ARGON PPML/PDF

presented to

PODI Technical Meeting
Orlando, Florida
November 6, 7, 8 - 2000

by

WWW.THINK121.COM

www.think121.com ▪ 657 Hite Road ▪ Harwick, PA 15049 ▪ 1 724 275 1230 ▪ 1 724 275 1277 fax

Presentation Overview

Brief History of think121

pdfExpress Overview

pdfExpress Workflow

Job Preparation

Analysis of PPML

PPML/PDF

Canopy PDF ProofStreamer

ARGON Viewer

Demonstration

Please feel free to interrupt with questions at any time!

History of think121

Prior two decades in software development (flight systems, compilers, database), typesetting, Postscript, and variable data printing.

1982 • First commercial 3rd-party PostScript driver (Unilogic/Scribe)

1990 • DataStreams: Programmable multi-platform/multi-SQL database publishing system.

1992 • Lexographic Imager: PostScript-based variable data solution.
- RIP-independent in-RIP caching of data.
- PageMaker front end.

1994 • Founded MarketPlace Direct: Information Distribution Service
- Full service (prepress, print, imaging, color, bindery, fulfillment, database)
- $0 - $3 Million in sales in four years: profitable, no debt.
- Utilized Lexographic Imager solution exclusively.

1998 • Sold MarketPlace Direct, pdfExpress development begins.
• Internal production with pdfExpress begins.

History of think121 (cont.)

1999

- pdfExpress patent applied for.
- **think121** founded.
- Introduced pdfExpress at Seybold 1999 San Francisco.
- First commercial pdfExpress production use.
- Development of pdfExpress Workflow begins.
- Canopy ProofStreamer conceived as extension of pdfExpress.

2000

- Demonstration of ProofStreamer concept.
- Expansion of commercial pdfExpress production.
- pdfExpress capabilities expanded.
- Canopy ProofStreamer product development begins.
- PPML standard released.
- Canopy ProofStreamer enters commercial use.
- ARGON Viewer is born...

pdfExpress Overview

What is pdfExpress?

- An interactive markup model, implemented as an Adobe Acrobat plug-in for Mac or PC, symmetrical across text and graphics, for defining variability in PDF documents.

- An engine for the symbolic manipulation of multiple, simultaneous PDF documents, with or without markup, for the purpose of combining, merging, layering and assembling the documents to form new PDF documents.

Key Features and Capabilities

Efficient, real-time manipulation of underlying PDF document and page structure at the Cos level.

Replacement of text with
- simple ASCII
- font-specific, symbolic character representations

pdfExpress Overview (cont.)

Replacement of marked graphics with

- external data (TIFF, JPEG, etc.)
- graphic objects embedded in other PDF files
- other PDF files

Overlay one PDF page onto another with full CTM in real-time.

Replacement and overlays happen in page Z-order.

Automatic structural sharing of like PDF content objects.

ASCII and XML script driven from command line, script files, COM or AppleScript.

Free of limits imposed by Acrobat Forms.

Designed for efficient performance and scalability.

Available as Acrobat plug-in and Adobe-license-free server.

ARGON PPML/PDF

pdfExpress Workflow

Overview

Database driven, not layout program driven.

All graphical components are created as PDF pages. Components can be created by any scriptable

- PDF-producing application, e.g. Mathematica, VTEX, InDesign, Illustrator; or
- PostScript-producing application via Acrobat Distiller, e.g., Quark; or
- PDF library such as pdfLib by Thomas Merz.

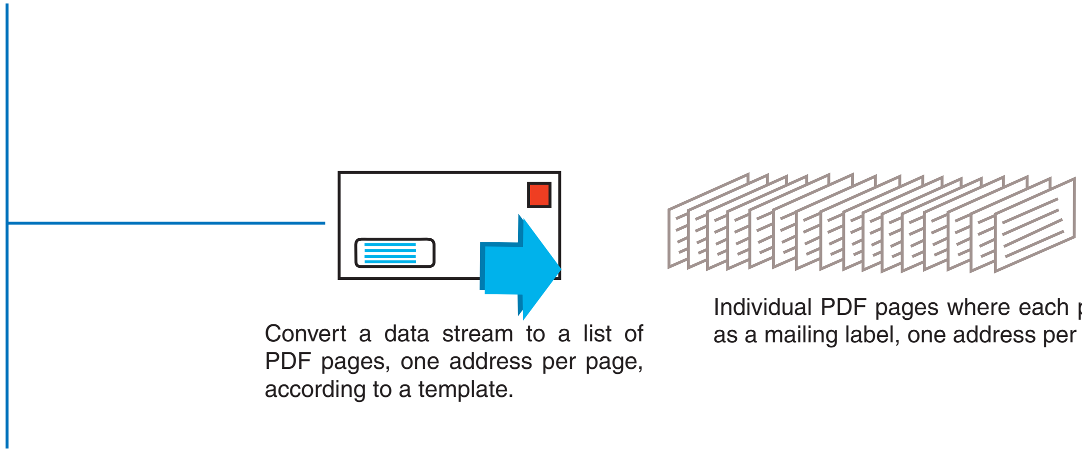
Components, pages, documents and jobs are assembled under database control.

Completed pages are defined by a base page (which may be blank) and one or more replacements or overlays on that page.

pdfExpress Workflow (cont.)

Example of Graphical Component Generation

database
"name","address","city","state",'zip",...
"Mr. J. Smith","1234 Main St.","Anytown","US","99999",...
"Ms. M. Jones","23 Grove Road","Mailville","US","99999",...

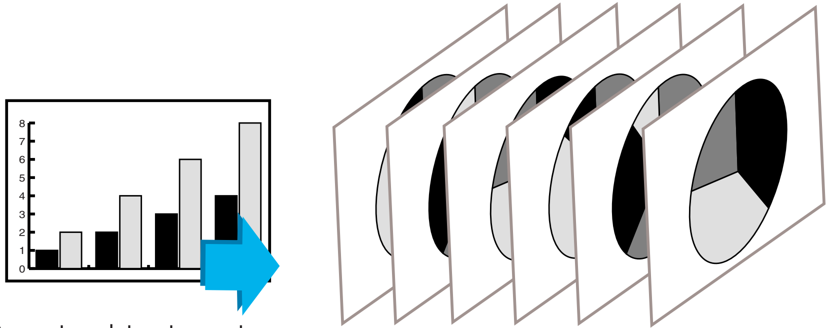


Convert a data stream to a list of PDF pages, one address per page, according to a template.

Individual PDF pages where each page acts as a mailing label, one address per page.

pdfExpress Workflow (cont.)

Example of Graphical Component Generation (cont.)

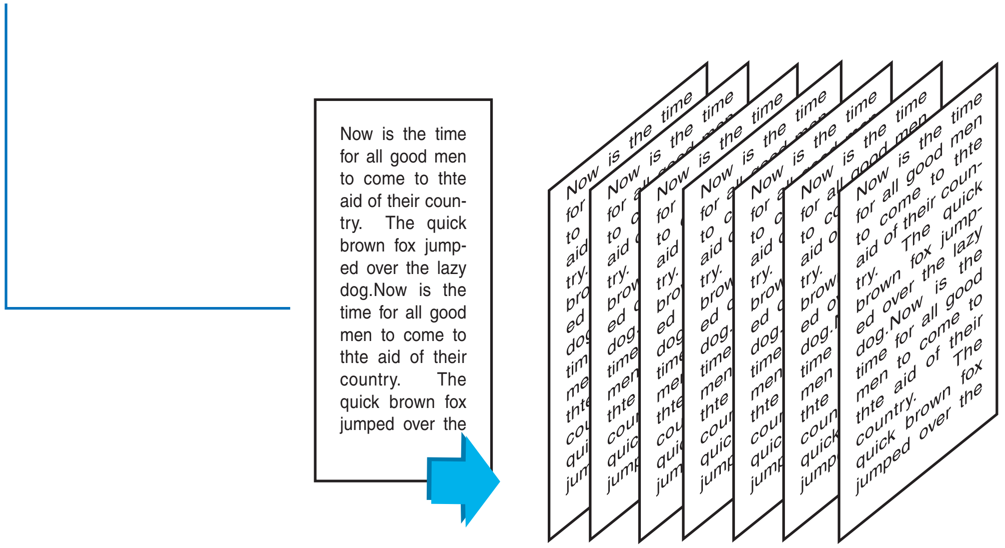


Convert a data stream to a list of PDF pages, one chart or graph per page, according to a template.

Individual PDF pages where each page carries a unique chart or graph.

pdfExpress Workflow (cont.)

Example of Graphical Component Generation (cont.)

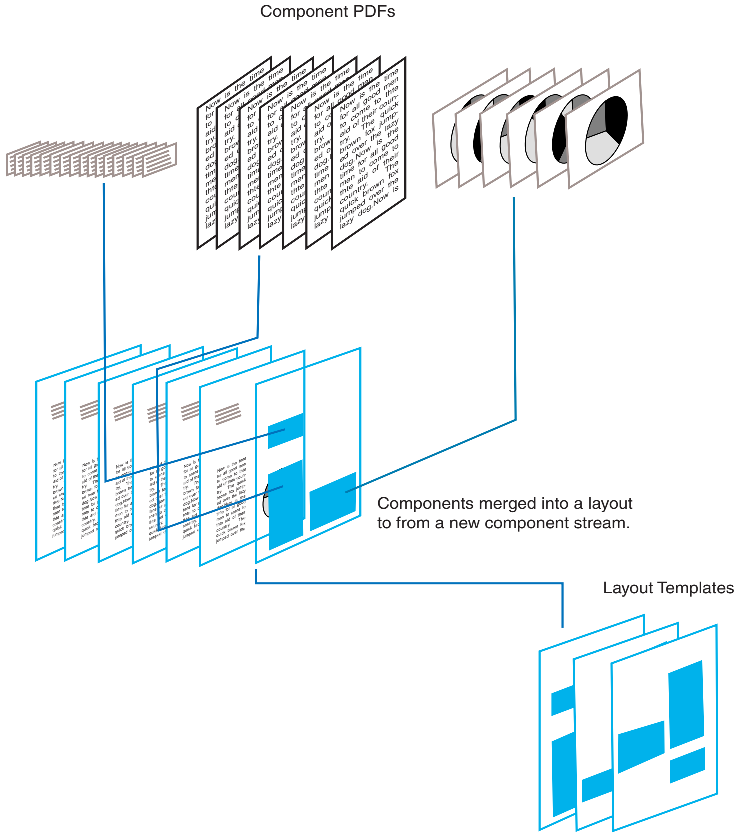

Now is the time for all good men to come to thte aid of their country. The quick brown fox jumped over the lazy dog.Now is the time for all good men to come to thte aid of their country. The quick brown fox jumped over the

Convert a data stream to a set of PDF pages where each page acts as a galley of text. Page structure, layout, and style are controlled from a data stream or template.

Individual PDF pages each representing a galley of text.

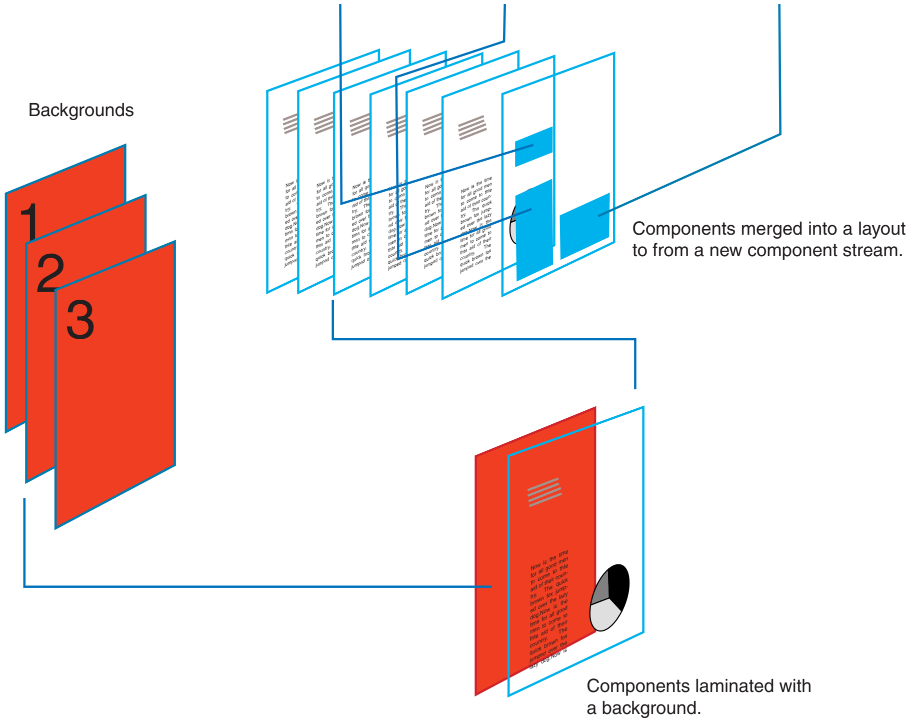
pdfExpress Workflow (cont.)

Example of Page Assembly


Component PDFs
Components merged into a layout to from a new component stream.
Layout Templates

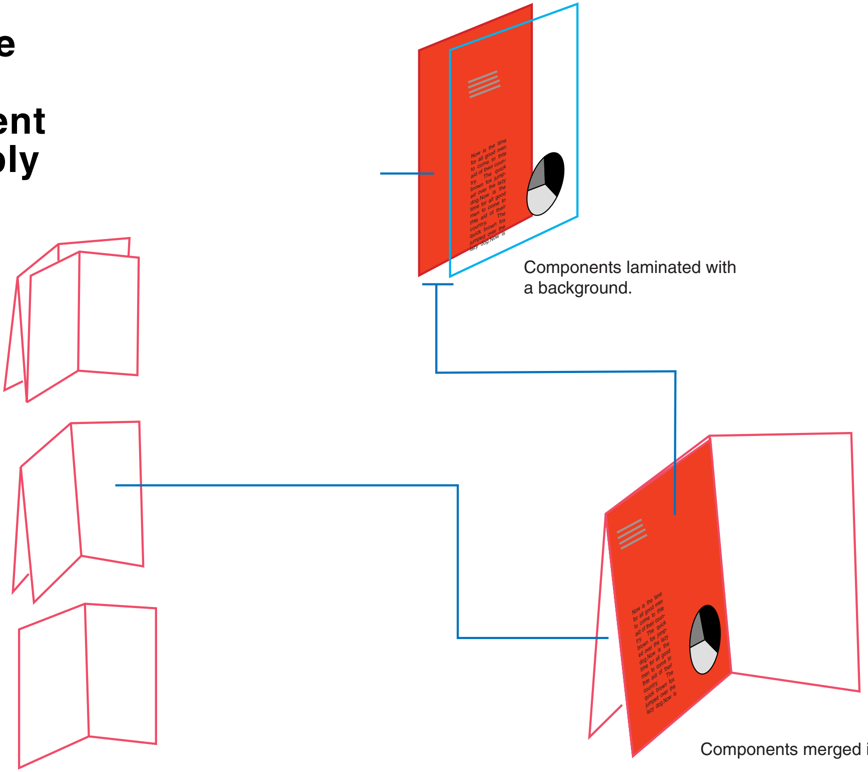
pdfExpress Workflow (cont.)

Example of Page Assembly (cont.)


Backgrounds
1
2
3
Components merged into a layout to from a new component stream.
Components laminated with a background.

pdfExpress Workflow (cont.)

Example of Document Assembly



Components laminated with a background.
Impositions
Components merged into impositions.

pdfExpress Workflow (cont.)

How pdfExpress is used in the workflow...

Compositing PDF objects for

creating page layout,
improving overall RIP time,
simplifying file management on the RIP, and
reducing overall job component count.

For previewing and preflight to

check graphic alignment,
create composites for customer review and sign-off,
validate imposition schemes, and
validate merged data.

Merging data.

Job Preparation

Job Preparation Overview

Set-Theoretic Job Decomposition - Creating the smallest unique set of graphical job components necessary based on the data.

Evaluating the Assembly Cost - Determining the optimal assembly instructions for the decomposed job.

Assembling the Job - Creating the appropriate scripting to run the job.

Impacting Job Throughput - How job decomposition and RIP characteristics interact.

Job Preparation (cont.)

Set-Theoretic Job Decomposition

1. All graphical components are decomposed into a set of uniquely identifiable objects. Uniqueness is determined as necessary by content, size, position, CTM, physical assembly, delivery or finishing requirement. Uniqueness is job specific.

2. Each object (graphical component, layer, page, document, imposition and run) is assigned a unique identifier (or key).

3. Variable text, variable images, and static components are associated with layers.

4. Layers are aggregated into collections of pages.

5. Unique documents are aggregated from collections of pages.

6. Impositions and runs are aggregated from collections of documents.

Job Preparation (cont.)

Set-Theoretic Job Decomposition (cont.)

7. In order from layers to pages to documents to impositions to runs each object is associated with a set $\{t_1, t_2, t_3, \ldots, t_n\}$ such that t_1 is the unique identifier for the object and t_2 through t_n represents the identifiers of components that make up that object. For example,

$$
\begin{aligned}
&layer_6 = \{ 6, gc_1, gc_5, gc_{10}, \ldots, gc_n \} \\
&pg_4 = \{ 4, layer_3, layer_{20}, layer_{31}, \ldots, layer_n \} \\
&doc_{134} = \{ 134, pg_{12}, pg_{14}, pg_{307}, \ldots, pg_n \} \\
&imp_{12} = \{ 12, doc_{1123}, doc_{2078}, doc_{3088}, \ldots, doc_n \} \\
&run_1 = \{ 1, imp_{12}, imp_{13}, imp_{14}, \ldots, imp_n \} \\
&job_1 = \{ 1, run_1, run_2, run_3, \ldots, run_n \}
\end{aligned}
$$

8. Each set of data values in the database associated with an object is assigned a unique identifier. These identifiers are linked to the appropriate graphical object. For example,

$$gc_6 = \{ data_{17}, data_{53} \}$$

Job Preparation (cont.)

Set-Theoretic Job Decomposition (cont.)

9. The tree-structured, set association of graphical components and data is traversed to form a collection of 3rd normal database tables where each object (graphical component, layer, page, etc.) is given its own table and each row in the table is identified by the key for that object, i.e., a table of all pages, a table of all documents, etc.

Properties of a Set-Theoretically Decomposed Job

- Third normal form relational database *construction tables* that carry, for each type of object (layer, page, etc.), the data values and sub-object references needed to construct that object.

- A complete collection of graphical layers (PDFs, PostScripts, etc.) that contain all the graphical information about the job.

Job Preparation (cont.)

Evaluating Assembly Cost

An assembly cost is assigned to each type of graphical component and layer from simplest (graphical component) to most complex (job), table by table.

Assembly cost is intimately related to the decomposition process used to identify low-level graphical elements.

At a minimum, this cost must include RIPing, RIP assembly and compositing.

- RIP cost is the amount of time it will take to RIP the specific object including start-up and shutdown of the RIP.
- RIP assembly cost is the amount of time it will take the RIP to layer the RIPed subcomponents (if it can).
- Compositing cost is the time to assemble the specific object with pdfExpress.

Job Preparation (cont.)

Evaluating Assembly Cost (cont.)

Some sample cost calculations could include:

$$\text{RIP Cost}(\text{obj}_1) = \text{RIP}_{\text{startup}} + \text{RIP}(\text{obj}_1) + \text{RIP}_{\text{terminate}}$$

$$\text{RIP Cost}(\text{aggregated obj}_{1..5}) = \text{RIP}_{\text{startup}} + \text{RIP}_{\text{terminate}} + \sum_{n=1}^{\text{number of objects}} \text{RIP}(\text{obj}_n)$$

$$\text{RIP Assembly Cost }(\text{page}_1) = \text{RIP}_{\text{startup}} + \text{layer}_1 + \text{layer}_2 + \text{layer}_3 + \text{RIP}_{\text{terminate}}$$

$$\text{Compositing Cost }(\text{page}_1) = \text{EXE}_{\text{startup}} + \text{layer}_1 + \text{layer}_2 + \text{layer}_3 + \text{EXE}_{\text{terminate}}$$

Job Preparation (cont.)

Assembly Cost Example

Assuming

$\mathrm{RIP}_{\mathrm{startup}} = 3$ sec

$\mathrm{RIP}_{\mathrm{terminate}} = 3$ sec

$\mathrm{RIP}(\mathrm{obj}_n) = 1$ sec (avg.)

Number of objects to assemble = 1,000

then

Processing as 1,000 single, RIP objects yields a cost of

$$1{,}000 * (\mathrm{RIP}_{\mathrm{startup}} + \mathrm{RIP}(\mathrm{obj}_n) + \mathrm{RIP}_{\mathrm{terminate}}) = 7{,}000 \text{ sec}$$

Processing a single, 1,000 element aggregated RIP object yields a cost of

$$\mathrm{RIP}_{\mathrm{startup}} + (1{,}000 * \mathrm{RIP}(\mathrm{obj}_n)) + \mathrm{RIP}_{\mathrm{terminate}} = 1{,}006 \text{ sec}$$

Job Preparation (cont.)

Evaluating Assembly Cost (cont.)

By conducting a RIP-based static analysis of the constructed tables it is possible to determine how to optimally assemble a job *given a particular production environment, i.e., output device*.

Facts about evaluating the assembly cost for a given job.

- Optimal assembly cost is always RIP specific. *Therefore, optimal reuse of objects is RIP specific.*
- Assembly cost is a nonlinear, discontinuous function of the specific job data, RIP characteristics, and compositing characteristics. *Therefore, assembly cost does not scale linearly.*
- For nontrivial jobs, an algorithmic approach to determining the optimal assembly, much like a database query or compiler optimizer, yields the most effective results *for the novice*.

Job Preparation (cont.)

Assembling the Job

Creation of any assembly script, whether with PPML, BTF, manually, etc., can be optimally accomplished by algorithmically combining the traversal of the construction databases with the RIP-based static analysis.

Scripts typically assemble layers into pages, documents, impositions and jobs.

Consequences of this model:

- No single assembly script is universally optimal.
- Assembly script optimization hints and calculations of reusable objects for a given RIP will impact performance when used with another RIP.
- Mistakes in characterizing a job, whether in decomposition or analysis, can negatively impact performance by orders of magnitude.

Job Preparation (cont.)

Impacting Job Throughput

- We consider a job preparation successful if it takes 10% to 20% of the print production time to prepare a job (database -> RIP -> paper coming out of the device), e.g., 1 to 1.5 hours of preparation per 8 hours of production.

- Careful planning for and execution of the job preparation step often yields several orders of magnitude of improvement in production time.

Analysis of PPML

Based on our Job Preparation analysis, we see PPML as

- an adequate data exchange model for variable data, and
- potentially containing extraneous or useless optimization information.

Additional PPML issues include:

- The loose nature of reusable objects, views, and scoping allow an infinite number of PPML job representations where one will do.
- Devices consuming PPML from unknown sources will have to recompute reuse information.
- Operations that modify global scope will have to be protected with a Java-like “sandbox” to prevent inadvertent destruction of customer data.
- CLIP_RECT’s interaction with respect to TRANSFORM is ill-specified.

PPML/PDF

PPML benefits from exclusive use with PDF in several ways:

1. Data representation in PPML is limited to integral PDF files.
2. Whole PDF pages become first class objects.
3. Graphical representation is platform and PPML independent.
4. Layering PDF's provides a relatively simple, well-defined graphical interaction model for marking a page.
5. Symbolic manipulation of PDF representations can replace the need for RIPing, particularly for previewing and validation.
6. Debugging of PDF's (color model, font inclusion, etc.) on the target output device is separate from overall job construction.
7. PDF generated by commercial applications usually offers a more compact and easier to manage file representation than PostScript.

PPML/PDF (cont.)

8. PDF files may be generated by any number of existing, PDF producing applications such as Quark, PageMaker, Word, etc.

9. PPML/PDF facilitates the development of Web-based PPML consuming applications for delivery of PDF over the Internet.

ARGON PPML/PDF

Canopy PDF ProofStreamer

Canopy™
Integrators of Xeikon Technology

Mr. Bill Marchiony
Manager of Training and Consulting Services
Canopy Print

ARGON

Generic ARGON Viewer:

- Freely distributed.
- Ignores all private data.
- Validates against pure PPML/PDF V1.01 only.
- No PPML export.
- No conversion support.
- Plug-in architecture for vendor-specific features.

ARGON Vendor Specific plug-ins:

- Previews to device-supported media sizes.
- Supports vendor and device-specific private data.
- Validates against device-specific constraints.
- Exports to vendor and device-specific PPML, JDF, BTF and other formats.
- Converts vendor and device-specific PPML, JDF, BTF, etc. to generic PPML/PDF

ARGON PPML/PDF

STACE

The STACE Abstraction Model for Variable Data

A collection of “set operators” for variable data components defined within a PPML/PDF context.

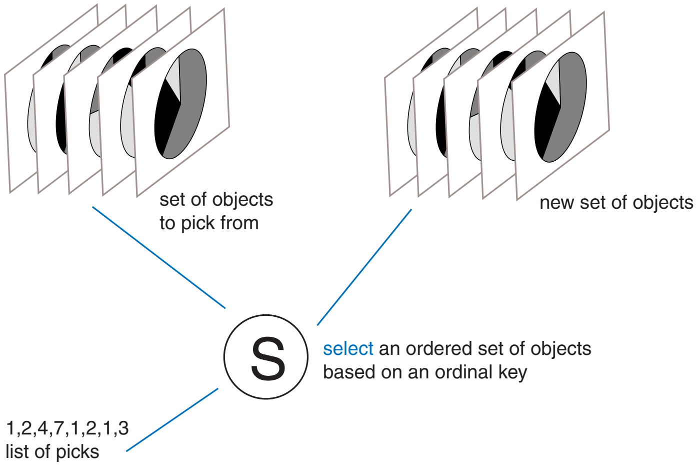
STACE sets are ordered collections of PDF pages.

STACE provides five low-level operations with which a user (either a human or program) can manipulate sets of PDF pages (STACE sets) for the purpose of preparing a variable data job. Some of these operations require out-of-band information to be carried within the PDF’s.

The representation of the inputs and result of these operations is valid PPML/PDF and valid STACE sets.

STACE (cont.)

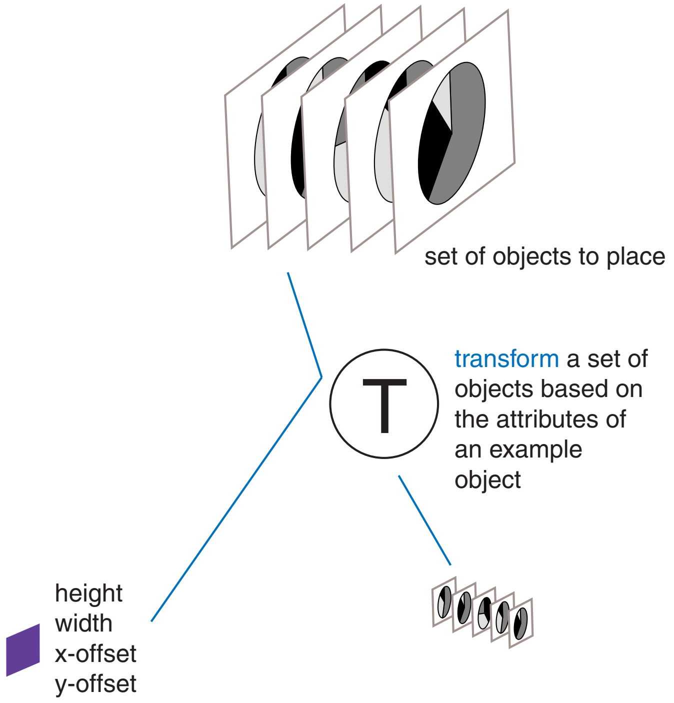
The first operation is SELECT. Select lets the user apply an ordered set of page numbers (range 1..maximum number of pages in the STACE set) to the STACE set in order to produce a new set of pages. The new set of pages makes a new STACE set.



ARGON PPML/PDF

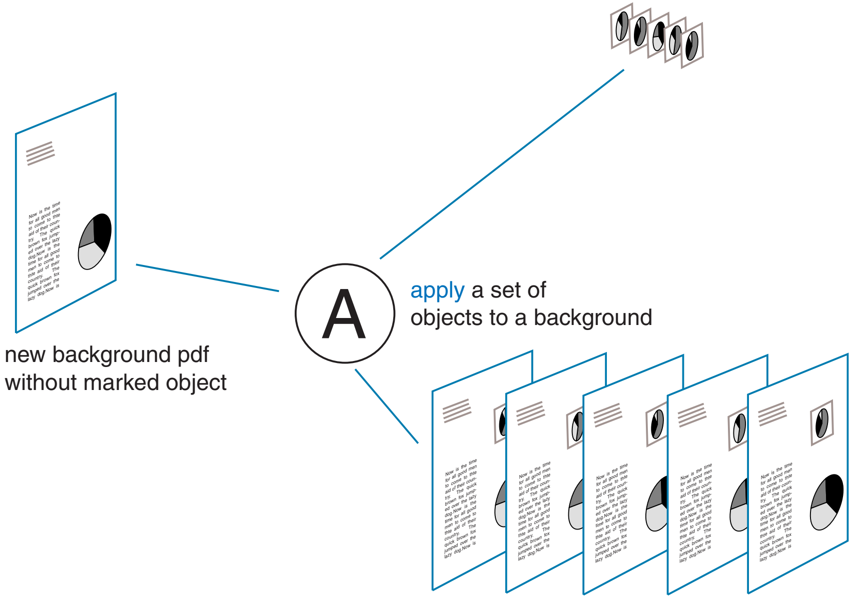
STACE (cont.)

The TRANSFORM operation allows a STACE set to be transformed through a CTM and clipping space into a new STACE set. The description of the transformation can be explicit, i.e., input by a user, or implicit, i.e., taken from an example object obtained via EXTRACT (see below). Information about the example object is transferred to the elements in the transformed set.


set of objects to place
T
transform a set of objects based on the attributes of an example object
height
width
x-offset
y-offset

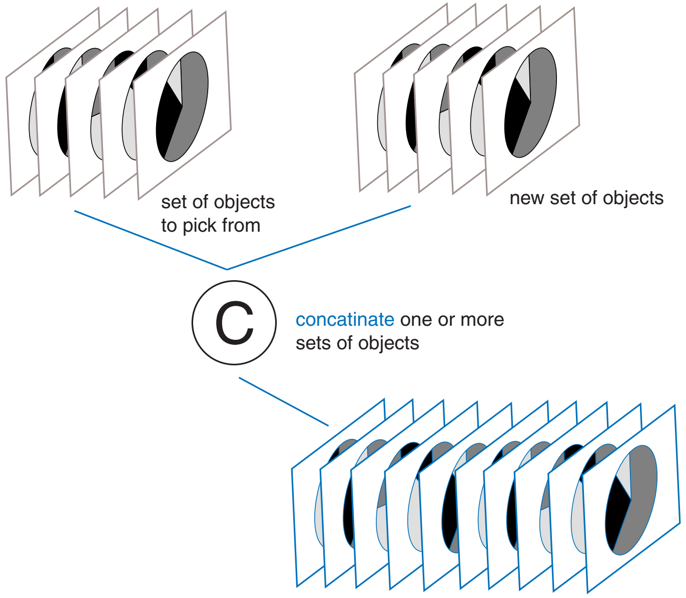
STACE (cont.)

The APPLY operation takes EXTRACTed, TRANSFORMed, or user-supplied elements and “laminates” them onto an existing STACE set element. The position of the element is determined either by its origin or can be explicitly defined by the user.



STACE (cont.)

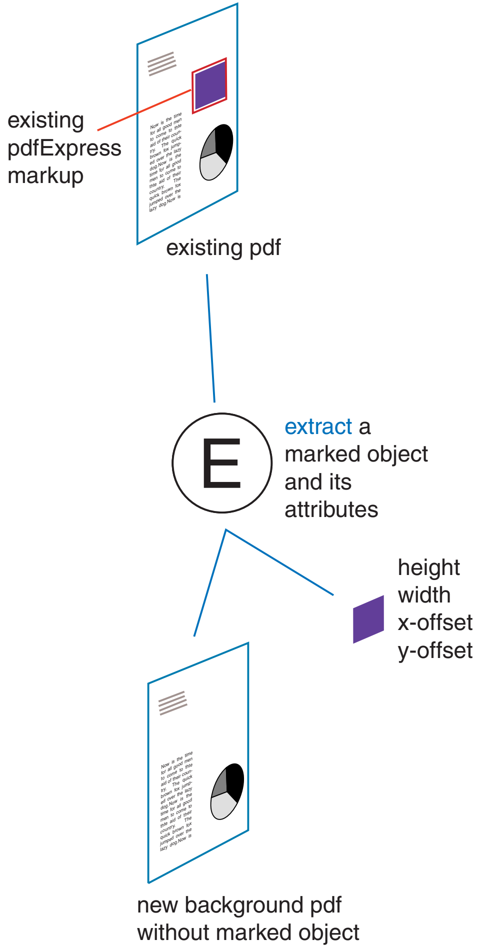
The CONCATENATE operation allows two or more STACE sets to be appended in order. The concatenation process does not affect any content information carried by the participants.



STACE (cont.)

Finally, the EXTRACT operation allows the user to select an example graphic arts object (TEXT or BITMAP) from an existing element in a STACE set element and remove it from that element.

Extract creates a new set consisting of the single element. This element appears as the original except that the graphic arts object has been removed.


existing pdfExpress markup
existing pdf
E
extract a marked object and its attributes
height
width
x-offset
y-offset
new background pdf without marked object