ink, plates, etc., and waits for new resources; sub-term of <i>Pause</i> .
<i>WaitForApproval</i>	<i>stopped</i>	<i>stopped</i>	Production has been stopped because a required approval is still missing; sub-term of <i>Pause</i> .

<i>ShutDown</i>	<i>stopped</i>	<i>down</i>	Machine stopped (may be switched off), restart requires a run up.
<i>BreakDown</i>	<i>stopped</i>	<i>down</i>	Breakdown of the device, repair required.
<i>Repair</i>	<i>stopped</i>	<i>down</i>	After a breakdown the device is being repaired.
<i>Failure</i>	<i>stopped</i>	<i>stopped</i>	Failure of the device; requires some maintenance in order to restart the device.
<i>PaperJam</i>	<i>stopped</i>	<i>stopped</i>	Paper jam in the device; sub-term of <i>Failure</i> .
<i>Maintenance</i>	<i>stopped</i>	<i>stopped</i>	Maintenance of the device.
<i>BlanketChange</i>	<i>stopped</i>	<i>stopped</i>	Changing of blankets; sub-term for <i>Maintenance</i> .
<i>SleeveChange</i>	<i>stopped</i>	<i>stopped</i>	Changing of sleeves; sub-term for <i>Maintenance</i> .

Appendix G ModuleType Supported Strings

Both the `ModuleStatus` element (see Table 5.38 Contents of the `ModuleStatus` element) and the `ModulePhase` element (see Table 3.25 Contents of the `ModulePhase` element) contain a *ModuleType* attribute that defines individual modules within a machine. The following table defines individual *ModuleType* values.

Table G.1 *ModuleType* definition for conventional printing devices

ModuleType	Description
<i>Feeder</i>	Feeder module, feeds the device with paper.
<i>PrintModule</i>	Unit for printing a color.
<i>CoatingModule</i>	Unit for coatings, for example, full coating of varnish.
<i>Drier</i>	Module for drying the previously panted color or varnish.
<i>PerfectingModule</i>	Unit for perfecting, reversing device.
<i>ExtensionModule</i>	Unit for extending the distance between modules, for example to increase the distance between the last printing module and the delivery module.
<i>Delivery</i>	Delivery module, unit for gathering the printed sheets.
<i>Imaging</i>	Imaging Module in a direct to plate machine.
<i>Numbering</i>	Numbering unit.

Appendix H Supported Error Codes in JMF

The following list defines the standard *ReturnCode* for messaging. The ID numbers are decimal. Error messages below 100 are reserved for protocol errors. Error messages above 100 are used for device and controller errors and error messages above 200 for job and pipe specific errors.

Table H.1 Return codes for JMF

ReturnCode	Description
0	Success
1 - 99	Protocol errors
1	General error
2	Internal error
3	XML parser error (. for instance, if a MIME file is sent to an XML controller)
4	XML validation error
5	Query/command not implemented
6	Invalid parameters
7	Insufficient parameters
8	Device not available (controller exists but not the device or queue)
100 – 199	Device and controller errors
100	Device not running
101	Device incapable of fulfilling request for example, a RIP that has been asked to cut a sheet)
102	No executable node exists in the JDF
103	Job ID not known by controller
104	JobPartID not known by controller
105	Queue entry not in queue
106	Queue request failed because queue entry is already executing
107	Queue entry is already executing, late change is not accepted
108	Selection or applied filter results in an empty list
109	Selection or applied filter results in an incomplete list. A buffer cannot provide the complete list queried for.
200 - ...	Job and pipe specific errors
200	Invalid resource parameters
201	Insufficient resource parameters
202	PipeID unknown
203	Unlinked resource link

Appendix I Event Types and Values

The **Notification** element is used for messaging and logging of events. It is defined in section 3.9.1.2 **Notification**. Notifications are grouped into five classes: *event*, *information*, *warning*, *error*, and *fatal*. Beyond this classification, the attributes *Type* and *Value* of the **Notification** element provide a container for detailed information about the event.

The attribute *Value* of the **Notification** element is of data type string which represents a void data type. In connection with a certain event type, other data types may be more appropriate. Then the content of the *Value* may be casted to that data type. Supported event types (table heading: *Type*), the associated data type of the attribute *Value* (table heading: *Value Data Type*) and their descriptions (table heading: *Description*) are listed in the following tables.

Table I.1 Event types and associated values of notification class: *event* (pure events)

Type	Value Data Type	Description
<i>Barcode</i>	<i>string</i>	A barcode has been scanned. <i>Value</i> : contains the scanned barcode.
<i>FCN-Key</i>	<i>integer</i>	A function key has been activated at a console. <i>Value</i> : contains the number of that function key.
<i>SystemTimeSet</i>	<i>timeInstant</i>	The system time of a device/controller/agent has been set (for example: readjusted, changed to daylight saving time, etc.). <i>Value</i> : contains the new time.
<i>CounterReset</i>	-	The production counter of a device has been reset.

Table I.2 Event types and associated values of notification classes: *warning...fatal*

Type	Class	Value Data Type	Description
<i>Error</i>	<i>error</i>	<i>string</i>	Internal Error ID of the application that declares the error.
<i>Fatal-Error</i>	<i>fatal</i>	<i>string</i>	Internal ID of the fatal error of the application that declares the fatal error.
<i>Warning</i>	<i>warning</i>	<i>string</i>	Internal Warning ID of the application that declares the warning.

Appendix J Examples

Note that these examples are still heavily under construction and should be used for general overview only. The emphasis is *not* on the individual bytes, e.g. capitalization or exact keywords.

Brief Example

J.1.1 Before Processing

This is a simple example of a JDF that describes color conversion for one file.

```
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="HDM20000821152850" Type="ColorSpaceConversion" JobID="HDM20000821152850"
Status="waiting" Version="0.9">
  <!--(c) Heidelberger Druckmaschinen AG 1999-2000-->
  <!--Warning: preliminary format; use at your own risk-->
  <NodeInfo/>
  <ResourcePool>
    <RunList ID="Link0011" Class="Parameter" Status="available">
      <Run Pages="0~-1">
        <LayoutElement>
          <FileSpec FileName="colortest.pdf"/>
        </LayoutElement>
      </Run>
    </RunList>
    <ColorSpaceConversionParams ID="Link0012" Class="Parameter" Status="available"
FinalTargetDevice="File::SMProcessCMYK.icc">
      <ColorSpaceConversion SourceCS="RGB" Operation="Convert"
SourceObjects="ImagePhotographic ImageScreenShot SmoothShades"
SourceProfile="File::image.icc" RenderingIntent="Perceptual"/>
      <ColorSpaceConversion SourceCS="RGB" Operation="Convert" SourceObjects="Text
LineArt" SourceProfile="File::text.icc" RenderingIntent="Perceptual"/>
    </ColorSpaceConversionParams>
    <RunList ID="Link0013" Class="Parameter" Status="unavailable">
      <Run Pages="0~-1">
        <LayoutElement>
          <FileSpec FileName="colortest.pdf"/>
        </LayoutElement>
      </Run>
    </RunList>
  </ResourcePool>
  <ResourceLinkPool>
    <RunListLink rRef="Link0011" Usage="input"/>
    <ColorSpaceConversionParamsLink rRef="Link0012" Usage="input"/>
    <RunListLink rRef="Link0013" Usage="output"/>
  </ResourceLinkPool>
  <AuditPool>
    <Created Author="Rainer's GATWriter 0.2000" TimeStamp="2000-08-21T15:28:50+02:00"/>
  </AuditPool>
</JDF>
```

J.1.2 After Processing

This is a simple example of a JDF that describes color conversion for one file after the color conversion process has been executed.

```
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="HDM20000821152850" Type="ColorSpaceConversion" JobID="HDM20000821152850"
Status="completed" Version="0.9">
  <!--(c) Heidelberger Druckmaschinen AG 1999-2000-->
  <!--Warning: preliminary format; use at your own risk-->
  <ResourcePool>
```

```

<RunList ID="Link0011" Class="Parameter" Status="available">
  <Run Pages="0~-1">
    <LayoutElement>
      <FileSpec FileName="colortest.pdf"/>
    </LayoutElement>
  </Run>
</RunList>
<ColorSpaceConversionParams ID="Link0012" Class="Parameter" Status="available"
FinalTargetDevice="File::SMPProcessCMYK.icc">
  <ColorSpaceConversion SourceCS="RGB" Operation="Convert"
SourceObjects="ImagePhotographic ImageScreenShot SmoothShades"
SourceProfile="File::image.icc" RenderingIntent="Perceptual"/>
  <ColorSpaceConversion SourceCS="RGB" Operation="Convert" SourceObjects="Text
LineArt" SourceProfile="File::text.icc" RenderingIntent="Perceptual"/>
</ColorSpaceConversionParams>
<RunList ID="Link0013" Class="Parameter" Status="available">
  <Run Pages="0~-1">
    <LayoutElement>
      <FileSpec FileName="colortest.pdf"/>
    </LayoutElement>
  </Run>
</RunList>
</ResourcePool>
<ResourceLinkPool>
  <RunListLink rRef="Link0011" Usage="input"/>
  <ColorSpaceConversionParamsLink rRef="Link0012" Usage="input"/>
  <RunListLink rRef="Link0013" Usage="output"/>
</ResourceLinkPool>
<AuditPool>
  <Created Author="Rainer's GATWriter 0.2000" TimeStamp="2000-08-21T15:28:50+02:00"/>
  <Modified Author="EatJDF Complete: task=*" TimeStamp="2000-08-21T16:58:58+02:00"/>
  <RunListAudit rRef="Link0011" Usage="input"/>
  <ColorSpaceConversionParamsAudit rRef="Link0012" Usage="input"/>
  <RunListAudit rRef="Link0013" Usage="output"/>
  <PhaseTime End="2000-08-21T16:58:58+02:00" Start="2000-08-21T16:58:58+02:00"
Status="setup" TimeStamp="2000-08-21T16:58:58+02:00"/>
  <PhaseTime End="2000-08-21T16:58:58+02:00" Start="2000-08-21T16:58:58+02:00"
Status="running" TimeStamp="2000-08-21T16:58:58+02:00"/>
  <PhaseTime End="2000-08-21T16:58:58+02:00" Start="2000-08-21T16:58:58+02:00"
Status="cleanup" TimeStamp="2000-08-21T16:58:58+02:00"/>
</AuditPool>
</JDF>

```

J.2 Product JDF to Process

The following Example describe a pair of college textbooks, one teachers edition and one students edition as product intent. Most intent resources are intentionally left empty.

```

<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="HDM20000821155440" Type="Product" JobID="HDM20000821155440" Status="waiting"
Version="0.9">
  <!--(c) Heidelberger Druckmaschinen AG 1999-2000-->
  <!--Warning: preliminary format; use at your own risk-->
  <ResourcePool>
    <Component ID="Link0003" Class="Quantity" Amount="100" Status="unavailable"
DescriptiveName="Book"/>
    <Component ID="Link0005" Class="Quantity" Amount="2000" Status="unavailable"
DescriptiveName="Cover">
      <!--This cover is reused by both-->
    </Component>
    <SizeIntent ID="Link0006" Class="Parameter" Status="available">
      <NumberSpan Name="Height" Range="756-828" Preferred="792"/>
      <NumberSpan Name="Width" Range="540-612" Preferred="576"/>
    </SizeIntent>
    <SizeIntent ID="Link0008" Class="Parameter" Status="available">
      <NumberSpan Name="Height" Range="756-828" Preferred="792"/>
      <NumberSpan Name="Width" Range="540-612" Preferred="576"/>

```

```

    <IntegerSpan Name="Pages" Preferred="240"/>
  </SizeIntent>
  <Component ID="Link0011" Class="Quantity" Amount="1000" Status="unavailable"
DescriptiveName="Book">
    <!--Students Book Intent-->
  </Component>
  <SizeIntent ID="Link0014" Class="Parameter" Status="available">
    <NumberSpan Name="Height" Range="756-828" Preferred="792"/>
    <NumberSpan Name="Width" Range="540-612" Preferred="576"/>
    <IntegerSpan Name="Pages" Preferred="198"/>
  </SizeIntent>
</ResourcePool>
<AuditPool>
  <Created Author="Rainer's GATWriter 0.2000" TimeStamp="2000-08-21T15:54:40+02:00"/>
</AuditPool>
<JDF ID="Link0002" Type="Product" Status="waiting" JobPartID="0"
DescriptiveName="Teacher's Edition">
  <ResourcePool>
    <Component ID="Link0009" Class="Quantity" Amount="100" Status="unavailable"
DescriptiveName="Insert"/>
  </ResourcePool>
  <ResourceLinkPool>
    <ComponentLink rRef="Link0003" Usage="output" Amount="100"/>
    <ComponentLink rRef="Link0009" Usage="input" Amount="100"/>
    <ComponentLink rRef="Link0005" Usage="input" Amount="100"/>
  </ResourceLinkPool>
  <JDF ID="Link0007" Type="Product" Status="waiting" JobPartID="2"
DescriptiveName="Teacher's Insert">
    <ResourceLinkPool>
      <SizeIntentLink rRef="Link0008" Usage="input"/>
      <ComponentLink rRef="Link0009" Usage="output" Amount="100"/>
    </ResourceLinkPool>
  </JDF>
</JDF>
<JDF ID="Link0004" Type="Product" Status="waiting" JobPartID="1"
DescriptiveName="Cover">
  <ResourceLinkPool>
    <ComponentLink rRef="Link0005" Usage="output" Amount="2000"/>
    <SizeIntentLink rRef="Link0006" Usage="input"/>
  </ResourceLinkPool>
</JDF>
<JDF ID="Link0010" Type="Product" Status="waiting" JobPartID="3"
DescriptiveName="Student's Edition">
  <ResourcePool>
    <Component ID="Link0013" Class="Quantity" Amount="1000" Status="unavailable"
DescriptiveName="Insert"/>
  </ResourcePool>
  <ResourceLinkPool>
    <ComponentLink rRef="Link0011" Usage="output" Amount="1000"/>
    <ComponentLink rRef="Link0013" Usage="input" Amount="1000"/>
    <ComponentLink rRef="Link0005" Usage="input" Amount="1000"/>
  </ResourceLinkPool>
  <JDF ID="Link0012" Type="Product" Status="waiting" JobPartID="4"
DescriptiveName="Student's Insert">
    <ResourceLinkPool>
      <ComponentLink rRef="Link0013" Usage="output" Amount="1000"/>
      <SizeIntentLink rRef="Link0014" Usage="input"/>
    </ResourceLinkPool>
  </JDF>
</JDF>
</JDF>

```

J.3 16-Page 4-up Brochure—Layout, RunList, Cut, Fold

J.4 Spawning and Merging

The following set of examples show a JDF job in the relevant stages of spawning and merging. One example defines a simple brochure with a cover and an insert. The red node, which defines the cover, is spawned, modified and subsequently merged. Blue elements represent meta-data that apply to spawning and merging.

J.4.1 Example 2 Component JDF before Spawning

The following JDF file describes a two-component brochure. The resources are not fleshed out.

```
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="HDM20000912181238" Type="Product" JobID="HDM20000912181238"
Status="waiting" Version="0.9" JobPartID="Part1">
  <NodeInfo/>
  <ResourcePool>
    <Component ID="Link0002" Class="Quantity" Amount="10000"
Status="unavailable" DescriptiveName="complete 16-page Brochure"/>
    <BindingIntent ID="Link0003" Class="Intent" Status="available"/>
    <Component ID="Link0005" Class="Quantity" Status="unavailable"
DescriptiveName="Cover Component"/>
    <Component ID="Link0009" Class="Quantity" Status="unavailable"
DescriptiveName="Insert Component"/>
  </ResourcePool>
  <ResourceLinkPool>
    <ComponentLink rRef="Link0002" Usage="output"/>
    <BindingIntentLink rRef="Link0003" Usage="input"/>
    <ComponentLink rRef="Link0005" Usage="input"/>
    <ComponentLink rRef="Link0009" Usage="input"/>
  </ResourceLinkPool>
  <JDF ID="Link0004" Type="Product" Status="waiting" JobPartID="Part2"
DescriptiveName="Cover">
    <NodeInfo/>
    <ResourcePool>
      <SizeIntent ID="Link0006" Class="Intent" Status="available"/>
      <ColorIntent ID="Link0007" Class="Intent" Status="available"/>
    </ResourcePool>
    <ResourceLinkPool>
      <ComponentLink rRef="Link0005" Usage="output"/>
      <SizeIntentLink rRef="Link0006" Usage="input"/>
      <ColorIntentLink rRef="Link0007" Usage="input"/>
    </ResourceLinkPool>
    <AuditPool/>
  </JDF>
  <JDF ID="Link0008" Type="Product" Status="waiting" JobPartID="Part3"
DescriptiveName="Insert">
    <NodeInfo/>
    <ResourcePool>
      <SizeIntent ID="Link0010" Class="Intent" Status="available"/>
      <ColorIntent ID="Link0011" Class="Intent" Status="available"/>
    </ResourcePool>
    <ResourceLinkPool>
      <ComponentLink rRef="Link0009" Usage="output"/>
      <SizeIntentLink rRef="Link0010" Usage="input"/>
      <ColorIntentLink rRef="Link0011" Usage="input"/>
    </ResourceLinkPool>
    <AuditPool/>
  </JDF>
</JDF>
```

J.4.2 Example 2 Component JDF Parent after spawning the cover node

The following JDF is the parent JDF after spawning. The **Component** that describes the cover is marked as *spawned_RW*, since it was copied into the spawned node and may be modified. A **Spawned** audit was inserted into the Cover nodes parent's **AuditPool**, and the Spawned node itself has a **Status** of *spawned*.

```
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="HDM20000912181238" Type="Product" JobID="HDM20000912181238"
Status="waiting" Version="0.9" JobPartID="Part1">
  <ResourcePool>
    <Component ID="Link0002" Class="Quantity" Amount="10000"
Status="unavailable" DescriptiveName="complete 16-page Brochure"/>
    <BindingIntent ID="Link0003" Class="Intent" Status="available"/>
    <Component ID="Link0005" Class="Quantity" SpawnStatus="spawned_RW"
Status="unavailable" DescriptiveName="Cover Component"/>
    <Component ID="Link0009" Class="Quantity" Status="unavailable"
DescriptiveName="Insert Component"/>
  </ResourcePool>
  <ResourceLinkPool>
    <ComponentLink rRef="Link0002" Usage="output"/>
    <BindingIntentLink rRef="Link0003" Usage="input"/>
    <ComponentLink rRef="Link0005" Usage="input"/>
    <ComponentLink rRef="Link0009" Usage="input"/>
  </ResourceLinkPool>
  <AuditPool>
    <Spawned URL="File:../spawn.jdf" jRef="Link0004" TimeStamp="2000-
09-12T18:12:39+02:00" rRefsRWCopied="Link0005"/>
  </AuditPool>
  <JDF ID="Link0004" Type="Product" Status="spawned" JobPartID="Part2"
DescriptiveName="Cover">
    <ResourcePool>
      <SizeIntent ID="Link0006" Class="Intent" Status="available"/>
      <ColorIntent ID="Link0007" Class="Intent" Status="available"/>
    </ResourcePool>
    <ResourceLinkPool>
      <ComponentLink rRef="Link0005" Usage="output"/>
      <SizeIntentLink rRef="Link0006" Usage="input"/>
      <ColorIntentLink rRef="Link0007" Usage="input"/>
    </ResourceLinkPool>
  </JDF>
  <JDF ID="Link0008" Type="Product" Status="waiting" JobPartID="Part3"
DescriptiveName="Insert">
    <ResourcePool>
      <SizeIntent ID="Link0010" Class="Intent" Status="available"/>
      <ColorIntent ID="Link0011" Class="Intent" Status="available"/>
    </ResourcePool>
    <ResourceLinkPool>
      <ComponentLink rRef="Link0009" Usage="output"/>
      <SizeIntentLink rRef="Link0010" Usage="input"/>
      <ColorIntentLink rRef="Link0011" Usage="input"/>
    </ResourceLinkPool>
  </JDF>
</AncestorPool/>
</JDF>
```

J.4.3 Example 2 Component JDF spawned node

The Component that represents the cover was copied into the spawned node, since it is the output resource. It is not locked, since it was spawned in RW mode. The existence of an AncestorPool denotes the node as spawned and defines the parent node.

```
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="Link0004" Type="Product" JobID="HDM20000912181238"
Status="waiting" Version="0.9" JobPartID="Part2"
DescriptiveName="Cover">
<NodeInfo/>
  <ResourcePool>
    <SizeIntent ID="Link0006" Class="Intent" Status="available"/>
    <ColorIntent ID="Link0007" Class="Intent" Status="available"/>
    <Component ID="Link0005" Class="Quantity" Status="unavailable"
DescriptiveName="Cover Component"/>
  </ResourcePool>
  <ResourceLinkPool>
    <ComponentLink rRef="Link0005" Usage="output"/>
    <SizeIntentLink rRef="Link0006" Usage="input"/>
    <ColorIntentLink rRef="Link0007" Usage="input"/>
  </ResourceLinkPool>
  <AncestorPool>
    <Ancestor ID="HDM20000912181238"/>
  </AncestorPool>
</JDF>
```

J.4.4 Example 2 Component JDF after merging

In this example, it is assumed that the cover output component was created by some processor that processed the spawned node. This resulted in the Component becoming available. The Component was also removed from the copy of the spawned node, since it would otherwise exist twice.

```
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="HDM20000912181238" Type="Product" JobID="HDM20000912181238"
Status="waiting" Version="0.9" JobPartID="Part1">
  <ResourcePool>
    <Component ID="Link0002" Class="Quantity" Amount="10000"
Status="unavailable" DescriptiveName="complete 16-page Brochure"/>
    <BindingIntent ID="Link0003" Class="Intent" Status="available"/>
    <Component ID="Link0005" Class="Quantity" Status="available"
DescriptiveName="Cover Component"/>
    <Component ID="Link0009" Class="Quantity" Status="unavailable"
DescriptiveName="Insert Component"/>
  </ResourcePool>
  <ResourceLinkPool>
    <ComponentLink rRef="Link0002" Usage="output"/>
    <BindingIntentLink rRef="Link0003" Usage="input"/>
    <ComponentLink rRef="Link0005" Usage="input"/>
    <ComponentLink rRef="Link0009" Usage="input"/>
  </ResourceLinkPool>
  <AuditPool>
    <Spawned URL="File:../spawn.jdf" jRef="Link0004" TimeStamp="2000-
09-12T18:12:39+02:00" rRefsRWCopied="Link0005"/>
    <Merged URL="File:../spawn.jdf" jRef="Link0004" TimeStamp="2000-09-
12T18:12:39+02:00" rRefsOverwritten="Link0005"/>
  </AuditPool>
```



```

<JDF ID="Link0004" Type="Product" Status="waiting" Version="0.9"
JobPartID="Part2" DescriptiveName="Cover">
  <ResourcePool>
    <SizeIntent ID="Link0006" Class="Intent" Status="available"/>
    <ColorIntent ID="Link0007" Class="Intent" Status="available"/>
  </ResourcePool>
  <ResourceLinkPool>
    <ComponentLink rRef="Link0005" Usage="output"/>
    <SizeIntentLink rRef="Link0006" Usage="input"/>
    <ColorIntentLink rRef="Link0007" Usage="input"/>
  </ResourceLinkPool>
</JDF>
<JDF ID="Link0008" Type="Product" Status="waiting" JobPartID="Part3"
DescriptiveName="Insert">
  <ResourcePool>
    <SizeIntent ID="Link0010" Class="Intent" Status="available"/>
    <ColorIntent ID="Link0011" Class="Intent" Status="available"/>
  </ResourcePool>
  <ResourceLinkPool>
    <ComponentLink rRef="Link0009" Usage="output"/>
    <SizeIntentLink rRef="Link0010" Usage="input"/>
    <ColorIntentLink rRef="Link0011" Usage="input"/>
  </ResourceLinkPool>
</JDF>
</JDF>

```

J.5 Conversion of PJTF to JDF

J.5.1 PJTF input

The following code defines 4-up duplex impositioning of a 17 page pdf document in Adobe PJTF format:

```

%JTF-1.2
1 0 obj
<<
/A [ 3 0 R ]
/V 1.1
/Cn [ 2 0 R ]
>>
endobj
2 0 obj
<<
/Type /JobTicketContents
/D [ 6 0 R ]
/PL 8 0 R
>>
endobj
3 0 obj
<<
/D (D:19991111173640)
/JTM (Default JT Creator)
/C (JT created)
>>
endobj
4 0 obj
<<
/Type /Catalog
/JT 1 0 R
>>
endobj
5 0 obj

```

```
<<
/Producer (HD PDFWrite vs. 0.1)
>>
endobj
6 0 obj
<<
/Fi [ 7 0 R ]
>>
endobj
7 0 obj
<<
/Fi (panrt17a.pdf)
>>
endobj
8 0 obj
<<
/Si 9 0 R
>>
endobj
9 0 obj
<<
/S 10 0 R
>>
endobj
10 0 obj
[ 11 0 R ]
endobj
11 0 obj
<<
/MS
<<
/Cl (sheet of paper)
/Me 12 0 R
>>
/Fr 13 0 R
/B 18 0 R
>>
endobj
12 0 obj
<<
/Dm [ 842 1191 842 1191 ]
>>
endobj
13 0 obj
<<
/PO [ 14 0 R 15 0 R 16 0 R 17 0 R ]
>>
endobj
14 0 obj
<<
/CTM [ 0.45 0 0 0.45 21 624 ]
/O 0
/Cl [ 21 624 399 1159 ]
>>
endobj
15 0 obj
<<
/CTM [ 0.45 0 0 0.45 442 624 ]
/O 1
/Cl [ 442 624 820 1159 ]
>>
endobj
16 0 obj
<<
/CTM [ 0.45 0 0 0.45 21 29 ]
/O 2
/Cl [ 21 29 399 564 ]
>>
endobj
17 0 obj
<<
```

```
/CTM [ 0.45 0 0 0.45 442 29 ]
/O 3
/C1 [ 442 29 820 564 ]
>>
endobj
18 0 obj
<<
/PO [ 19 0 R 20 0 R 21 0 R 22 0 R ]
>>
endobj
19 0 obj
<<
/CTM [ 0.45 0 0 0.45 21 624 ]
/O 4
/C1 [ 21 624 399 1159 ]
>>
endobj
20 0 obj
<<
/CTM [ 0.45 0 0 0.45 442 624 ]
/O 5
/C1 [ 442 624 820 1159 ]
>>
endobj
21 0 obj
<<
/CTM [ 0.45 0 0 0.45 21 29 ]
/O 6
/C1 [ 21 29 399 564 ]
>>
endobj
22 0 obj
<<
/CTM [ 0.45 0 0 0.45 442 29 ]
/O 7
/C1 [ 442 29 820 564 ]
>>
endobj
xref
0 23
0000000000 65535 f
0000000009 00000 n
0000000071 00000 n
0000000146 00000 n
0000000233 00000 n
0000000283 00000 n
0000000338 00000 n
0000000377 00000 n
0000000419 00000 n
0000000453 00000 n
0000000487 00000 n
0000000516 00000 n
0000000608 00000 n
0000000660 00000 n
0000000722 00000 n
0000000810 00000 n
0000000900 00000 n
0000000985 00000 n
0000001072 00000 n
0000001134 00000 n
0000001222 00000 n
0000001312 00000 n
0000001397 00000 n
trailer
<<
/Root 4 0 R
/Info 5 0 R
/Size 23
>>
startxref
1484
```

```
%%EOF
```

J.5.2 JDF output

This JDF file describes the Imposition process defined by the PJTF file.

```
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="HDM20000821165026" Type="Imposition" JobID="HDM20000821165026" Status="waiting"
Version="0.9">
  <!--(c) Heidelberger Druckmaschinen AG 1999-2000-->
  <!--Warning: preliminary format; use at your own risk-->
  <NodeInfo/>
  <ResourcePool>
    <Layout ID="Link0002" Class="Parameter" rRefs="Cos13 Cos18 Cos11" Status="available">
      <Signature ID="Cos9">
        <Sheet rRef="Cos11"/>
      </Signature>
    </Layout>
    <Surface ID="Cos13" Side="Front">
      <PlacedObjects>
        <ContentObject CTM="0.45 0 0 0.45 21 624" Ord="0" ClipBox="21 624 399 1159"/>
        <ContentObject CTM="0.45 0 0 0.45 442 624" Ord="1" ClipBox="442 624 820 1159"/>
        <ContentObject CTM="0.45 0 0 0.45 21 29" Ord="2" ClipBox="21 29 399 564"/>
        <ContentObject CTM="0.45 0 0 0.45 442 29" Ord="3" ClipBox="442 29 820 564"/>
      </PlacedObjects>
    </Surface>
    <Surface ID="Cos18" Side="Back">
      <PlacedObjects>
        <ContentObject CTM="0.45 0 0 0.45 21 624" Ord="4" ClipBox="21 624 399 1159"/>
        <ContentObject CTM="0.45 0 0 0.45 442 624" Ord="5" ClipBox="442 624 820 1159"/>
        <ContentObject CTM="0.45 0 0 0.45 21 29" Ord="6" ClipBox="21 29 399 564"/>
        <ContentObject CTM="0.45 0 0 0.45 442 29" Ord="7" ClipBox="442 29 820 564"/>
      </PlacedObjects>
    </Surface>
    <Sheet ID="Cos11" rRefs="Cos18 Cos13">
      <Surface rRef="Cos18"/>
      <Surface rRef="Cos13"/>
      <MediaSource Class="sheet of paper">
        <Media ID="Cos12" Dimensions="842 1191 842 1191"/>
      </MediaSource>
    </Sheet>
    <RunList ID="Link0003" Class="Parameter" Status="unavailable">
      <Run Pages="0~17">
        <LayoutElement>
          <FileSpec FileName="panrt17a.pdf"/>
        </LayoutElement>
      </Run>
    </RunList>
    <RunList ID="Link0004" Class="Parameter" Status="unavailable">
      <Run Pages="0~4">
        <LayoutElement>
          <FileSpec FileName="Layout.pdf"/>
        </LayoutElement>
      </Run>
    </RunList>
  </ResourcePool>
  <ResourceLinkPool>
    <RunListLink rRef="Link0003" Usage="input"/>
    <LayoutLink rRef="Link0002" Usage="input"/>
    <RunListLink rRef="Link0004" Usage="output"/>
  </ResourceLinkPool>
  <AuditPool>
    <Created Author="Rainer's GATWriter 0.2000" TimeStamp="2000-08-21T16:50:26+02:00"/>
    <Created Author="PJTF2JDF" TimeStamp="2000-08-21T16:50:26+02:00"/>
  </AuditPool>
</JDF>
```

J.6 Conversion of PPF to JDF

J.7 Messages