PPML

Personalized Print Markup Language

for XML-based, efficient printing
of documents with reusable and variable content

Functional Specification

Version 2.1

July 31, 2002
The PPML Working Group

 http://www.podi.org

PPML

The Personalized Print Markup Language

http://www.podi.org

Feedback and Developer Participation

PODi welcomes feedback on this specification, and offers the following services to support widespread adoption of the specification:

- ### Specification Updates

  The PPML specification is distributed free of charge. If you are a developer who will be implementing the PPML standard, you should subscribe to the free PPML updates and tech note service.

  Additional PPML features are already planned, and some aspects of the specification are likely to be refined as development proceeds. The spec document itself will be updated, and technical notes will be published containing clarifications, implementation notes, and so on.

- ### Developer Support web site

  If you are a software or hardware developer interested in supporting PPML, you can register to participate in the PPML Developers discussion group. At present, there is no charge for this service.

To participate in the PPML initiative in any of the above ways, send an email to ppmlinfo@podi.org.

PODi

The Digital Printing Initiative

Web: www.podi.org

Table of Contents

Appendices

Chapter 1: Introduction

1.1 Purpose of the PPML language

This document describes the PPML (Personalized Print Markup Language) data format. The PPML format was developed by members of PODi, a market development and education initiative. Information about PODi is available at http://www.podi.org.

The main purpose of the PPML language, compared to most earlier languages, is to provide **object-level addressability** and **reusability**. More information on these features and their target applications is available in the PODi document *Introduction to PPML*.

1.2 The PPML 2.0 Architecture

PPML provides an open, XML-based architecture for digital print projects. It was first introduced to the market at the worldwide "drupa" exhibition in Dusseldorf in May, 2000, and has become the first widely-adopted print stream based entirely on an open standard.

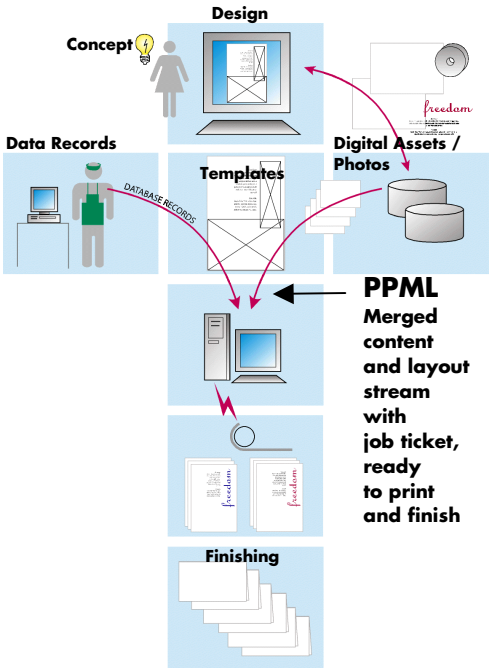
1.2.1 Additional potential

Compared to any previous print data format, PPML offers significant advantages. Examples of new applications expected in the next year include:

- **Automatic, template-based page layout.** A wide range of XML software tools are available today that can convert a stream of data into a ready-to-print stream of PPML documents. Some XML tools are free, including some that are Open Source; all can work with PPML, because it is XML.
- **Photo finishing:** Most PPML Consumers can accept photos in their native file format (JPEG or TIFF), which avoids the need for the photo system to wrap images in a container such as EPS. This significantly simplifies workflow and speeds production. Similarly, for applications that require printing the same photo repeatedly on a single sheet, PPML's reusability feature means the photo can be transmitted once to a digital press and imaged onto a sheet multiple times at different sizes. This saves tremendous amounts of network bandwidth, data storage, and RIP time, with no sacrifice in quality.

1.2.2 Basic PPML Workflow

PPML can be generated by any process, automated or operator-controlled. Its natural affinity for data-driven applications means the workflow concept shown here is a common PPML application:



- The project concept is converted to a page design template by an operator at a workstation. This may be done using graphical tools or by creating logical expressions in a templating language.
- To create a print run, data records and digital assets (such as photos) are blended using the template. The result is a stream of fully marked-up PPML documents.
- The PPML is fed to a digital print system, which processes the pages, prints the documents, and (in suitably equipped systems) feeds them to automated finishing equipment.

In addition to being part of an open workflow, the PPML specification is format-neutral, allowing content data to be supplied in any format that a machine supports. As such, it is not limited to the graphic arts or any other application segment, and its design can be extended in response to new opportunities and applications that are recognized by member companies and PODi management.

Since version 1.0, PPML has been extended beyond being a content stream. PPML 2.0 provides a complete workflow architecture:

- **Device-independent document content.** Documents can be encoded into PPML without knowledge of the specific device that will print them.
- **Open to all content formats.** PPML does not specify content format; it provides metadata about document structure and layout. Thus, it is immediately adaptable to any new application that may arise that uses a different content format from those previously associated with digital print. Among other things, this means any new PPML-based print system can easily be driven by all PPML-producing software, even if the new system is in a market that's not normally associated with digital print.
- **Device-independent job ticketing.** Common processing parameters such as media selection, RIPping parameters, and finishing instructions can be inserted into the PPML stream without knowledge of the specific device that will print them. The open PPML ticketing architecture allows this to be done using the JDF standard or other ticket formats. These features are defined in the separate document *PPML Job Ticketing*, also available from PODi.
- **A design for packaging the job content, layout, and job ticket for reliable transport.** An appendix to this specification defines rules for creating a PPML print project (job content, layout, and job ticket) on one system and transporting it to another, where it can be unpacked and printed reliably, even in cross-platform applications.

Because the entire PPML architecture is XML-based, all of the content, structure, and job ticket data in a PPML 2.0 project can be generated, manipulated, extracted, subsetted, and processed in any way that is supported by common XML data tools. In addition, metadata and other types of non-printing content can be embedded in PPML through the use of the XML namespace mechanisms. This sort of flexibility and versatility has never before been available in a print stream, illustrating the power of the PPML design.

For more information, contact PODi at info@ppml.org.

1.3 Organization of this document

This document reflects the hierarchical structure of PPML data.

Chapter 1 is this introduction.

Chapter 2 discusses the data format: XML.

Chapter 3 introduces terminology: the anatomy of a PPML document, job, etc.

Chapter 4 then presents the structure of PPML data, down to the level where documents are composed of pages.

Chapter 5 presents the make-up of the PPML page, including "Objects" and "Marks," the printable page image elements that go onto pages. The language features in this chapter are the source of the power of the PPML language.

Chapter 6 presents the Print Layout elements: page size, sheet size, imposition, step and repeat.

Chapter 7 discusses Production Specifications: aspects of how the finished document is "manufactured."

Chapter 8 discusses Resources – the additional items such as fonts that are required for production of the pages.

1.4 Notation used in this document

The following typographic notation is used in this document.

- Code excerpts, element names, and attributes: `Letter Gothic`
- The vertical bar character signifies the logical OR operator: |
  For instance, `"SOURCE | OCCURRENCE_REF"` means
  `"SOURCE` or `OCCURRENCE_REF".`
- Because many PPML element names are common English words, it is often convenient and accurate to use them conversationally. In this document, when an element name appears in text *not* in a monospaced font, but with Initial Capitals, it is specifically referring to the PPML item that bears that name. When it appears with no capitalization, the word is being used with no special PPML significance. Example:
  The `SOURCE` element contains one or more component files.

In an `OBJECT` element, the Source may contain data in any of several formats. Customers may submit image data that was gathered from a number of different sources.

- In tables of XML attributes, when the data type is Number or Integer, a multiplication sign indicates a string of numbers separated by spaces. For instance, "Number × 4" indicates that the value of the attribute should be four numbers, such as "`1.234 2.0 3 4.567`."

1.5 Additional resources

See the PODi web site, http://www.podi.org, for additional documents about PPML and personalized printing in general.

1.6 Feedback

Feedback on this specification is welcome. Send email to ppmlinfo@podi.org.

Chapter 2:
The PPML Data Format

2.1 XML

PPML is an application of XML, the Extensible Mark-up Language.

2.1.1 Introduction to XML

Data objects in an XML stream are called elements, and each type of element can be defined as having certain attributes. This specification defines the elements for the PPML data format, the hierarchy requirements for the structure of a PPML document, and the attributes for each element.

Readers who are not yet familiar with XML are directed to these resources:

- Appendix 2 of this document is a brief description of how XML works.
- XML.ORG (http://www.xml.org) is an industry web portal operated by OASIS, the Organization for the Advancement of Structured Information Standards.
- OASIS's "The SGML/XML Web Page" (http://www.oasis-open.org/cover/sgml-xml.html) contains many excellent links to reference information.
- "The XML.commune" (http://www.xml.com) is a collaborative partnership between Seybold Publications and Songline Studios, an affiliate of O'Reilly & Associates. The site includes Tim Bray's excellent annotated version of the XML syntax recommendation.
- Project Cool XML Zone (http://www.projectcool.com/developer/xmlz/) is one of the best sites for developers, with a fairly good introduction to the basics of XML.

2.1.2 Notation for specifying optional elements

Within one XML element, other elements may be required or optional. In standard XML syntax notation optional elements are denoted by placing a punctuation mark next to the subordinate element:

Symbol	Meaning
?	0 or 1 (may or may not be present)
+	1 or more (at least one is required)
*	0 or more

Example: As will be described later, the `PAGE` element may or may not contain a Required Resources section, and may contain zero or more `PRIVATE_INFO` elements and zero or more Marks. This structure would be denoted:

```
PAGE (REQUIRED_RESOURCES?, PRIVATE_INFO*, MARK*)
```

This notation, with the "child" elements enclosed in parentheses, is sometimes referred to as the element's model.

2.1.3 PPML Capitalization conventions

In XML, the names of elements and their attributes are case-sensitive, so capitalization is significant in the code examples in this document.

The PPML capitalization convention is:

Element names: `ALL_CAPS_WITH_UNDERSCORE_BETWEEN_WORDS`.

Attributes: `TitleCase`, with no space between words.

Example of a `DOCUMENT_SET` tag with attributes "`Name`" and "`DocumentCount`":

```
<DOCUMENT_SET  Label="MyJob"  DocumentCount="150">
```

2.1.4 DTD and Schema

PPML is specified both in a Document Type Definition (DTD, http://www.w3.org/TR/1998/REC-xml-19980210#dt-doctype) and in an XML Schema (http://www.w3.org/XML/Schema). All versions of the DTD and Schema will always be available at http://www.podi.org/ppml.

Each DTD will have a ".dtd" suffix and each XML Schema will have a ".xsd" suffix. Each DTD and Schema will be named "ppmlXXX" with the version number extended to two decimal places minus the decimal point replacing the "XXX". The DTD for PPML 2.1 is named ppml210.dtd and the corresponding XML Schema is named ppml210.xsd.

XML processors wishing to validate a PPML document may do so by incorporating a DOCTYPE declaration or by specifying an XML Namespace (http://www.w3.org/TR/REC-xml-names/) in the top level, PPML, element.

For example, to validate a PPML Version 2.1 document by means of a DOCTYPE declaration, one would include the following XML markup immediately following the XML declaration and any intervening whitespace:

```
<! DOCTYPE PPML PUBLIC
    "-//PODi//DTD PPML 2.10//EN" "http://www.podi.org/ppml/ppml210.dtd">
```

Some XML processors, when validating against an XML Schema, require additional information beyond the namespace declaration. To enable validation on the widest variety of XML processors, a conforming Producer must also include the attributes `xsi:schemaLocation` and `xmlns:xsi`, as in these examples:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.podi.org/ppml/ppml210.xsd"
```

Notes:

1. The value of `xsi` must be the URI to the XML schema instance definition, which is exactly as shown above.
2. The value of `schemaLocation` must be the location of the schema as specified by the Producer, and must be accessible to the Consumer. Typically the value will be the URL to the PPML schema on the PODi web site, as shown above. But standard XML practice allows caching a schema at some other location, in which case that location should be used as the value of schemaLocation.

To validate a PPML Version 2.1 document by means of an XML Namespace declaration, one would start the actual PPML element with:

```
<PPML xmlns="http://www.podi.org/ppml/ppml210.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.podi.org/ppml/ppml210.xsd" ...
```

or:

```
<ppml:PPML xmlns:ppml="http://www.podi.org/ppml/ppml210.dtd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation=http://www.podi.org/ppml/ppml210.xsd
```

where "ppml" in the strings "xmlns:ppml=" and "ppml:PPML" above is an arbitrary NMTOKEN (http://www.w3.org/TR/1998/REC-xml-19980210#NT-Nmtoken).

A valid PPML document must include one of these methods of version identification.

2.1.5 Character sets

PPML elements may contain characters as defined in the XML specification at http://www.w3.org/TR/REC-xml#charsets.

2.2 Non-XML data

2.2.1 Introduction

Non-XML data is an important part of printing. For instance, it is used for PostScript code fragments, bitmap images and compressed data such as fonts. But as of the writing of this specification, XML elements cannot readily incorporate such data.[1,2]

A process is underway in the XML movement to solve this, but results are not expected within the timeframe required by the PPML initiative. When a standard XML solution for non-XML data has been announced, the intention is that PPML will support it. In the meantime another approach is required.

In any event, this topic has no bearing on the central issues of how PPML defines pages and documents that have reusable content; it relates only to how the various data objects are "packaged" for transport from the Producer to the Consumer.

[1] The XML specification states "CDATA sections may occur anywhere character data may occur; they are used to escape blocks of text containing characters which would otherwise be recognized as markup. CDATA sections begin with the string "<![CDATA[" and end with the string "]]>."

[2] Another application dealing with this issue is medical imaging. See http://www.xml.com/xml/pub/98/07/binary/binary.html.

2.2.2 External references

This method includes no non-XML data within the XML stream; rather, it is kept in external files and pointed to as external resources. Example:

```
<EXTERNAL_DATA  Src="ftp://uc.wisc.edu/logo.eps" .../>
```

2.2.3 Wrap the non-XML data and the XML structure, in segments, in MIME as a means of transporting the dataset in a single stream.

Some applications will always absolutely require in-stream non-XML data. Other applications may include hundreds or thousands of single-use images, which would be a nuisance to store online for access via external reference; it's simpler to just download the data within the job stream, print it, and throw it away. Clearly, therefore, a method is needed to embed non-XML data in the data stream.

When the W3C solution arrives it will be possible to send a stream of PPML elements interspersed with non-XML data. The Consumer will receive the stream, extract the non-XML data and deal with it, and parse and handle the XML segments.

Today, the same thing could be done by using MIME as an encoding filter:[3]

- The Producer, instead of putting the additional data between start and end tags, inserts MIME separators between segments.
- The Consumer, seeing that the start of the stream is MIME, not `<?xml version="1.0"?>`, unpacks the MIME pieces and reassembles the XML and non-XML pieces.
- Processing then proceeds the same as it will when the W3C solution is implemented.

This method has the advantage of already being permissible in XML.

[3]Resources about MIME:

- RFC 2557 (MIME E-mail Encapsulation of Aggregate Documents, such as HTML (MHTML)) http://www.landfield.com/rfcs/rfc2557.html shows how to encapsulate HTML and external data; the same method is valid for XML.
- RFC2393 (Content-ID and Message-ID Uniform Resource Locators) http://www.landfield.com/rfcs/rfc2392.html defines cid URIs. This method is used when you email a complete web page to someone, instead of just the URI.
- RFC 2387 (The MIME Multipart/Related Content-type) describes the MIME method that makes sense for PPML applications: "Several applications of MIME … require multiple body parts that make sense only in the aggregate."

Chapter 3:
Terminology and Basic Concepts

3.1 Producers and Consumers

- A "PPML Producer" (or simply "Producer") is anything that generates PPML files. This may be a standalone application, a system-level driver, or anything else.
- A "PPML Consumer" (or simply "Consumer") is typically a RIP or DFE (digital front end to a digital printing device), but it may be any other device (or process or system) that reads and interprets PPML files. PPML Consumers only differ in these regards:
  - the data formats they can process in `SOURCE` elements (see section 5.8)
  - their degree of imposition support (See section 6.1.1 for details).
  - the data encoding formats they support. The only required encoding format is Base64.

Note that a PPML Consumer may also be a PPML Producer. For instance, an application could read PPML files, interpret their contents, modify the content or structure, and produce new PPML files.

3.2 Anatomy of a Personalized Print project

- **Project** is all activities involving both the initial setup phase and the subsequent production runs. A Project is an on-going activity, consisting of multiple Jobs, as opposed to a conventional print job which is typically produced once and archived.
- **Job** is the collection of activities and data to fulfill a single personalized printing work order, or to prepare the templates, objects, etc. that will later be used in fulfilling production work orders. In personalized printing, a Job is part of a Project.
- **Page** is a single side of a trimmed sheet after all trimming and binding has been performed. Some personalization projects produce documents that have multiple Pages, others (e.g. a single-sided postcard) produce documents of only one Page. A Page consists of static and/or varying Objects, each in a specified position and orientation. ("Page" can also refer to the internal representation of a Page in a PPML file.)
- **Content Data** is source data (e.g. a picture, a text block, an EPS file) which may be placed on various Pages in various combinations of scaling, position, rotation, etc. A piece of Content Data may be used by more than one Object.
- **Object** is a discrete piece of Content Data in a specific combination of scaling, rotation, etc. Objects may be **Disposable** (single-use, RIP once and discard) or **Reusable**.
- **Instance Document** is the end result of the PPML manufacturing process: a set of one or more Pages, bound or loose, produced from a single record in the variable-data file. (This term also refers to any representation of such a document, such as an on-screen preview.)

- **Personalized Document:** an Instance Document.
- **Static Document:** a print job that contains no data-driven content – specifically, the page content is not generated from variable-data fields.[4] Such a document may include one or more Reusable Objects, such as a PowerPoint background or a forms overlay.
- **Sheet surface** is one side of a press sheet, typically containing one or more instance Pages, imposed into position for manufacturing of an Instance Document.
- **Template** is the set of instructions for composing Personalized Documents. It defines which Pages may be in an Instance Document, what goes on each Page, and the logic rules by which each Page will be populated in response to the variable data.
- **Dataset:** a PPML element, typically containing one or more Jobs and/or Reusable Object definitions and related elements required to process them.

3.3 Additional terminology

- **RIP:** a Raster Image Processor – a hardware device or software application that reads a source file in a particular language and converts it to a raster – a pattern of scan lines in a data format that is suited to the machine on which the printing will take place.
- **Pre-flight:** a procedure, automated or manual, that is applied to a print file in the graphic arts to ensure that when production begins, all output will be as expected. This includes checking that all required resources are available. (The term "pre-flight" alludes to an aircraft pilot's pre-flight checklist, which is a process intended to ensure that nothing is overlooked before the flight begins.)
- **Streaming:** a type of print application in which the Consumer begins output of the job before it has received all pages.

  Typically streaming applications are found in long-run transactional printing applications, such as printing of thousands of phone bills, not in the graphic arts. Usually this means the job has tens of thousands of pages, essentially infinite, and it means that the resource usage is unknown when output begins.

  A single streaming job can occupy a printer continuously for days printing unique pages. To avoid data underrun (where the RIP processing doesn't keep up with the print engine and the engine either stops or prints blank pages), the Consumer system must consider carefully how many resources and pages to preprocess before the stream output begins so that in the steady state, RIP processing does keep up with the print engine.

  Effective management of cache resources requires reliable information about the *future* need for a given cached object; in streaming applications, that information is typically not available when output begins.

[4] Looking at the PPML code for a document, it's not possible to tell whether or not it was generated from variable data fields. (The PPML format exists at the point in the workflow where all page content decisions have already been made.) It's nonetheless worth defining this term because such documents can be a valid PPML application if they contain reusable objects.

3.4 Terms related to PPML Job Ticketing

The following terms appear in this document, specific to the PPML Job Ticket.

Job Ticket: the data required to produce a set of printed documents, beyond the document content and layout specified in PPML.

PPML Job Package: one PPML dataset plus its accompanying PPML Job Ticket. Together, these specify the ticket, layout, and content required for print production.

Ticket State: The state of a PPML Consumer relative to all parameters that can be controlled by a Job Ticket. See section 4.11 for more information.

3.5 Detection of Errors

When an error is detected with the information in a PPML file, the behavior of the Consumer is not specified. Some Consumers may halt the job at that point. Others may generate a warning message, ignore the offending PPML, and proceed as best they can.

Chapter 4:
The Structure of PPML Data

4.1 Hierarchy, Scope, and Inheritance

4.1.1 PPML is Hierarchical

PPML is a hierarchical structure, in which the properties and resources of an element are inherited from its enclosing ("parent") structure. The contents of the child element may temporarily override (or mask) the parent's properties and resources; when the child element ends, the previous state is restored.

- A PPML element (the highest level) can contain resource definitions and DOCUMENT_SET elements.
- A DOCUMENT_SET element (a set of personalized documents) can contain resource definitions and DOCUMENT elements.
- A DOCUMENT element (which prints one complete document, of one or more pages) can contain resource definitions and PAGE elements.
- A PAGE element can contain resource definitions and MARK elements. MARK elements are what actually cause page content to be printed onto a page, using ink or toner.

PPML, DOCUMENT_SET, DOCUMENT and PAGE are known as **levels** in the PPML hierarchy.

- A MARK element (which places image marks on a page) can contain two kinds of content elements: OBJECT and OCCURRENCE_REF. (Each of these content elements contains smaller elements as well.)

4.1.2 Reusable Objects; caching

An important resource in PPML is the Reusable Object. As explained later in this specification, a reusable piece of page content is expressed as an OCCURRENCE of a REUSABLE_OBJECT element and is accessed using OCCURRENCE_REF. This construct is central to PPML's productivity improvement.

The reusability feature (enabled by elements such as REUSABLE_OBJECT and SOURCE) allows the data for a picture (or any other page content) to be sent once to the Consumer, where it can be RIPped (prepared for imaging on pages) and saved (cached) for reuse in subsequent Pages, Documents, Document Sets, and Datasets. Typically, this improves efficiency by avoiding two redundant burdens on the system: redundant downloading and redundant computation of the content's appearance. But there is no *requirement* that the Occurrence be cached; how reusability is implemented in a Consumer is not defined in the PPML language.

Caching would ordinarily improve print speed (by avoiding re-RIPping), but it is valid for a PPML Consumer not to cache but instead to regenerate the Occurrence, e.g. by re-fetching the source data and/or reRIPping the object, each time it is used in an OCCURRENCE_REF.

4.1.3 Scope

Two important attributes of Occurrences are their *Name* and their *Scope*.

The name is the mechanism by which MARK elements can place the Occurrence on a page.

The scope defines how long the Consumer must remember the Occurrence, so that the Producer can access it by name. Possible values are Page, Document, DocSet, PPML, and Global.[5] (Note that the content elements OBJECT and MARK are *not* scopes.)

When an Occurrence is *in scope* the Consumer is required to recognize the Occurrence's name and be able to use it. When the Occurrence's specified scope level ends, the Occurrence becomes *out of scope*. For instance, if an Occurrence has scope "Document", then at the end of the current Document (i.e. when a </DOCUMENT> tag is encountered) the Occurrence goes out of scope.

The scope of a PPML element defines where this element is *known*. Each named element is known within the enclosing element specified by its scope (DocSet, Document, etc), from the point where it is first defined until the end of that element.

The Occurrence can be defined with a scope larger than the current enclosing element. For instance, within a DOCUMENT element an Occurrence can be defined with Scope="DocSet". In that case the Occurrence will be known beyond the end of the DOCUMENT element, until the end of the enclosing DOCUMENT_SET element.

Any element in the hierarchy inherits the names known to its enclosing element (i.e. a PAGE knows of all elements that are defined in its enclosing DOCUMENT etc.).

Scope is discussed at greater length in section 5.10 of this document.

[5] Global objects have an additional attribute, Environment, which can be used to categorize global objects for project management purposes. See further discussion in section 5 of this document.

4.2 The <PPML> Element

4.2.1 Description

The `PPML` element is the top level, encompassing all other elements of the dataset.

4.2.2 Model

```
PPML          (CONFORMANCE*,
              TICKET?,
              SUPPLIED_RESOURCES?,
              REQUIRED_RESOURCES?,
              IMPOSITION*,
              (PRINT_LAYOUT | PAGE_DESIGN)?,
              PRIVATE_INFO*,
              (TICKET_SET | TICKET_REF | REUSABLE_OBJECT | SEGMENT_ARRAY |
                 (DOCUMENT_SET | JOB) )*
              )
```

4.2.3 Attributes

Attribute	Required /Optional	Type	Description
Label	Optional	String	An identifying label for this PPML element.
Creator	Optional	String	Identifies application or person that created the file, for instance to potentially aid in post-processing
CreationDate	Optional	String	Time stamp, in date/hours/minutes/seconds, using the subset of ISO 8601 described in the W3C's http://www.w3.org/TR/NOTE-datetime. Example: "1997-07-16T19:20:30+01:00"
ResourcesIncluded	Optional	Boolean	Values: `Yes` or `No`. If `Yes`, promises a Consumer that all referenced content data, fonts, and other resources are supplied with the dataset. See section 10.2.3, "Details of ResourcesIncluded".
SheetLayoutIncluded	Optional	Boolean	Values: `Yes` or `No`. If `Yes`, declares that this dataset includes the `SHEET_LAYOUT` element and requires that the imposition defined in `SHEET_LAYOUT` must be honored. (See Section 6.1.1 for discussion of optional imposition support in PPML). Consumers that do not support `SHEET_LAYOUT` must reject the dataset if this attribute's value is `Yes`.

4.2.4 Implementation notes

Note that a PPML dataset is allowed to *not* contain any Document Sets. A valid dataset could contain nothing but a set of Reusable Object definitions with `Scope="Global"` which are being sent to the Consumer for pre-processing and storage in the Consumer system.

4.3 The <DOCUMENT_SET> Element

> **Note**
>
> JOB is a synonym for DOCUMENT_SET: they can be used interchangeably. Similarly, in attribute values, Job is a synonym for DocSet.

4.3.1 Description

A Document Set is a set of Instance Documents. Typically an Instance Document represents the binding of layout information (e.g. a template) and a record of data (e.g. from a database).

A Document Set is a group of Instance Documents that are to be treated as a unit, perhaps because the documents are intended for a single recipient, such as a cover letter, a brochure, and a postcard. PPML does not require that Document Set must be used in this way; it is merely a convenient grouping mechanism.

4.3.2 Model

```
DOCUMENT_SET (SUPPLIED_RESOURCES?,
              REQUIRED_RESOURCES?,
              IMPOSITION*,
              (PRINT_LAYOUT | PAGE_DESIGN)?,
              PRIVATE_INFO*,
              (TICKET_SET | TICKET_REF | REUSABLE_OBJECT | SEGMENT_ARRAY |
              DOCUMENT)+)
```

4.3.3 Attributes

Attribute	Required /Optional	Type	Description
Label	Optional	String	An identifying label for this Document Set
DocumentCount	Optional	Integer	Number of Instance Documents in this Document Set. If this attribute is used, it must be accurate; if the actual document count is different, it's an error.

4.3.4 Context

The DOCUMENT_SET element appears only within a PPML element. It is optional: a PPML dataset may contain zero or more Document Sets.

4.4 The <DOCUMENT> Element

4.4.1 Description

The DOCUMENT element marks a single Instance Document. Typically an Instance Document represents the binding of layout information (e.g. a template) and a record of data from some data set (e.g. a database). Example: when printing personalized information for people on a mailing list, the Document tag delimits each individual set of pages that will be sent to one recipient on the list.

A Document may be larger or smaller than one sheet of substrate. The Document may be hundreds of pages long or one page, and in either case, each page could be any size, from a full press sheet to something as small as a postage stamp, so that many Documents could be printed on a single sheet. The term "Document" is thus not a physical term but a logical one.

The default is to print Instance Documents in the same sequence as they appear in the PPML stream, unless the Consumer is specifically instructed to do otherwise, e.g. via a REPEAT structure.

4.4.2 Model

```
DOCUMENT     (SUPPLIED_RESOURCES?,
             REQUIRED_RESOURCES?,
             PAGE_DESIGN?, PRIVATE_INFO*,
             (TICKET_SET | TICKET_REF | REUSABLE_OBJECT | SEGMENT_ARRAY |
             PAGE)+)
```

4.4.3 Attributes

Attribute	Required /Optional	Type	Description
Label	Optional	String	An identifying label for this Document.
Dimensions	Optional	Number × 2	Width and height of pages in this Document, in PPML units. Example: for a U.S. letter page, Dimensions="612 792". Use of this attribute is no longer recommended. Use the PAGE_DESIGN or PAGE_LAYOUT element instead. This attribute is ignored if a PAGE_DESIGN or PAGE_LAYOUT element is in effect. If no such element is in effect, this attribute is equivalent to <PAGE_DESIGN TrimBox="0 0 w h"/>
PageCount	Optional	Integer	Number of pages in the document. If this attribute is used, it must be accurate; if the actual page count is different, the result is an error.
DocumentCopies	Optional	Integer	How many copies to print of this Instance Document. (If the current PRINT_LAYOUT element has an NCopies attribute, the total copies printed will be NCopies times DocumentCopies.)

4.4.4 Context

The DOCUMENT element occurs only within a DOCUMENT_SET element.

4.5 The <PAGE> Element

4.5.1 Description

The PAGE element delimits the contents of each individual page in each Instance Document.

4.5.2 Model

```
PAGE          (SUPPLIED_RESOURCES?,
               REQUIRED_RESOURCES?,
               PAGE_DESIGN?,
               PRIVATE_INFO*,
               (TICKET_SET | TICKET_REF)*,
               (REUSABLE_OBJECT | SEGMENT_ARRAY | MARK)*)
```

4.5.3 Attributes

Attribute	Required /Optional	Type	Description
Label	Optional	String	An identifying label for this Page.
Dimensions	Optional	Number × 2	Width and height of this Page, in PPML units. Example: for a U.S. letter page, Dimensions="612 792". Use of this attribute is no longer recommended. Use the PAGE_DESIGN or PAGE_LAYOUT element instead. This attribute is ignored if a PAGE_DESIGN or PAGE_LAYOUT element is in effect. If no such element is in effect, this attribute is equivalent to <PAGE_DESIGN TrimBox="0 0 w h"/>.

4.5.4 Context

The PAGE element appears only within a DOCUMENT element.

4.5.5 Blank pages

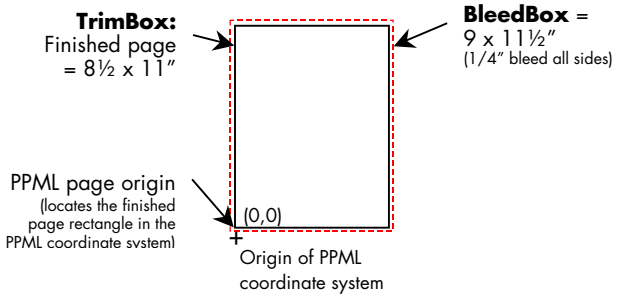
A PAGE element that does not contain any MARK elements instructs the Consumer to print a blank page.

4.6 The <PAGE_DESIGN> Element

4.6.1 Description

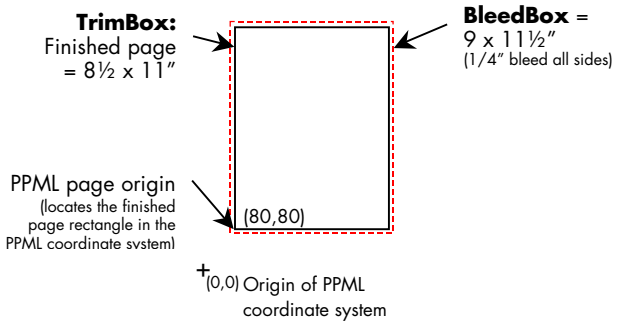
The PAGE_DESIGN element specifies the finished rectangular area of a Page as well as optional bleed box information. Examples:

```
<PAGE_DESIGN TrimBox="0 0 612 792"
    BleedBox="-18 -18 630 810"
    />
```



The same page, with its origin offset from the origin of the PPML coordinate system:

```
<PAGE_DESIGN TrimBox="80 80 692 872"
    BleedBox="62 62 710 890"
    />
```



The "Trim Box"

The required TrimBox attribute indicates the rectangular region of interest of the page design and defines the intended finished page size. The TrimBox origin is defined in the PPML coordinate system which is the same coordinate system in which all marks for the page are specified.

This information is useful to a PPML processor such as a PPML viewer, page proofer, or imposition layout tool only interested in page design definitions. An imposition tool, for example, may use the TrimBox information as the description of the intended finished page design, and use its dimensions to locate cut marks on imposed sheets as needed.

The "Bleed Box"

The optional BleedBox attribute indicates that page content extends outside of the design rectangle specified by the TrimBox attribute and recommends to a Consumer, such as an imposition processor, a preferred bleed extent.

The BleedBox attribute if specified must completely contain the rectangular region defined by the TrimBox or be equal to it.

If no BleedBox is specified then no hint is provided to the consumer of the existence of bleed edges of the intended finished page.

4.6.2 Model

```
PAGE_DESIGN  EMPTY
```

4.6.3 Attributes

Attribute	Required /Optional	Type	Description
TrimBox	Required	Number × 4	Coordinates, in 1/72", of the page content area.
BleedBox	Optional	Number × 4	Coordinates, in 1/72", of the page's bleed area.

4.6.4 Context

The `PAGE_DESIGN` element appears within `PPML`, `DOCUMENT_SET`, `DOCUMENT` and `PAGE`.

4.6.5 Page orientation

All dimensions in the attributes are to be listed in "upright" orientation. For instance, a portrait letter-size page will have `PAGE_DESIGN TrimBox="0 0 612 792"` and a landscape letter-size page will have `PAGE_DESIGN TrimBox="0 0 792 612"`. Thus, no separate Orientation attribute is needed.

Note that any page may be rotated later when it is used in imposition (see Chapter 6:). But the page itself, and its content, are independent of imposition and printing.

4.6.6 Similarity with `PAGE_LAYOUT` in imposition

`PAGE_DESIGN` expresses the designer's intent regarding the finished dimensions of the page. As described in Chapter 6, later production processes may involve placing pages onto sheets ("imposition"). The imposition may be expressed using PPML's imposition features or by using alternate imposition layout expression formats.

Note that PPML's imposition includes a `PAGE_LAYOUT` element, which appears similar to `PAGE_DESIGN` because both have a `TrimBox` and `BleedBox` attribute. The difference is that `PAGE_DESIGN` only expresses the designer's intent, in the context of the page content stream (`PPML`, `DOCUMENT_SET`, `DOCUMENT`, `PAGE`), while `PAGE_LAYOUT` defines the dimensions of page *cells* (see section 6.9) in the context of imposition (assigning pages to sheets). Therefore the `TrimBox` and `BleedBox` attributes of `PAGE_LAYOUT` require a graphical clipping behavior, and the `TrimBox` and `BleedBox` attributes of `PAGE_DESIGN` do not – they leave the determination of that behavior to the consuming PPML processor.

At least one `PAGE_LAYOUT` or `PAGE_DESIGN` element must be in effect for each Page.

4.7 The <CONFORMANCE> Element

4.7.1 Description

The optional CONFORMANCE element declares that the enclosing dataset conforms to a specific PPML subset. (See Chapter 10: Conformance Subsets.) The model allows multiple CONFORMANCE elements, since it's conceivable that a future dataset could conform to more than one subset.

This element occurs at the start of the model for the PPML element so that a Consumer can know, from the very beginning, that nothing in the dataset exceeds the restrictions of a defined subset.

The CONFORMANCE element simply informs the PPML Consumer that the dataset meets the subset's requirements. The Consumer may use this information, but is not required to do anything with it.

Parties who wish to define a subset should register their desired Subset string with PODi. Write to ppmlinfo@podi.org for information.

4.7.2 Model

```
CONFORMANCE  EMPTY
```

4.7.3 Attributes

Attribute	Required/ Optional	Type	Description
Subset	Required	String	Declares which PPML subset the dataset conforms to. The identifying string for each defined subset will be stated in the section of this specification that defines the subset.
Level	Optional	String	Optional qualifier, further identifying the features within the subset.

4.7.4 Context

CONFORMANCE can occur in PPML.

4.7.5 Example

The following is the start of a dataset that conforms to two hypothetical subsets:

```
<PPML>
   <CONFORMANCE  Subset="GA"/>
   <CONFORMANCE  Subset="TR"/>
   <SUPPLIED_RESOURCES ...>
       ...
```

4.8 The <TICKET> element

4.8.1 Description

A TICKET element may be used to explicitly identify the job ticket data for the dataset. The ticket data may either be stored inline within the TICKET element, or stored in a separate file and accessed by an external reference. See examples and application notes below.

If a TICKET element is not present in a PPML instance, all TICKET_REF elements defined in that PPML instance are considered references to ticket data implicitly identified at the application level, such as the PPML job package (see section 4.9 for more information about TICKET_REF).

4.8.2 Model

```
TICKET       (EXTERNAL_DATA | INTERNAL_DATA)
```

4.8.3 Attributes

Attribute	Required /Optional	Type	Description
Format	Required	String	Unique identifier of a previously defined ticket item.

4.8.4 Context

A single TICKET element can occur in the root PPML element prior to the definition of the first DOCUMENT_SET element.

4.8.5 Examples

The following examples illustrate methods of including JDF job ticket data. It is important to point out that although these examples use JDF syntax, the PPML language itself does not specify any particular ticket data format.

Example 1: Reference to an external job ticket file

In this method, the TICKET element contains a single EXTERNAL_DATA element, which references an external job ticket file.

```
<TICKET Format="application/jdf">
  <EXTERNAL_DATA Src="MyTicket.jdf"/>
</TICKET>
```

Example 2: Internal ticket data

In this method the entire ticket file is imported into the PPML dataset.

```
<TICKET Format="application/jdf">
   <INTERNAL_DATA Encoding="text/xml">
     <jdf:JDF xmlns:jdf="http://www.CIP4.org/JDFSchema_1">
        <jdf:JDF Class="Product" ... >
           ...
        </jdf:JDF>
     </jdf:JDF>
   </INTERNAL_DATA>
</TICKET>
```

4.8.6 Considerations about internal vs. external ticket data:

The external method is expected to be preferred in most situations.

The internal method forfeits a key advantage of the external method: neither the ticket nor the dataset can be reused independently of each other. The advantage is that the dataset and ticket are in a single file, reducing the chances of their becoming separated during production. Additional considerations:

- Since PPML workflows typically involve many other external files, the internal job ticket method is not expected to provide a substantial change in reliability of the workflow. (In either case, the shop must still keep track of the files that comprise a dataset such as a TLC package.)
- It is expected that in some cases, the ticket data may be edited in prepress. In those cases, it would be more convenient to simply edit a separate ticket data file since it will be much smaller than a PPML file containing the ticket data.
- Similarly, if the ticket data is to be edited, it's safer to keep it separate from the dataset, to eliminate any chance of unintentionally modifying the dataset.

4.9 The <TICKET_REF> element

4.9.1 Description

A TICKET_REF can occur in various places within the PPML hierarchy and is used to reference uniquely identified pieces of ticket information defined in the ticket data associated with the PPML data. The ticket data itself is either explicitly identified by a TICKET element (refer to section 4.8), or in the absence of a TICKET element, identified implicitly from the context of the dataset at the application level (for instance in a package as described in Appendix D: "Packaging PPML datasets for transport using ZIP files or removable media)."

These pieces of ticket information referenced from within the PPML hierarchy may be used to specify characteristics of the print product definition such as media type and single and two sided printing for the various pages, as well as process control parameter resources, or any other production information unrelated to the definition a PPML page's content. The semantics of the information referenced by TICKET_REF are not defined by this specification and are instead defined in the specification of the particular format identified by the value of the Format attribute of the TICKET element.

A TICKET_REF element may have either a Ref attribute or an ExtIDRef attribute, but not both. ExtIDRef directly references a single piece of ticket information; Ref can indirectly reference one or more pieces of ticket information, by referencing a previously defined TICKET_SET element.

4.9.2 Model

```
TICKET_REF    EMPTY
```

4.9.3 Attributes

Attribute	Required /Optional	Type	Description
ExtIDRef	Optional	String	Unique identifier of a previously defined ticket item.
Ref	Optional	String	Unique identifier of a previously defined Ticket Set element.

4.9.4 Occurs in

The TICKET_REF element can occur in PPML, DOCUMENT_SET, DOCUMENT, PAGE, TICKET_SET, and TICKET_STATE.

4.9.5 Scope

The scope in which the ticket data referenced by a TICKET_REF takes effect is implied by the location of the TICKET_REF in the PPML hierarchy; there is no explicit Scope attribute. Any ticket data specified by a TICKET_REF only affects content that occurs after the TICKET_REF; the ticket data stays in effect until the end of the containing element, or until overridden by a subsequent TICKET_REF.

A resource will expire at a place that is determined by where its Ticket Ref occurs. Here are two examples in which a Ticket Ref accesses a duplex-printing resource in two different ways:

```
<!-- Example 1 ->
<DOCUMENT_SET>
    <TICKET_REF ExtIDRef="TwoSidedLong".../>
    <!-- The above duplexing instruction persists to the end of the
        containing element - the current DOCUMENT_SET, including all
        subsequent documents, or until overridden by another duplexing
        instruction -->
    <DOCUMENT.../>
    <DOCUMENT.../>
    <DOCUMENT.../>

<!-- Example 2 ->
<DOCUMENT_SET>
    <DOCUMENT.../>
    <DOCUMENT...>
        <TICKET_REF ExtIDRef="TwoSidedLong".../>
        <!-- The above persists only to the end of the containing element
            (DOCUMENT, in this case), at which point the previous duplexing
            state, if any, returns -->
    </DOCUMENT>
    <DOCUMENT.../>
    :
```

4.9.6 Inheritance of ticket information

Ticket inheritance between document structure elements

The activated resource shall be inherited by lower levels of the PPML hierarchy (`PPML`, `DOCUMENT_SET`, `DOCUMENT`, `PAGE`, `MARK`), unless it is later overridden by other `TICKET_REF` elements at those lower levels. Below is an example, presuming that the job ticket has defined two resources whose IDs are Media_White (the default for this ticket) and Media_Goldenrod:

```
<!-- at first, the initial setting "Media_White" is active for all documents-->
    <DOCUMENT...>
        <PAGE.../>
        <PAGE...>
            <TICKET_REF ExtIDRef="Media_Goldenrod"/>
            <!-- Goldenrod applies to this page only -->
        </PAGE>
        <PAGE.../>
    </DOCUMENT>

    <DOCUMENT...>
        <TICKET_REF ExtIDRef="Media_GoldenrodMedia"/>
        <!-- Goldenrod applies to all pages in the document -->
        <PAGE.../>
        <PAGE.../>
        <PAGE...>
            <TICKET_REF ExtIDRef="Media_WhiteMedia"/>
            <!-- White applies to this page only -->
        </PAGE>
        <PAGE...>
            <!-- back to Goldenrod -->
```

```
      </PAGE>
   </DOCUMENT>
   <!-- initial job setting "White" is again active -->
```

Ticket inheritance in definition of reusable content:

Ticket information in Reusable Object definitions is independent of ticket information in the dataset.

- `REUSABLE_OBJECT` elements do not inherit ticket resources from their XML parents in the PPML stream. Rather, each `REUSABLE_OBJECT` element inherits the initial settings of the job ticket, if any, specified in the dataset's `TICKET` element. Within the `REUSABLE_OBJECT` element, any subsequence `TICKET_REF`s are relative to those initial values.
- `TICKET_REF`s inside a `REUSABLE_OBJECT` element have no effect on the PPML Pages, Documents, etc. in which they occur.

Example: Assume that a dataset's job ticket, specified in its `TICKET` element, has two resources that activate or deactivate a feature called "RIPfeature12," and those resource have IDs `Feature12on` and `Feature12off`. Assume the ticket establishes an initial setting of `Feature12off`.

```
<!-- at first, "Feature12off" is active for all documents-->
   <DOCUMENT...>
   <TICKET_REF ExtIDRef="Feature12on"/>
   <DOCUMENT...>
      <REUSABLE_OBJECT...>
         ...
         <OCCURRENCE_LIST>
            <OCCURRENCE....>
               <!-- This OCCURRENCE element contains no TICKET_REF for
                  Feature12.  The initial setting for "RIPFeature12" will be
                  the initial setting in the job ticket.  If this dataset is
                  processed with different job tickets, the setting for
                  RIPfeature12 on this OCCURRENCE may vary. -->
               ...
            </OCCURRENCE>
            <OCCURRENCE....>
               <!-- This OCCURRENCE element does contain a TICKET_REF for
                  Feature12.  The setting for RIPfeature12 on this OCCURRENCE
                  is known. -->
               <TICKET_REF ExtIDRef="Feature12off"/>
            </OCCURRENCE>
         </OCCURRENCE_LIST>
      </REUSABLE_OBJECT>
      <!-- The document content stream resumes; TICKET_REFs within the
         REUSABLE_OBJECT element are ignored -->
      <PAGE...>   ...
```


4.9.7 Recommended Feature ID Strings

The following are recommended (not mandatory) `ExtIDRef` strings for controlling features that are commonly available on digital print equipment. These are the ID strings used for these features in the PPML Job Ticket specification, which is based on JDF, the recommended format for graphic arts applications.

This list in no way limits what features may be accessed through the PPML `TICKET_REF` element in any given ticket format. This list only suggests a convention for the features shown here.

The detailed semantics implied by each of these strings are implementation-dependent; these strings are simply recommended as a way of accessing the feature if it exists. It is expected that in many cases, detailed machine setup will be performed by a person operating the equipment, so these "on/off" controls simply activate or deactivate the feature, however it has been set up.

Feature	**Action**	**Recommended ExtIDRef**
Trimming	Enable	`ExtIDRef="TrimmingOn"`
	Disable	`ExtIDRef="TrimmingOff"`
Collate	Enable	`ExtIDRef="CollateEnabled"`
	Disable	`ExtIDRef="CollateDisabled"`
Folding	Enable	`ExtIDRef="SaddleFold"`
	Disable	`ExtIDRef="NoFold"`
Duplex	Long Edge	`ExtIDRef="TwoSidedLongEdge"`
	Short Edge	`ExtIDRef="TwoSidedShortEdge"`
	Off	`ExtIDRef="OneSided"`
Output order	First page first, face up	`ExtIDRef="SameOrderFaceUp"`
	First page first, face down	`ExtIDRef="SameOrderFaceDown"`
	Last page first, face up	`ExtIDRef="ReverseOrderFaceUp"`
	Last page first, face down	`ExtIDRef="ReverseOrderFaceDown"`
Stitching (stapling)	SaddleStitch	`ExtIDRef="StitchSaddle"`
	Top Left	`ExtIDRef="StitchTopLeft"`
	Bottom Left	`ExtIDRef="StitchBottomLeft"`
	Top Right	`ExtIDRef="StitchTopRight"`
	Bottom Right	`ExtIDRef="StitchBottomRight"`
	Dual Right Edge	`ExtIDRef="StitchDualRightEdge"`
	Dual Left Edge	`ExtIDRef="StitchDualLeftEdge"`
	Dual Top Edge	`ExtIDRef="StitchDualTopEdge"`
	Dual Bottom Edge	`ExtIDRef="StitchDualBottomEdge"`
	None	`ExtIDRef="StitchNone"`

Feature	**Action**	**Recommended ExtIDRef**
Auto booklet making	From unimposed pages	`ExtIDRef="BookletWithReorder"`
	From pre-imposed sheets	`ExtIDRef="BookletPreOrdered"`
	Off	`ExtIDRef="BookletOff"`
Jogging (offset)	Enable	`ExtIDRef="JogOffsetOn"`
	Disable	`ExtIDRef="JogOffsetOff"`
Holemaking	3 hole left	`ExtIDRef="HoleType3HoleLeft"`
	3 hole right	`ExtIDRef="HoleType3HoleRight"`
	2 hole top	`ExtIDRef="HoleType2HoleTop"`
	2 hole bottom	`ExtIDRef="HoleType2HoleBottom"`
	None	`ExtIDRef="HoleTypeNone"`
Color model	Color (CMYK)	`ExtIDRef="CMYKColorModel"`
	Device Gray	`ExtIDRef="DeviceGray"`
Black Overprint	Enable	`ExtIDRef="BlackOverprintEnabled"`
	Disable	`ExtIDRef="BlackOverprintDisabled"`
Spot Color	Declare a named color to be a spot color (so it is not color-separated)	`ExtIDRef="SpotColor_MyColor"` *(Substitute the actual color name after the underscore)*
Screen selector	Identify a screen definition to be used in processing of images	`ExtIDRef="ScreenSelector_MyScreen"` *(Substitute the actual screen name after the underscore)*
Media selection	Identify media to use (A media name, location, or other identifier that's defined in the job ticket or is otherwise known to the Consumer)	`ExtIDRef="Media_MyMedia"` *(Substitute the actual media name after the underscore)*

4.10 The <TICKET_SET> element

4.10.1 Overview

The TICKET_SET element is an aggregation of several TICKET_REF elements.

4.10.2 Model

```
TICKET_SET    (TICKET_REF*)
```

4.10.3 Attributes

Attribute	Required /Optional	Type	Description
ID	Required	String	Unique identifier of this Ticket Set.

4.10.4 Occurs in

The TICKET_SET element can occur in PPML, DOCUMENT_SET, DOCUMENT, PAGE, MARK, REUSABLE_OBJECT and OCCURRENCE_LIST.

4.11 The <TICKET_STATE> element

4.11.1 Overview

The TICKET_STATE element provides hints during the definition of Reusable Object Occurrences regarding Ticket States that may be in effect when the Occurrence is referenced.

The Ticket State is the state of the Consumer relative to all parameters that can be controlled by a Job Ticket. This list of parameters will vary from Consumer to Consumer, because different products have different features that a Job Ticket can control.

Note that the Ticket State of a Consumer will vary as jobs are produced. Examples:

- immediately upon power-up the system will have one Ticket State
- an operator may change some RIP settings manually, e.g. set the resolution to 600 dpi
- after the system processes a Job Ticket the state will usually be different
- often, parameters will change while jobs are running.

When the Occurrence is subsequently referenced with OCCURRENCE_REF, the Ticket State at that time should match one of the Ticket States listed in the OCCURRENCE definition. If it does not, the Consumer may not have had adequate information about how to prepare the Occurrence at time of definition, so the result will be implementation-dependent. See section 5.14.8 for more information about how Consumers may process Occurrences.

4.11.2 Model

```
TICKET_STATE (TICKET_REF*)
```

4.11.3 Attributes

None.

4.11.4 Context

The TICKET_STATE element appears only in OCCURRENCE.

4.11.5 Computing the current Ticket State

The Ticket State defined by a TICKET_STATE element is:

- the Ticket State of the Consumer after processing the job ticket specified in the TICKET element, plus
- the effect of the TICKET_REFs contained in the TICKET_STATE element.

Note that each TICKET_STATE element defines a different, separate Ticket State; the elements are not cumulative. See example below.

4.11.6 Example: multiple TICKET_STATE elements in an OCCURRENCE

Each `TICKET_STATE` element defines a different, separate Ticket State. This example illustrates two different parameters that often affect processing of Occurrences: ScreenSelector and Color Model. Presume that the current Job Ticket contains IDs "ScreenSelector_AdobeAccurateScreens" and "CMYKColorModel".

```
<REUSABLE_OBJECT>
   <OBJECT Position=...>
      ...
   </OBJECT>
   <VIEW/>
   <OCCURRENCE_LIST>
      <OCCURRENCE  Name=...>
         <VIEW/>
         <TICKET_STATE>
            <!--This Ticket State is the state after the job ticket was first
               processed, plus the effect of the CMYKColorModel Ticket Ref and
               the effect of the ScreenSelector Ticket Ref. -->
            <TICKET_REF  ExtIDRef="CMYKColorModel"/>
            <TICKET_REF  ExtIDRef="ScreenSelector_AdobeAccurateScreens"/>
         </TICKET_STATE>
         <TICKET_STATE>
            <!--This Ticket State is the state after the job ticket was first
               processed, plus the effect of the CMYKColorModel Ticket Ref.
               The ScreenSelector for this Ticket State is inherited from the
               dataset's original Ticket State, whatever that may be.-->
            <TICKET_REF  ExtIDRef="CMYKColorModel"/>
         </TICKET_STATE>
         <TICKET_STATE>
            <!--This Ticket State is the state after the job ticket was first
               processed, plus the effect of this ScreenSelector Ticket Ref.
               The Color Model for this Ticket State is inherited from the
               dataset's original Ticket State, whatever that may be.-->
            <TICKET_REF  ExtIDRef="ScreenSelector_AdobeAccurateScreens"/>
         </TICKET_STATE>
      </OCCURRENCE>
   </OCCURRENCE_LIST>
</REUSABLE_OBJECT>
```

Chapter 5: The PPML page

5.1 The PPML Coordinate System

The PPML coordinate system is the same as the Cartesian coordinates used by PostScript®:

- the origin (0,0) is at the bottom left corner of the page
- units are 1/72 of an inch
- *x* increases to the right
- *y* increases upward.

All PPML units are base 10. The following definitions apply:

Integer: In PPML, an "integer" is specified as an optional sign character ('+' or '–', with '+' being the default) followed by one or more digits "0" to "9". The range for a PPML integer encompasses (at a minimum) –2147483648 to +2147483647.

Number: In PPML, a "number" is either an "integer" or an optional sign character ('+' or '–', with '+' being the default) followed by zero or more digits "0" to "9" followed by a dot (.) followed by zero or more digits "0" to "9" with at least one digit required either before or after the dot. The digits after the dot may be followed by an optional exponent. The exponent is the letter 'E' or 'e' followed by an "integer." A "number" has the capacity for at least a single-precision floating point number (see [ICC32]) and has a range (at a minimum) of –3.4e+38F to +3.4e+38F.

[ICC32] refers to "ICC Profile Format Specification, version 3.2", 1995. Available at ftp://sgigate.sgi.com/pub/icc/ICC32.pdf.

5.2 A Page contains Marks

PPML constructs a page image by placing a series of *Marks* on the page. Marks can consist of graphics, text and/or images defined in some external content data format. A Mark can reference either *non-reusable* or *reusable* content data. Reusable content data are data which may have multiple occurrences in a PPML page, document, document set, dataset or environment. The PPML code defines the data as reusable, which permits the PPML consumer to cache these items in some format which may permit highly efficient reproduction.

5.3 The <MARK> Element

5.3.1 Description

The MARK element specifies the actual placement of marks on a page. It is used either for the placement of Objects (section 5.7) or for placing an Occurrence of a Reusable Object (section 5.12).

The Consumer places MARKs on a page in the order in which they are listed in the PAGE element. MARKs later in a PAGE element are placed on top of the earlier ones.

Each MARK's Position attribute defines its location on the page, while the associated VIEW allows selecting (clipping) and transforming (e.g. scaling) the MARK to create the desired page content.

Conceptually, each MARK defines a rectangular raster image that consists of "marked" and "transparent" pixels. Each MARK is rasterized independently from any other MARKs on the page. When a MARK overlaps MARKs previously placed on the page, its marked pixels completely obscure the previous MARKs' pixels, and the transparent pixels leave the previous MARKs' pixels unaffected. Which pixels in a MARK's raster image are marked and which are transparent depends on the MARK's content data and the content data format, and is outside of the scope of the PPML Specification.

Notes:

1. In the case of PostScript and PDF content data, the MARK's raster image starts out consisting of transparent pixels. Only those pixels marked by imaging operators are "marked" in the MARK's raster image.
2. In the case of non-transparent TIFF content data, the original rectangular area defined by the TIFF source is completely marked. If the data is not rotated by a VIEW transformation, the rectangular raster image resulting from the MARK completely obscures every pixel beneath it. If the data is rotated, then only the pixels beneath the parallelogram resulting from the transformed TIFF data are obscured.
3. Other content formats likewise include the concept of transparent (or "clear") pixels as well as white and colored pixels. Any such transparent pixels will allow pixels from previous MARKs to show through unaffected.

Some content formats describe pixels (or objects) that are only partially transparent. The interaction of these pixels with other pixels or objects defined by the same content data from a single SOURCE used to generate the raster image for a particular MARK is defined by the content data format, and is outside of the scope of the PPML specification. However, any such pixels are considered "marked" for the purposes of determining the effect of MARK overlaps: if the raster image for a MARK contains "partially transparent" pixels that overlap pixels from a previous MARK, the "partially transparent" pixels of the MARK that is on top are considered as "marked" pixels and completely obscure the previous MARK's pixels.

5.3.2 Model

```
MARK        ((VIEW?, OBJECT+) | OCCURRENCE_REF | SEGMENT_REF)
```

5.3.3 Attributes

Attribute	Required /Optional	Type	Description
Position	Required	Number × 2	Specifies a translation to be applied to the object's coordinate space in order to position the object on the page. This translation is concatenated with any prior transformations applied to the original data.

5.3.4 Context

`MARK` can occur in `PAGE`.

5.3.5 Implementation note

The `Position` attribute on `MARK` and `OBJECT` defines the placement of these objects. Note that this placement is also affected by other transformations applied to the elements. For example, if the `OBJECT` is a rectangle whose lower left corner is at (0, 0), that corner will be placed at the point specified by `Position`. If the rectangle's upper left corner is at (0, 0), that corner will be placed at the `Position` point.

5.4 The <VIEW> Element

5.4.1 Description

The VIEW element combines a TRANSFORM with a CLIP_RECT to form a description of how a particular set of content data is to be rendered.

5.4.2 Model

```
VIEW          (TRANSFORM?, CLIP_RECT?)
```

5.4.3 Attributes

None.

5.4.4 Context

VIEW can occur in MARK, OBJECT, REUSABLE_OBJECT and OCCURRENCE.

5.4.5 Empty VIEW elements

An empty VIEW element (<VIEW/>) means the identity transform with no clipping.

5.5 The <TRANSFORM> Element

5.5.1 Description

The TRANSFORM element represents a two-dimensional homogeneous transformation matrix.

5.5.2 Model

TRANSFORM EMPTY

5.5.3 Attributes

Attribute	Required /Optional	Type	Description
Matrix	Required	Number × 6	Supplies the components of a two dimensional homogeneous transformation matrix. See the *PostScript Language Reference Manual* for details.

5.5.4 Context

TRANSFORM can occur in VIEW.

5.6 The <CLIP_RECT> Element

5.6.1 Description

The CLIP_RECT element specifies the corners of a rectangle to be used for clipping the content data with which the CLIP_RECT is associated.

5.6.2 Model

```
CLIP_RECT    EMPTY
```

5.6.3 Attributes

Attribute	Required /Optional	Type	Description
Rectangle	Required	Number × 4	Supplies the x and y coordinates of the lower left and upper right corners of a rectangle to be used for clipping.

5.6.4 Context

CLIP_RECT can occur in VIEW.

5.7 The <OBJECT> Element

5.7.1 Description

The OBJECT element associates a VIEW with a SOURCE to specify the clip, scale and orientation of an item of appearance data within a MARK or a REUSABLE_OBJECT.

The Position attribute specifies a translation to be applied to the SOURCE's coordinate space in order to position the SOURCE in relation to other SOURCE elements within a MARK or REUSABLE_OBJECT. This translation is concatenated with any prior transformations applied to the original data. (See the implementation note regarding object origin in section 5.3.5.)

5.7.2 Model

```
OBJECT      (SOURCE, VIEW?)
```

5.7.3 Attributes

Attribute	Required /Optional	Type	Description
Position	Required	Number × 2	Specifies a translation to be applied to the object's coordinate space in order to position the object on the page. This translation is concatenated with any prior transformations applied to the original data.

5.7.4 Context

The OBJECT element can occur in MARK and REUSABLE_OBJECT.

5.8 The <SOURCE> Element

5.8.1 Description

The SOURCE element defines a set of one or more content elements (EXTERNAL_DATA, INTERNAL_DATA), of a single format, to be collected into a single sequence of appearance data. The content data from all enclosed elements are concatenated in the order the elements appear, and are processed as a single unit by the format processor, the same as if all the data had been submitted to the Consumer as a single object.

Note that some file format specifications allow non-content data, which must be removed by Consumers that accept that format. For instance, the format type for EPS files is application/postscript, but Windows EPS files contain a non-PostScript binary preview (See the *PostScript Language Reference Manual*, appendix H.5.2.), which the Consumer system must remove.

5.8.2 Model

```
SOURCE      ((INTERNAL_DATA | EXTERNAL_DATA)+ | EXTERNAL_DATA_ARRAY)
```

5.8.3 Attributes

Attribute	Required /Optional	Type	Description
Format	Required	Keyword	Indicates format of the data (e.g., PostScript, PDF, TIFF, etc.). Value: any format name registered with the Internet Assigned Numbers Authority (IANA).[6]
Dimensions	Required	Number × 2	The width *w* and height *h* of a rectangle that encloses the content data contained in this element. See 5.8.5, "Dimensions and ClippingBox" below.
ClippingBox	Optional	Number × 4	Supplies the coordinates of the lower left and upper right corners of the rectangle containing the desired area of the content data, in PPML default coordinates.

5.8.4 Context

SOURCE can occur in OBJECT.

5.8.5 Dimensions and ClippingBox

- For SOURCE elements whose content format is dimensionless, the Dimensions attribute states what width and height the Consumer should *assume*.
- If ClippingBox is not present, Dimensions specifies an implicit clipping rectangle "0 0 *w h*".

[6] These formats are listed at http://www.isi.edu/in-notes/iana/assignments/media-types/media-types.

- If both `Dimensions` and `ClippingBox` are present, both of them clip. The effective clipping boundary is the intersection of the clipping rectangle implied by `Dimensions` and the specified `ClippingBox`.

5.9 The <EXTERNAL_DATA> Element

5.9.1 Description

An EXTERNAL_DATA element identifies, by location and access method, a single content datum (e.g. a source file or job ticket). A content appearance datum, which may be in any of the supported formats (e.g., PostScript, PDF, PCL, TIFF, etc.), can be used by itself or in combination with other content elements to construct components which appear on the printed page.

5.9.2 Model

```
EXTERNAL_DATA   EMPTY
```

5.9.3 Attributes

Attribute	Required /Optional	Type	Description
Src	Required	URI	URI (Uniform Resource Identifier) string identifying the external data. See RFC2396 for full details of URIs.[7]
Checksum	Optional	String	Hexadecimal-encoded string, provided as a hint to the Consumer. Consumers are not required to support this attribute.
ChecksumType	Optional	String	Identifies the type of checksum. If this attribute is present, the Checksum attribute must also be present. Default="MD5".
SourceUsage	Optional	Keyword	"Single" or "Multiple" or "Unknown" (default). A hint to the Consumer: will data from this source be used only once, or in other elements? See 5.9.5, "The SourceUsage attribute" below.

5.9.4 Context

EXTERNAL_DATA may occur within SOURCE and TICKET.

5.9.5 The SourceUsage attribute

For content appearance data, SourceUsage="Multiple" means the data in this source file may be used again later. Thus, the Consumer may wish to cache the unprocessed source data to avoid retrieving it again later.

[7] RFC2396 is at www.ietf.org/rfc/rfc2396.txt. A good overview of URIs and URLs is at www.w3.org/Addressing/Overview.html.

5.10 The <EXTERNAL_DATA_ARRAY> Element

5.10.1 Description

An EXTERNAL_DATA_ARRAY element identifies, by location and access method, a multi-segment source datum. A multi-segment source is one that contains multiple content descriptions that can be accessed individually, e.g. a multi-page PostScript or PDF file.

Only one EXTERNAL_DATA_ARRAY element may be used in a SOURCE element.

5.10.2 Model

```
EXTERNAL_DATA_ARRAY    EMPTY
```

5.10.3 Attributes

Attribute	Required /Optional	Type	Description
Src	Required	URI	See section 5.9.3, attributes of EXTERNAL_DATA.
Checksum	Optional	String	Hexadecimal-encoded string, provided as a hint to the Consumer. Consumers are not required to support this attribute.
ChecksumType	Optional	String	Identifies the type of checksum. If this attribute is present, the Checksum attribute must also be present. Default="MD5".
Index	Optional	Integer	Indicates which segment is to be selected for use in this instance. The default (and minimum) value is "1", which corresponds to the very first segment of the referenced source file.
IndexUsage	Optional	Keyword	Single or Multiple or Unknown (default). A hint to the Consumer, meaning "will additional segments be used later in the same graphics state?" See section 5.10.5, "The IndexUsage attribute" below.

5.10.4 Context

EXTERNAL_DATA_ARRAY may occur within SOURCE.

5.10.5 The IndexUsage attribute

IndexUsage= "Multiple" means that although this instance only uses one of the segments in this multi-segment file, additional instances of EXTERNAL_DATA_ARRAY may call for other segments. Thus, as an optimization, a Consumer may wish to process all the segments in the source, not just the one segment specified by the Index attribute.

5.11 The <INTERNAL_DATA> Element

5.11.1 Description

An INTERNAL_DATA element is the same as an EXTERNAL_DATA element except that it contains the actual data, instead of referring to it. Therefore it has no Src attribute.

Like the content datum referred to by an EXTERNAL_DATA element, an INTERNAL_DATA content datum may be in any of the supported formats (e.g., PostScript, PDF, PCL, TIFF, etc.) and can be used by itself or in combination with other content elements to construct components which appear on the printed page.

Note that the content data itself, contained in the INTERNAL_DATA element, must be valid XML content – it must conform to the character sets identified in section 2.1.5, "Character sets."

5.11.2 Model

```
INTERNAL_DATA    ANY
```

5.11.3 Attributes

Attribute	Required /Optional	Type	Description
Encoding	Optional	Keyword	Encoding scheme of the data: None (default) or any encoding name registered with the Internet Assigned Numbers Authority (IANA).[8] However, note that Consumers are only required to support Base64.
CharacterSet	Optional	String	Specifies the character set of the decoded data. For use with text content or any other media type containing characters. Value: any character set name registered with the Internet Assigned Numbers Authority (IANA).[9]
Label	Optional	String	Any arbitrary string to identify this element, for instance in case an error message is necessary.
Creator	Optional	String	Identifies the application that created this content.

5.11.4 Context

INTERNAL_DATA may occur within SOURCE and TICKET.

[8] The valid encoding name strings are listed at http://www.isi.edu/in-notes/iana/assignments/transfer-encodings.

[9] The valid character set name strings are at http://www.isi.edu/in-notes/iana/assignments/character-sets.

5.12 The <REUSABLE_OBJECT> Element

5.12.1 Description

The REUSABLE_OBJECT element defines a component of page appearance which is intended for multiple use, and may therefore be stored by the PPML consumer in some optimized format.

Reusable Objects exist for efficiency: to store frequently used items so they can be accessed without redundant processing. Each individual use (Occurrence) of a Reusable Object may have its own different VIEW, but there may be some transformations that are shared. For instance, a photo may be clipped and rotated, and then be scaled to several different sizes. The VIEW on the Reusable Object could perform the clipping and rotating once; then several different Occurrences could be defined, each with a VIEW that performs additional scaling.

5.12.2 Model

```
REUSABLE_OBJECT (OBJECT+, VIEW?, OCCURRENCE_LIST)
```

5.12.3 Attributes

None.

5.12.4 Context

REUSABLE_OBJECT can occur in PPML, DOCUMENT_SET, DOCUMENT and PAGE.

5.12.5 Note regarding TICKET_REF

See section 4.9.6, "Inheritance of ticket information," for rules regarding inheritance of job ticket information in REUSABLE_OBJECT elements.

5.13 The <OCCURRENCE_LIST> Element

5.13.1 Description

Within a REUSABLE_OBJECT definition element, the OCCURRENCE_LIST element declares each viewing transformation which may be applied to the object, and may provide hints of the relative importance of each transformation.

5.13.2 Model

```
OCCURRENCE_LIST (OCCURRENCE)+
```

5.13.3 Attributes

None.

5.13.4 Context

OCCURRENCE_LIST can occur in REUSABLE_OBJECT.

5.14 The <OCCURRENCE> Element

5.14.1 Description

The OCCURRENCE element specifies the VIEW and relative importance with which a particular rendition of a Reusable Object will occur. By specifying Occurrence information in the definition of a Reusable Object, the PPML Producer facilitates optimization of rendering and storage by the eventual Consumer.

Note that the element model contains no explicit statement of the dimensions of the content image area that will be created when the Consumer generates this Occurrence. A Consumer that wishes to anticipate the dimensions should do so by accumulating the clipping boxes defined in the REUSABLE_OBJECT element.

5.14.2 Model

```
OCCURRENCE   (VIEW?, TICKET_STATE*)
```

5.14.3 Attributes

Attribute	Required /Optional	Type	Description
Name	Required	String	Name to be used when referring to this OCCURRENCE. The name must be unique within the Occurrence's scope or environment; see 5.14.5, "Policies for Name collisions" below.
Environment	Required if Scope ="Global"; not needed otherwise	String	Specifies the environment in which a global object should be defined. (There is no default environment.)
Scope	Optional	Keyword	Specifies the scope of this object's use. Possible values are Global, PPML, DocSet, Document and Page. By default, the scope is the containing element in which the object is defined. A higher value may be specified in this attribute, but a lower value is an error.
Overwrite	Optional	Boolean	Defines what the Consumer should do if Scope="Global" and the name already exists in the specified Environment: Yes means "overwrite the existing Occurrence", No means "ignore this element." Default= No. This attribute has no meaning unless Scope="Global".
Weight	Optional	Number	A number from 1 (minimum importance) to 100 (maximum) describing, qualitatively, the relative importance of this Occurrence. See 5.14.6, "Statistics about Reuse: the Weight attribute."

5.14.4 Context

The `OCCURRENCE` element can occur in `OCCURRENCE_LIST`.

5.14.5 Policies for Name collisions

The value of `Name` must be unique within the scope (or within the Environment, if `Scope="Global"`). The following policies define how the Consumer should handle the case where `Name` already exists at the specified Scope:

- If `Scope="Global"` then the `Overwrite` attribute defines what action should be taken: overwrite the existing attribute, or ignore this element?
- If Scope is not Global, an error occurs.

5.14.6 Statistics about Reuse: the `Weight` attribute

How efficiently a given system (Producer or Consumer) handles reusable content is expected to be a major differentiating factor compared to other PPML systems. System designers are therefore advised to give thought to efficient design regarding this feature.

Typically, when a PPML Consumer receives a Reusable Object definition, it will pre-process it (RIP it) into the data format required by the target print engine, and then save the resulting Occurrences somewhere (cache them), e.g. in RAM, on internal disk, or on some attached storage system. Sometime later, in the same print run or some other run, the data stream will call for that Occurrence by name, and the Consumer will be able to recall it from storage and image it without pausing to process it "on the fly."

A Consumer must make informed decisions about what to cache and for how long. A Consumer with large amounts of RAM may be able to hold all of a job's Occurrences in RAM at once; this approach will usually produce the fastest possible throughput. But as system price declines, RAM tends to be more limited, which forces the Consumer to make decisions about what to cache and what not to – especially as jobs become complicated and the quantity of Occurrences increases.

Imagine, for instance, a print run that includes two Occurrences. If one will be used 800 times and the other only twice, it's clear which one should be cached.

But the Consumer cannot make that decision unless it knows the relative importance of the Occurrences. Producers therefore play an important role in supporting productive printing: only the Producer knows how often an Occurrence will be used in a given print run, and if the Producer wishes to support optimized printing, it should feed that information to the Consumer via the `Weight` attribute in the `OCCURRENCE` element.

In the absence of `Weight`, a Consumer can still base a caching strategy on such factors as scope or least recently used.

5.14.7 What to cache and for how long

The Consumer is responsible for its caching technology and caching strategy; there is no requirement in this specification that the Consumer provide any particular caching functionality. However, the major goal of the PPML initiative is to improve efficiency of reuse, so Consumers that substantially improve throughput are likely to be much more successful.

It's also important to understand that there is no requirement to cache at any particular stage of the RIPping process. The communication between Producer and Consumer on this subject is limited to the Producer providing two types of information: `Weight` hints and `Scope` declarations.

Note that even if an Occurrence goes out of scope, the Consumer is not required to purge it (nor take any other action). In fact the Producer has no certain knowledge of the Occurrence's status. Even the scope declaration is just a "hint" that the Consumer may or may not use.

5.14.8 Strategies for Processing of Occurrences; effect of TICKET_REF

Every time an Occurrence is referenced, the resulting object must conform to the `SOURCE` and `VIEW`s specified in the `OCCURRENCE` element. This requirement is central to PPML's goal of interoperability.

However, as noted above, this specification does not mandate how and when the Consumer must *process* the Occurrence.

- A Consumer may choose to pre-RIP the Occurrence at the point of definition.
- Another Consumer may choose to defer the processing until the point of `OCCURRENCE_REF`.
- A third Consumer may choose to perform part of the processing at definition time and defer part until the point of reference.

Performance (speed) and expected visual results will affect these decisions.

Note that job ticket data, specified within or outside the PPML stream (see section 4.8), may influence how some Consumers ultimately render the Occurrence. All job ticket settings are considered "hints" to the PPML Consumer. This includes any `TICKET_REF`s within the Occurrence definition.

A Consumer may take the hints at face value and fully process the Occurrence at time of definition, using the ticket information available at that time. Another Consumer may defer processing until the Occurrence is referenced, using ticket data available at that time. Any other strategy is acceptable as well – for example, pre-processing at time of definition and re-processing at time of reference.

When the Occurrence is referenced, the Ticket State at point of OCCURRENCE_REF should match one of the states listed in a TICKET_STATE element in the OCCURRENCE definition. If it does not, the result is Consumer-dependent. See section 4.11 for more information.

If a PPML Producer is concerned with consistency of output across different Consumers, it should specify all relevant job ticket parameters when the Occurrence is defined, and ensure that the same parameters are in effect when the Occurrence is referenced. Alternatively, the Producer may specify no parameters, or only some, and leave all other rendering decisions to the individual Consumer.

For example, given the following PPML, where the JPEG is of a brown shoe and the initial job ticket setup has specified color output using the CMYK color model:

```
<DOCUMENT>
  <REUSABLE_OBJECT ...>
    ... Src="brown_shoes.jpg"...
    <OCCURRENCE Name="brown_shoes_no_scale">
    ...
  </REUSABLE_OBJECT>
```



```
<PAGE>
  <TICKET_REF ExtIDRef="DeviceGrey"/>
  ...
  <OCCURRENCE_REF Ref="brown_shoes_no_scale"/>
  ...
```

It is implementation defined whether the `OCCURRENCE_REF` above prints in brown or black and white.

Again, performance and expected results will affect such decisions, as vendors design products for different markets. But in all cases the output must conform to the original definition in the `OCCURRENCE` element.

5.14.9 Implementation note: Effects of imposition

Consumers are advised to take into account the possibility that imposition will require the Occurrence to be imaged in more than one orientation.

5.15 The <OCCURRENCE_REF> Element

5.15.1 Description

The OCCURRENCE_REF element creates a reference to an Occurrence of a Reusable Object. The Reusable Object and Occurrence to which the OCCURRENCE_REF refers must have been defined earlier in the dataset or globally via a named environment.

5.15.2 Model

```
OCCURRENCE_REF  EMPTY
```

5.15.3 Attributes

Attribute	Required /Optional	Type	Description
Ref	Required	String	Name of a previously defined Occurrence for this object.
Environment	Optional	String	The environment in which the name of a Global Occurrence should be interpreted. (This attribute is required if the scope of the Occurrence is Global; otherwise, this attribute has no meaning.)

5.15.4 Context

OCCURRENCE_REF can occur in MARK, SHEET_MARK, VER_TRIM_MARKS, HOR_TRIM_MARKS, VER_FOLD_MARKS and HOR_FOLD_MARKS.

5.16 Notes on REUSABLE_OBJECTs, OCCURRENCES, Scope, and Environment

5.16.1 Implementation notes

Note that Occurrences with `Scope="Global"` will never go out of scope. Therefore, they will accumulate wherever the Consumer stores its resources, e.g. its disk or a file server. This means that any Consumer system may want to consider whether, and how, to manage the storage of Reusable Objects and their Occurrences.

5.16.2 Protection of an Environment's global resources

It is the Consumer's responsibility to protect global-scoped Occurrences from being accidentally erased by subsequent downloads. Therefore, Consumer vendors may want to require authorization before any dataset can create or access an Environment. This is left as an implementation decision for the Consumer.

One approach could be to use some unique identifier as part of the Environment, perhaps including the domain name of the print job's originator. In either case, PPML merely considers it to be a simple text string, but accidental duplication of Environment would be unlikely. Examples:

```
Scope="Global" Environment="FordJob@Dclark@MyCompany.com"
Scope="Global" Environment="MyCompany/Dclark/"
```

5.16.3 Scope

The Occurrence's `Scope` attribute defines how long the Occurrence must be available: for the current Page, the current Document, the current Document Set, the entire PPML dataset, or permanently (Global). For instance, in a Consumer that caches Occurrences, when the Consumer completes the defined scope (e.g. the current Document), the Occurrence can be flushed from cache memory.

Scoping is mostly for lifespan: when an Occurrence goes out of scope, the Consumer is permitted to recover the resources it used. (It also has a namespace effect – for instance, "Ford logo" may have a different meaning in a particular job than it does for most projects.) However, global scope is somewhat different. Most uses of global scope will be for Occurrences that persist over a considerable period of time: weeks or months, as in a continuing project, perhaps even years, such as company logos). It is expected that in typical production work such Occurrences will be loaded into the Consumer system before production jobs begin, and they will then be referenced repeatedly in multiple jobs or projects.

5.16.4 Resolving Occurrence names

When a Mark contains an Occurrence Reference, the referenced Occurrence name is resolved by searching from lowest to highest level. If the Occurrence was defined within the current Page, that definition is used; if not, each higher level is searched: Document, Document Set, then PPML. Global Occurrences are only searched if the Occurrence Reference has an Environment attribute. If

it has, only global Occurrences in that Environment are searched and Occurrences at lower scopes are ignored. It is an error if no Occurrence is found.

5.16.5 Downloading reusable objects for caching for future use

In real world workflows, the source data for some reusable objects typically becomes available to production workers before other objects become available. To minimize workload at deadline time, it's a good idea to download such objects to the Consumer for caching when they become available, rather than waiting until all objects are available.

To do this, construct a PPML dataset that contains no Document Sets, just Reusable Object definition elements. Set each element's Scope attribute to `Global` and define a value for the `Environment` string attribute.

5.17 The <SEGMENT_ARRAY> element

5.17.1 Description

The SEGMENT_ARRAY element defines a collection of reusable objects whose contents are contained in a multi-page source file. Once the SEGMENT_ARRAY has been declared, individual segments can be placed on a page by use of a SEGMENT_REF.

A SEGMENT_ARRAY element may identify the source data via the Src attribute, or via an EXTERNAL_DATA or INTERNAL_DATA element, but not both.

5.17.2 Model

```
SEGMENT_ARRAY    (VIEW?, (INTERNAL_DATA | EXTERNAL_DATA)?)
```

5.17.3 Attributes

Attribute	Required/ Optional	Type	Description
ClippingBox	Optional	Number × 4	Supplies the coordinates of the lower left and upper right corners of the rectangle containing the desired area of the content data, in PPML default coordinates.
Dimensions	Required	Number × 2	The width w and height h of a rectangle that encloses the content data contained in this element.
Environment	Required if Scope = "Global"; not needed otherwise	String	Specifies the environment in which a global object should be defined. (There is no default environment.)
Format	Required	Keyword	Indicates the format of the data (e.g., PostScript, PDF, TIFF, etc.) Value: any format name registered with the Internet Assigned Numbers Authority (IANA). (See Appendix 3.)
IndexRange	Required	Comma-separated list of ranges, e.g. 1-10 or 1-5,7,10-12	Specifies which of the segments within the source to fully process and cache within the Consumer. Segments which are skipped may require some processing to locate the start of data for subsequent segments. The list is specified as either a single index or a range of indices given as l-h ("low to high"). The index values must increase monotonically.
Name	Required	String	Name to be used when referencing SEGMENT_ARRAY elements.
Overwrite	Optional	Boolean	Defines what the Consumer should do if Scope="Global" and the name already exists in the specified environment. For each segment specified in the

Attribute	Required/ Optional	Type	Description
			IndexRange, a value of "Yes" instructs the Consumer to replace a prior definition with the same Name and index by the newly supplied value and leaves any other segment unchanged (any segment not in the current IndexRange is taken from the existing definition). It is allowed that the maximum index of the current IndexRange is greater than the one of the existing definition. A value of "No" (the default) instructs the Consumer to leave the prior value of the segment.
Scope	Optional	Keyword	Specifies the scope of this element's use. Possible values are `Global`, `PPML`, `DocSet`, `Document` and `Page`. By default, the scope is the containing element in which the object is defined. A higher value may be specified with this attribute. Specifying a lower scope level is an error.
Src	Optional	URI	See section 5.9.3. Use of this attribute is discouraged. To identify an external file, Producers are advised to instead use `EXTERNAL_DATA`.
Checksum	Optional	String	Hexadecimal-encoded string, provided as a hint to the Consumer. Consumers are not required to support this attribute.
ChecksumType	Optional	String	Identifies the type of checksum. If this attribute is present, the Checksum attribute must also be present. Default="`MD5`".
Weight	Optional	Number	A number from 1 (minimum importance) to 100 (maximum) describing, qualitatively, the relative importance of this Segment Array. See section 5.14.6, "Statistics about Reuse: the `Weight` attribute."

5.17.4 Context

The `SEGMENT_ARRAY` element can occur within `PPML`, `DOCUMENT_SET`, `DOCUMENT` and `PAGE`.

5.17.5 Implementation note: Effects of `IndexRange` and `Overwrite`

When combining `IndexRange` with `Overwrite="Yes"`, it is possible that segments in the same `SEGMENT_ARRAY` have different values for `ClippingBox`, `View` and `Dimensions`.

5.17.6 Implementation note: Effects of nested scopes

A redefinition of a `SEGMENT_ARRAY` on a lower scope completely hides the ones on a higher scope. As a consequence a reference to a segment that is not in the IndexRange of the `SEGMENT_ARRAY` on the lowest scope results in an empty mark and is not resolved by a possible segment on a higher scope.

5.18 The <SEGMENT_REF> element

5.18.1 Description

The SEGMENT_REF element creates a reference to a member of a Segment Array. The SEGMENT_ARRAY element to which this element refers must have been previously defined in a scope containing the reference.

5.18.2 Model

```
SEGMENT_REF   EMPTY
```

5.18.3 Attributes

Attribute	Required/ Optional	Type	Description
Environment	Optional	String	Specifies the environment in which the name of the global-scoped element is defined.
Index	Optional	Integer	Indicates which segment is to be selected for use in this instance. If Index refers to a segment that falls outside the specified IndexRange (see also the Overwrite attribute in SEGMENT_ARRAY), this Mark is empty. The default (and also minimum) value is "1", which corresponds to the very first segment of the referenced source file.
Ref	Required	String	Specifies the name of the previously defined Segment Array to which this element refers.

5.18.4 Context

A SEGMENT_REF can occur in MARK.

5.19 Definition of PPML Extent Boxes

The extent box of a SOURCE element is its effective clipping boundary determined by its Dimensions and ClippingBox attributes. See Section 5.8.5, Dimensions and ClippingBox for the definition of the clipping boundary.

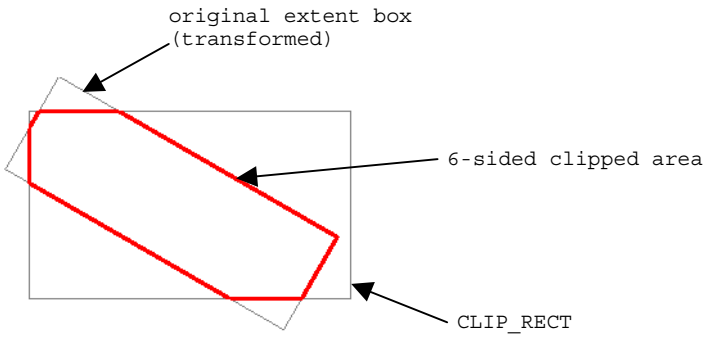
To apply a VIEW to an extent box, the Consumer must use the following procedure:

- Apply the transformation specified in the TRANSFORM attribute to the current extent box. This results in a four-sided figure.
- If the VIEW has a CLIP_RECT attribute, clip the four-sided figure using the clipping rectangle. This results in a figure that can have up to eight sides.
- Compute the bounding box of this figure: it is the new extent box.

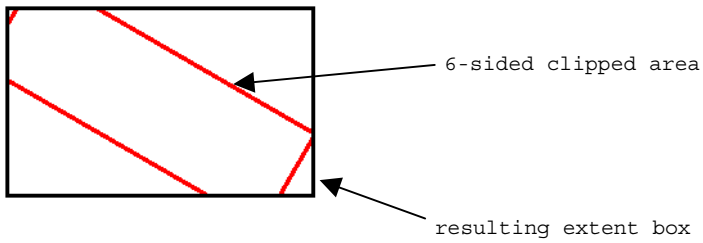
To combine two or more extent boxes, compute the bounding box of the positioned extent boxes: it is the new extent box.

5.19.1 Applying a VIEW to an Extent Box

This example shows how a VIEW is applied to an extent box:

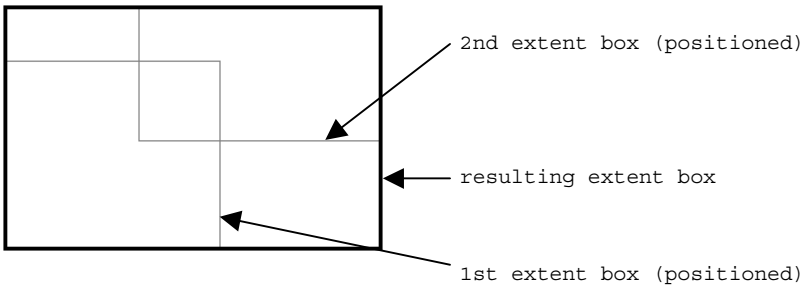


The resulting extent box is shown below:

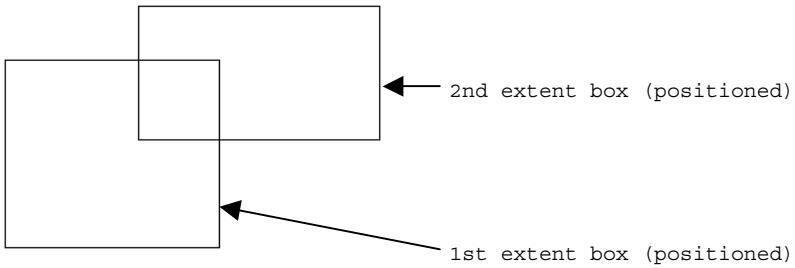


5.19.2 Combining Extent Boxes

When two or more objects are combined in a single mark, their extent boxes are combined as follows, then the view is applied as shown in the previous section:



The resulting extent box is shown below:



The next section contains examples of how extent boxes are used.

5.20 Notes on Transforming, Clipping and Positioning

The following two examples show how to process a simple case of a MARK on a PPML page: a single EPS file is transformed and clipped in various ways, and placed on a page. All the instructions in the first example will be contained in the MARK element; the second example shows how the same result could be accomplished using a REUSABLE_OBJECT element.

Both examples use the same original EPS file – a few words of text, which fits into a box 100 units high and 150 units wide. The result we want to achieve is a part of this EPS file, reduced, cropped, and rotated, as shown at the right.

Source

Desired Result

5.20.1 Self-Contained MARK Example

A self-contained MARK has this structure:

- The simplest possible MARK contains a VIEW and one OBJECT.
- An OBJECT is a VIEW of a single SOURCE.
- Each of the VIEWs can contain a TRANSFORM and a CLIP_RECT.

To process a MARK, the Consumer must first process each OBJECT inside it. And to do that, it first processes the SOURCE in the OBJECT. Here is the resulting sequence the Consumer must follow:

- Process the SOURCE, applying its ClippingBox if any
- Take the result and transform it using the TRANSFORM from the OBJECT's VIEW
- Take the result and clip it using the CLIP_RECT from the OBJECT's VIEW

This produces one OBJECT that will be contained in the MARK.

Now, position the OBJECT in the MARK's coordinate space.

Repeat the above for each OBJECT in the MARK.

Now, apply the MARK's VIEW:

- Take the set of (one or more) OBJECTs and transform it using the TRANSFORM from the MARK's VIEW
- Take the result and clip it using the CLIP_RECT from the MARK's VIEW

This produces the final piece of page content that will appear on the page. The last step will be to position it on the page, using the MARK's Position attribute.

The following PPML fragment achieves our desired result using a self-contained MARK:

```
<MARK Position="30 40">
   <VIEW>
      <TRANSFORM Matrix="0.75 0 0 0.75 0 0" />
      <CLIP_RECT Rectangle="0 0 75 75" />
   </VIEW>
   <OBJECT Position="-20 -20">
      <SOURCE Dimensions="150 100" ClippingBox="30 50 160 90">
         <EXTERNAL_DATA Src="ppml.eps" />
      </SOURCE>
      <VIEW>
         <TRANSFORM Matrix="0.866 -0.5 0.5 0.866 -25.98 31.7" />
         <CLIP_RECT Rectangle="20 20 120 120" />
      </VIEW>
   </OBJECT>
</MARK>
```

Note that the Format attribute of the SOURCE *has been omitted for clarity.*

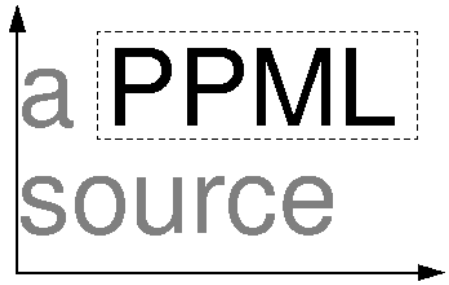
A PPML Consumer processes this fragment using the steps shown on the following pages.

1. Read the SOURCE element in the OBJECT

First, the Consumer finds the SOURCE element inside the MARK:

```
<MARK Position="30 40">
   <VIEW>
      <TRANSFORM Matrix="0.75 0 0 0.75 0 0" />
      <CLIP_RECT Rectangle="0 0 75 75" />
   </VIEW>
   <OBJECT Position="-20 -20">
      <SOURCE Dimensions="150 100" ClippingBox="30 50 160 90">
         <EXTERNAL_DATA Src="ppml.eps" />
      </SOURCE>
      <VIEW>
         <TRANSFORM Matrix="0.866 -0.5 0.5 0.866 -25.98 31.7" />
         <CLIP_RECT Rectangle="20 20 120 120" />
      </VIEW>
   </OBJECT>
</MARK>
```

The ClippingBox attribute crops the edges of the EPS file, as shown by the dashed line:



☑ *Current coordinate space: the* SOURCE.

The result is shown below. **This is the content defined by this** SOURCE **element:**

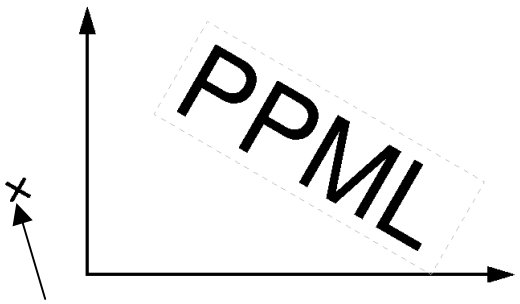
SOURCE's origin

2. Completing the OBJECT: VIEW the SOURCE

Next, the Consumer applies the OBJECT's VIEW, starting with the TRANSFORM element:

```
<MARK Position="30 40">
   <VIEW>
      <TRANSFORM Matrix="0.75 0 0 0.75 0 0" />
      <CLIP_RECT Rectangle="0 0 75 75" />
   </VIEW>
   <OBJECT Position="-20 -20">
      <SOURCE Dimensions="150 100" ClippingBox="30 50 160 90">
         <EXTERNAL_DATA Src="ppml.eps" />
      </SOURCE>
      <VIEW>
         <TRANSFORM Matrix="0.866 -0.5 0.5 0.866 -25.98 31.7" />
         <CLIP_RECT Rectangle="20 20 120 120" />
      </VIEW>
   </OBJECT>
</MARK>
```

The transformation component of this VIEW specifies a translation of (-25.98,31.7) and a rotation of –30°.



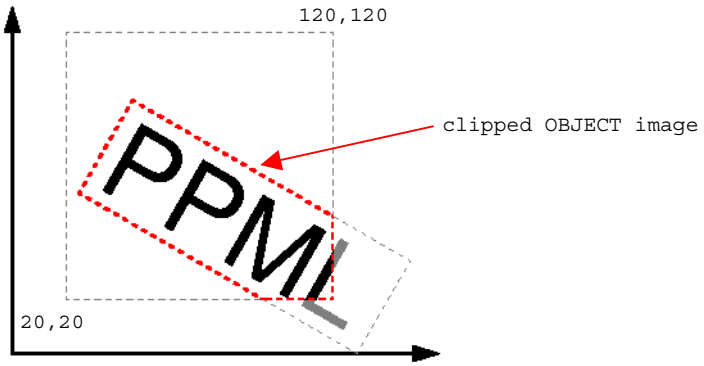
SOURCE origin
Offset -25.98,31.7
from OBJECT origin,
rotated -30°

☑ *Current coordinate space: the* OBJECT.

Now process the OBJECT's CLIP_RECT. This completes the VIEW, and thus completes the content of the OBJECT:

```
<MARK Position="30 40">
   <VIEW>
      <TRANSFORM Matrix="0.75 0 0 0.75 0 0" />
      <CLIP_RECT Rectangle="0 0 75 75" />
   </VIEW>
   <OBJECT Position="-20 -20">
      <SOURCE Dimensions="150 100" ClippingBox="30 50 160 90">
         <EXTERNAL_DATA Src="ppml.eps" />
      </SOURCE>
      <VIEW>
         <TRANSFORM Matrix="0.866 -0.5 0.5 0.866 -25.98 31.7" />
         <CLIP_RECT Rectangle="20 20 120 120" />
      </VIEW>
   </OBJECT>
</MARK>
```

The CLIP_RECT (20,20 to 120,120) clips the rotated image like this:

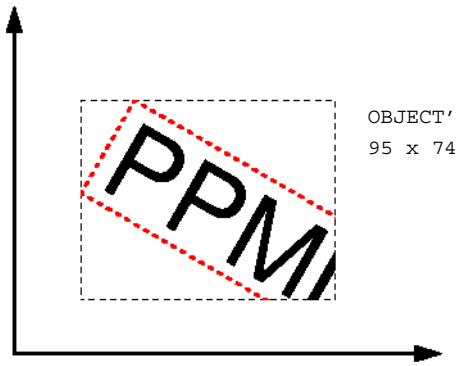


☑ *Current coordinate space: the* OBJECT.

Note

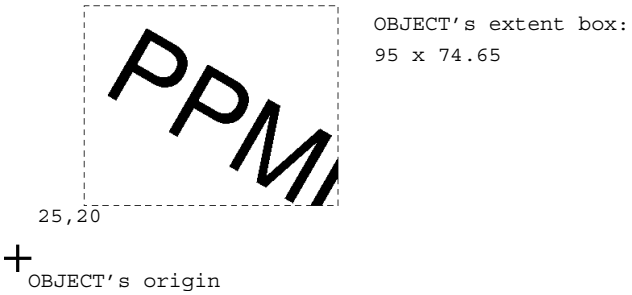
The drawings use color to highlight the clipping area.

Next, determine the extent box of this OBJECT element:



☑ *Current coordinate space: the* OBJECT.

The result is shown below. **This is the content that this** OBJECT **element defines:**

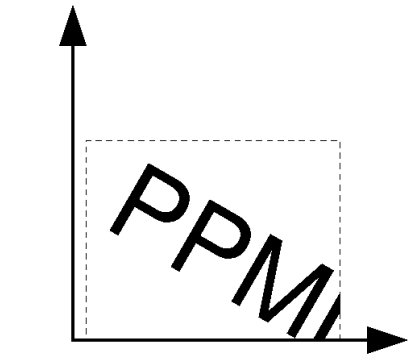


3. Place the OBJECT in the MARK, and apply the MARK's VIEW

A MARK can contain several OBJECTs, each with its own position. Thus, when each OBJECT is complete, its origin can be placed anywhere within the coordinates of its enclosing MARK element. This is done using the OBJECT element's Position attribute.

In this example the MARK contains only one OBJECT, positioned at (-20,-20).

```
<MARK Position="30 40">
   <VIEW>
      <TRANSFORM Matrix="0.75 0 0 0.75 0 0" />
      <CLIP_RECT Rectangle="0 0 75 75" />
   </VIEW>
   <OBJECT Position="-20 -20">
      <SOURCE Dimensions="150 100" ClippingBox="30 50 160 90">
         <EXTERNAL_DATA Src="ppml.eps" />
      </SOURCE>
      <VIEW>
         <TRANSFORM Matrix="0.866 -0.5 0.5 0.866 -25.98 31.7" />
         <CLIP_RECT Rectangle="20 20 120 120" />
      </VIEW>
   </OBJECT>
</MARK>
```

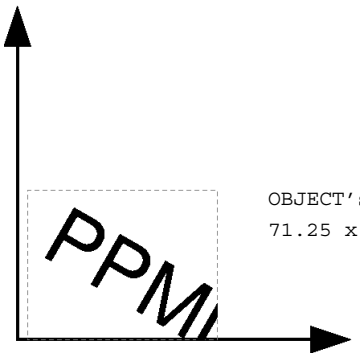


☑ Current coordinate space: the *MARK*.

Next, apply the MARK's TRANSFORM: scale the OBJECT to 75% of its original size:

```
<MARK Position="30 40">
   <VIEW>
      <TRANSFORM Matrix="0.75 0 0 0.75 0 0" />
      <CLIP_RECT Rectangle="0 0 75 75" />
   </VIEW>
   <OBJECT Position="-20 -20">
      <SOURCE Dimensions="150 100" ClippingBox="30 50 160 90">
         <EXTERNAL_DATA Src="ppml.eps" />
      </SOURCE>
      <VIEW>
         <TRANSFORM Matrix="0.866 -0.5 0.5 0.866 -25.98 31.7" />
         <CLIP_RECT Rectangle="20 20 120 120" />
      </VIEW>
   </OBJECT>
</MARK>
```

Result:



☑ Current coordinate space: the MARK.

Next, apply the MARK's CLIP_RECT: in this case, it does no extra clipping.

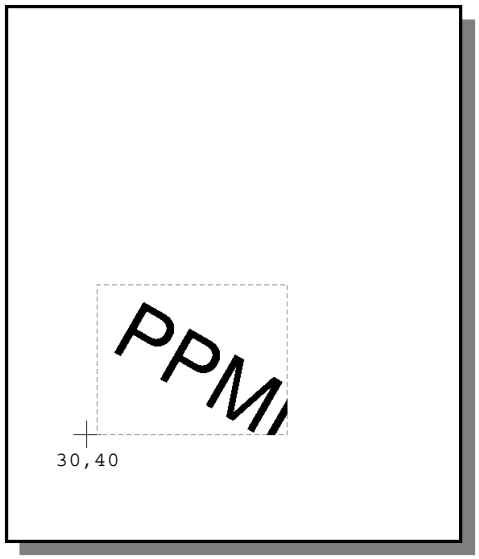
```
<MARK Position="30 40">
   <VIEW>
      <TRANSFORM Matrix="0.75 0 0 0.75 0 0" />
      <CLIP_RECT Rectangle="0 0 75 75" />
   </VIEW>
   <OBJECT Position="-20 -20">
      <SOURCE Dimensions="150 100" ClippingBox="30 50 160 90">
         <EXTERNAL_DATA Src="ppml.eps" />
      </SOURCE>
      <VIEW>
         <TRANSFORM Matrix="0.866 -0.5 0.5 0.866 -25.98 31.7" />
         <CLIP_RECT Rectangle="20 20 120 120" />
      </VIEW>
   </OBJECT>
</MARK>
```

The MARK's content is now complete. The content can now be positioned on the page, as shown below.

4. Position the MARK on the page.

The only remaining step is to process the MARK element's Position attribute.

```
<MARK Position="30 40">
   <VIEW>
      <TRANSFORM Matrix="0.75 0 0 0.75 0 0" />
      <CLIP_RECT Rectangle="0 0 75 75" />
   </VIEW>
   <OBJECT Position="-20 -20">
      <SOURCE Dimensions="150 100" ClippingBox="30 50 160 90">
         <EXTERNAL_DATA Src="ppml.eps" />
      </SOURCE>
      <VIEW>
         <TRANSFORM Matrix="0.866 -0.5 0.5 0.866 -25.98 31.7" />
         <CLIP_RECT Rectangle="20 20 120 120" />
      </VIEW>
   </OBJECT>
</MARK>
```



☑ Current coordinate space: the PAGE.

The entire MARK is now complete: the content has been marked onto the page.

```
<MARK Position="30 40">
  <VIEW>
    <TRANSFORM Matrix="0.75 0 0 0.75 0 0" />
    <CLIP_RECT Rectangle="0 0 75 75" />
  </VIEW>
  <OBJECT Position="-20 -20">
    <SOURCE Dimensions="150 100" ClippingBox="30 50 160 90">
      <EXTERNAL_DATA Src="ppml.eps" />
    </SOURCE>
    <VIEW>
      <TRANSFORM Matrix="0.866 -0.5 0.5 0.866 -25.98 31.7" />
      <CLIP_RECT Rectangle="20 20 120 120" />
    </VIEW>
  </OBJECT>
</MARK>
```

The following PostScript code could be placed before the EPS source to produce this result:

```
30 40 translate                                  % MARK position
0 0 75 75 rectclip                               % MARK clipping
[0.75 0 0 0.75 0 0] concat                       % MARK transform
-20 -20 translate                                % OBJECT position
20.0 20.0 100.0 100.0 rectclip                   % OBJECT clipping
[0.866 -0.5 0.5 0.866 -25.98 31.7] concat        % OBJECT transform
30.0 50.0 120.0 40.0 rectclip                    % SOURCE clipping
% insert content of file "ppml.eps" here
```

5.20.2 REUSABLE_OBJECT Example

This example renders the same MARK as the previous one, but uses a REUSABLE_OBJECT.

A REUSABLE_OBJECT has this structure:

- The simplest possible REUSABLE_OBJECT contains a VIEW, one OBJECT, and an OCCURRENCE_LIST with one OCCURRENCE.
- Each OCCURRENCE specifies a VIEW of all the OBJECTs in this REUSABLE_OBJECT.
- A MARK can include a particular OCCURRENCE of a REUSABLE_OBJECT by including an OCCURRENCE_REF.
- It only makes sense to use REUSABLE_OBJECT if its OCCURRENCEs are used in more than one MARK; it is probable (but not required) that the PPML Consumer will optimize the OBJECT for reuse.

To process a REUSABLE_OBJECT, the Consumer must first process each OBJECT inside it. And to do that, it first processes the SOURCE in the OBJECT. It is the same sequence as is used for OBJECTs within a MARK:

- Process the SOURCE, applying its ClippingBox if any
- Take the result and transform it using the TRANSFORM from the OBJECT's VIEW
- Take the result and clip it using the CLIP_RECT from the OBJECT's VIEW

This produces one OBJECT that will be contained in the REUSABLE_OBJECT.

Now, position the OBJECT in the REUSABLE_OBJECT's coordinate space.

Repeat the above for each OBJECT in the REUSABLE_OBJECT.

Now, apply the REUSABLE_OBJECT's VIEW:

- Take the set of (one or more) OBJECTs and transform it using the TRANSFORM from the REUSABLE_OBJECT's VIEW
- Take the result and clip it using the CLIP_RECT from the REUSABLE_OBJECT's VIEW

Now, apply each OCCURRENCE's VIEW:

- Take the result and transform it using the TRANSFORM from the OCCURRENCE's VIEW
- Take the result and clip it using the CLIP_RECT from the OCCURRENCE's VIEW
- Repeat the above for each OCCURRENCE in the OCCURRENCE _LIST

This process produces the final piece of page content for each OCCURRENCE. They are now ready to be included on a page with an OCCURRENCE_REF. The last step will be to position the content on the page, using the MARK's Position attribute.

The following PPML fragment achieves our desired result using a REUSABLE_OBJECT:

```
<REUSABLE_OBJECT>
   <OBJECT Position="-20 -20">
      <SOURCE Dimensions="150 100" ClippingBox="30 50 160 90">
         <EXTERNAL_DATA Src="ppml.eps" />
      </SOURCE>
      <VIEW>
         <TRANSFORM Matrix="0.866 -0.5 0.5 0.866 -25.98 31.7" />
         <CLIP_RECT Rectangle="20 20 120 120" />
      </VIEW>
   </OBJECT>
  <VIEW />
   <OCCURRENCE_LIST>
      <OCCURRENCE Name="example">
         <VIEW>
            <TRANSFORM Matrix="0.75 0 0 0.75 0 0" />
            <CLIP_RECT Rectangle="0 0 75 75" />
         </VIEW>
      </OCCURRENCE>
   </OCCURRENCE_LIST>
</REUSABLE_OBJECT>

<MARK Position="30 40">
   <OCCURRENCE_REF Ref="example" />
</MARK>
```

Note that the Format attribute of the SOURCE *has been omitted for clarity.*

A PPML Consumer processes this fragment using the following steps.

1. Create the OBJECT specified in the REUSABLE_OBJECT.

Use steps 1 and 2 from the previous example to obtain the OBJECT by reading its SOURCE and applying its VIEW.

```
<REUSABLE_OBJECT>
   <OBJECT Position="-20 -20">
      <SOURCE Dimensions="150 100" ClippingBox="30 50 160 90">
         <EXTERNAL_DATA Src="ppml.eps" />
      </SOURCE>
      <VIEW>
         <TRANSFORM Matrix="0.866 -0.5 0.5 0.866 -25.98 31.7" />
         <CLIP_RECT Rectangle="20 20 120 120" />
      </VIEW>
   </OBJECT>
  <VIEW />
   <OCCURRENCE_LIST>
      <OCCURRENCE Name="example">
         <VIEW>
            <TRANSFORM Matrix="0.75 0 0 0.75 0 0" />
            <CLIP_RECT Rectangle="0 0 75 75" />
         </VIEW>
      </OCCURRENCE>
   </OCCURRENCE_LIST>
</REUSABLE_OBJECT>
```

The result is shown below. **This is the content that this OBJECT element defines:**

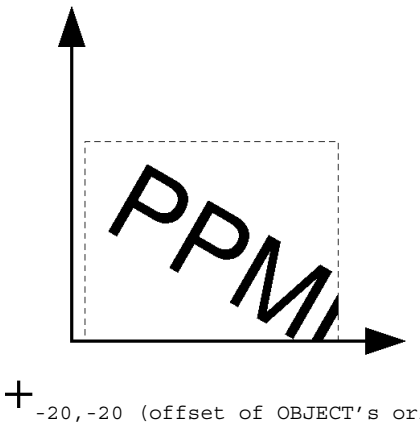
☑ *Current coordinate space: the* OBJECT

2. Place the OBJECT, and apply the REUSABLE_OBJECT's and OCCURRENCE's VIEWs.

A REUSABLE_OBJECT can contain several OBJECTs, each with its own position. Thus, when each OBJECT is complete, its origin can be placed anywhere within the coordinates of its enclosing REUSABLE_OBJECT element. This is done using the OBJECT element's Position attribute.

In this example, the OBJECT is positioned at (-20,-20).

```
<REUSABLE_OBJECT>
   <OBJECT Position="-20 -20">
      <SOURCE Dimensions="150 100" ClippingBox="30 50 160 90">
         <EXTERNAL_DATA Src="ppml.eps" />
      </SOURCE>
      <VIEW>
         <TRANSFORM Matrix="0.866 -0.5 0.5 0.866 -25.98 31.7" />
         <CLIP_RECT Rectangle="20 20 120 120" />
      </VIEW>
   </OBJECT>
  <VIEW />
   <OCCURRENCE_LIST>
      <OCCURRENCE Name="example">
         <VIEW>
            <TRANSFORM Matrix="0.75 0 0 0.75 0 0" />
            <CLIP_RECT Rectangle="0 0 75 75" />
         </VIEW>
      </OCCURRENCE>
   </OCCURRENCE_LIST>
</REUSABLE_OBJECT>
```



-20,-20 (offset of OBJECT's origin in the REUSABLE_OBJECT)

☑ Current coordinate space: the *REUSABLE_OBJECT*.

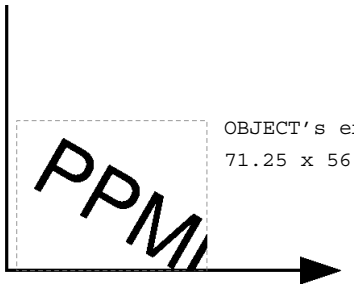
Next, apply the REUSABLE_OBJECT's VIEW: transform and clip the OBJECT as specified. In this example, the REUSABLE_OBJECT's VIEW is empty and no processing is required.

```
<REUSABLE_OBJECT>
  <OBJECT Position="-20 -20">
    <SOURCE Dimensions="150 100" ClippingBox="30 50 160 90">
      <EXTERNAL_DATA Src="ppml.eps" />
    </SOURCE>
    <VIEW>
      <TRANSFORM Matrix="0.866 -0.5 0.5 0.866 -25.98 31.7" />
      <CLIP_RECT Rectangle="20 20 120 120" />
    </VIEW>
  </OBJECT>
  <VIEW />
  <OCCURRENCE_LIST>
    <OCCURRENCE Name="example">
      <VIEW>
        <TRANSFORM Matrix="0.75 0 0 0.75 0 0" />
        <CLIP_RECT Rectangle="0 0 75 75" />
      </VIEW>
    </OCCURRENCE>
  </OCCURRENCE_LIST>
</REUSABLE_OBJECT>
```

Next, apply the OCCURRENCE's TRANSFORM: scale the OBJECT to 75% of its current size:

```
<REUSABLE_OBJECT>
  <OBJECT Position="-20 -20">
    <SOURCE Dimensions="150 100" ClippingBox="30 50 160 90">
      <EXTERNAL_DATA Src="ppml.eps" />
    </SOURCE>
    <VIEW>
      <TRANSFORM Matrix="0.866 -0.5 0.5 0.866 -25.98 31.7" />
      <CLIP_RECT Rectangle="20 20 120 120" />
    </VIEW>
  </OBJECT>
  <VIEW />
  <OCCURRENCE_LIST>
    <OCCURRENCE Name="example">
      <VIEW>
        <TRANSFORM Matrix="0.75 0 0 0.75 0 0" />
        <CLIP_RECT Rectangle="0 0 75 75" />
      </VIEW>
    </OCCURRENCE>
  </OCCURRENCE_LIST>
</REUSABLE_OBJECT>
```

Result:



☑ Current coordinate space: the OCCURRENCE.

Next, apply the OCCURRENCE's CLIP_RECT: in this case, it does no extra clipping.

```
<REUSABLE_OBJECT>
   <OBJECT Position="-20 -20">
      <SOURCE Dimensions="150 100" ClippingBox="30 50 160 90">
         <EXTERNAL_DATA Src="ppml.eps" />
      </SOURCE>
      <VIEW>
         <TRANSFORM Matrix="0.866 -0.5 0.5 0.866 -25.98 31.7" />
         <CLIP_RECT Rectangle="20 20 120 120" />
      </VIEW>
   </OBJECT>
  <VIEW />
   <OCCURRENCE_LIST>
      <OCCURRENCE Name="example">
         <VIEW>
            <TRANSFORM Matrix="0.75 0 0 0.75 0 0" />
            <CLIP_RECT Rectangle="0 0 75 75" />
         </VIEW>
      </OCCURRENCE>
   </OCCURRENCE_LIST>
</REUSABLE_OBJECT>
```

The OCCURRENCE's content is now complete.

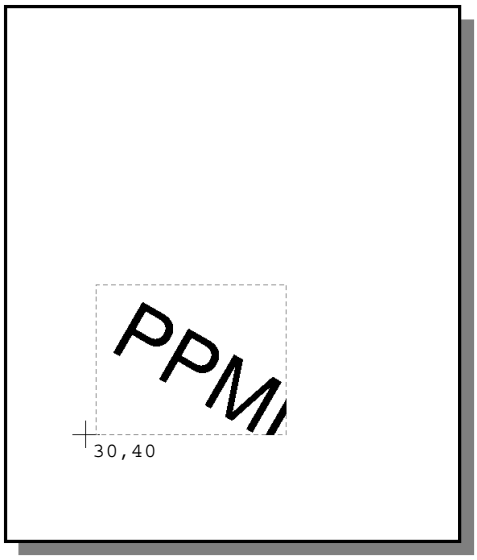
```
<REUSABLE_OBJECT>
   <OBJECT Position="-20 -20">
      <SOURCE Dimensions="150 100" ClippingBox="30 50 160 90">
         <EXTERNAL_DATA Src="ppml.eps" />
      </SOURCE>
      <VIEW>
         <TRANSFORM Matrix="0.866 -0.5 0.5 0.866 -25.98 31.7" />
         <CLIP_RECT Rectangle="20 20 120 120" />
      </VIEW>
   </OBJECT>
  <VIEW />
   <OCCURRENCE_LIST>
      <OCCURRENCE Name="example">
         <VIEW>
            <TRANSFORM Matrix="0.75 0 0 0.75 0 0" />
            <CLIP_RECT Rectangle="0 0 75 75" />
         </VIEW>
      </OCCURRENCE>
   </OCCURRENCE_LIST>
</REUSABLE_OBJECT>
```

3. Position the OCCURRENCE on the PAGE.

The only remaining step is to apply the MARK element's Position attribute to the OCCURRENCE created in step 2:

```
<REUSABLE_OBJECT>
   <OBJECT Position="-20 -20">
      <SOURCE Dimensions="150 100" ClippingBox="30 50 160 90">
         <EXTERNAL_DATA Src="ppml.eps" />
      </SOURCE>
      <VIEW>
         <TRANSFORM Matrix="0.866 -0.5 0.5 0.866 -25.98 31.7" />
         <CLIP_RECT Rectangle="20 20 120 120" />
      </VIEW>
   </OBJECT>
   <VIEW />
   <OCCURRENCE_LIST>
      <OCCURRENCE Name="example">
         <VIEW>
            <TRANSFORM Matrix="0.75 0 0 0.75 0 0" />
            <CLIP_RECT Rectangle="0 0 75 75" />
         </VIEW>
      </OCCURRENCE>
   </OCCURRENCE_LIST>
</REUSABLE_OBJECT>
```

```
<MARK Position="30 40">
   <OCCURRENCE_REF Ref="example" />
<MARK/>
```



☑ Current coordinate space: the PAGE.

The entire MARK is now complete: the content has been marked onto the page.

```
<MARK Position="30 40">
   <OCCURRENCE_REF Ref="example" />
</MARK>
```

Chapter 6: Print Layout – Page Layout and Imposition

6.1 Introduction

6.1.1 Imposition in personalized printing

In addition to its personalization features, the PPML language includes another important feature not found in most print languages: imposition. It's important to understand what imposition is and is not, especially in the context of personalized documents, which are a main purpose of the PPML language.

- Imposition is the process of positioning page images on sheets of paper in the printer (or in a digital printing press), as part of the process of producing finished documents.
- In addition to the page images, various marks can be added to the sheets, to aid in the production process. For instance, marks can be added to show where the paper should be folded or trimmed.
- Imposition has no effect on the content of any individual page – it only affects where the pages are placed on a press sheet.

Note: in this document, "imposition" (lowercase) refers to the functions described above. It does not refer to processing of the `IMPOSITION` element. "Imposing Consumers" are ones that process the `SHEET LAYOUT` element.

> **Note**
>
> PPML Consumers are not required to support the `SHEET_LAYOUT` element, nor the `Ncopies` and `Collate` attributes on `PRINT_LAYOUT`. This means a complex production job intended for a large-format digital printing press can be proof-printed on a simpler, small-format desktop printer. Similarly, a single-page production printer can print the dataset's document content stream (including copies and collation), ignoring imposition instructions.
>
> It also means a post-processing system can extract the document content stream (Document Sets, Documents and Pages) from a PPML dataset, and use other methods to assign pages to sheets, add sheet marks, etc.

Personalized printing requires imposition instructions that have never before been necessary.

In non-personalized printing, imposition is the placement of unchanging master pages onto a reproduction master, such as a printing plate.

But in digital printing of personalized documents, every copy is unique. Therefore, in addition to the regular imposition instructions, the language must also specify where to place each sequential copy of

the document (each Instance Document). Sometimes the next document starts on a separate sheet, sometimes it starts in the next row of the same sheet, sometimes it starts in the next column of the same sheet.

6.1.2 Overview of PPML elements for laying out the print job

This section provides a conceptual overview of how PPML pages are printed onto sheets as part of the overall production process. Each element is defined in its own section below.

Top level elements

```
<PRINT_LAYOUT>   includes:
   <PAGE_LAYOUT>   defines page size and cropping.
   <SHEET_LAYOUT>      defines the size of the sheet, the sheet marks (e.g. crop marks),
                       and all imposition instructions
```

Sheet layout elements

Sheet layout elements include imposition elements plus certain production marks that are associated with each sheet.

```
<SHEET_LAYOUT> includes:
   <SHEET_MARK>
   Imposition elements
```

Imposition elements

Imposition elements contain signature definitions and `REPEAT` elements:

```
<IMPOSITION>    includes:
   <SIGNATURE>     or
   <REPEAT>
```

Impositions can have a Name and can be referenced with `IMPOSITION_REF`.

Signature elements

Finally, the `SIGNATURE` element (and its surrounding `REPEAT`s, if any) define what is to be printed on a single sheet:

```
<SIGNATURE>
   <CELL>     defines the page order of each available position in the imposition layout:
              which location should receive the first Page, the second Page, and so on,
              and whether the Page should be rotated.
   Gutter locations & sizes (spaces between cells)
   Fold marks
   Trim Marks
```

6.1.3 Production Marks

"Production marks" are marks added to the sheet to assist in production; they are not part of document content. The Consumer may add production marks to a sheet after all the pages have

been imaged, or before the pages, or both. Production marks and PAGEs are imaged in the order they appear.

The following sections define each of the elements presented in this overview.

6.2 The <PRINT_LAYOUT> Element

6.2.1 Description

PRINT_LAYOUT is the master element that includes the page dimensions and how the Pages are to be laid out onto sheets by the Consumer.

6.2.2 Model

```
PRINT_LAYOUT (PAGE_LAYOUT, SHEET_LAYOUT?)
```

6.2.3 Example

The following illustrates a simple setup for printing letter-size pages onto 12x18" sheets. (Lower-level elements are omitted for this illustration.)

```
<PRINT_LAYOUT>
  <PAGE_LAYOUT  TrimBox="0 0 612 792"/>
  <SHEET_LAYOUT  HSize="1296" VSize="864">
    ...
  </SHEET_LAYOUT>
</PRINT_LAYOUT>
```

6.2.4 Attributes

Attribute	Required /Optional	Type	Description
Ncopies	Optional	Integer	How many copies to print of each sheet (for an imposing Consumer) or each page (for a non-imposing Consumer). Default=1.
Collate	Optional	Keyword	"Document" (default) = print the entire first copy of the document (all sheets, all pages), then print the entire second copy of the same document, etc. "DocSet" = print one copy of the entire Document Set (one copy of each document), then print the entire set again (another copy of each document), etc. "No" = print all copies of the first sheet (or page) of the first document, then print all copies of the second sheet (or page) of that document, etc.

Notice the distinction between the non-imposing consumer (which sees only pages) and the imposing consumer (which sees sheets). Both Ncopies and Collate make sense for both environments. The non-imposing consumer will copy and collate individual pages, while an imposing consumer will copy and collate full sheets.

Generally, uncollated output makes fewer demands on the Consumer's memory and may thus be the preferred mode when outputting to lower-powered products.

6.2.5 Context

PRINT_LAYOUT can occur in PPML and DOCUMENT_SET.

6.3 The <PAGE_LAYOUT> Element

6.3.1 Description

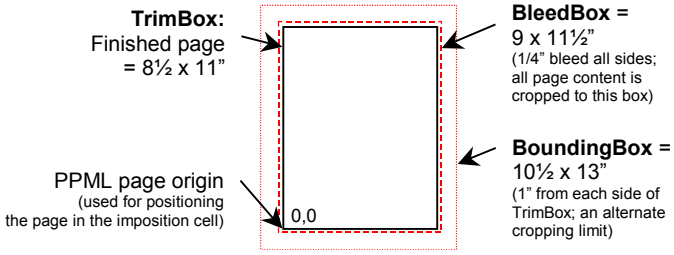
The PAGE_LAYOUT element describes page cropping information when using PPML's imposition. This element appears similar to PAGE_DESIGN because both have a TrimBox and BleedBox attribute. See section 4.6.6 for a discussion of the similarities and differences.

The PAGE_LAYOUT element states the rectangular dimensions of the Page. Three different dimensions are given: the trim box, the bleed box, and the bounding box. Example:

```
<PAGE_LAYOUT TrimBox="0 0 612 792"
   BleedBox="-18 -18 630 810"
   BoundingBox="-72 -72 684 864"
   />
```

The "Trim Box"

TrimBox indicates the final page size after trimming. The lower left corner of the trimmed page is the origin: when BleedBox or BoundingBox extends outside the trimmed page, its lower left corner will have negative coordinates, as shown in the PAGE_LAYOUT example above.



The "Bleed Box"

"Bleed" is the practice of intentionally allowing page content to extend a small distance beyond TrimBox. This is done to compensate for normal imperfections in the finishing process: if the trimming is not perfectly accurate, blank paper might be visible along the edge of the page. Extending the page image beyond TrimBox (i.e. using bleeds) avoids this.

The PAGE_LAYOUT element's BleedBox attribute specifies how far the image area is *allowed* to extend outside the page, but the allowed amount may not always be used. For instance, at the edge of a sheet, the entire specified bleed area is used. But within an imposed sheet (i.e. between two adjacent pages), the bleed extends into the gutter between the pages (if there is one) as follows:

- If there is no space (gutter) between the pages, then no bleed is needed at that edge. On that side, the Consumer crops the content of each page at each Page's TrimBox (which in this case is also the line where the two pages meet). At the outside borders of the signature the bleed would still be used.
- If there is a gutter, and it's less than or equal to the bleed, then the bleed fills the gutter. (The bleed from each page stops in the middle of the gutter.)
- If the gutter is wider than the bleed, the Consumer crops the page image at the BleedBox.

If no BleedBox is specified, BleedBox defaults to TrimBox.

TrimBox should not extend beyond BleedBox, but if it does, TrimBox will prevail.

The "Bounding Box"

The bounding box states the farthest uncropped extent of all objects on the page. In rare circumstances this may be useful as an alternative cropping boundary. It is expected to be used less frequently than `BleedBox` but will be of value in appropriate applications.

For instance, a PPML Page could consist of a single full-page object created by a desktop publishing application. Output from such applications typically includes production marks that fall outside the page area: crop marks, file identification information, etc. When the PPML page containing this object is imposed, the Producer has typically set `BleedBox` to a small value, so that all the application's production mark information is cropped out.

But when the same PPML page is proof-printed on a non-imposing printer, it may be preferable not to crop out those marks. With `BoundingBox`, a Producer can indicate the farthest uncropped extent of all objects on the page. The Consumer can honor `BoundingBox` instead of `BleedBox`, which allows printing page proofs that show the original application-provided marks outside the bleed area.

If `BoundingBox` is not specified, it defaults to `BleedBox`. If `BleedBox` extends beyond `BoundingBox`, then `BoundingBox` is set to the intersection of the two.

6.3.2 Model

```
PAGE_LAYOUT  EMPTY
```

6.3.3 Attributes

Attribute	Required /Optional	Type	Description
TrimBox	Required	Number × 4	Coordinates, in 1/72", of the trimmed size of the final page (i.e. after finishing).
BleedBox	Optional	Number × 4	Coordinates, in 1/72", of the page's bleed area. Defaults to `TrimBox`.
BoundingBox	Optional	Number × 4	Coordinates, in 1/72", of the maximum area that may need to be printed. Defaults to `BleedBox`.

6.3.4 Context

The `PAGE_LAYOUT` element appears within `PRINT_LAYOUT` and `SHEET_LAYOUT`.

6.3.5 Page orientation

All dimensions in the attributes are to be listed in "upright" orientation. For instance, a portrait letter-size page will have `TrimBox="0 0 612 792"` and a landscape letter-size page will have `TrimBox="0 0 792 612"`. Thus, no separate Orientation attribute is needed.

Note that any Page may be rotated when it is used in `IMPOSITION` and/or `SHEET_LAYOUT`. But the Page itself, and its content, are independent of imposition and printing.

6.4 The <SHEET_LAYOUT> Element

6.4.1 Description

In general, the SHEET_LAYOUT element contains all the elements that define what goes where on which sheet. It declares any marks that are associated with the sheet itself and what imposition instructions to use.

6.4.2 Model

```
SHEET_LAYOUT (SHEET_MARK | (PAGE_LAYOUT?, (IMPOSITION | IMPOSITION_REF)))*
```

6.4.3 Attributes

Attribute	Required /Optional	Type	Description
Hsize	Required	Number	Horizontal size of the sheet in 1/72"
Vsize	Required	Number	Vertical size of the sheet in 1/72"
GangDocuments	Optional	Boolean	Yes means all Instance Documents in a Document Set are to be concatenated into a single stream of pages for imposition. No (the default) means each Instance Document must start a new sheet. See also the PageOrder attribute of the CELL element (section 6.9.5, "Using expressions in the PageOrder attribute").

6.4.4 Context

SHEET_LAYOUT occurs within PRINT_LAYOUT.

6.4.5 Usage

Note that the model allows SHEET_MARK elements to come before or after imposition, or before *and* after Imposition elements. The Consumer must image the sheet in the sequence specified in SHEET_LAYOUT.

If SHEET_LAYOUT contains no child elements, then it defines nothing but the sheet size – it defines no imposition or sheet marks. In this case each page is centered on a sheet. If the PAGE's BleedBox is bigger than the sheet size, then the sheet size is used for cropping.

An optional PAGE_LAYOUT element may precede IMPOSITION or IMPOSITION_REF, in which case it replaces the previous PAGE_LAYOUT. This allows combining several different imposition schemes on the same sheet, including (optionally) different page sizes.

One set of page numbers applies to the whole sheet, even if it contains more than one IMPOSITION.

6.5 The <SHEET_MARK> Element

6.5.1 Description

The SHEET_MARK element places a Reusable Object at a specified location on every sheet. Applications of this feature are expected to include color control strips, the print shop's logo, or job ID information.

Note that a sheet mark may be placed anywhere on the sheet: the Producer may place sheet marks on top of page image areas if desired.The name of the Occurrence Reference is resolved immediately when the Sheet Mark element is encountered. That is, the OCCURRENCE content object named in the OCCURRENCE_REF element is retrieved immediately, such that even if the OCCURRENCE is renamed while the job is running, the appearance of the SHEET_MARK will not be affected.

Note that this element can only exist at the PPML or Document Set level (not Document or Page) because its enclosing SHEET_LAYOUT element can only appear at those levels. Therefore, the Occurrence used in a Sheet Mark cannot have a scope of Document or Page.

6.5.2 Model

```
SHEET_MARK (OCCURRENCE_REF)
```

6.5.3 Attributes

Attribute	Required /Optional	Type	Description
Position	Required	Number × 2	Location where the bottom left corner of the mark's bounding box is to be placed on the sheet.
Face	Optional	Keyword	Whether the Sheet Mark is to appear on the top of the sheet (Face="Up") or bottom of the sheet (Face="Dn"). Default=Up.

6.5.4 Context

SHEET_MARK occurs in SHEET_LAYOUT

6.5.5 Future considerations: variable sheet marks

Future versions may include the ability to imprint variable information in a sheet mark. Examples might include the date and time of the press run, a text string to identify which machine printed the sheets, a sheet number within the run or within the job, and so on.

6.6 The <IMPOSITION> Element

6.6.1 Description

The IMPOSITION element creates an imposition template, which immediately becomes the active imposition. The optional Name attribute allows saving it as a reusable template so it can be recalled with IMPOSITION_REF.

The IMPOSITION element can have two possible content structures:

- For multi-sheet applications, IMPOSITION can contain a SIGNATURE, e.g.:

```
<IMPOSITION>
  <SIGNATURE>  ...  </SIGNATURE>
</IMPOSITION>
```

- For applications where the document is smaller than one sheet, the IMPOSITION element can contain one REPEAT element (which may be nested) around one SIGNATURE element, for instance:

```
<IMPOSITION>
  <REPEAT Direction="Stack">
    <REPEAT Direction="Hor">
      <REPEAT Direction="Ver">
        <SIGNATURE>  ...  </SIGNATURE>
      </REPEAT>
    </REPEAT>
  </REPEAT>
</IMPOSITION>
```

6.6.2 Model

```
IMPOSITION  (SIGNATURE | REPEAT)
```

6.6.3 Attributes

Attribute	Required /Optional	Type	Description
Name	Optional	String	Optional identifying string for reference in a subsequent IMPOSITION_REF element.
Environment	Required if Scope= "Global"; not needed otherwise	String	Specifies the environment in which the Imposition should be defined. (There is no default environment.)
Scope	Optional	Keyword	Specifies the scope of this Imposition template's use. Possible values for Scope are Global, PPML, and DocSet. By default, the scope is the containing element in which the imposition is defined. A higher value may be specified in this attribute, but a lower value is an error.
Rotation	Optional	Integer	Rotation of the IMPOSITION content structure (the imposed set of signatures) relative to the sheet, counterclockwise: 0, 90, 180, 270 degrees. Default=0.
Position	Optional	Number × 2	Location where the bottom left corner of the rotated IMPOSITION content structure is to be placed on the sheet. If the Position attribute is not used, the entire structure is centered on the sheet. The imposition content structure is the logical structure that contains all the cells (including any empty cells) in the imposition scheme. It does not include any trim or fold marks.

6.6.4 Context

IMPOSITION can occur in SHEET_LAYOUT, DOCUMENT_SET, and PPML.

6.7 The <IMPOSITION_REF> Element

6.7.1 Description

The IMPOSITION_REF element recalls an imposition template that was previously defined. This enables the convenience of creating a library of standard imposition setups and reusing them.

6.7.2 Model

IMPOSITION_REF EMPTY

6.7.3 Attributes

Attribute	Required /Optional	Type	Description
Name	Required	String	Name attribute of the imposition template previously defined.
Environment	Optional	String	The environment in which the name of a Global imposition template should be interpreted. (This attribute is required if the scope of the template is Global; otherwise, this attribute has no meaning.)
Rotation	Optional	Number	Rotation of the IMPOSITION content structure, counterclockwise: 0, 90, 180, 270 degrees. Default=0.
Position	Optional	Number × 2	Location where the bottom left corner of the IMPOSITION content structure is to be placed on the sheet. If the Position attribute is not used, the entire structure is centered on the sheet.

6.7.4 Context

The IMPOSITION_REF element occurs in SHEET_LAYOUT.

6.7.5 Implementation notes

Calling for a stored imposition template by name has advantages but also has a side effect. Producers should be conscious of this.

One advantage is that the dataset can be marginally smaller. Another is that it may be simpler for the Producer to output a simple name than to regenerate all the imposition instructions. Perhaps most important, though, is that if a dataset uses IMPOSITION_REF to call for a template by name, then the latest version of that template will automatically be retrieved. This means that if a shop has refined its template, the updates will automatically be implemented in any dataset that uses that template.

But it also means that the dataset no longer has complete control of the imposition: by definition, IMPOSITION_REF means "I don't care what imposition is stored under this name – use it."

If the Producer requires absolute control of the imposition for a job, it should explicitly define the imposition in the dataset, using IMPOSITION and its child elements. (The dataset can still use IMPOSITION_REF to call the imposition by name later in the dataset; the point is that the imposition is only certain if it's defined within the dataset that references it.)

6.8 The <SIGNATURE> Element

6.8.1 Description

A **signature** is a set of one or more pages from an Instance Document, printed on a single sheet of paper. The pages are arranged in a specific sequence, and are printed on one or both sides of the sheet.

The SIGNATURE element defines a uniform cell grid defined by Nrows and Ncols. The size of the cells in the grid is not specified by the imposition layout, but is defined by the TrimBox attribute of the PAGE_LAYOUT of the document that is imposed. The HOR_GUTTER and VER_GUTTER elements define the spacing between the cells in the grid.

Note that every cell has the same size. Specifically, the Rotation attribute of the CELL is not used to determine the size of a cell.

Once this grid is defined, the BleedBox in the PAGE_LAYOUT defines the clipping rectangle for each cell depending on the gutters and the relative position in the grid.

The Rotation and Position in the IMPOSITION element determine how and where this grid is positioned.

6.8.2 Model

```
SIGNATURE    (CELL+, HOR_TRIM_MARKS?, VER_TRIM_MARKS?,
             HOR_GUTTER*, VER_GUTTER*, HOR_FOLD_MARKS*, VER_FOLD_MARKS*)
```

6.8.3 Attributes

Attribute	Required /Optional	Type	Description
Nrows	Required	Integer	The number of rows in this signature.
Ncols	Required	Integer	The number of columns in this signature.
PageCount	Optional	Integer	The number of *different* pages consumed by this signature. (See section 6.8.5 below.) Default is the number of CELL elements in this signature.

6.8.4 Context

The SIGNATURE element can occur in IMPOSITION and REPEAT.

6.8.5 PageCount applications

PageCount specifically states how many different pages the Producer has assigned to this Signature. Typically this equals the number of CELL elements, but that is not required.

For instance, in an eight-page Signature the Producer may choose to assign only four or six pages to the signature, and that's the number that would be assigned to PageCount. As another example, a Producer may want to assign the same page to multiple locations in the same signature – for

instance it may duplicate the second page on the signature, for some reason. In that case when the Producer calculates PageCount, it would ignore those duplicates, counting only how many *different* pages are assigned to the Signature.

6.9 The <CELL> Element

6.9.1 Description

The CELL element assigns Pages to specific locations on a Signature. For each Page, it specifies the row and column position within the signature, whether the Page is to be printed on the face-up or face-down side of the sheet, and whether the page content is to be rotated in the cell.

One CELL element may be used for each page position on either side of the signature. No CELL element has to be specified for positions that are empty.

The TrimBox attribute of the PAGE_LAYOUT used to instantiate the IMPOSITION template determines the actual size of every Cell within the Signature.

The Rotation attribute of the CELL determines how the page content is placed inside the Cell. It does not affect its size or bleed area. E.g. if the Rotation is 90, the page content is rotated 90 degrees counterclockwise around the center of the cell.

No trim marks will be generated for missing cells.

6.9.2 Model

```
CELL   EMPTY
```

6.9.3 Attributes

Attribute	Required /Optional	Type	Description
Row	Required	Integer	Row number of the cell being defined. Top row=1.
Col	Required	Integer	Column number of the cell being defined. Left column=1.
Face	Optional	Keyword	Whether the Page is to appear on the top of the sheet (Face="Up") or bottom of the sheet (Face="Dn"). Default=Up.
PageOrder	Required	String	Defines the sequence number of the Page to be placed in this cell. Can be an integer or an expression. See description and example in paragraphs 6.9.5 and 6.9.6 below.
Rotation	Optional	Integer	Rotation of the Page, counterclockwise: 0, 90, 180, 270. Default=0.

6.9.4 Context

The CELL element occurs in SIGNATURE.

6.9.5 Using expressions in the PageOrder attribute

Expressions can use the operators +, – , *, /, and parentheses, operating on integers and two variables: ***s*** for sheet number (starting at 1) and ***n*** for number of pages to be imposed. Expressions are evaluated with normal 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, PPML imposition templates can automatically assign pages to the correct Signature and Cell position. To use this feature, the Producer should specify the `PageOrder` attribute using expressions based on ***n***.

The variable ***n*** depends on ***p***, the total number of pages that need to be imposed.
This number ***p*** on its turn depends on the value of the `GangDocuments` attribute of `SHEET_LAYOUT`:

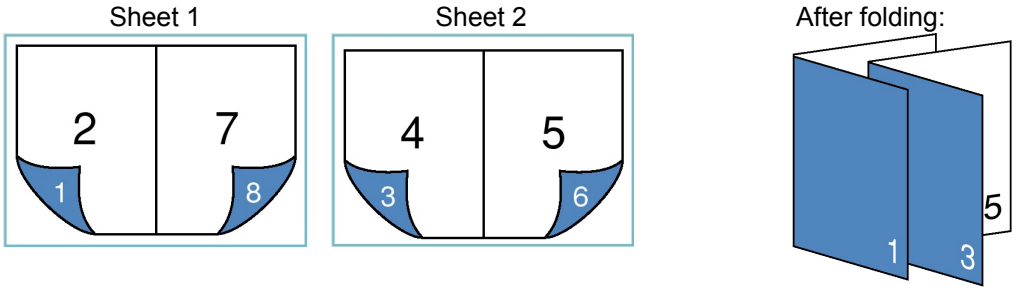
- `GangDocuments="No"` means each Instance Document must start on a new Sheet. In this case, ***p*** refers to the number of pages in the current Instance Document, and the Consumer will evaluate the `PageOrder` expression separately for each Instance Document.
- `GangDocuments="Yes"` means all Instance Documents are to be concatenated into a single stream of pages for imposition. In this case, ***p*** refers to the total page count (the sum of page counts for all documents in the Document Set) and `PageOrder` refers to a page's position in the concatenated stream of pages, not its position within its parent Document.

In both cases, ***n*** is derived from ***p*** according to the following rule: let ***c*** be the sum of all the `PageCount` attributes of all the `SIGNATURE` elements in the `SHEET_LAYOUT`, then ***n*** is ***p*** rounded up to the nearest multiple of ***c***. The number of signatures generated will be ***n/c***.

Any cell that has a resulting `PageOrder` attribute greater than ***p*** or less than 1 is left blank. For instance, if ***c*** equals 4, and an Instance Document contains 7 pages, then ***n*** for that document is 8, and the last cell will have no content.

6.9.6 Examples

This example shows an eight-page job being assigned to the cells of two four-page signatures.



The cell assignments shown in the above diagram for the eight pages are as follows. Pages that get assigned to the second signature are shown center-aligned so they're easy to recognize; notice that within each signature, the page sequence (as shown in the illustrations) is Down Up Up Down.

Page #	Signature	Row	Column	Face
1	1	1	1	Down
2	1	1	1	Up
3	2	1	1	Down
4	2	1	1	Up
5	2	1	2	Up
6	2	1	2	Down
7	1	1	2	Up
8	1	1	2	Down

This two-signature imposition can be described in a more general form, so that it handles any number of pages, and will automatically generate additional signatures as needed to accommodate those pages. This is done by using *one* 4-cell `SIGNATURE` element, with each `PageOrder` attribute being an expression f of ***s***, the sheet number in the above table. The general form will be as follows. (Note: "$f_n(\boldsymbol{s})$" is not part of the PPML code – it will be explained below.)

```
<IMPOSITION Name="2 x 2-UP Bundled">
  <SIGNATURE Nrows="1", Ncols="2">
    <CELL Row="1" Col="1" PageOrder= f1(s) Face="Up" Rotate="0"/>
    <CELL Row="1" Col="1" PageOrder= f2(s) Face="Dn" Rotate="0"/>
    <CELL Row="1" Col="2" PageOrder= f3(s) Face="Up" Rotate="0"/>
    <CELL Row="1" Col="2" PageOrder= f4(s) Face="Dn" Rotate="0"/>
  </SIGNATURE>
</IMPOSITION>
```

Each of the expressions $f_1(\boldsymbol{s})$ … $f_4(\boldsymbol{s})$ is of the form $f_n(\boldsymbol{s})$=a*$\boldsymbol{s}$ –b.

The following illustrates how to determine f_2(s), the expression for the second `CELL` element.

We find the values of a and b by rewriting the expression "a*$\boldsymbol{s}$ –b" for the two entries in the above table that have `Row=1`, `Col=1` and `Face=Down` (the first and third entry from the table). We know the result must be the page number shown in the first column of the table:

```
1 = a*1 - b    (first entry from table)
3 = a*2 - b    (third entry from table)
```

Resolving this for a and b gives a=2 and b=1. So the second `CELL` element becomes:

```
<CELL Row="1" Col="1" PageOrder="2*s-1" Face="Dn" Rotate="0"/>
```

Doing this for all four cells, the final code is:

```
<IMPOSITION Name="2 x 2-UP Bundled">
  <SIGNATURE Nrows="1", Ncols="2">
     <CELL Row="1" Col="1" PageOrder="2*s" Face="Up" Rotate="0"/>
     <CELL Row="1" Col="1" PageOrder="2*s-1" Face="Dn" Rotate="0"/>
     <CELL Row="1" Col="2" PageOrder="9-2*s" Face="Up" Rotate="0"/>
     <CELL Row="1" Col="2" PageOrder="10-2*s" Face="Dn" Rotate="0"/>
  </SIGNATURE>
</IMPOSITION>
```

The true power of using expressions in the `PageOrder` attribute is shown by generalizing the above for any n-page document, n being a multiple of 4:

```
<IMPOSITION Name="2 x 2-UP Bundled">
  <SIGNATURE Nrows="1", Ncols="2">
     <CELL Row="1" Col="1" PageOrder="2*s" Face="Up" Rotate="0"/>
     <CELL Row="1" Col="1" PageOrder="2*s-1" Face="Dn" Rotate="0"/>
     <CELL Row="1" Col="2" PageOrder="n+1-2*s" Face="Up" Rotate="0"/>
     <CELL Row="1" Col="2" PageOrder="n+2-2*s" Face="Dn" Rotate="0"/>
  </SIGNATURE>
</IMPOSITION>
```

If we instead want to fold each sheet first, then gather them together, the page assignment scheme would follow the same generic sequence but it would allocate pages 1-4 to the first signature, and 5-8 to the second signature, as follows:

Page #	Signature	Row	Column	Face
1	1	1	1	Down
2	1	1	1	Up
3	1	1	2	Up
4	1	1	2	Down
5	2	1	1	Down
6	2	1	1	Up
7	2	1	2	Up
8	2	1	2	Down

The resulting imposition is:

```
<IMPOSITION Name="2 x 2-UP">
  <SIGNATURE Nrows="1", Ncols="2">
     <CELL Row="1" Col="1" PageOrder="4*s-2" Face="Up" Rotate="0"/>
     <CELL Row="1" Col="1" PageOrder="4*s-3" Face="Dn" Rotate="0"/>
     <CELL Row="1" Col="2" PageOrder="4*s-1" Face="Up" Rotate="0"/>
     <CELL Row="1" Col="2" PageOrder="4*s-0" Face="Dn" Rotate="0"/>
  </SIGNATURE>
</IMPOSITION>
```

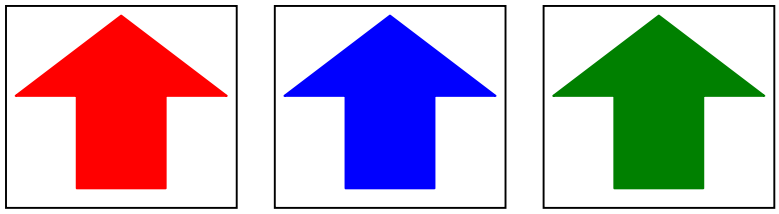
Notice that in this second example, every sheet is independent from the previous one, which is reflected by the `PageOrder` expressions being independent of ***n***.

6.9.7 Cell Rotation Example

This example shows the effect of the `Rotation` attribute in a `CELL`. Note that some attributes have been omitted for clarity.

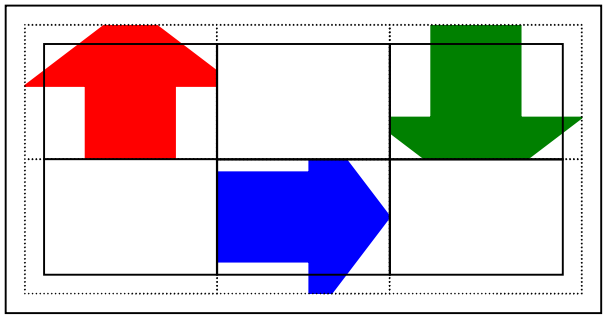
```
<SHEET_LAYOUT>
  <IMPOSITION>
    <SIGNATURE Nrows="2" Ncols="3">
      <CELL Row="1" Col="1" PageOrder="3*s" Rotation="0" />
      <CELL Row="2" Col="2" PageOrder="3*s+1" Rotation="270" />
      <CELL Row="1" Col="3" PageOrder="3*s+2" Rotation="180" />
    </SIGNATURE>
  </IMPOSITION>
</SHEET_LAYOUT>
```

The input document contains these three pages :



The active `PAGE_LAYOUT` is as follows:

Executing the imposition gives the following result:

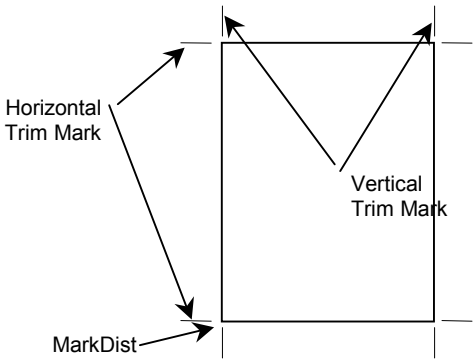


Note that the trim and bleed boxes are shown in this drawing. They will not be visible in the actual PPML output.

6.10 The <HOR_TRIM_MARKS> Element

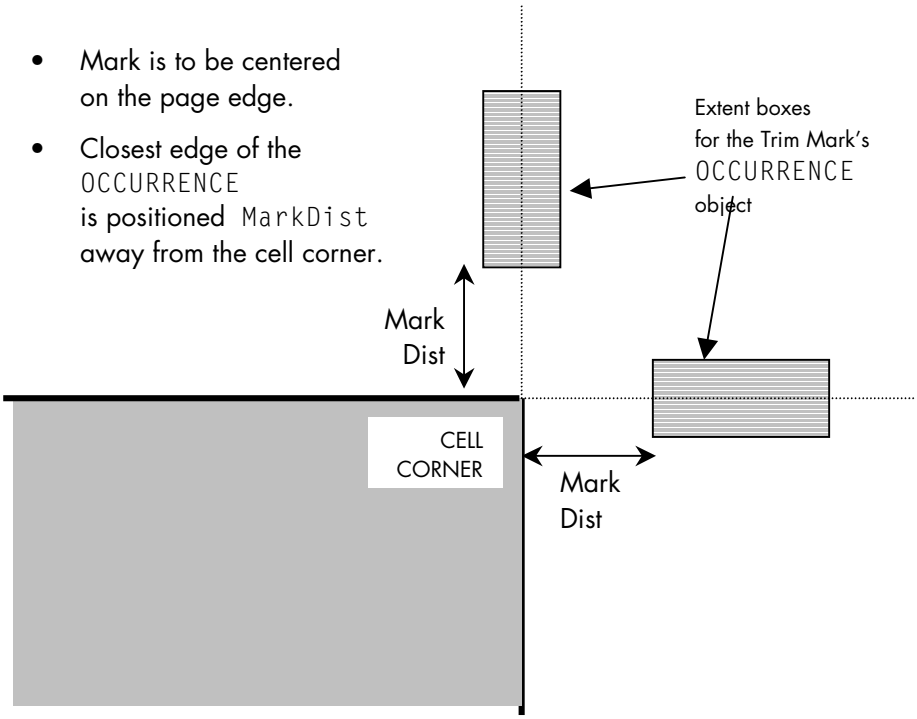
6.10.1 Description

Trim Marks are Reusable Object Occurrences that can be automatically placed by the Consumer on each sheet, at the corners of the final pages, on both sides of the sheet (front and back), using the `HOR_TRIM_MARKS and VER_TRIM_MARKS` elements.



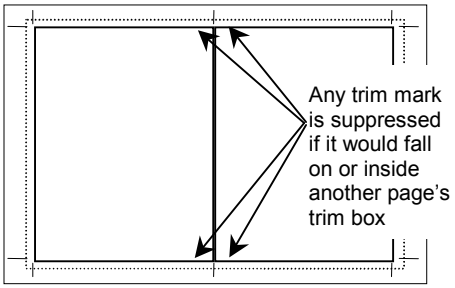
Each mark is centered on one boundary of the TrimBox. The attribute `MarkDist` specifies the mark's distance from the corner of the page.

- Mark is to be centered on the page edge.
- Closest edge of the `OCCURRENCE` is positioned `MarkDist` away from the cell corner.



The mark is printed without rotation or mirroring – for instance the vertical trim mark at the top of the page will be identical, relative to the sheet, to the mark at the bottom of the page.

If a signature has pages that touch, or nearly touch, as shown at right, some trim marks would fall onto the TrimBox of their neighboring pages. A trim mark is suppressed if any part of its bounding box falls closer than MarkDist to a neighboring trim box. An optional attribute `AllowOnPage="Yes"` (default = `"No"`) can overrule this suppression.



The OCCURRENCE_REF may only refer to a reusable object with a scope at least as high as the scope of the IMPOSITION element enclosing this mark. It is an error to refer to a mark which is in scope, but which has a scope lower than that of the enclosing IMPOSITION element.

6.10.2 Model

```
HOR_TRIM_MARKS (OCCURRENCE_REF)
```

6.10.3 Attributes

Attribute	Required /Optional	Type	Description
MarkDist	Optional	Number	Distance of the mark away from the page, in 1/72"
AllowOnPage	Optional	Boolean	Default= "No". If Yes, Trim Marks will *not* be suppressed if they fall on or inside another page's trim box.

6.10.4 Context

HOR_TRIM_MARKS can occur in SIGNATURE.

6.10.5 Example

The following example shows how Trim Marks would be coded using two Reusable Object Occurrences named VerTrim and HorTrim. Each is to be positioned six points from the corner of its page.

Note that the Trim Marks elements are unaffected by how many cells are in the signature; they simply declare whether or not the signature has trim marks.

```
<IMPOSITION>
   <SIGNATURE Nrows="1" Ncols="2">
      <CELL .../>
      <CELL .../>
      <HOR_TRIM_MARKS MarkDist="6">
         <OCCURRENCE_REF Ref="HorTrim">
      </HOR_TRIM_MARKS>
      <VER_TRIM_MARKS MarkDist="6">
         <OCCURRENCE_REF Ref="VerTrim">
      </HOR_TRIM_MARKS>
   </SIGNATURE>
</IMPOSITION>
```

6.11 The <VER_TRIM_MARKS> Element

6.11.1 Description

The VER_TRIM_MARKS element is the vertical equivalent to HOR_TRIM_MARKS. See HOR_TRIM_MARKS (section 6.10) for description, illustration, and example.

The OCCURRENCE_REF may only refer to a reusable object with a scope at least as high as the scope of the IMPOSITION element enclosing this mark. It is an error to refer to a mark which is in scope, but which has a scope lower than that of the enclosing IMPOSITION element.

6.11.2 Model

```
VER_TRIM_MARKS (OCCURRENCE_REF)
```

6.11.3 Attributes

Attribute	Required /Optional	Type	Description
MarkDist	Optional	Number	Distance of the mark away from the page, in 1/72"
AllowOnPage	Optional	Boolean	Default= "No". If Yes, Trim Marks will *not* be suppressed if they fall on or inside another page's trim box.

6.11.4 Context

VER_TRIM_MARKS can occur in SIGNATURE.

6.12 The <HOR_GUTTER> Element

6.12.1 Description

The horizontal gutter is a horizontal strip of space between two rows of cells in a signature.

The `BetweenRows` attribute specifies the set of rows between which this gutter should be inserted. For instance, the following code shows a signature with `NRows="3"` and gutters between all rows:

```
<IMPOSITION>
   <SIGNATURE Nrows="3" Ncols="2">
      <CELL Row="1" Col="1" PageOrder="3" Face="Up" Rotate="180"/>
      ...
      <CELL Row="2" Col="2" PageOrder="8" Face="Dn" Rotate="0"/>
      <HOR_GUTTER BetweenRows="1 3" Distance="18"/>
   </SIGNATURE>
</IMPOSITION>
```

It is also possible to specify a different `HOR_GUTTER` element for each space between rows:

```
<IMPOSITION>
   <SIGNATURE Nrows="2" Ncols="2">
      <CELL Row="1" Col="1" PageOrder="3" Face="Up" Rotate="180"/>
      ...
      <CELL Row="2" Col="2" PageOrder="8" Face="Dn" Rotate="0"/>
      <HOR_GUTTER BetweenRows="1 2" Distance="36"/>
      <HOR_GUTTER BetweenRows="2 3" Distance="18"/>
   </SIGNATURE>
</IMPOSITION>
```

Each `HOR_GUTTER` element affects only the rows identified in BetweenRows – previous gutter settings for other gutters are unaffected. For instance, this code for an 8-row signature defines uniform spacing for all rows, then changes the value for the middle gutter to 1":

```
   <SIGNATURE Nrows="8" Ncols="1">
      ...
      <HOR_GUTTER BetweenRows="1 8" Distance="18"/>
      <HOR_GUTTER BetweenRows="4 5" Distance="72"/>
   </SIGNATURE>
```

6.12.2 Model

```
HOR_GUTTER  EMPTY
```

6.12.3 Attributes

Attribute	Required /Optional	Type	Description
Distance	Required	Number	Size (height) of the gutter, in 1/72″
BetweenRows	Required	Integer ×2	Identifies the set of rows between which this Distance applies. See examples. Top row = 1.

6.12.4 Context

HOR_GUTTER occurs in SIGNATURE.

6.13 The <VER_GUTTER> Element

6.13.1 Description

The VER_GUTTER element is identical to HOR_GUTTER except that it defines a vertical strip of space between two columns, not rows, of cells in a signature. See the description of HOR_GUTTER, section 6.12.1, for examples and explanation of attributes.

6.13.2 Model

```
VER_GUTTER   EMPTY
```

6.13.3 Attributes

Attribute	Required /Optional	Type	Description
Distance	Required	Number	Size (width) of the gutter, in 1/72″
BetweenCols	Required	Integer × 2	Identifies the set of columns between which this Distance applies. See examples. Columns are numbered from left to right.

6.13.4 Context

VER_GUTTER occurs in SIGNATURE.

6.14 The <HOR_FOLD_MARKS> Element

6.14.1 Description

The HOR_FOLD_MARKS element specifies a pair of horizontal fold marks between two specified rows of a Signature – an Occurrence of a Reusable Object that will print outside the left and right edges of the Signature.

If fold marks are defined between two cells, the trim marks on the two corners of each cell closest to the fold marks are suppressed. Fold marks are also suppressed if any part of its bounding box falls closer than MarkDist from the trim box of a neighboring cell.

The name of the Reusable Object Occurrence is resolved immediately when this element is encountered. The OCCURRENCE_REF may only refer to a reusable object with a scope at least as high as the scope of the IMPOSITION element enclosing this mark. It is an error to refer to a mark which is in scope, but which has a scope lower than that of the enclosing IMPOSITION element.

6.14.2 Model

```
HOR_FOLD_MARKS (OCCURRENCE_REF)
```

6.14.3 Attributes

Attribute	Required /Optional	Type	Description
BetweenRows	Required	Integer × 2	Rows between which the fold line exists
MarkDist	Optional	Number	Distance, in 1/72", between the outermost page of the signature and the start of the Reusable Object. (Positive value = away from the signature.)

6.14.4 Context

HOR_FOLD_MARKS occurs in SIGNATURE.

6.14.5 Example

```
<HOR_FOLD_MARKS BetweenRows="1 2" MarkDist="6">
   <OCCURRENCE_REF  Ref="HDashedLine"/>
</HOR_FOLD_MARKS>
```

6.15 The <VER_FOLD_MARKS> Element

6.15.1 Description

The VER_FOLD_MARKS element specifies a pair of vertical fold marks between two specified columns of a Signature – an Occurrence of a Reusable Object that will print outside the top and bottom edges of the Signature.

The mark will be centered on the fold line, at a distance specified by the MarkDist attribute.

If fold marks are defined between two cells, the trim marks on the two corners of each cell closest to the fold marks are suppressed. Fold marks are also suppressed if any part of its bounding box falls closer than MarkDist from the trim box of a neighboring cell.

The name of the Reusable Object Occurrence is resolved immediately when this element is encountered. The OCCURRENCE_REF may only refer to a reusable object with a scope at least as high as the scope of the IMPOSITION element enclosing this mark. It is an error to refer to a mark which is in scope, but which has a scope lower than that of the enclosing IMPOSITION element.

6.15.2 Model

```
VER_FOLD_MARKS (OCCURRENCE_REF)
```

6.15.3 Attributes

Attribute	Required /Optional	Type	Description
BetweenCols	Required	Integer × 2	Columns between which the fold line exists
MarkDist	Optional	Number	Distance, in 1/72", between the outermost page of the signature and the start of the Reusable Object. (Positive value = away from the signature.)

6.15.4 Context

VER_FOLD_MARKS occurs in SIGNATURE.

6.15.5 Example

```
<VER_FOLD_MARKS BetweenCols="1 2" MarkDist="6">
   <OCCURRENCE_REF  Ref="VFoldMark"/>
</VER_FOLD_MARKS>
```

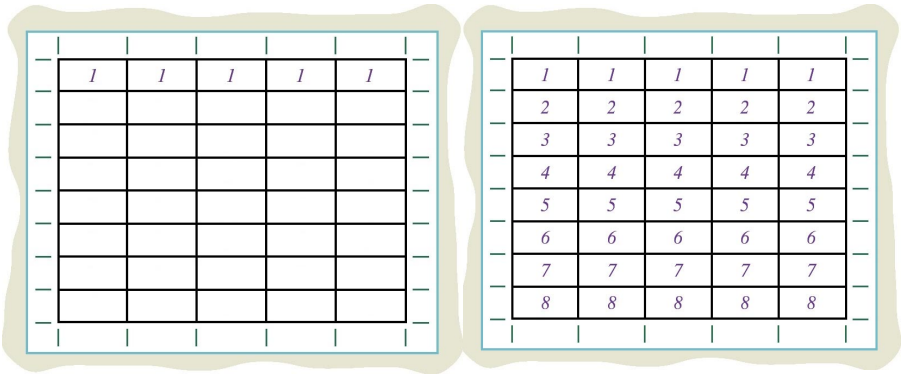
6.16 The <REPEAT> Element

6.16.1 Description

An imposition template allows printing multiple *pages* on a signature, from one Instance Document. In contrast, the `REPEAT` element allows printing signatures from multiple *documents* on a single sheet. It also controls the distribution of different Instance Documents throughout the print run.

Three attributes control the effect of each `REPEAT` element: Direction, Action, and Count. The elements can be nested, with different values in each element. When `REPEAT` elements are nested, they are executed from innermost to outermost. For instance, the following code could be used in creating a sheet of five identical columns of eight different business cards (see illustration). (Inner elements have been omitted for this illustration.)

```
<REPEAT Direction="Ver" Action="Increment" Count="8">
   <REPEAT Direction="Hor" Action="Duplicate" Count="5">
      <SIGNATURE...>.... </SIGNATURE>
   </REPEAT>
</REPEAT>
```



1: The inner `REPEAT` prints the first card in the top row of each column:
`<REPEAT Direction="Hor" Action="Duplicate" Count="5">`

2: The second `REPEAT` does the same on each row, with a different card:
`<REPEAT Direction="Ver" Action="Increment" Count="8">`

Printing pre-sorted stacks: If the attribute values are `Direction="Stack" Action="Increment"`, `REPEAT` puts the next Signature on the next sheet. That is, the signatures will repeat through the stack of sheets, producing a stack of pre-sorted documents.

In such applications, a Consumer may wish to print the *last* sheet first, so it ends up at the bottom of the stack. To support such applications, the optional attribute `Order="Descending"` can be used.

Nested `REPEAT`s using `Action="Increment"`: When multiple nested `REPEAT`s have `Action="Increment"`, an additional counter ***d*** (document counter) is applied. In every step of a `REPEAT` with action="Increment", ***d*** is incremented by 1, while ***s*** remains the overall sheet counter. For instance, in the following example the inner `REPEAT` has `Direction="Ver" Count="3"` so the Consumer will first put three signatures in a column. The outer `REPEAT` has

`Direction="Hor" Count="4"` so the whole column will be repeated four times, incrementing the counter ***d*** continuously:

```
<REPEAT  Action="Increment"  Direction="Hor"  Count="4">
   <REPEAT Action="Increment" Direction="Ver"  Count="3">
      <SIGNATURE....>
   </REPEAT>
</REPEAT>
```

These are the values of ***d*** for the resulting sheet:

1	4	7	10
2	5	8	11
3	6	9	12

If the signature has one cell, with `PageOrder="s"`, then one should impose on the first sheet the first page of document 1, below it the first page of document 2 and so on.

When the counting of documents is incremented in the stack direction the counter ***s*** starts over from 1. If different documents start on the same sheet and they have different number of pages, then the next set of documents starts only after the longest document ends. For example:

```
<REPEAT Action="Increment" Direction="Hor" Count="2">
 <SIGNATURE Nrows="1" Ncols="1">
   <CELL Row="1" Col="1" PageOrder="s"/>
 </SIGNATURE>
</REPEAT>
```

If document 1 has 1 page and document 2 has 2 pages then this is the page distribution:

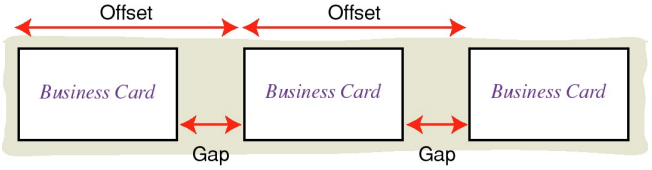
```
Page 1:  document 1 page 1, document 2 page 1   (S=1)
Page 2:                   , document 2 page 2   (S=2)
Page 3:  document 3 page 1, document 4 page 1   (S=1)
```

Notice also that when the count of documents imposed reaches the total count in the imposition template, the consumer keeps imposing the following documents, as if a global imaginary `REPEAT` with count "infinity" encompasses all other `REPEAT`s. In other words, let N be the number of documents that a `REPEAT` and all its nested `REPEATS` consume. This is equal to the product of all Count attributes of the `REPEAT` (and all its nested ones), that have an attribute `Action=increment`. When all the sheets for these N documents are generated, the whole process starts again for the next N documents.

In the last example the imposition template imposes two documents (one `REPEAT` element with count="2"). In such a case the Consumer imposes the first two documents and then imposes the next two documents and so on until all documents are imposed.

If there is more than one `IMPOSITION` element in `SHEET_LAYOUT`, the counter ***d*** increments independently for each `IMPOSITION`.

Spacing of Signatures – the attributes `Spacing` and `SpacingMethod`: By default, signatures are repeated with no space between them: the TrimBox of the next Instance Document touches the TrimBox of the previous one. Optionally, the `Spacing` attribute can specify a distance between the documents.



`Spacing` can have two different meanings, depending on the value of another attribute, `SpacingMethod`,

which has values `Gap` or `Offset`. By default, `Spacing` specifies the gap between the signatures, as shown in the illustration. If `SpacingMethod= "Offset"` then `Spacing` is the distance from the *start* of one signature to the start of the next.

Multiple multi-page Instance Documents per sheet: Note that `REPEAT` repeats a signature, which is defined as "a set of one or more pages from an Instance Document, printed on a single sheet of paper." The business card example above shows the trivial case of a one-page Signature, where each cell equals an entire Instance Document.

It is also possible to repeat a multi-page (multi-cell) Signature on a single sheet. For instance, a personalized folded card, such as an invitation or a greeting card, could be repeated, placing two Instance Documents on each sheet using the following code:

```
<REPEAT Direction="Ver" Count="2" Action="Increment">
  <SIGNATURE Nrows="2" Ncols="1">
    <CELL Row="1" Col="1" PageOrder="4*s-0" Face="Up" Rotate="0"/>
    <CELL Row="1" Col="2" PageOrder="4*s-3" Face="Up" Rotate="0"/>
    <CELL Row="1" Col="1" PageOrder="4*s-2" Face="Dn" Rotate="0"/>
    <CELL Row="1" Col="2" PageOrder="4*s-1" Face="Dn" Rotate="0"/>
  </SIGNATURE>
</REPEAT>
```

6.16.2 Model

```
REPEAT (REPEAT | SIGNATURE)
```

6.16.3 Attributes

Attribute	Required /Optional	Type	Description
Direction	Required	Keyword	Specifies which direction this `REPEAT` element is defining. Allowable values: `Ver` (vertical), `Hor` (horizontal), `Stack` (from sheet to sheet).
Action	Required	Keyword	What to print in the next location: use the same value of the signature counter *s* again, or increment it. Values: `Duplicate` or `Increment`.
Order	Optional	Keyword	Values: `Ascending` (default) or `Descending`.
Count	Required	Integer	Total count of repeated instances.
Spacing	Optional	Number	Distance, in 1/72", between Signatures. Default=0. See the `SpacingMethod` attribute for the effect of this value.
SpacingMethod	Optional	Keyword	Values: `Gap | Offset`. Defines the meaning of the Spacing value: If `Gap`, then `Spacing` is the gap distance *between* signatures. If `Offset`, then `Spacing` is the distance from the *start* of one signature to the start of the next signature.

6.16.4 Context

The `REPEAT` element occurs in `IMPOSITION` and `REPEAT`.

Chapter 7:
Production Specifications

7.1 Introductory remarks

Most of the preceding PPML elements concern the appearance of individual pages. It is often useful to provide additional information that supports the automated production ("manufacturing") of finished documents from those pages. Such information has no bearing on the *content* of individual pages; rather, it concerns production on a particular machine: how the pages should be rendered on that machine or instructions to inline finishing equipment.

As much as possible, PPML is intended to be device-independent, presuming that the machine has the RIPs (processor resources) required by the dataset. Therefore, the PPML philosophy is to keep all production specifications clearly separate. If it becomes necessary to retarget a job to a different Consumer, this structure makes it easy to identify and perhaps modify all elements that are not device-independent.

Still, practical reality in current product implementations (and expected implementations in the foreseeable future) is that much production information is specific to individual manufacturers: even the raw feature set varies substantially. Therefore, at present the PPML philosophy is that the language should only specify production parameters that are true no matter what device will be used for printing.

7.2 The <PRIVATE_INFO> Element

7.2.1 Description

Some applications on some systems need additional "private" information, e.g. device-specific features that aren't part of the PPML language. This element allows inclusion of any arbitrary data.

Private Info elements are private; their content is ignored by systems that don't know the meaning of the enclosed data.

One expected application for this feature is to include extracts from the widely used PPD (PostScript Printer Description) file format. Such functionality may be explicitly added to PPML in future editions; in any event, the `PRIVATE_INFO` element can safely be used to convey information from PPDs or any other printer description file format (or any other allowable XML content), and it will be ignored by any Consumer that has no use for it. Another example of a possible application would be to provide data regarding a CMS (color management system) profile.

7.2.2 Model

```
PRIVATE_INFO    (#PCDATA)
```

7.2.3 Attributes

Attribute	Required /Optional	Type	Description
Creator	Required	String	The creator (person, application, system etc) of this element
Identifier	Optional	String	An arbitrary string identifying what information or feature is provided by the content of this element.
Encoding	Optional	String	Identifies the encoding, if any, used in the content of this element
CharacterSet	Optional	String	Identifies the character set used in the content of this element.

7.2.4 Context

The `PRIVATE_INFO` element can occur in `PPML`, `DOCUMENT_SET`, `DOCUMENT`, and `PAGE`.

Chapter 8: Resources

8.1 The <REQUIRED_RESOURCES> Element

8.1.1 Description

The optional Required Resources element can appear at any level (PPML, Document Set, Document, Page). It specifies all the resources required (e.g. a font or a PostScript procedure set) to process every page and every object at and below the current level (the "enclosed pages"). There is no required use for this element, but it exists for two purposes:

1. **Pre-flight checks:** so that a Consumer can ensure that all resources required for a print run are available before the processing and printing starts.
2. **Subsets:** To facilitate extraction of self-sufficient subsets of the larger PPML dataset that include all the resources required to print the subset successfully.

8.1.2 Model

```
REQUIRED_RESOURCES  (FONT*,
                     EXTERNAL_DATA*,
                     PROCESSOR*,
                     SUPPLIED_RESOURCE_REF*)
```

8.1.3 Context

The `REQUIRED_RESOURCES` element can occur at any level: `DOCUMENT_SET`, `DOCUMENT`, `PAGE`, or the entire `PPML` element.

8.1.4 Attributes

None.

8.1.5 Application notes

A PPML Producer can choose the level (or levels) at which it will place the Required Resources element, based on the functionality desired for the target application.

Consumers should note that there may be an interaction between `SUPPLIED_RESOURCE` and `REQUIRED_RESOURCE` which presents an opportunity for optimization. For instance, the input stream might name an `EXTERNAL_DATA` Required Resource that's previously been supplied. In a simplest implementation, the Consumer can simply concatenate the external file within the `SOURCE` element whenever it's needed. In contrast, a more sophisticated Consumer may choose to add code to process the Resource in a way that makes it persistent, and then insert code that loads it later, when needed.

8.2 The <FONT> Element

8.2.1 Description

The FONT element identifies a font resource required for processing the pages enclosed in the current level.

8.2.2 Model

```
FONT          EMPTY
```

8.2.3 Attributes

Attribute	Required /Optional	Type	Description
FontName	Required	String	Name of the font as referenced by the content of the SOURCE elements in which it is used.
Format	Required	String	Data format of the font. Value: any format name registered with the Internet Assigned Numbers Authority (IANA).

8.2.4 Context

FONT occurs in REQUIRED_RESOURCES.

8.3 The <PROCESSOR> Element

8.3.1 Description

The PROCESSOR element names a file format interpreting resource, e.g. a RIP or similar interpreter, required for processing the pages enclosed in the current level.

8.3.2 Model

```
PROCESSOR    EMPTY
```

8.3.3 Attributes

Attribute	Required /Optional	Type	Description
Format	Required	String	Name of the language or file format. Value: any format name registered with the Internet Assigned Numbers Authority (IANA).
Revision	Optional	String	Any identifying string that will be useful to a Consumer in identifying whether its available processor resources are appropriate for the enclosed data.

8.3.4 Context

PROCESSOR occurs in REQUIRED_RESOURCES.

8.4 The <SUPPLIED_RESOURCES> Element

8.4.1 Description

SUPPLIED_RESOURCES is an umbrella element containing one or more child SUPPLIED_RESOURCE elements.

8.4.2 Model

```
SUPPLIED_RESOURCES (SUPPLIED_RESOURCE+)
```

8.4.3 Attributes

None.

8.4.4 Context

SUPPLIED_RESOURCES occurs within PPML, DOCUMENT, DOCUMENT_SET, and PAGE.

8.5 The <SUPPLIED_RESOURCE> Element

Note

The `Src` attribute is deprecated in favor of the model used elsewhere in PPML.: `INTERNAL_DATA | EXTERNAL_DATA`

8.5.1 Description

The Supplied Resource is a definition of a reusable resource such as a font, PostScript ProcSet, and other reusable resources for later use. To be used, the Supplied Resource must be referenced by a `SUPPLIED_RESOURCE_REF` in a `REQUIRED_RESOURCES` element.

Resources are independent of each other. They may be processed in any order, but they must appear before they are referenced.

8.5.2 Model

```
SUPPLIED_RESOURCE   (INTERNAL_DATA | EXTERNAL_DATA)?
```

8.5.3 Attributes

Attribute	Required /Optional	Type	Description
Name	Required	String	An identifying name for this resource for use in SUPPLIED_RESOURCE_REF.
ResourceName	Required	String	Name of the resource as referenced by the content of the SOURCE elements in which it is used.
Src *deprecated in version 2.0*	Optional	URI	Location of the resource file. Required if the element is empty, i.e. if no `INTERNAL_DATA` or `EXTERNAL_DATA` is specified.
Format	Required	String	Data format of the resource. Value: any format name registered with the Internet Assigned Numbers Authority (IANA).
Type	Required	Keyword	The resource type: `Font | ProcSet`. Other types may be defined in the future. A ProcSet is a PostScript ProcSet as defined in the PostScript Language Reference Manual.
SubType	Optional	String	Optional resource subtype, e.g. (Type1, TrueType etc.)
Scope	Optional	String	Specifies how long the Consumer must ensure that the resource will be needed: to the end of the current `PPML`, `DOCUMENT_SET`, `DOCUMENT`, or `PAGE` element.

8.5.4 Context

The `SUPPLIED_RESOURCE` element can occur in `SUPPLIED_RESOURCES`.

8.6 The <SUPPLIED_RESOURCE_REF> Element

8.6.1 Description

This element embodies a reference to a previously named SUPPLIED_RESOURCE element. This permits a SUPPLIED_RESOURCE element to be declared once, and referenced in multiple REQUIRED_RESOURCES elements.

8.6.2 Model

```
SUPPLIED_RESOURCE_REF   EMPTY
```

8.6.3 Attributes

Attribute	Required /Optional	Type	Description
Name	Required	String	Supplies the name of a previously encountered and named SUPPLIED_RESOURCE element.

8.6.4 Context

The SUPPLIED_RESOURCE_REF element can occur in REQUIRED_RESOURCES.

Chapter 9:
Future Capabilities

The following are in addition to future capabilities mentioned elsewhere in this specification.

9.1 Transparency / overprinting

In the current version of PPML each `MARK` defines a raster image that consists of "marked" and "transparent" pixels. When a `MARK` overlaps a `MARK` that was previously placed on the page, its marked pixels completely obscure the previous `MARK`'s pixels, and the transparent pixels leave the previous `MARK`'s pixels unaffected. This specification only applies to the interaction of `MARK`s: it does not preclude the content data format used for a particular `MARK` from using transparency to specify the color of the marked pixels in the `MARK`'s raster image.

Later versions of this specification may allow the placement of a `MARK` to modify rather than obscure `MARK`s that were previously placed on the page. Note, however, that since different `MARK`s may have been generated by content data in different content data formats using different color models, the definition of how a "partially transparent" overlying pixel would interact with an underlying pixel is a complex process.

9.2 Color Management

Future versions of PPML may include direct support for CMS (color management system) profiles. In the current version, color profiles can be supported via `PRIVATE_INFO` or `EXTERNAL_DATA` elements.

9.3 PPML Consumer Profile

Differences between Consumers (e.g. which data formats they can accept, level of imposition support, color separations available) may be documented in a standardized Consumer Profile file format. In the current version of PPML, such information can optionally be conveyed in `PRIVATE_INFO` elements.

Chapter 10: Conformance Subsets

10.1 Introduction

The PPML language allows a practically limitless range of data formats. This gives the language great flexibility for present and future applications, but also creates the possibility of valid PPML datasets that no machine could consume. To enable greater predictability, PODi may define subsets designed to meet the needs of various markets and applications.

Conformance to a particular subset can be declared using the `CONFORMANCE` element (see section 4.7). Each subset described below has one or more identifying strings for use in the `Subset` and `Level` attributes of `CONFORMANCE`.

10.2 Graphic Arts subset

This subset is intended to meet the needs of typical graphic arts workflows.

`Subset` string: `GA`

`Level` string: `1` or `2`

10.2.1 Levels

The relationships between PPML Producers and PPML Consumers can be categorized as informal, semi-formal, and formal. The PPML Graphic Arts Conformance Subset is intended for informal and semi-formal relationships. It is not intended for formal relationships. If a conforming dataset specifies `ResourcesIncluded=Yes`, then the dataset is suitable for informal blind-exchange. If a conforming dataset specifies `ResourcesIncluded=No`, then the dataset is suitable for semi-formal partial-blind-exchange.

Level 1: informal relationship, "blind exchange"

An informal relationship allows "blind exchange" between Producer and Consumer. All data needed for the job is transmitted with the job itself. There is no reliance on any previous exchanges between Producer and Consumer.

The Producer must ensure the job conforms to the Subset and that all resources are included in the job itself. The Consumer must ensure it can correctly process any PPML that conforms to this subset.

Level 2: semi-formal, "partial blind exchange"

A semi-formal relationship allows partial-blind-exchange between Producer and Consumer. Some of the data needed for this job may have been sent in a previous exchange and has been kept by the Consumer for use by future jobs.

The Producer must ensure that the PPML data conforms to this Conformance Subset, and that all needed data is either in the job stream or already present at the Consumer. The Consumer must ensure it can correctly process any Conformance Subset PPML.

Open exchange (formal relationship)

A formal relationship allows open exchange of data between Producer and Consumer. The Producer knows which Consumer it is sending data to and forms the data according to what the Consumer needs. For these relationships, no Conformance Subset is needed. However, PPML datasets prepared for one Consumer may not print correctly if sent to another Consumer. It is up to the Producer to guarantee that the PPML can be processed by the Consumer.

10.2.2 Overview of PPML Changes

PPML that conforms to the Graphic Arts Subset is restricted as follows:

The SOURCE element

The `SOURCE` element Format attribute may only have one of these values:

```
application/postscript
application/pdf
image/tiff
image/jpeg.
```

Further restrictions on these data formats (e.g. revision levels) are explained in detail below. A conforming PPML Producer may produce any or all of these formats, and therefore conforming PPML Consumers must support all of them.

Job ticketing

The PPML Job Ticket format[10] shall specify all Production Instructions. `PRINT_LAYOUT` elements contained within `PPML` and `DOCUMENT_SET` elements are allowed, but will be overridden by a `PRINT_LAYOUT` element within the PPML job ticket.

PRIVATE_INFO

`PRIVATE_INFO` cannot alter the content or layout of objects on the page.

The `ResourcesIncluded` attribute

The `PPML` element's attribute `ResourcesIncluded` promises a Consumer that all referenced content data, fonts, and other resources are supplied with the dataset. Note that this attribute can

[10] The original wording of this requirement, when first written, was "The to-be-defined PPML digital print ticket format".

have the value Yes or No. Either value is valid for compliance with the Graphic Arts subset. A value of Yes means the dataset is suitable for the "blind exchange" business relationship model.

10.2.3 Details of ResourcesIncluded

A PPML dataset that specifies ResourcesIncluded=Yes must conform to these rules:

1. All content data is transmitted with the dataset. For example, if the dataset is carried in MIME, all content data is also included in that MIME stream. All references to external data (EXTERNAL_DATA, EXTERNAL_DATA_ARRAY, SUPPLIED_RESOURCE) will refer only to data transmitted with the job.
2. All REQUIRED_RESOURCES elements (if present) must include only SUPPLIED_RESOURCE_REF and PROCESSOR elements. No FONT elements are allowed.
3. No element shall specify Scope=Global. This guarantees that the data carried with the dataset will be used and not some global data from a previous dataset.

A PPML dataset that specifies ResourcesIncluded=Yes but does not conform to the above rules is an invalid PPML dataset.

10.2.4 Content Format Details

Color Spaces

Some color data does not specify a calibrated color space to determine its color characteristics: TIFF, JPEG, and PostScript/PDF in color spaces DeviceCMYK and DeviceRGB. All such color data shall be assumed to be calibrated to the SWOP standard (Specifications Web Offset Publications, available at http://www.swop.org) for four-component data, or to the sRGB standard (IEC61966-2.1, available at http://www.srgb.com) for three-component data.

PostScript

SOURCE elements with Format=application/postscript conform to the Graphic Arts subset if they refer to content data that obey these restrictions:

- Content data adheres to the PostScript Language Reference Manual, Third Edition (PLRM). For example, language extensions for particular printers are not allowed.
- Content data do not rely on the execution of illegal operators as defined in "Encapsulated PostScript File Format Specification Version 3.0", Adobe Technical Note #5002 and as amended by Appendix G, "Operator Usage Guidelines" of the PLRM. A PPML Consumer is free to redefine these illegal operators to consume their operands and do nothing else.
- Content data do not use any restricted operators as defined in "Encapsulated PostScript File Format Specification Version 3.0", except as allowed in Appendix G, "Operator Usage Guidelines" of the PLRM. A PPML Consumer is free to redefine these restricted operators to perform only permitted uses.
- Any external resources, such as fonts, that are not included directly in the content data are specified in the REQUIRED_RESOURCES element that pertains to this SOURCE element.

- OPI comments for image replacement must be ignored. Any image replacement, such as that specified by OPI comments, has already been accomplished before the PPML Consumer receives the PPML dataset.

PDF

`SOURCE` elements with `Format=application/pdf` conform to the Graphic Arts subset if they refer to content data that obey these restrictions:

- Content data contain only PDF operators as specified in the Portable Document Format Reference Manual, Version 1.3.
- Any external resources, such as fonts, that are not included directly in the content data are specified in the REQUIRED_RESOURCES element that pertains to this SOURCE element.
- No image object will contain an OPI Dictionary.

TIFF

`SOURCE` elements with `Format=image/tiff` conform to the Graphic Arts subset if they refer to content data that obey these restrictions:

- Content data conform to TIFF Revision 6.0[11], except:
- Content data do not specify Compression=6, which is ill-defined and can't guarantee successful parsing of JPEG data, and
- Content data can specify Compression=7, which is well-defined JPEG, [12] and widely used.

Note that Compression=5 (LZW compression) is supported, but requires a license from Unisys. Conforming PPML Producers and Consumers are required to obtain such a license themselves or use products from companies that already have a license.

JPEG

`SOURCE` elements with `Format=image/jpeg` conform to the Graphic Arts subset if they refer to content data that obey these restrictions:

- Content data conform to Huffman-encoded Lossy JPEG (any of these Start Frame Markers: SF0, SF1, SF2, SF5, SF6).
- Resolution is deduced from the Dimensions attribute of the element. Only one JPEG image file is allowed per `SOURCE` element, so that `Dimensions` will be correct.

[11] Available at http://partners.adobe.com/asn/developer/PDFS/TN/TIFF6.pdf

[12] ftp://ftp.sgi.com/graphics/tiff/TTN2.draft.txt

Appendix A: Acknowledgements

PPML Working Group participants – version 1.0

The PPML specification would not have been possible without the substantial efforts of the following companies and their designated participants. In alphabetical order, they are:

Adobe Systems: John Green
Agfa: Roger Baeten and Marcus Delhoune
Barco: Dirk De Bosschere
EFI: Margaret Motamed
HP: Bob Taylor
IBM: D. R. Palmer
Indigo: Sigal Krumer and Ouri Poupko
NexPress: David Blaszyk, Tim Donahue, Wayne Minns
Pageflex: Peter Davis
Scitex: Jacob Aizikowitz, Israel Roth, Reuven Sherwin
Xeikon: Anthony Porter
Xerox: Steve Strasen

Prior work

While PPML as a standardized data format is new, the technology of variable data printing (VDP) is not.

PPML concepts were largely contributed by skilled developers of established VDP products from several members of PODi, including:

- Agfa variable data machines and Personalizer X software
- Barco's Book Ticket Format (BTF) and Imposition Templates for PrintStreamer
- Indigo™ Yours Truly™ Personalization® architecture, SNAP® personalization software and software applications
- Pageflex's MPower variable data composition software, based on the NuDoc composition engine
- Scitex's VI Digital Front Ends, Darwin software, and VPS™ language. Scitex is a co-founder of PODi. Before PPML, VPS was the format that was most widely supported by third-party applications.
- Xeikon's "Private-I" software

Origins of PPML

PPML 1.0 grew out a combined proposal approved in July 1999 by the PPML Working Group. This proposal was a merger of proposals from Scitex, Barco and Pageflex: Scitex, by way of its VPS language, contributed the foundation for the basic object model, object-level granularity, and job structure of PPML; Barco contributed the foundation for the production-centric parts of the specification, including major work on imposition; PageFlex contributed the original proposal for an XML-based language called PPML. NexPress contributed substantial work from its proposed vPDF specification, and Xerox presented additional information at the July conference based on its substantial experience with its VIPP PostScript-based variable data software.

Versions 1.5 and 2.0

Major contributors

Adobe Systems: Craig Benson
Barco: Wim Sandra
Edmond Research & Development: Paul Jones
Electronics For Imaging: Boris Aronshtam
Hewlett-Packard: Kevin Currans, Steve Hiebert
IBM: D. R. Palmer, Art Ford, Claudia Alimpich
Indigo: Sigal Krumer, Ouri Poupko
NexPress: Tim Donahue, David Blaszyk
Pageflex: Peter Davis
Xeikon: Marcus Delhoune, Roger Baeten
Xerox: Steve Strasen, Robert Herriot
Xmpie: Reuven Sherwin

Additional contributors

Creo (formerly Scitex and CreoScitex): Avinoam Beinglass
Electronics For Imaging: Richard Falk, Mike Robinson
Hewlett-Packard: Robert Taylor
IBM: Hitesh Bhindi
Océ: Werner Engbrocks
PrintSoft: Keith Adeney
Xerox: Jean-Yves Bouche

Appendix B: Introduction to XML

The PPML data format is based on the XML (eXtensible Mark-up Language) syntax. This is analogous to saying that XML is the programming language in which the PPML application is written. To understand PPML, therefore, it's helpful to have some basic knowledge of how XML works.

Elements: In XML, data can be grouped into tagged elements, like this:

```
<TAG>This sentence is data of type TAG.</TAG>
```

Here, <TAG> is the start tag, and </TAG> is the end tag. The end tag uses the same tag name as the start tag, but the name is preceded by a "/". Whatever lies between the start and end tags is considered to be of type TAG.

Nesting: Elements can be, and usually are, nested:

```
<TAG1>This is TAG1 text.
   <TAG2>And this is TAG2 text.</TAG2>
</TAG1>.
```

Note that the end tags are in the reverse order from the start tags, so that TAG2 lies entirely inside TAG1. This means the elements are properly nested. The following would NOT be syntactically valid because the outer tag (TAG1) is closed off while the inner tag (TAG2) is still left open:

```
<TAG1>This is TAG1 text.
   <TAG2>And this is TAG2 text.
</TAG1>
</TAG2>.
```

Elements with no content: In some cases, a tagged element will have no content between the opening and closing tags. This can be abbreviated with a single tag that has the "/" character at the end. In other words, <TAG/> is equivalent to <TAG></TAG>.

Attributes: Elements can specify attributes, which are properties of the particular instance of the element. For example, element NOTE_TEXT could be defined to have the property that the color is normally red, but I may override this in a specific instance by specifying:

<NOTE_TEXT Color="blue">This text will be blue.</NOTE_TEXT>

In this example, `Color` is an attribute of the element `NOTE_TEXT`.

Comments: Finally, comments (information which is not processed by software) can be placed in the XML file for users who may wish to look directly at the file. Such comments are embedded between
<!-- and --> delimiters, for instance:

```
<!-- This is a comment. -->
```

White space (returns, tabs, and spaces) are allowed within a comment.

The DTD: An XML application, such as PPML, specifies exactly which tags are defined, which elements can (or must) exist within other elements, and what attributes and values can be specified for each element, via a file called the Document Type Definition (DTD).

Appendix C: Strings to use for the Format attribute of SOURCE

The following are examples of the strings approved by IANA (the Internet Assigned Numbers Authority) that are to be used in the value of the `Format` attribute in the `SOURCE` element. These strings were developed for use in identifying the media type in a MIME stream; PPML is adopting them by reference because they are an existing standard that is well suited to PPML needs.

Most of these strings are from http://www.isi.edu/in-notes/iana/assignments/media-types/media-types.

Format	**IANA identifier**	
PostScript	`application/postscript`	`RFC2045, RFC2046`
Encapsulated PostScript (EPS)	`application/postscript`	
PDF	`application/pdf`	
PCL	`application/vnd.hp-PCL`	
PCL XL	`application/vnd.hp-PCLXL`	
AFP	`application/vnd.ibm.modcap`	
TIFF	`image/tiff`	`RFC2302`
JPEG	`image/jpeg`	`RFC2045, RFC2046`
GIF	`image/gif`	`RFC2045, RFC2046`
SVG (scaleable vector graphics)	`image/svg-xml`	

Appendix D: Packaging PPML datasets for transport using ZIP files or removable media

D.1 Introduction

MIME and other transport methods are extremely flexible, and provide a highly generalized interface between many kinds of systems. However, current practice often involves the simple case of creating an entire project on one machine and physically transporting it to another machine. For this case, a much simpler method of transport can be defined. This section describes rules for reliably packaging a PPML dataset created on one machine for transport to another machine, where it can be unpacked so that all references will still function as expected.

In this workflow, all related resource files are typically within a single directory tree. This practice allows for a simple case for constructing the PPML references and packaging the project: the entire directory may be copied to removable media such as a CD, or the directory may be packed into a single file using compression software.

PKZIP[13] is one such packing application. It is supported on many platforms (Windows, Macintosh, Unix/Linux) and is available as open source.

The rules described here for references and file naming will allow for reliable transport of a PPML dataset from one machine to another, and even between platforms.

PPML Consumers are not required to accept ZIP files.[14] If a PPML Producer generates datasets and ZIP packages that conform to these rules, the receiving system can use any unzipping tools to unpack the package, and the references should work successfully.[15]

---

[13] www.pkware.com

[14] In the workflows where this method is intended to be used, unpacking ZIP files is a common practice. Nonetheless, Consumers that can directly read ZIP files will offer two competitive advantages: simpler workflow and savings of disk space. Since some datasets can be large, there can be a real advantage to building this functionality into the Consumer.

[15] In a closed, formal environment where the Producer and Consumer know each other's systems, they are free to use any conventions they want. However, datasets and packages that do not conform to these rules will not transport and unpack reliably on unknown receiving systems. For portability, or if the receiving system is unknown when the dataset is generated, the rules defined in this section should be obeyed.

D.2 Requirements

This appendix specifies rules for naming the PPML file and its attachment files. It also prescribes the relative locations of such files. These rules implement the following requirements.

When the files are extracted in a single operation from the ZIP file on the target platform or copied from removable media to directory on a target platform:

1) The PPML file can be located easily
2) Each URI in the PPML file references the intended attachment file using the target platform's software for mapping a URI to a specific file.

Note that these rules describe the characteristics of the files on the *target* platform after they are extracted from a ZIP file or copied from a removable media. The files on the source platform are likely to have similar characteristics, in order to simplify the zipping or copying operation, but they are not required to adhere to these rules.

The following terms are used in this appendix:

Conforming Package: all the files and directories in a ZIP file or on a removable media.

Conforming Package Extraction: all the files and directories on a Consumer's platform that come from a single Conforming Package, either by extracting them from a ZIP file, or by copying them from a removable media.

The rules are phrased in terms of the Conforming Package Extraction because it is easier to specify the files and directories on the Consumer's platform than to specify the representation of files and directories on a ZIP file. It is best to treat a ZIP file as a black box with add and extract APIs. The extract API produces the Conforming Package Extraction. A rule about a Conforming Package Extraction implies what must be in a Conforming Package in order for the ZIP extract operation or the removable-media copy operation to work properly.

D.3 Rules for Files and Directories

1. **Directory Structure:** A Conforming Package Extraction shall contain a single top level directory. All files and directories in the Conforming Package Extraction shall reside in that top-level directory, or in directories under it.

2. **PPML File:** A Conforming Package Extraction shall contain only one PPML file, whose name must conform to the rules in this Appendix, and whose suffix is "`.ppml`". The PPML file shall reside in the top-level directory.

3. **Character set:** Each character of a file name or directory name within a Conforming Package Extraction must be a printable character from ISO 696 IRV (i.e. those in the range 32 to 126 inclusive) that is not one of the nine characters in the table below.[16]

34 ('"' double quote)	42 ('*' asterisk)	47 ('/' slash)
58 (':' colon)	60 ('<' less-than)	62 ('>' greater-than)
63 ('?' question mark)	92 ('\' backslash)	124 ('|' vertical bar)

   The first character of a file name or directory name must not be '`.`' (dot). This is due to limitations of some Windows systems.

4. **Case sensitivity:** Each directory within a Conforming Package Extraction on a case-sensitive platform (i.e. the top-level directory or any directories under it) shall not contain multiple files whose names are identical except for the case of one or more letters. For instance, a directory cannot contain both `test.eps` and `TEST.EPS`, or `test.eps` and `Test.eps`. Also, a directory cannot contain both `Foo` and `foo`, where each is either a directory or file.

   Because the above rule indirectly applies to a Conforming Package, the above rule implies that on a case-insensitive platform a Conforming Package Extraction contains exactly the same files and directories that are in the Conforming Package, e.g.. there won't be both a `test.eps` and `Test.eps`. Note: a Producer creates a Conforming Package without the knowledge about whether the Consumer is on a case-sensitive or case-insensitive platform.

   See rule 2 in the section entitled "Rules for URIs". It gives the corresponding rule for case sensitivity of URIs that reference files whose names are governed by this rule.

5. **Length restrictions:**

   a. The maximum length for each name of a **file or directory** in a Conforming Package Extraction shall be 31 characters, including extension.[17] Example: the

[16] These characters are excluded from PPML filenames because Windows does not allow them in filenames. Windows is the most restrictive. Mac OS9 excludes the colon ":" and Linux excludes only "/", both of which are excluded from Windows filenames.

[17] Filenames in Macintosh OS9 are limited to 31 characters in length.

name of the PPML file in the package must be limited to 26 characters plus 5 characters for ".ppml".

b. The maximum length for each **path name** of a file in a Conforming Package Extraction (relative to the top-level directory of the Conforming Package Extraction) shall be 127 characters, including slashes, period and file suffix.

Windows is the motivation for this constraint. On Windows the maximum length of an absolute path is 254 characters; every file created in a Windows directory must result in an absolute path no more than 254 characters long, or the file creation will fail. Arbitrarily dividing 254 in half allows:

- 127 characters for the complete pathname of the top-level directory of the Conforming Package Extension, e.g. "c:/projects/projectFoo".
- 127 characters for the path of each file in the Conforming Package Extension relative to its top-level directory, e.g. "images/sunset.gif" is the relative path of the file (limit 127 characters) and "c:/projects/projectFoo/images/sunset.gif" is the complete pathname of the file (limit 254 characters).

These limits are considered reasonable for expected production needs, and they guarantee that any package can be unpacked on the most restrictive system (Windows), into any directory whose pathname doesn't exceed 127 characters, and the resulting absolute paths will never exceed the 254 character limit.[18]

D.4 Rules for URIs

1. **Allowed URIs:** URIs in the PPML file of a Conforming Package Extraction (e.g. EXTERNAL_DATA) shall meet the requirements of either rel_path or AbsoluteURI in RFC 2396. Examples of a rel_path URI are: "myfile.eps", "./myfile.eps" and "images/myfile.eps". (Note that URIs use slashes, not backslashes.) Examples of an AbsoluteURI URI are "http://foo.com/test.eps" and "ftp://ftp.foo.com/test.eps".

Each rel_path URI must reference a file that is within the Conforming Package Extraction.

If absolute URIs are used, it is the responsibility of the Producer and Consumer to ensure that the Consumer can access all referenced data – it is not a function of this packaging specification. For instance, to access http://foo.com/test.eps, the Consumer must have HTTP access.

This rule prohibits abs_path URIs (e.g. "/working/test1/test2.eps") and platform centric URIs (e.g. "file:///c:foo/bar.gif").

[18] It is the responsibility of the person or program that picks the location of the top-level directory to ensure that the target directory's pathname is 127 characters or less. Again, this issue only exists on Windows systems; other systems have no practical limit, in the simple environments for which this specification is intended.

2. **Preserve case:** For each letter in a filename or directory of a Conforming Package Extraction, the corresponding letter in its referencing URI (in the PPML file) must be identical, including case. For example, file "images/Foo.gif" must be referenced in the URI as "images/Foo.gif" and not as "images/foo.gif", "Images/foo.gif" or "IMAGES/FOO.GIF". This rule ensures that each file in Conforming Package Extraction on a case-sensitive platform, such as Unix/Linux, can be referenced by its corresponding URI.

3. **Use Escape Characters:** According to RFC 2396, certain characters must not appear directly within a URI. Instead the character must be escaped by using a "%" followed the two digit hex value of the character. For example, the rel_path URI that references the file "first time" would be "first%20time" where 0x20 is the value of the space character. The excluded characters for the abs_path portion of the URI are the 10 characters space, "#", "%", ";", "[", "]", "^", "`","{" or "}". For the first segment of the rel_path URI (called rel_segment), the excluded characters are the same except that the ":" is an excluded character and the ";" is not. See RFC 2396 for the full grammar.

   If a URI in a PPML file of a Conforming Package Extraction references a file whose name includes characters that cannot be directly represented in a URI as described above, then each such character must be escaped as described above.

D.5 Example

The following example shows a simple PPML dataset conforming to these rules: a PPML file that references three objects. Shown below are:

1. The content of the PPML file, with the URIs highlighted.
2. A Windows directory listing of the entire dataset: the PPML file and its three referenced files.
3. A directory listing of a ZIP file created from this dataset.

The highlighting in these listings is not significant; it only draws attention to relevant information.

D.5.1 PPML file

```
<?xml version="1.0" encoding="UTF-8"?>
<! DOCTYPE PPML PUBLIC
   "-//PODi//DTD PPML 2.00//EN" "http://www.podi.org/ppml/ppml200.dtd">
<PPML Creator="Test">
   <DOCUMENT_SET Label="URI example">
      <DOCUMENT Dimensions="594 840">
         <PAGE Label="Page 1">
            <MARK Position="0 800">
               <VIEW/>
               <OBJECT Position="0 0">
                  <SOURCE Format="application/postscript" Dimensions="40 40">
                     <EXTERNAL_DATA Src="object-1.eps"/>
                  </SOURCE>
                  <VIEW/>
               </OBJECT>
            </MARK>
            <MARK Position="50 800">
               <VIEW/>
               <OBJECT Position="0 0">
                  <SOURCE Format="application/postscript" Dimensions="40 40">
                     <EXTERNAL_DATA Src="object-2.eps"/>
                  </SOURCE>
                  <VIEW/>
               </OBJECT>
            </MARK>
            <MARK Position="100 800">
               <VIEW/>
               <OBJECT Position="0 0">
                  <SOURCE Format="application/postscript" Dimensions="40 40">
                     <EXTERNAL_DATA Src="images/object-1.eps"/>
                  </SOURCE>
                  <VIEW/>
               </OBJECT>
            </MARK>
         </PAGE>
      </DOCUMENT>
   </DOCUMENT_SET>
</PPML>
```

D.5.2 Windows directory listing of the files to be packaged

```
Directory of E:\PPML
URIS~1   PPM           873  08-22-01  1:35p URIs.ppml
IMAGES         <DIR>        08-22-01  1:42p images
OBJECT-1 EPS       190,677  10-18-00 10:13a object-1.eps
OBJECT-2 EPS        37,122  10-17-00  8:14a object-2.eps

Directory of E:\PPML\images
OBJECT-1 EPS       193,366  10-16-00  1:39p object-1.eps
```

Note that the entire project, including the PPML file itself, was created within the directory E:\PPML, but that directory names never appear in the relative URIs in the PPML. In fact the project could have been created in any directory on any drive, and the PPML code and the ZIP package would look the same as shown here, because both the PPML and the ZIP package use relative locations.

D.5.3 Directory listing of conforming ZIP file

```
PKZIP(R)  Version 2.50  Compression Utility for Windows 95/NT  4-15-1998
 Viewing .ZIP: URItest.zip

  Length  Method    Size  Ratio   Name
  ------  ------    ----- -----   ----
     873 DeflatN     383 56.2%   URIs.ppml
       0 Stored        0  0.0%   images/
  190677 DeflatN   62047 67.5%   object-1.eps
   37122 DeflatN   14145 61.9%   object-2.eps
  193366 DeflatN   62483 67.7%   images/object-1.eps
  ------           ------ -----   ----
  422038           139058 67.1%      5
```

Appendix E: Job ticketing formats

E.1 Introduction

As noted in section 4.8, "the PPML language itself does not specify any particular ticket data format." Instead, a PPML `TICKET` element identifies a ticket datum (e.g. a file) that contains named objects, and any object can be referenced from within the PPML stream by providing its name in a `TICKET_REF` element. This can be done without any knowledge of the data format or internal workings of the ticket file; the Producer only needs to know what name to use for each function in the ticket via the `ExtIDRef` attribute.

This allows Producers and Consumers to implement PPML job ticketing at whatever technical level is appropriate for their business environment. As an example, a Producer who has an existing business relationship with a specific Consumer shop can use PPML Job Ticketing to support any ticket data format they want.

E.2 Example: an arbitrary job ticket format

The following is a sample fragment of an imaginary job ticket format. It happens to be encoded in XML, but it need not be. The only thing required of a job ticket is that it contain the ID strings that the Producer will use in `TICKET_REF` elements in the accompanying PPML dataset. (The shading, boldface and underlining shown here are used to highlight how the ID strings connect the ticket data to the PPML stream.)

The job ticket might include these lines:

```
<MyTicket  xmlns:"http://www.mysite.com/schema/myticket.xsd">

   <Feature  ID="TwoSidedLongEdge">
      <ActivationCode>   activation code goes here  </ActivationCode>
   </Feature>

   <Feature  ID="OneSided">
      <ActivationCode>   activation code goes here  </ActivationCode>
   </Feature>
   ...
```

Here is an example of how a dataset might access these features using `TICKET_REF`:

```
<PPML  ...>
   ...
   <TICKET_REF  ExtIDRef="OneSided"/>
   <DOCUMENT ...>
      <PAGE ...>...</PAGE>
      ...
   </DOCUMENT>
```

```
<TICKET_REF  ExtIDRef="TwoSidedLongEdge"/>
<DOCUMENT...>
    <PAGE ...>...</PAGE>
    ...
```

For this example, the location and meaning of the TICKET_REFs is not important; they may occur at various points in the PPML stream. The important thing to understand is that the connection between TICKET content and TICKET_REF is accomplished through the ID strings that occur in both places.

E.3 JDF as an instance of a PPML Job Ticket

E.3.1 Introduction

JDF ("Job Definition Format," http://www.cip4.org) is an industry-standard data format for electronic job ticketing. It provides a device-independent way to describe the finished result of the printing process, such as an 8-page stapled booklet, and the production processes that may be used to produce it. JDF syntax is recommended for interoperable job tickets, especially for graphic arts applications. See the PPML Job Ticketing specification for details.

E.3.2 Example encoded in JDF

Here is an example of how the same job ticket information might be encoded in JDF:

```
<LayoutPreparationParams ID=.... >
    ...
    <LayoutPreparationParamsUpdate UpdateID="OneSided"
        Sides="OneSidedFront"/>
    <LayoutPreparationParamsUpdate UpdateID="TwoSidedLongEdge"
        Sides="TwoSidedFlipY"/>
    ...
</LayoutPreparationParams>
```

In the PPML stream, this ticket would be referenced in exactly the same way as shown above:

```
<PPML  ...>
    ...
    <TICKET_REF  ExtIDRef="OneSided"/>
    <DOCUMENT ...>
        <PAGE ...>...</PAGE>
        ...
    </DOCUMENT>

    <TICKET_REF  ExtIDRef="TwoSidedLongEdge"/>
    <DOCUMENT...>
        <PAGE ...>...</PAGE>
        ...
```

In these examples the PPML stream is identical, even though the ticket data formats are different. The basic mechanism of the PPML job ticket scheme is that the ExtIDRef attributes identify the pieces of the job ticket file that are being requested at that point in the dataset.

Appendix F: Embedding text in a PPML stream

F.1 Introduction

Page content in a PPML Object can be expressed as either EXTERNAL_DATA or INTERNAL_DATA. This appendix describes the relative advantages of embedding text in a PPML dataset using INTERNAL_DATA, in particular the advantages of using Scaleable Vector Graphics (SVG) as a method of encoding the text.

As a baseline for comparison, this appendix begins by reviewing the advantages of EXTERNAL_DATA. The relative advantages of INTERNAL_DATA are then discussed, and finally the additional advantages of SVG are presented.

F.2 Advantages and Applications of External Data

For reusable content objects, EXTERNAL_DATA is commonly used, as in this example:

```
<OBJECT Position="0 0">
   <SOURCE Format="application/postscript" Dimensions="400 50">
      <EXTERNAL_DATA Src="MyCarPhoto.eps"/>
   </SOURCE>
   <VIEW/>
</OBJECT>
```

EXTERNAL_DATA offers several advantages that have substantial benefits in some applications:

- Files referenced by EXTERNAL_DATA can contain binary content; INTERNAL_DATA is restricted to XML content, which cannot be binary. This allows using any content format (e.g. images and PDF) efficiently.
- EXTERNAL_DATA allows a content object to be generated by any software application, yet it can be referenced by any PPML Producer.
- EXTERNAL_DATA allows a content object to be created asynchronously, before creation of the PPML dataset that references it.

F.3 Advantages and Applications of Internal Data

For applications that involve "disposable text" (such as the name and address on an Instance Document), INTERNAL_DATA allows embedding the text directly into the PPML stream, as in this example, which uses PostScript encoding to typeset a line in 12 point Futura:

```
<OBJECT Position="72 500">
   <SOURCE Format="application/postscript" Dimensions="400 50">
      <INTERNAL_DATA>
         /Futura findfont 12 scalefont setfont
         (Dear Jan Watkins,) show
      </INTERNAL_DATA>
   </SOURCE>
   <VIEW/>
</OBJECT>
```

INTERNAL_DATA offers different advantages from EXTERNAL_DATA:

- **The entire print job can be expressed in a single stream, encoded in XML.** This was one of the original requirements for PPML – the ability to encode everything in a single stream. Some applications absolutely require this, so that when the same dataset is reprinted later, there is no question that the job will run the same.
- **Avoids the need for many separate files.** By its nature, disposable text is used only once, so placing it into a file that's used only once is not efficient. High-volume print streams can have thousands of pages, each of which might have several content objects; encoding each one into a separate file can be extremely inefficient. Creating and consuming thousands of separate files involve substantial overhead for both the Producer and the Consumer. INTERNAL_DATA allows bundling the content into a single stream.

F.4 Additional advantages of SVG

SVG (www.svg.org) is an accepted W3C recommendation (http://www.w3.org/TR/SVG/) that provides a way of describing text and vector graphics in XML. The following SVG example does the same thing as the preceding PostScript example, but all properties and content are expressed using XML constructs:

```
<OBJECT Position="72 500">
   <SOURCE Format="image/svg-xml" Dimensions="400 50">
      <INTERNAL_DATA>
         <svg:svg xmlns:svg=... width="400" height="50">
            <svg:text x="0" y="0" font-size="12"
               font-family="Futura-Book" fill="black">
               Dear Jan Watkins,
            </svg:text>
         </svg:svg>
      </INTERNAL_DATA>
   </SOURCE>
   <VIEW/>
</OBJECT>
```

SVG provides a way for PPML content data to gain all the advantages of other XML content:

- **Parse and validate the document content using XML tools.** Encoding the content data itself in XML means the entire dataset, including the variable content, can be checked using the same XML tools that check the PPML itself – and in the same single step in the workflow.
- **Locate and extract document content using XML tools.** PPML content encoded in SVG can be extracted from a dataset using generic XML tools, which means the output of a PPML Producer can be used in many ways. For instance, it can be cataloged, archived, or added to a database, which in turn would make it available for a variety of other applications: archiving, indexing, searching.

As use of PPML expands into applications that involve substantial quantities of disposable text, as well as reusable content, it is expected that there will be increased use of `INTERNAL_DATA` in general, and SVG in particular.

Change History

Version 1.0, March 15, 2000

Initial release.

Version 1.01, May 18, 2000:

- **Inside front cover:** modify text and email address related to reader participation.
- **1.2 Organization of this Document:** add "and Marks"
- **2.1.4 DTD:** Add reference to the official online version of the PPML DTD.
- **4.4.3 (attributes of DOCUMENT and PAGE):** reposition Label attribute in the table. *(This does not affect functionality.)*
- **5.3.3 Attributes of** `MARK`, new **5.5.3 Implementation Note:** New definition of the `Position` attribute.
- **5.7.1 Description of** `OBJECT` **element:** add a second paragraph, clarifying intent related to the change in 5.5.3 above.
- **5.7.3 Attributes of** `OBJECT`: see 5.3.3 above.
- **5.8.2 Model of** `SOURCE`**:** add `EXTERNAL_DATA_ARRAY`, consistent with contexts listed in 5.10.3.
- **Appendix 3:** add SVG.
- **Reference card:** update per the above; document the list of allowed attribute values where appropriate, and show which choice is the default.

Version 1.02, December 14, 2000:

New features and substantial additions

- **Add support for multi-page source files:**
  - Created two new elements, `SEGMENT_ARRAY` (section 5.17) and `SEGMENT_REF` (section 5.18);
  - Added `SEGMENT_REF` to the model for `MARK`, and added `SEGMENT_ARRAY` to the model for PPML, Job [Document Set in version 2.x], Document, and Page.
- **Illustrations** of how PPML content objects are created and placed on a page:
  - Added new section **5.19 Definition of PPML Extent Boxes**
  - Added section **5.20 Notes on Transforming, Clipping and Positioning**
- **Imaging model re transparency & overprint:** Modify the following sections regarding the interaction of marks on a page:

5.2 A Page contains Marks
5.3.1 The MARK Element – Description
9.1 Transparency / overprinting

Additional changes and clarifications

- **2.1.4 DTD:** Add PUBLIC identifier; change statement regarding DTDs stored on the Web.
- **2.2 Non-XML Data:** remove sentence about a possible separate specification regarding transport issues.
- **5.8.1 SOURCE:** Add paragraph regarding non-content data, such as binary previews on Windows EPS files.
- **5.10.3 EXTERNAL_DATA_ARRAY:** Clarify minimum value of `Index` attribute.
- **6.6.3 IMPOSITION `Position` attribute:** Declare that the imposition structure does not include any trim or fold marks, so the marks do not affect position on the sheet,
- **6.8.1 SIGNATURE description:** Explain CELL positioning and rotation
- **6.9 The CELL Element:** Expand description (6.9.1), add rotation example (6.9.7), add "PageOrder <1" case at end of 6.9.5.
- **6.10 The HOR_TRIM_MARKS Element:** Add illustration of position of trim marks; clarify wording of mark suppression in the "touching pages" case.
- **Scope of OCCURRENCE_REF in sheet marks:** State in 6.10.1, 6.11.1, 6.14.1, 6.15.1 that the scope of a sheet mark's Occurrence Ref must be at least as high as the enclosing `IMPOSITION`.
- **6.14.1, HOR_FOLD_MARKS:** clarify suppression of trim marks near fold marks.
- **8.2.3, Attributes of FONT:** Add `Format` attribute. Also, change the `Name` attribute to `FontName` and add a descriptive note about its intent. ("`Name`" in other PPML elements is merely an arbitrary identifying string; in the `FONT` element, it denotes the actual name of the font, e.g. `Helvetica-BoldOblique`. Also, add `Format` attribute.
- **8.5 SUPPLIED_RESOURCE:**
  - **8.5.1 Description:** stipulate that the resource must be referenced to be used; stipulate that resources can be processed in any order.
  - **8.5.3 Attributes:** add required `ResourceName` attribute; clarify that the `Name` attribute is for use in `SUPPLIED_RESOURCE_REF`; `Type` attribute has only two possible values (`Font` or `ProcSet`); add definition of ProcSet.

Version 1.5, May 31, 2001:

New features and substantial additions

- **Conformance subsets**
  - Add new Chapter 10, Conformance Subsets, particularly Section 10.2, Graphic Arts subset, with full definition of file formats and their constraints.
  - Add new `CONFORMANCE` element (Section 4.7) and `ResourcesIncluded` attribute on `PPML`.
- **Page Dimension information:** For non-imposing Consumers (see below), add new `PAGE_DESIGN` element (section 4.6); add corresponding text in `PAGE_LAYOUT`; deprecate the use of the `Dimensions` attribute on `DOCUMENT` and `PAGE`.

Additional changes and clarifications

- **Imposing and non-imposing Consumers:** clarify the term "imposition" as used in this specification (section 6.1.1) and update the boxed note in Section 6.1 regarding what features a Consumer may or may not support; add `SheetLayoutIncluded` attribute on `PPML`.
- **Enhanced REPEAT functionality** for imposing Consumers: in the `PageOrder` attribute of `CELL`, change the counter ***s*** to refer to sheets (not signatures) and add document counter ***d***.

Errata in Version 1.5

- Copyright date on table of contents page should be 2001
- DTD information in section 2.1.4 should refer to version 1.5
- Model for `PPML`:
  - Section 4.2.2 should say `CONFORMANCE?`, not `CONFORMANCE_LEVEL?`
  - Reference Card's model should say `CONFORMANCE?` not `CONFORMANCE`

Version 2.0, April 4, 2002:

New features and substantial additions

- **PPML Architecture**
  - Add new section 1.2, **PPML Architecture**, detailing and extending the potential uses and applications of future versions of PPML
- **Job ticketing**
  - Create separate spec document *PPML Job Ticketing*
  - New section 3.4: Add section for job ticketing in Definitions section
  - New sections 4.8 (`TICKET`) and 4.9 (`TICKET_REF`)

- Add TICKET_REF to models for PPML, JOB/DOCUMENT_SET, DOCUMENT, PAGE, MARK, REUSABLE_OBJECT and OCCURRENCE_LIST
- Add TICKET to model for PPML
- Change model for INTERNAL_DATA to ANY to accept job ticket data as content; change EXTERNAL_DATA Description and Context text to include use of EXTERNAL_DATA with TICKET.
- Refine wording of section 10.2 (Graphic Arts subset) regarding job ticketing
- Add **Appendix E, Job Ticketing Formats** to this document

- **Packaging jobs for transport:**
  - Add **Appendix D, Packaging** (incorporates Tech Note TN1 into the specification)

- **Schema support**
  - Rewrite section 2.1.4, "DTD and Schema"
  - Created XML schema for PPML 2.0

Additional changes and clarifications

- Correct all errata from Version 1.5 (see above)
- Improve accuracy of CDATA footnote in section 2.2, NonXML Data
- Deprecate JOB in favor of DOCUMENT_SET (section 4.3 and throughout document)
  This change was modified in version 2.1 – see below
- EXTERNAL_DATA, EXTERNAL_DATA_ARRAY, SEGMENT_ARRAY: Add Checksum and ChecksumType attributes
- SUPPLIED_RESOURCE: change model to INTERNAL_DATA | EXTERNAL_DATA; deprecate the Src attribute
- New **Appendix F, Embedding Text**
- Conformance subset strings (section 4.7): add instructions on how to submit strings to PODi
- Update **Appendix A, Acknowledgements**
- Add missing Level attribute to table in 4.7.3
- Remove duplicated section 5.20.2 (illustration of transformation and clipping)

Version 2.1, July 31, 2002:

- ***Refinements to Job Ticketing of page content elements***
  - Remove TICKET_REF from between content elements inside PAGE. (See model for Page, section 4.5.2.) Thus, the smallest level to which TICKET_REF applies is an entire Page.
  - Add TICKET_SET (an aggregation of TICKET_REFs, for convenience):
    - Define TICKET_SET in new section 4.10
    - Add to model for PPML, DOCUMENT_SET, DOCUMENT, PAGE
  - Definition of Reusable Objects:
    - In section 3.4, "Terms related to PPML Job Ticketing," add definition of Ticket State
    - Add new element TICKET_STATE (section 4.11) with description of Ticket State concept.
    - In Reusable Object definitions, move all TICKET_REFs down to the lowest level (the individual OCCURRENCE) and thus remove all inheritance of ticket info from individual page content elements. See section 4.11 for discussion.

      Specifically, remove TICKET_REF from model for REUSABLE_OBJECT and OCCURRENCE_LIST and add TICKET_STATE to model for OCCURRENCE.
- ***Additional changes***
  - JOB vs DOCUMENT_SET: un-deprecate JOB (section 4.3 and throughout). (Keep DOCUMENT_SET, but allow JOB as a synonym. *(This change avoids invalidating existing datasets that use* JOB.*)*
  - Clerical changes, including: remove "::" notation in "PAGE_DESIGN::TrimBox" (section 4.6.5); correct the name of job ticketing spec in front matter; fix multiplication signs in attribute tables in chapter 6

PPML (Personalized Print Markup Language) Reference Card

http://www.podi.org

Version 2.1 • July 31, 2002

See legend at bottom

Structure

Page ELEMENT	ATTRIBUTES	CONTAINS
15 PPML *The top level ("dataset" level) which encompasses all others*	Label, Creator, CreationDate, ResourcesIncluded, SheetLayoutIncluded	(CONFORMANCE*, TICKET?, SUPPLIED_RESOURCES?, REQUIRED_RESOURCES?, IMPOSITION*, (PRINT_LAYOUT | PAGE_DESIGN)?, PRIVATE_INFO*, (TICKET_REF | REUSABLE_OBJECT | SEGMENT_ARRAY | (DOCUMENT_SET | JOB))*)
16 DOCUMENT_SET (synonym: JOB) *A set of documents*	Label, DocumentCount	(SUPPLIED_RESOURCES?, REQUIRED_RESOURCES?, IMPOSITION*, (PRINT_LAYOUT | PAGE_DESIGN)?, PRIVATE_INFO*, (TICKET_REF | REUSABLE_OBJECT | SEGMENT_ARRAY | DOCUMENT)+)
17 DOCUMENT *A single Instance Document (one or more Pages)*	Label, Dimensions, PageCount, DocumentCopies	(SUPPLIED_RESOURCES?, REQUIRED_RESOURCES?, PAGE_DESIGN?, PRIVATE_INFO*, (TICKET_REF | REUSABLE_OBJECT | SEGMENT_ARRAY | PAGE)+)
18 PAGE	Label, Dimensions	(SUPPLIED_RESOURCES?, REQUIRED_RESOURCES?, PAGE_DESIGN?, PRIVATE_INFO*, TICKET_REF*, (REUSABLE_OBJECT | SEGMENT_ARRAY | MARK)*)
19 PAGE_DESIGN	TrimBox, BleedBox	EMPTY
21 CONFORMANCE	Subset, Level	EMPTY
22 TICKET	Format	(EXTERNAL_DATA | INTERNAL_DATA)
24 TICKET_REF	ExtIDRef, Ref	EMPTY
29 TICKET_SET	ID	TICKET_REF*
30 TICKET_STATE	*None*	TICKET_REF*

Page Content

34 MARK *Places content on a Page*	Position	((VIEW?, OBJECT+) | OCCURRENCE_REF | SEGMENT_REF)
36 VIEW	*None*	(TRANSFORM?, CLIP_RECT?)
37 TRANSFORM	Matrix	EMPTY
38 CLIP_RECT	Rectangle *(number x 4)*	EMPTY
39 OBJECT *A view of a source element*	Position *(within the enclosing element's coordinate space)*	(SOURCE, VIEW?)
40 SOURCE *One or more content data items*	Format, Dimensions, ClippingBox	((INTERNAL_DATA | EXTERNAL_DATA)+ | EXTERNAL_DATA_ARRAY)
42 EXTERNAL_DATA *Points to an external data source (file, URL, etc)*	Src *(URI)*, Checksum, ChecksumType, SourceUsage	EMPTY SourceUsage attribute = Single | Multiple | Unknown
43 EXTERNAL_DATA_ARRAY *Points to a multi-segment external data source*	Src (URI), Checksum, ChecksumType, Index, IndexUsage	EMPTY IndexUsage attribute = Single | Multiple | Unknown
44 INTERNAL_DATA *In-stream content data*	Label, Creator, CharacterSet, Encoding	(#PCDATA)
45 REUSABLE_OBJECT *A content object that will be used repeatedly, & how it will be viewed*	*None*	(OBJECT+, VIEW?, OCCURRENCE_LIST)
46 OCCURRENCE_LIST	*None*	(OCCURRENCE)+
47 OCCURRENCE *One View of a Reusable Object, with hints about frequency of use*	Name, Environment, Scope, Overwrite, Weight	(VIEW?, TICKET_STATE*)
51 OCCURRENCE_REF *Calls up a stored Occurrence for printing on a Page or Sheet*	Ref (the name of the saved Occurrence), Environment	EMPTY
54 SEGMENT_ARRAY *Creates a collection of reusable objects from a single source file*	ClippingBox, Dimensions, Environment, Format, IndexRange, Name, Overwrite, Scope, ~~Src (URI)~~, Checksum, ChecksumType, Weight	(VIEW?, (EXTERNAL_DATA | INTERNAL_DATA)?)
56 SEGMENT_REF *Calls up a member of a Segment Array for printing*	Environment, Index, Ref	EMPTY

Print Layout (page layout, imposition)

80 PRINT_LAYOUT	Ncopies, Collate (*Document* | Job | No)	(PAGE_LAYOUT, SHEET_LAYOUT?)
81 PAGE_LAYOUT *page size*	TrimBox, BleedBox, BoundingBox	EMPTY
83 SHEET_LAYOUT *sheet size, marks*	Hsize, Vsize, GangDocuments (Yes | *No*)	(SHEET_MARK | (PAGE_LAYOUT?, (IMPOSITION | IMPOSITION_REF)))*
84 SHEET_MARK	Position, Face (*Up* | Dn)	(OCCURRENCE_REF)
85 IMPOSITION *defines rules for Step & Repeat, and how to distribute pages in multi-sheet docs*	Name, Scope, Environment, Position & Rotation *(of all Signatures in this imposition)*	(SIGNATURE | REPEAT) (All Rotation attributes = *0* | 90 | 180 | 270)
87 IMPOSITION_REF *Recalls a stored Imposition*	Name, Environment; Position & Rotation *(overriding original attributes)*	EMPTY
88 SIGNATURE *assigns Pages of a Document to cell positions on a sheet; incl. cell spacing & marks*	Nrows, Ncols, PageCount	(CELL+, HOR_TRIM_MARKS?, VER_TRIM_MARKS?, HOR_GUTTER*, VER_GUTTER*, HOR_FOLD_MARKS*, VER_FOLD_MARKS*)
90 CELL *A single page position in a Signature, on the face-up or -down side of the sheet*	Row, Col, Face (*Up* | Dn), Rotation, PageOrder *(integer or expression)*	EMPTY
95 HOR_TRIM_MARKS *marks on a Signature at Cell's trim size*	MarkDist, AllowOnPage (Yes | *No*)	(OCCURRENCE_REF)
97 VER_TRIM_MARKS *marks on a Signature at Cell's trim size*	MarkDist, AllowOnPage (Yes | *No*)	(OCCURRENCE_REF)
98 HOR_GUTTER ***space to add between rows in a Signature***	BetweenRows (range of rows), Distance	EMPTY
100 VER_GUTTER *space to add between columns in a Signature*	BetweenCols (range of columns), Distance	EMPTY
101 HOR_FOLD_MARKS *optional marks on a fold line between rows*	BetweenRows (row numbers), MarkDist	(OCCURRENCE_REF)
102 VER_FOLD_MARKS *optional marks on fold line between cols*	BetweenCols (column numbers), MarkDist	(OCCURRENCE_REF)
103 REPEAT *Duplicates a Signature on a sheet. Note: can be nested in all three dimensions.*	Direction (Ver | Hor | Stack), Action (Duplicate | Increment), Order (*Ascending* | Descending), Count, Spacing, SpacingMethod	(REPEAT | SIGNATURE) (SpacingMethod attribute = *Gap* | Offset)

Production Specs

108 PRIVATE_INFO	Creator, Identifier, Encoding, CharacterSet	(#PCDATA)
109 REQUIRED_RESOURCES	*none*	(FONT*, EXTERNAL_DATA*, PROCESSOR*, SUPPLIED_RESOURCE_REF*)
110 FONT	FontName, Format	EMPTY
111 PROCESSOR	Format, Revision	EMPTY
112 SUPPLIED_RESOURCES	*none*	(SUPPLIED_RESOURCE+)
113 SUPPLIED_RESOURCE	Name, ResourceName, ~~Src (URI)~~, Format, Type, SubType, Scope	(INTERNAL_DATA | EXTERNAL_DATA)? Type attribute = Font | ProcSet
114 SUPPLIED_RESOURCE_REF	Name	EMPTY

Legend: *1. Required attributes are underlined. ~~Strikethrough~~ indicates nodes that are being deprecated.*
2. In a list of possible values, the default (if any) is italicized, e.g. "(Single \| Multiple \| Unknown)"
3. Indents indicate that an element's function is related to the unindented element above it. (The indent is only an aid to readability; it has no significance in the PPML language.)